



ACTAS JORNADAS SARTECO 2016

*Arturo Gonzalez-Escribano; Diego R. Llanos; Sergio Cuenca Asensi;
Jesús González Peñalver (Eds.)*



Actas Jornadas Sarteco 2016

ARTURO GONZALEZ-ESCRIBANO; DIEGO R. LLANOS; SERGIO CUENCA ASENSI;
JESÚS GONZÁLEZ PEÑALVER (EDS.)

Actas Jornadas Sarteco 2016


Ediciones Universidad
Salamanca

AQUILAFUENTE, 218



Ediciones Universidad de Salamanca y
de cada autor

Motivo de cubierta:
Diseñadora María Alonso Miguel
Diseño del logotipo de las Jornadas SARTECO 2016: Javier Fresno Bausela

1.º edición: septiembre, 2016

ISBN: 978-84-9012-626-4 (PDF)

Ediciones Universidad de Salamanca
www.cusal.es
cusal@usal.es

Realizado en España – Made in Spain

Todos los derechos reservados.

Ni la totalidad ni parte de este libro pueden reproducirse ni transmitirse sin permiso escrito de Ediciones Universidad de Salamanca

Obra sometida a proceso de
evaluación mediante sistema de revisión por pares a tenor de las normas del congreso



Usted es libre de: Compartir — copiar y redistribuir el material en cualquier medio o formato
Ediciones Universidad de Salamanca no revocará mientras cumpla con los términos:

 Reconocimiento — Debe reconocer adecuadamente la autoría, proporcionar un enlace a la licencia e indicar si se han realizado cambios. Puede hacerlo de cualquier manera razonable, pero no de una manera que sugiera que tiene el apoyo del licenciador o lo recibe por el uso que hace.

 NoComercial — No puede utilizar el material para una finalidad comercial.

 SinObraDerivada — Si remezcla, transforma o crea a partir del material, no puede difundir el material modificado.



Ediciones Universidad de Salamanca es miembro de la UNE
Unión de Editoriales Españolas www.une.es



Catalogación de editor en ONIX accesible en <https://www.dilve.es/>

Prólogo

La Sociedad de Arquitectura y Tecnología de Computadores (SARTECO) presenta una vez más las Jornadas SARTECO, las cuales integran las XXVII Jornadas de Paralelismo (JP2016) y las I Jornadas de Computación Empotrada y Reconfigurable (JCER2016), que unifican en un único evento previas jornadas denominadas Jornadas de Computación Empotrada y Jornadas de Computación Reconfigurable y Aplicaciones.

Además, en el contexto de las jornadas SARTECO se celebran también el VI Concurso de Programación Paralela y la segunda edición del Concurso “Tu tesis en 3 minutos” que pretende premiar los mejores trabajos de tesis recientes en el área.

La celebración conjunta de estas actividades y eventos de carácter científico-técnico constituye un referente nacional imprescindible para la comunidad científica agrupada en SARTECO. Estas jornadas reúnen a un nutrido grupo de investigadores, procedentes de diferentes universidades y centros de investigación, con el objeto de intercambiar experiencias, presentar y debatir resultados de investigación, facilitar colaboraciones y sinergias entre grupos, y potenciar nuestras oportunidades de transferencia tecnológica a la industria.

Este año las Jornadas se celebran en Salamanca, en el contexto del V Congreso Español de Informática (CEDI 2016), que engloba multitud de eventos y actividades relacionadas con la Informática, con el objeto de obtener una mayor visibilidad e impacto social.

Desde la organización de estas nuevas Jornadas SARTECO, os deseamos a todos y a todas, un fructífero trabajo y una placentera estancia en Salamanca.

Comités de coordinación

XVII Jornadas de Paralelismo

Ramón Bevide Palacios (UC)
Jesús Carretero Pérez (UC3M)
José Duato Marín (UPV)
Inmaculada García Fernández (UAL)
Antonio Garrido Del Solo (UCLM)
Emilio López Zapata (UMA)
Emilio Luque Fadón (UAB)
Pedro de Miguel Anasagasti (UPM)
Alberto Prieto Espinosa (UGR)
Francisco José Quiles Flor (UCLM)
Ana Ripoll Aracil (UAB)
Francisco Tirado Fernández (UCM)
Mateo Valero Cortés (UPC)
Victor Viñals Yúferas (UZ)

I Jornadas de Computación Em-potrada y Reconfigurable

Jesús González Peñalver (UGR)
Sergio Cuenca Asensi (UA)
Miguel A. Vega Rodríguez (UNEX)
Miguel Damas Hermoso (UGR)
Antonio Martínez Alvarez (UA)
Gustavo Sutter (UAM)
Ignacio Bravo (UAH)
José Torres (UV)
Jordi Carrabina (UPC)
Juan Suardiá (UPCT)
Jesús Barba Romero (UCLM)
Goituria Sagardui Mendieta (UMON-DRAGON)
Jorge Portilla Berruenco (UPM)

Comité de Organización

Diego R. Llanos Ferraris (UVa)
Arturo González Escribano (UVa)
Jesús González Peñalver (UGR)
Sergio Cuenca Asensi (UA)
Daniel Barba García (UVa)
Ana Moretón Fernández (UVa)
Eduardo Rodríguez Gutiez (UVa)

ÍNDICE

Parte I.- XXVII Jornadas de Paralelismo

Aplicaciones de la computación de altas prestaciones:

Una Adaptación Paralela Basada en ε-Dominancia del Algoritmo de las Luciérnagas para la Inferencia de Filogenias Multiobjetivo	
SERGIO SANTANDER-JIMÉNEZ, MIGUEL A. VEGA-RODRÍGUEZ	17
PFIRE: hacia un simulador de propagación de incendios forestales multiesquema paralelo	
NICOLÁS CHIARAVIGLIO, ÁNGEL FARGUELL, LUCAS PELEGRIN, ANA CORTÉS, TOMÁS MARGALEF	25
FARSITE vs. WRF-SFIRE: simulación de lapropagación de incendios forestales, una cuestión de precisión y de tiempo	
ÁNGEL FARGUELL, NICOLÁS CHIARAVIGLIO, ANA CORTÉS, TOMÁS MARGALEF, JOSEP RAMÓN MIRÓ	33
Algoritmo de pre-análisis para la estimación de movimiento en HEVC	
GABRIEL CEBRIÁN-MÁRQUEZ, JOSÉ LUIS MARTÍNEZ, PEDRO CUENCA	41
Robust High-speed Eye Pupil Tracking System on GPUs	
JUAN MOMPEÁN, JUAN L. ARAGÓN, PEDRO M. PRIETO, PABLO ARTAL	47
Stereo Matching using SGM on the GPU	
D. HERNANDEZ-JUAREZ, A. CHACÓN, A. ESPINOSA, D. VÁZQUEZ, J. C. MOURE, A. M. LÓPEZ	53
Acceso eficiente a memoria para selección de características multi-objetivo en GPUS	
JUAN JOSÉ ESCOBAR, JULIO ORTEGA, JESÚS GONZÁLEZ, MIGUEL DAMAS, CONSOLACIÓN GIL	61
Nueva Implementación Paralela en GPUs para la Extracción de Firmas Espectrales Puras con Información Espacial y Espectral	
LUIS IGNACIO JIMÉNEZ, JAVIER PLAZA, ANTONIO PLAZA, SERGIO SÁNCHEZ, GABRIEL MARTÍN	71
Clasificación de imágenes de teledetección mediante ELM kernel y perfiles morfológicos en GPU	
ALBERTO S. GAREA, DORA B. HERAS, FRANCISCO ARGÜELLO	81
Análisis del efecto del particionado en tiles sobre una versión paralela del codificador HEVC	
OTONIEL LÓPEZ, PABLO PÍNOL, HÉCTOR MIGALLÓN, VICENTE GALIANO, MANUEL P. MALUMBRES	91
MARL-Ped+Hitmap: aumentando la productividad de simulaciones basadas en agentes con una herramienta de arrays distribuidos	
EDUARDO RODRIGUEZ-GUTIEZ, FRANCISCO MARTINEZ-GIL, JUAN MANUEL ORDUÑA-HUERTAS, ARTURO GONZALEZ-ESCRIBANO	97
Migración portable y de altas prestaciones de aplicaciones MATLAB a C++: deconvolución esférica de datos de resonancia magnética por difusión	
JAVIER GARCÍA BLAS, J. DANIEL GARCÍA, MANUEL F. DOLZ, JESÚS CARRETERO, YASSER ALEMÁN, ERICK JORGE CANALES-RODRÍGUEZ	105

Tecnologías Grid, clúster, plataformas distribuidas y Big Data:

Towards full GPU virtualization in cloud infrastructures with rCUDA	
JOSE M. CLAVER, CARLOS PÉREZ CONDE, JUAN GUTIERREZ-AGUADO	113
Sentiment Analysis on Multilingual Tweets using Big Data Technologies	
RODRIGO MARTÍNEZ-CASTAÑO, JUAN C. PICHEL, PABLO GAMALLO	119
Evaluación del rendimiento de una implementación Cloud para un clasificador neuronal aplicado a imágenes hiperespectrales	
JUAN MARIO HAUT, MERCEDES PAOLETTI, JAVIER PLAZA, ANTONIO PLAZA	127
Aplicando un procedimiento de optimización paralelo <i>Teaching-Learning</i> para el enfoque automático de heliostatos	
N.C. CRUZ, J.L. REDONDO, J.D. ÁLVAREZ, M. BERENGUEL, P.M. ORTIGOSA	135
Estudio del rendimiento de plataformas de computación voluntaria para aplicaciones intensivas en datos	
SAÚL ALONSO MONSALVE, FÉLIX GARCÍA CARBALLEIRA, ALEJANDRO CALDERÓN MATEOS	141

Leveraging the Power of Big Data Tools for Large Scale Molecular Dynamics Analysis	
OMAR A. MURES, EMILIO J. PADRÓN, BRUNO RAFFIN	149
HPSEngine: motor de alto rendimiento y baja latencia para el procesamiento distribuido en tiempo real	
RAFAEL LEIRA, PAULA ROQUERO, CARLOS VEGA, IVÁN GONZALEZ, JAVIER ARACIL	157
Cloudification of a Legacy Hydrological Simulator using Apache Spark	
SIIVINA CAÑO-LORES, ANDREI LAPIN, PETER KROPP, JESÚS CARRETERO	167

Lenguajes, compiladores y herramientas de programación y ejecución paralela:

Revisiting Conventional Task Schedulers to Exploit Asymmetry in ARM big.LITTLE Architectures for Dense Linear Algebra	
LUIS COSTERO, FRANCISCO D. IGUAL, KATZALIN OLCOZ, ENRIQUE S. QUINTANA-ORTÍ, FRANCISCO TIRADO	175
Evaluación del Consumo Energético de la Memoria Transaccional Software en Procesadores Heterogeneos	
EMILIO VILLEGAS, ALEJANDRO VILLEGAS, ÁNGELES NAVARRO, RAFAEL ASENJO, OSCAR PLATA	185
Análisis comparativo del uso de STMs en códigos de reducción irregulares	
MANUEL PEDRERO, ELADIO GUTIÉRREZ, SERGIO ROMERO, ÓSCAR PLATA	193
Extendiendo rCUDA con soporte para copias P2P entre GPUs remotas	
CARLOS REAÑO, FEDERICO SILLA	203
Programación Paralela Estructurada del paradigma Divide y Vencerás. Una Propuesta Metodológica	
MARIO ROSSAINZ LÓPEZ, MANUEL I. CAPEL, FERNANDO HERNÁNDEZ POLO, IVO H. PINEDA TORRES	209
Extendiendo Paraldroid para la Generación Automática de Código OpenCL	
SERGIO AFONSO, ALEJANDRO ACOSTA, FRANCISCO ALMEIDA	219
Detección automática de los patrones de paralelismo Map y Farm en códigos secuenciales	
DAVID DEL RIO ASTORGA, MANUEL F. DOLZ, LUIS MIGUEL SANCHEZ, J. DANIEL GARCÍA	229
Técnica de ocultación de malware	
JAVIER CARRILLO MONDEJAR, JOSÉ LUIS MARTÍNEZ MARTÍNEZ	237

Evaluación de Prestaciones

Reduciendo el Tiempo de Ejecución de una Aplicación de Cálculo de Riesgos Financieros a través del uso de GPUs Virtuales	
JAVIER PRADES, BLESSON VARGHESE, CARLOS REAÑO, FEDERICO SILLA	245
Parallelization Capabilities of Open-Source HEVC Codecs	
DAVID GARCIA-LUCAS, GABRIEL CEBRIÁN-MÁRQUEZ, PEDRO CUENCA	255
Sintonización de la cuantización uniforme en compresores perceptuales de imagen basados en la transformada wavelet	
M. O. MARTÍNEZ-RACH, O. LÓPEZ GRANADO, P. PIÑOL PERAL, M. PÉREZ MALUMBRES	263
Uso de aceleradores remotos para mejorar las prestaciones de la librería FFTW	
SANTIAGO MISLATA, FEDERICO SILLA	271
Dynamic load balancing for an enhanced energy efficiency on heterogeneous systems	
ALBERTO CABRERA, ALEJANDRO ACOSTA, FRANCISCO ALMEIDA, VICENTE BLANCO	281
Una herramienta de benchmarking para compiladores de OpenACC	
DANIEL BARBA, ARTURO GONZÁLEZ-ESCRIBANO, DIEGO R. LLANOS	289
Propuesta arquitectónica para la ejecución de tareas en Apache Spark para entornos heterogéneos	
ESTEFANÍA SERRANO, JAVIER GARCÍA BLAS, JESUS CARRETERO, MÓNICA ABELLA	299
Methodology for the efficient execution of applications on multi-core HPC environments	
CARLOS RANGEL, ALVARO WONG, DOLORES REXACHS, EMILIO LUQUE	305
Planificación Simbiótica de Procesos en el IBM POWER8	
JOSUÉ FELIU, STIJN EYERMAN, JULIO SAHUQUILLO, SALVADOR PETIT	315
Entorno de planificación para sistemas de tiempo real con un procesador SMT comercial	
CLARA FURIÓ, JOSUÉ FELIU, JULIO SAHUQUILLO, SALVADOR PETIT, HOUCINE HASSAN	325

Redes y Comunicaciones:

Una nueva metodología para encaminamiento tolerante a fallos en topologías KNS ROBERTO PEÑARANDA, ERNST GUNNAR GRAN, TOR SKEIE, MARÍA ENGRACIA GÓMEZ, PEDRO LÓPEZ	335
Pérdida de señal en redes on-chip ópticas JAVIER CANO, JUAN-JOSÉ CRESPO, FRANCISO J. ALFARO-CORTÉS, JOSE L. SÁNCHEZ	345
Redes en malla BLE para comunicaciones inteligentes y colaborativas DIEGO HORTELANO, TERESA OLIVARES, M. CARMEN RUIZ, CELIA GARRIDO-HIDALGO	353
Implementación real de un sistema de gestión eciente de WSN basado en una arquitectura BLE para IoT CELIA GARRIDO-HIDALGO, TERESA OLIVARES, M. CARMEN RUIZ, DIEGO HORTELANO	361
TrustedNet: protocolo para la creación de redes espontáneas basadas en la confianza JOSÉ VICENTE SORRIBES, LOURDES PEÑALVER	369
Evaluación del Rendimiento de la Difusión de Mensajes Utilizando Parada Forzada en Redes Oportunistas JORGE HERRERA-TAPIA, ANDRÉS TOMÁS, ENRIQUE HERNÁNDEZ-ORALLO, PIETRO MANZONI, CARLOS TAVARES CALAFATE, JUAN CARLOS CANO	379
Arquitectura para integrar los vehículos eléctricos en las Smart Cities del futuro V. TORRES-SANZ, JULIO A. SANGÜESA, PIEDAD GARRIDO, FRANCISCO J. MARTINEZ	385
Un nuevo enfoque para la adaptación de caudal en transmisiones multicast en SDWN ESTEFANÍA CORONADO, ROBERTO RIGGIO, JOSÉ VILLALÓN, ANTONIO GARRIDO	391
Un Esquema de Colas Eficiente para Reducir HoL Blocking en Topologías Dragonfly con Encaminamiento Determinista Mínimo PEDRO YEBENES, JESÚS ESCUDERO-SAHUQUILLO, PEDRO J. GARCÍA, FRANCISCO J. QUILES	399
Integración de Herramientas de Simulación con OpenFabrics Software para el Modelado de Redes de Interconexión InfiniBand GERMAN MAGLIONE-MATHEY, PEDRO YEBENES, JESUS ESCUDERO-SAHUQUILLO, PEDRO J. GARCÍA, FRANCISCO J. QUILES	409
Modelado realista de una NoC fotónica en un entorno de simulación CMP detallado JOSÉ PUCHE, SERGIO LECHAGO, SALVADOR PETIT, MARÍA E. GÓMEZ, JULIO SAHUQUILLO	415
Arquitectura NoC Híbrida con Conmutación de Paquete-Circuito Dirigida por el Protocolo de Coherencia MIGUEL GORGUES ALONSO, JOSÉ FLICH CARDO	421
Guaranteeing latencies in DVFS-based NoCs under unbalanced traffic loads JOSÉ V. ESCAMILLA, JOSÉ FLICH, MARIO ROBERTO CASU	431
Evaluando un escenario de pruebas para el IoT entre la emulación y el uso de dispositivos reales JORGE E. LUZURIAGA, MARCO ZENNARO, JUAN CARLOS CANO, CARLOS CALAFATE, PIETRO MANZONI	441
Un recorrido por el análisis forense JUAN MANUEL CASTELO GOMEZ, JOSÉ LUIS MARTÍNEZ MARTÍNEZ	447

Arquitecturas del subsistema de memoria y almacenamiento secundario:

Particionado dinámico de cachés compartidas para maximizar la equidad entre tareas VICENT SELFA, JULIO SAHUQUILLO, MARÍA E. GÓMEZ, SALVADOR PETIT	455
Leveraging OSD+ Devices to Efficiently Implement Data Objects in FPFS JUAN PIERNAS, PILAR GONZÁLEZ-FÉREZ	461
Mecanismo de clasificación de páginas basado en el paso de tokens entre TLBs ALBERT ESTEVE, ALBERTO ROS, ANTONIO ROBLES, MARÍA E. GÓMEZ	471
Modelando de forma precisa el subsistema de memoria de una GPU FRANCISCO CANDEL, SALVADOR PETIT, JULIO SAHUQUILLO, JOSÉ DUATO	481

Gestión de los Reemplazos de Bloques Limpios en Protocolos de Coherencia Basados en Directorio	
RICARDO FERNÁNDEZ-PASCUAL, ALBERTO ROS, MANUEL E. ACACIO	489
Gestión de Contenidos en Caches Operando a Bajo Voltaje	
ALEXANDRA FERRERÓN, JESÚS ALASTRUEY BENEDÉ, DARÍO SUÁREZ GRACIA, TERESA MONREAL ARNAL, PABLO IBÁÑEZ, VÍCTOR VIÑALS YÚFERA	497
Arquitecturas del procesador, multiprocesadores y chips multinúcleo:	
Exploración y optimización energética de arquitecturas heterogéneas con el framework gem5	
MARCOS HORRO, GABRIEL RODRÍGUEZ, JUAN TOURIÑO	509
Irrevocabilidad Relajada para Memoria Transaccional Hardware	
RICARDO QUILSANT, ELADIO GUTIÉRREZ, EMILIO L. ZAPATA, ÓSCAR PLATA	517
Reduciendo el consumo dinámico de energía con Tag Filter Cache	
JOAN J. VALLS, JULIO SAHUQUILLO, ALBERTO ROS, MARÍA E. GÓMEZ	525
Docencia en Arquitectura y Tecnologías de Computadores:	
WepSIM: simulador integrado de microprogramación y programación en ensamblador	
JAVIER PRIETO CEPEDA, FÉLIX GARCÍA-CARBALLEIRA, ALEJANDRO CALDERON MATEOS, SAÚL ALONSO MONSAIVE	535
MIPSpfga: Una infraestructura para la enseñanza de asignaturas del área de Arquitectura y Tecnología de Computadores	
SARAH HARRIS, ROBERT OWEN, ENRIQUE SEDANO, DANIEL CHAVER	543
Parte II.- I Jornadas de Computación Empotrada y Regonfigurable	
Diseño de Sistemas:	
Implementación de técnicas de estimación de la presión arterial a partir de ECG y PPG	
JOSÉ M. BOTE, JOAQUÍN RECAS, ROMÁN HERMIDA, MARIAN DÍAZ-VICENTE	551
Versat, a Runtime Partially Reconfigurable Coarse-Grain Reconfigurable Array using a Programmable Controller	
JOÃO D. LOPES, RUI SANTIAGO, JOSÉ T. DE SOUSA	561
Ciberseguridad en robots autónomos: Análisis y evaluación multiplataforma del bastionado ROS	
FRANCISCO JAVIER RODRÍGUEZ LERA, VICENTE MATELLÁN, JESÚS BALSA, FERNANDO CASADO	571
Plataforma de Verificación para Circuitos Digitales basada en Dispositivos Reconfigurable	
C. A. TORRES CERNA, J. J. RAYGOZA PANDURO, E. C. BECERRA ÁLVAREZ, S. ORTEGA CISNEROS, J. RIVERA DOMINGUEZ	579
Arquitecturas, Diseños de Referencia y Plataformas:	
Análisis y evaluación de un módulo IME hardware para el codificador HEVC usando una plataforma SoC	
ESTEPANÍA ALCOCER, OTONIEL LÓPEZ-GRANADO, MANUEL P. MALUMBRES, ROBERTO GUTIÉRREZ	589
Using the ChipCflow Project as a learning tool for programming Dataflow Language	
JORGE LUIZ E SILVA, BRUNO DE ABREU SILVA	595
Diseño Novedoso de Neuronas Digitales de Impulsos en FPGA para procesamiento en Tiempo Real	
S. BARRIOS-DV, J. J. RAYGOZA-PANDURO, E. C. BECERRA-ÁLVAREZ, A. SALAMANCA-CHAVARIN, E. A. CASTELLANOS-ALVARADO	603
Arquitectura Paralela de Grano Fino para la Aceleración de Predicciones mediante Factorización Matricial	
ARTURO DURÁN DOMÍNGUEZ, JUAN ANTONIO GÓMEZ PULIDO, DAVID RODRÍGUEZ LOZANO	611

Aplicaciones y Retos de la Sociedad:

Positioning System for Recreated Reality Applications implemented on a multi-processing embedded system	
PATRICIA MARTÍNEZ MEDJAVILLA, EUGENIO VILLAR BONET	619
Intelligent machine tool monitoring and control system	
AITOR DUO ZUBIAURRE, MIREN ILLARRAMENDI REZABAL, ROSA BASAGOITI ASTIGARRAGA, PEDRO ARRAZOLA ARRIOLA, EXABIER HORMAETXE FERNANDEZ, JAVIER APERRIBAY ZUBIA	627
Prototipado de sistemas sobre hardware de bajo coste para la regulación automática de glucosa en diabéticos tipo 1	
CÉSAR VÁZQUEZ, IGNACIO BRAVO, ALFREDO GARDEL, JESÚS BERIÁN, JOSÉ LUIS LÁZARO	637
Monitorización de la Contaminación Ambiental Mediante Sensores Móviles	
WILLIAN ZAMORA, ÓSCAR AIVEAR, CARLOS T. CALAFATE, JUAN-CARLOS CANO, PIETRO MANZONI	645
Reconstrucción de jugadas de billar en 3D	
FCO. J RODRÍGUEZ-LOZANO, JOSÉ M. PALOMARES, J. OLIVARES	653
Consumo Energético de Aplicaciones Antivirus en Android	
ELSA VERA, JOSÉ ARTEAGA, JORGE PINCAY, WILLIAN ZAMORA, ALEX SANTAMARÍA	661
SmartWaterSigfox: IoT con Sigfox para la gestión de contadores de agua	
J. JIMÉNEZ-ORTEGA, M. DAMAS, F. GÓMEZ	667
Simulación de Tráfico Vehicular en base a Trazas Reales	
JORGE L. ZAMBRANO, CARLOS T. CALAFATE, DAVID SOLER, JUAN-CARLOS CANO, PIETRO MANZONI	673
DAGRI: Detección Automática de Grietas en Pavimentos	
A. CUBERO-FERNANDEZ, JOSE M. PALOMARES, JOAQUIN OLIVARES, R. VILLATORO, F. LEÓN	681
Conectividad de Sistemas:	
Propagation models in single/multi-radio scenarios: overview and influence analysis	
J.M. GARCÍA, F.J. ESTÉVEZ, G. REBEL, J.M. CASTILLO-SECILLA, J. GONZÁLEZ, P. GLÖSEKÖTTER	689
Reconocimiento de Patrones para Reglas Distribuidas en Sistemas de Bajas Prestaciones	
FERNANDO LEÓN, JOSE M. PALOMARES, JOAQUIN OLIVARES, A. CUBERO-FERNÁNDEZ	699
Merging WSN and IoT Technologies towards Smart Urban Networking	
GABRIEL MUJICA, JORGE PORTILLA, TERESA RIESGO	709
Algoritmo de Aprendizaje Automático para la Detección y Captura de Tráfico Usando el Sniffer Multicanal SnifferWSNTOS	
MIGUEL S. POLONIO, JOAQUÍN OLIVARES, JOSE M. PALOMARES, F. LEÓN, J.M. CASTILLO-SECILLA, A. CUBERO-FERNÁNDEZ	719
Sistema de Arquitectura Abierta: el Gesto Aplicado al Control Robótico	
GONZALO POMBOZA-JUNEZ, JUAN ANTONIO HOLGADO-TERRIZA	727
eK-Red de control, actuación y optimización de consumo eléctrico para enchufes	
JOSÉ MANUEL LÓPEZ MARTÍNEZ, JESÚS GONZÁLEZ PEÑALVER, ALBERTO PRIETO ESPINOSA	733
Sistema abierto para la adquisición de datos desde GCMs propietarios	
J. BERIÁN, A. GARDEL, I. BRAVO, C. VÁZQUEZ, J.L. LÁZARO	743
A Novel Indoor Localization Scheme for Autonomous Nodes in IEEE 802.15.4a Networks	
G. RÉBELY, J. GONZÁLEZY, P. GLÖSEKÖTTER, FRANCISCO ESTEVEZ, ADRIAN ROMERO	749

Parte I.- XXVII Jornadas de Paralelismo

Aplicaciones de la computación de altas prestaciones

Una Adaptación Paralela Basada en ε -Dominancia del Algoritmo de las Luciérnagas para la Inferencia de Filogenias Multiobjetivo

Sergio Santander-Jiménez¹ y Miguel A. Vega-Rodríguez¹

Resumen— La aplicación de técnicas multiobjetivo para resolver problemas de optimización biológicos representa una de las tendencias más importantes en el ámbito de la bioinformática. En este sentido, la combinación de paralelismo y computación bioinspirada representa una aproximación prometedora para abordar problemas de complejidad NP-completa en este área. En este trabajo, estudiamos diversas técnicas para mejorar una aproximación multiobjetivo basada en el algoritmo de las luciérnagas enfocada a la inferencia de hipótesis filogenéticas. Concretamente, integramos mecanismos basados en ε -dominancia y otras estrategias multiobjetivo para mejorar la calidad global de los frentes de Pareto generados por el algoritmo. La aproximación resultante es paralelizada con OpenMP con ánimo de explotar las capacidades de cómputo paralelo de un sistema multicore compuesto por 32 cores. Los experimentos efectuados sobre cuatro bases de datos biológicas dan cuenta de resultados paralelos y multiobjetivo significativos, confirmando la bondad de las estrategias aplicadas con respecto a la propuesta original y otros métodos biológicos de la literatura.

Palabras clave— Algoritmo de las Luciérnagas, Optimización Multiobjetivo, OpenMP, Bioinformática.

I. INTRODUCCIÓN

LOS recientes avances en desarrollo algorítmico y hardware han permitido la inclusión de supuestos más realistas en el modelado de sistemas biológicos. Como resultado, se ha producido un incremento significativo en el número de propuestas basadas en computación paralela y optimización multiobjetivo para resolver problemas bioinformáticos [1], [2]. La reconstrucción de historias filogenéticas que describan la evolución de los organismos vivos representa uno de los problemas NP-completos más relevantes en este contexto [3]. La literatura da cuenta del exitoso empleo de técnicas de computación multiobjetivo bioinspirada a este problema, principalmente enfocadas a la resolución de incongruencias durante el proceso de inferencia. Poladian y Jermin aplicaron optimización multiobjetivo con objeto de realizar análisis filogenéticos considerando fuentes conflictivas de información [4]. Otras propuestas trataron de abordar el problema de acuerdo a múltiple criterios de optimalidad. En este sentido, es posible resaltar los trabajos de Coelho et al. [5], quienes propusieron estrategias inmuno-inspiradas de acuerdo a criterios basados en distancias, y Cancino y Delbem [6], quienes diseñaron la herramienta PhyloMOEA

para realizar búsquedas filogenéticas de acuerdo a los principios de parsimonia y verosimilitud. El reto computacional adicional que supone el tratamiento multiobjetivo de este problema ha motivado la aplicación de computación paralela para minimizar los tiempos de ejecución. Por ejemplo, Cancino et al. propusieron en [7] esquemas de paralelización MPI y MPI+OpenMP para su propuesta, mientras que Santander-Jiménez y Vega-Rodríguez evaluaron diversos diseños paralelos evolutivos y de inteligencia de enjambre para clusters multicore en [8].

A la hora de resolver problemas de optimización multiobjetivo, el objetivo fundamental radica en encontrar un conjunto de soluciones que representen un compromiso entre $n \geq 2$ funciones objetivo [9]. El concepto de dominancia Pareto se usa frecuentemente para distinguir la calidad de las soluciones a lo largo del proceso de búsqueda. Dadas dos soluciones s_1 y s_2 , s_1 domina a s_2 si y solo si $\forall i \in [1, 2, \dots, n]$, $f_i(s_1)$ no es peor que $f_i(s_2)$ y $\exists i \in [1, 2, \dots, n]$ tal que $f_i(s_1)$ mejora a $f_i(s_2)$. En este contexto de optimización, se pueden definir dos propiedades fundamentales para medir la calidad de la salida devuelta por un algoritmo multiobjetivo: convergencia al frente de Pareto óptimo y diversidad de soluciones. El hecho de cumplir estos dos objetivos representa un problema complejo en escenarios de optimización reales, por lo que surgieron diversas estrategias para impulsar las capacidades de búsqueda de los algoritmos. Una aproximación representativa viene dada por la ε -dominancia [10], que extiende la dominancia Pareto de manera que no se permita que dos soluciones con una diferencia (o proporción) menor que ε_i en el i -ésimo objetivo se consideren dominadas la una a la otra. Este mecanismo puede resultar útil, por ejemplo, para identificar individuos prometedores en la población, considerándolos en las estrategias de aprendizaje del algoritmo para impulsar la generación de soluciones candidatas de mayor calidad.

En este trabajo estudiamos la aplicación de la ε -dominancia y otras estrategias para mejorar el algoritmo Multiobjetivo Firefly Algorithm (MO-FA) [8] para inferir filogenias conforme a los criterios de parsimonia y verosimilitud. Proponemos a su vez una adaptación paralela del algoritmo resultante, denominado ε -MO-FA, para explotar máquinas multicore con OpenMP [11]. El objetivo principal consiste en evaluar el rendimiento de la propuesta desde una perspectiva tanto paralela como multiobjetivo. Para ello, analizaremos en primer lugar la escalabilidad

¹Universidad de Extremadura, Departamento de Tecnología de los Computadores y de las Comunicaciones, Escuela Politécnica. Campus Universitario s/n, 10003, Cáceres, España. {sesaji, mavega}@unex.es

del algoritmo en un sistema de memoria compartida compuesto por 32 cores según las métricas de aceleración y eficiencia. En segunda lugar, introduciremos un estudio comparativo para evaluar ε -MO-FA con respecto a la propuesta original MO-FA usando tres métricas multiobjetivo: hipervolumen, relación de cobertura y espaciado. Este análisis experimental se efectuará sobre cuatro bases de datos de nucleótidos reales, realizando comparaciones con otros métodos filogenéticos de la literatura.

Este artículo se organiza del siguiente modo. La siguiente sección detalla la definición del problema y formula las funciones objetivo consideradas. La Sección III describe las características fundamentales de ε -MO-FA así como su paralelización con OpenMP. La Sección IV contiene la evaluación experimental de la propuesta. Finalmente, se incluyen conclusiones y líneas de trabajo futuras en la Sección V.

II. RECONSTRUCCIÓN FILOGENÉTICA

Los métodos de reconstrucción filogenética tratan de describir la historia evolutiva de las especies identificando relaciones ancestro-descendiente entre organismos vivos e hipotéticos. Una hipótesis filogenética viene dada por una estructura arborescente $T = (V, E)$. En ella, V representa el conjunto de nodos donde se identifica a los nodos internos como organismos ancestrales, mientras que los resultados del proceso evolutivo vienen dados por nodos hojas. Por su parte, E contiene las ramas empleadas para definir relaciones entre organismos a lo largo del curso de la evolución. Esta historia evolutiva es inferida a partir del procesamiento de las similitudes y divergencias observadas en un conjunto de N secuencias alineadas conteniendo M caracteres por secuencia correspondientes a las especies a estudiar (por ejemplo, secuencias de nucleótidos en el caso de análisis basados en ADN según el alfabeto $\Lambda = \{A, C, G, T\}$).

El proceso de inferencia se modela como un problema de optimización que versa en torno a la obtención de una filogenia óptima según un determinado criterio biológico (implementado como función objetivo). Sin embargo, diferentes funciones filogenéticas pueden dar lugar a relaciones conflictivas para un mismo conjunto de datos [12]. Para afrontar este problema, abordamos la reconstrucción filogenética desde una perspectiva multiobjetivo según dos funciones principales: parsimonia y verosimilitud. Por un lado, el principio de máxima parsimonia busca dar preferencia a la hipótesis evolutiva más simple, dada por la topología filogenética que minimice el número de cambios observados entre organismos emparentados (esto es, la filogenia que muestre un número mínimo de mutaciones por generación). La parsimonia de un árbol filogenético $T = (V, E)$ se calcula como [3]:

$$P(T) = \sum_{i=1}^M \sum_{(a,b) \in E} C(a_i, b_i), \quad (1)$$

donde $(a, b) \in E$ define la relación evolutiva entre dos nodos $a, b \in V$, a_i y b_i son los valores de estado

en el i -ésimo carácter de las secuencias de a y b , y $C(a_i, b_i)$ cuantifica la divergencia entre a_i y b_i .

Por su parte, las aproximaciones de máxima verosimilitud dirigen el proceso de inferencia de acuerdo a los supuestos dados por un modelo evolutivo, el cual define las probabilidades de observar eventos de mutación. La idea principal subyace en usar estas probabilidades para inferir la hipótesis evolutiva más probable que explique las características de los organismos de entrada. Dadas las probabilidades definidas por un modelo μ , la hipótesis de máxima verosimilitud viene descrita por el árbol evolutivo $T = (V, E)$ que maximice la siguiente expresión [3]:

$$L[T, \mu] = \prod_{i=1}^M \sum_{x, y \in \Lambda} \pi_x [P_{xy}(t_{ru}) L_p(u_i = y)] \times [P_{xy}(t_{rv}) L_p(v_i = y)], \quad (2)$$

donde π_x es la probabilidad estacionaria del estado $x \in \Lambda$, $P_{xy}(t)$ la probabilidad de mutación de x a un estado diferente y en un tiempo t , $r \in V$ el nodo raíz del árbol con descendientes $u, v \in V$, y $L_p(u_i = y)$, $L_p(v_i = y)$ las verosimilitudes parciales de observar y en el carácter i -ésimo de u y v , respectivamente.

Abordar la inferencia filogenética como un problema de optimización conlleva una dificultad elevada desde una perspectiva computacional. Esto es debido al hecho de que la reconstrucción de filogenias óptimas muestra una complejidad NP-completa. Las razones de dicha complejidad están vinculadas a las dimensiones N y M del conjunto de datos de entrada. En primer lugar, el tamaño del espacio de búsqueda filogenético depende del número de especies N de tal forma que se observa un crecimiento exponencial en el número de posibles topologías a medida que crece N . En segundo lugar, los tiempos de evaluación que implican el cálculo de las funciones objetivo dependen del tamaño M de las secuencias, experimentando un crecimiento lineal con M que puede resultar muy significativo en los alineamientos biológicos actuales. Estos dos problemas fundamentales explican por qué la inferencia filogenética es considerada como uno de los grandes retos computacionales en bioinformática, así como la necesidad de llevar a cabo análisis reales mediante técnicas de computación de altas prestaciones y computación bioinspirada.

III. UNA PROPUESTA PARALELA BASADA EN ε -DOMINANCIA

Para afrontar el problema de la filogenética, se propone una aproximación paralela que implementa ε -dominancia y otras técnicas multiobjetivo para mejorar las capacidades de búsqueda. Esta sección describe las características clave de la propuesta.

A. ε -Based Multiobjective Firefly Algorithm

El Firefly Algorithm (FA) [13] es un algoritmo bioinspirado que se fundamenta en las capacidades de bioluminiscencia de las luciérnagas. La idea básica

consiste en modelar las interacciones de estos organismos, basadas en patrones de atracción por emisiones de luz. La decisión de desplazarse hacia la posición de otra luciérnaga según este comportamiento depende de múltiples factores, tales como la intensidad de luz, la distancia entre luciérnagas y la absorción de luz por parte del entorno. En [8] describimos una primera adaptación multiobjetivo de este algoritmo (MO-FA) donde se aplicaba dominancia estricta para distinguir qué luciérnagas mostraban los patrones de luz más atractivos, esto es, qué soluciones mostraban mejor calidad desde un prisma multiobjetivo. A pesar de obtener soluciones de gran calidad (mejorando al estándar NSGA-II [14]), la forma general de los frentes de Pareto obtenidos presentaba margen de mejora atendiendo a su convergencia y diversidad. Este es el motivo por el cual proponemos una nueva adaptación, ε -MO-FA, basada en ε -dominancia para lograr capacidades de búsqueda mejoradas introduciendo mayor flexibilidad en los patrones de aprendizaje del algoritmo. Los parámetros de entrada de ε -MO-FA incluyen el número de luciérnagas en la población (*tamPop*), el número de evaluaciones dado como criterio de parada (*maxEval*), el factor de atracción (β_0), el coeficiente de absorción de luz (γ), un factor de aleatoriedad (α) y los valores épsilon en cada objetivo ($\varepsilon_1, \varepsilon_2$).

A fin de aplicar este algoritmo a filogenética, las soluciones serán representadas mediante matrices de distancia de tamaño $N \times N$, donde N es el número de especies de entrada. Estas estructuras contienen en cada entrada $m[x, y]$ una medida en coma flotante de la distancia evolutiva entre los organismos x e y . Esta codificación indirecta nos permite realizar búsquedas en un espacio matricial auxiliar cuyo procesamiento resulta apropiado mediante los operadores definidos en el diseño original del FA, aplicando a su vez el método de construcción de árboles BIONJ para inferir las topologías filogenéticas asociadas a dichas matrices. En el paso de inicialización de la población, se obtienen las matrices de partida mediante el procesamiento de topologías aleatoriamente seleccionadas de un repositorio generado mediante técnicas de *bootstrapping*, usando la librería bioinformática BIO++ [15] para codificar las estructuras necesarias.

En cada generación, se comparan las luciérnagas de la población según la ε -dominancia. Sea P_i una luciérnaga con matriz de distancias $P_i.m$ la cual es ε -dominada por otra luciérnaga P_j con matriz $P_j.m$ ($P_j \succ_\varepsilon P_i$). El procedimiento de atracción en ε -MO-FA genera una nueva solución P'_i desplazando P_i hacia P_j , calculando en un primer paso la distancia global δ_{ij} que separa a P_i y P_j :

$$\delta_{ij} = \sqrt{\sum_{x=1}^N \sum_{y=1}^x (P_i.m[x, y] - P_j.m[x, y])^2}. \quad (3)$$

Una vez calculado δ_{ij} , se computa la nueva matriz de distancias mediante la siguiente fórmula de movimiento, la cual viene gobernada por los parámetros de atracción β_0 , absorción γ y aleatoriedad α :

$$P'_i.m[x, y] = P_i.m[x, y] + \beta_0 e^{-\gamma \delta_{ij}} (P_j.m[x, y] - P_i.m[x, y]) + \alpha (\text{rnd}[0, 1] - \frac{1}{2}), \quad (4)$$

donde $\text{rnd}[0, 1]$ es un número aleatorio tomado de una distribución uniforme en el intervalo $[0, 1]$. Mientras que el segundo término de la fórmula denota el grado en que P_i aprende de P_j , el tercer término introduce aleatoriedad al movimiento para promover las capacidades de exploración del algoritmo. Estos pasos se repiten por cada luciérnaga en la población que ε -domine a P_i , de tal manera que la nueva matriz $P'_i.m$ es generada conforme a la información proporcionada por múltiples individuos, modelando de esta manera el comportamiento colectivo propio de los algoritmos de inteligencia de enjambre. Dado que la matriz resultante debe ser simétrica, se aplica el operador BLX- α [16] sobre aquellas entradas $m[x, y] \neq m[y, x]$. Posteriormente, se infiere y evalúa la filogenia correspondiente, repitiendo los cálculos de movimiento sobre cada luciérnaga ε -dominada de la población.

Al final de cada generación, las nuevas soluciones compiten con las luciérnagas originales con objeto de mantener las *tamPop* soluciones más prometedoras. Para ello, aplicamos los métodos de ordenación rápida no dominada y cálculos de distancia *crowding* para clasificar y ordenar nuestras soluciones mediante *rankings* Pareto y valores de densidad [14]. A continuación, actualizamos el frente de Pareto, comenzando la siguiente generación del algoritmo.

B. Diseño Paralelo

En el Pseudocódigo 1 presentamos un diseño paralelo basado en OpenMP de ε -MO-FA para sistemas multicore de memoria compartida. Al paralelizar este algoritmo, el reto principal desde una perspectiva paralela viene dado por el hecho de que existen dos fuentes de desbalanceo de carga en los movimientos de luciérnagas. En primer lugar, el procedimiento de atracción chequea la población completa bajo ε -dominancia, de manera que sólo se aplican movimientos sobre aquellas soluciones que son ε -dominadas por al menos una solución en la población. En segundo lugar, puesto que las soluciones pueden ser ε -dominadas por múltiples luciérnagas diferentes en la generación actual, los operadores de movimiento deben aplicarse un número variable de veces, dando como resultado tiempos de procesamiento variables para cada luciérnaga ε -dominada.

Nuestra propuesta paralela afronta estos dos problemas del siguiente modo. Al comienzo de cada generación, calculamos el número de luciérnagas ε -dominadas en la población (líneas 4-14 en el Pseudocódigo 1), almacenando sus identificadores así como los de las luciérnagas que las ε -dominan. Una vez identificadas las luciérnagas a procesar, aplicamos el bucle de movimiento sobre el número de luciérnagas ε -dominadas detectadas. De esta manera, es posible eliminar la condición *if* que gobernaba el cálculo de nuevas soluciones en el bucle de movimiento original de la versión serie, resolviendo la

Pseudocódigo 1 ε -MO-FA - Diseño OpenMP

```

1: #pragma omp parallel (num_hilos)
2: P, FrentePareto ← Inicializar (tamPop, num_hilos)
3: mientras ! criterio de parada (maxEval) hacer
4:   #pragma omp single
5:   idDominadas ← 0
6:   numeDominadas ← 0
7:   eDominantes ← 0
8:   para i = 1 hasta tamPop hacer
9:     si  $\exists P_j, P_k \succ_e P_i$  entonces
10:    idDominadas[numeDominadas] ← i
11:    eDominantes[j] ← eDominantes[j]  $\cup P_j$ 
12:    numeDominadas ← numeDominadas + 1
13:   fin si
14:   #pragma omp for schedule (dynamic)
15:   para i = 1 to numeDominadas hacer
16:     idDom ← idDominadas[i]
17:      $P_{\text{omPop}+i, m}$  ← Atraer Luciérnaga ( $P_{\text{idDom}, m}$ ,
18:     eDominantes[idDom],  $\beta_0, \gamma, \alpha$ )
19:      $P_{\text{omPop}+i, T}$  ← Inferir y Evaluar Árbol Filogenético
20:     ( $P_{\text{omPop}+i, m}$ )
21:   fin para
22:   #pragma omp single
23:   P ← Ordenación Rápida no Dominada y Cálculo Crowding (P, tamPop+numeDominadas)
24:   FrentePareto ← Actualizar Frente (P, FrentePareto)
24: fin mientras

```

primera fuente de desequilibrio de carga. El bucle de movimiento (líneas 15-20) es paralelizado mediante `#pragma omp for`, empleando una política de planificación dinámica para lidiar con la segunda fuente de desequilibrio, relacionada con el número cambiante de luciérnagas ε -dominantes que deben ser consideradas por iteración. Los pasos finales de una generación (líneas 21-23) se engloban en una directiva `#pragma omp single` debido a las dependencias de datos asociadas a la gestión de las estructuras de la población y el frente de Pareto. Obsérvese que este esquema de paralelización, en el que definimos una región paralela al comienzo y se aplican directivas `single / for` para definir fracciones de código serie / paralelo, tiene por objeto reducir el overhead por gestión de hilos que implicaría el uso continuado de `#pragma omp parallel for` dentro del bucle principal.

IV. RESULTADOS EXPERIMENTALES

Esta sección da cuenta de los resultados de los experimentos realizados para evaluar el rendimiento paralelo y multiobjetivo de ε -MO-FA. La experimentación llevada a cabo involucra el análisis de cuatro bases de datos reales de nucleótidos [6]: rbcL.55 (gen cloroplástico rbcL, N=55 secuencias, M=1314 nucleótidos por secuencia), mtDNA.186 (ADN mitocondrial humano, N=186, M=16608), RDPIL.218 (ARN procarionta, N=218, M=4182), y ZILLA.500 (gen rbcL, N=500, M=759). La configuración hardware considerada comprende dos procesadores de 16 cores AMD Opteron 'Abu Dhabi' 6376 (un total de 32 cores) a 2,3GHz con 48GB DDR3 RAM, usando Ubuntu 14.04 LTS y el compilador GCC 5.3.0.

Para evaluar el comportamiento de la propuesta en estos escenarios reales, se han usado diversos indicadores enfocados a medir rendimiento paralelo y calidad de soluciones. Por un lado, se han usado las métricas de aceleración (*speedup*) y eficiencia para estudiar la escalabilidad del algoritmo paralelo al considerar tamaños crecientes de problema y sistema,

usando como referencia los tiempos serie mostrados en la Tabla I. Por otro lado, la calidad multiobjetivo de los frentes de Pareto generados ha sido analizada mediante el empleo de 3 métricas multiobjetivo [9]: el hipervolumen I_H del espacio objetivo cubierto por las soluciones generadas, la relación de cobertura $SC(X, Y)$ que compara dos algoritmos X e Y mediante el cálculo de la fracción de soluciones en Y que son débilmente dominadas por X , y el espaciado SP entre soluciones del frente. La configuración de los parámetros de entrada de ε -MO-FA ha sido realizada mediante el estudio de distintos valores uniformemente distribuidos para cada uno, aplicando hipervolumen para cuantificar la calidad de las salidas generadas. Este estudio paramétrico apuntó a los siguientes valores óptimos: $tamPop=128$, $\beta_0=1$, $\gamma=0,5$, $\alpha=0,05$, y los valores $\varepsilon_1=0,005$ (objetivo de parsimonia) y $\varepsilon_2=0,0005$ (objetivo de verosimilitud). El criterio de parada fue establecido a 10000 evaluaciones y las ejecuciones fueron realizadas utilizando el modelo evolutivo $CTR + \Gamma$.

TABLA I: Tiempos serie (segundos) para ε -MO-FA

	rbcL.55	mtDNA.186	RDPIL.218	ZILLA.500
T_{exec}	5008,131	44927,917	45576,534	70179,939

A. Resultados Paralelos

La evaluación del rendimiento paralelo de ε -MO-FA se ha llevado a cabo considerándose tamaños crecientes del sistema (8, 16, 24 y 32 cores). Para cada configuración y conjunto de datos, hemos efectuado 11 ejecuciones independientes para obtener muestras estadísticamente relevantes de los tiempos de ejecución del algoritmo. La Tabla II muestra los factores de aceleración o *speedups* medianos (columnas 2, 4, 6 y 8) así como los valores de eficiencia (columnas 3, 5, 7 y 9) observados con respecto a los tiempos serie descritos en la Tabla I. Además, las columnas 10-11 en la Tabla II resumen el comportamiento medio del algoritmo según estas dos métricas de paralelismo, englobando los resultados observados en todas los conjuntos de datos estudiados.

En términos generales, nuestra propuesta paralela hace un uso efectivo de los recursos paralelos disponibles en el sistema multicore. Concretamente, se verifican aceleraciones en los rangos 7,2 - 7,8 (8 cores), 12,3 - 14,4 (16 cores), 16,3 - 19,8 (24 cores) y 19,6 - 24,0 (32 cores). Las eficiencias apuntan a que ε -MO-FA consigue una explotación satisfactoria de los recursos hardware, observándose en el conjunto de datos con mayor número de especies (ZILLA.500) un valor de eficiencia significativo del 75,1% cuando se emplea el sistema completo (32 cores). Es también destacable el hecho de que los factores de aceleración obtenidos están estrechamente relacionados a la complejidad de la instancia analizada, dado que un tamaño del problema creciente implica más cálculos sobre estructuras matriciales y arborescentes de mayor tamaño, motivando así un incremento de la

TABLA II: Factores de aceleración (speedup) y eficiencias (en %) obtenidas por ϵ -MO-FA

	rbcl_55		mtDNA_186		RDPIL_218		ZILLA_500		Media	
Cores	Speedup	Eficiencia	Speedup	Eficiencia	Speedup	Eficiencia	Speedup	Eficiencia	Speedup	Eficiencia
8	7,209	90,114	7,291	91,141	7,294	91,180	7,818	97,727	7,403	92,541
16	12,255	76,592	13,123	82,018	13,383	83,645	14,385	89,905	13,286	83,040
24	16,257	67,736	17,821	74,256	18,423	76,762	19,782	82,424	18,071	75,295
32	19,568	61,150	21,190	66,219	22,029	68,841	24,042	75,131	21,707	67,835

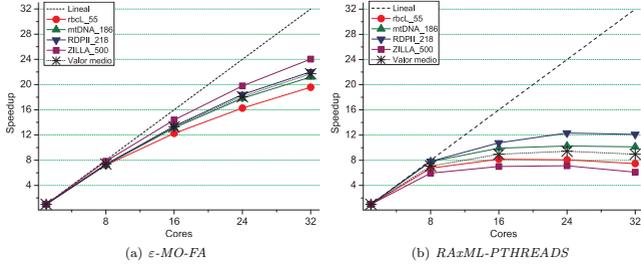


Fig. 1: Rendimiento paralelo - comparativa de factores de aceleración

fracción paralelizable de la aplicación.

Con ánimo de comprobar la bondad de estos resultados, a continuación presentamos comparativas con dos herramientas paralelas para reconstrucción filogenética: RAXML [17] (inferencia por máxima verosimilitud) y PhyloMOEA [7] (inferencia multiobjetivo). La Figura 1 muestra una comparativa gráfica de los *speedups* obtenidos por ϵ -MO-FA y RAXML en su versión multicore basada en hilos POSIX. Para configuraciones del sistema que involucran 16 o más cores, se puede observar una mejora significativa en el modo en que ϵ -MO-FA escala con respecto a RAXML-PTHREADS. De hecho, al emplear el sistema completo, ϵ -MO-FA es capaz de reportar un valor de *speedup* medio de 21,7, en comparación al valor de 8,95 observado en la herramienta de referencia. Con respecto a PhyloMOEA, la Tabla III introduce una comparativa con los *speedups* reportados en [7] (16 cores) para la versiones MPI y MPI+OpenMP de PhyloMOEA. Esta tabla da cuenta de cómo nuestra implementación paralela de ϵ -MO-FA muestra un mejor comportamiento desde una perspectiva paralela con respecto a las dos versiones del método multiobjetivo PhyloMOEA. En conclusión, este estudio comparativo con otras herramientas filogenéticas confirma que nuestra propuesta obtiene resultados paralelos significativos, dando lugar a una explotación precisa de los sistemas multicore actuales.

B. Resultados Multiobjetivo

A continuación afrontamos la evaluación de los frentes de Pareto inferidos por ϵ -MO-FA realizando comparaciones con nuestra propuesta original MO-FA. Para ello, hemos considerado los resultados medianos obtenidos de 31 ejecuciones independientes por conjunto de datos, los cuales son detallados en la

TABLA III: Comparativa de factores de aceleración con PhyloMOEA (16 cores)

	ϵ -MO-FA	PhyloMOEA MPI	PhyloMOEA MPI+OpenMP
rbcl_55	12,26	7,30	8,30
mtDNA_186	13,12	7,40	8,50
RDPIL_218	13,38	9,80	10,20
ZILLA_500	14,39	6,70	6,30

Tabla IV. En esta tabla se muestran los valores medianos de hipervolumen (junto con sus rangos intercuartílicos) obtenidos por cada algoritmo, los valores de espaciado que dan cuenta de la uniformidad en la distribución de soluciones de los frentes, y comparativas entre las salidas de los dos algoritmos según la relación de cobertura. En este sentido, obsérvese que valores mayores de hipervolumen y cobertura implican mejor calidad multiobjetivo, mientras que el espaciado es una métrica a minimizar. Además, la Figura 2 representa gráficamente los frentes de Pareto obtenidos por ϵ -MO-FA y MO-FA en las ejecuciones de hipervolumen mediano.

Atendiendo a la calidad global de los frentes inferidos, la Tabla IV apunta a que la nueva propuesta basada en ϵ -dominancia representa una mejora significativa sobre la adaptación original del algoritmo en todas las instancias analizadas. Si bien las diferencias en hipervolumen no resultan ser excesivamente altas, tanto el espaciado como la relación de cobertura dan soporte a la idea de que ϵ -MO-FA es capaz de obtener salidas más satisfactorias que MO-FA desde una perspectiva multiobjetivo. De hecho, el espaciado confirma que ϵ -MO-FA obtiene una relevante distribución de soluciones en los frentes de Pareto, solventando los problemas de diversidad presentados

TABLA IV: Rendimiento multiobjetivo - comparativa entre ϵ -MO-FA y MO-FA

	rbcL_55	mtDNA_186	RDPII_218	ZILLA_500
Hipervolumen				
$I_H(\epsilon\text{-MO-FA})$	71,55±0,01	70,02±0,01	74,81±0,13	73,00±0,08
$I_H(\text{MO-FA})$	71,47±0,08	70,00±0,01	74,73±0,08	72,96±0,02
Espaciado				
SP(ϵ -MO-FA)	0,097	0,078	0,028	0,037
SP(MO-FA)	0,128	0,103	0,033	0,051
Relación de cobertura				
SC(ϵ -MO-FA,MO-FA)	62,50 %	85,71 %	57,32 %	81,25 %
SC(MO-FA, ϵ -MO-FA)	30,00 %	18,75 %	22,12 %	13,24 %

TABLA V: Evaluación de resultados de parsimonia y verosimilitud

Instancia	Valor de parsimonia			Verosimilitud (GTR+ Γ)		Verosimilitud (HKY85+ Γ)	
	ϵ -MO-FA	TNT	PhyloMOEA	ϵ -MO-FA	RxML	ϵ -MO-FA	PhyloMOEA
rbcL_55	4874	4874	4874	-21782,64	-21788,57	-21813,81	-21889,84
mtDNA_186	2431	2431	2437	-39868,64	-39868,07	-39889,23	-39896,44
RDPII_218	41488	41488	41534	-134080,68	-134085,14	-134154,31	-134696,53
ZILLA_500	16218	16218	16219	-80568,24	-80599,77	-80967,27	-81018,06

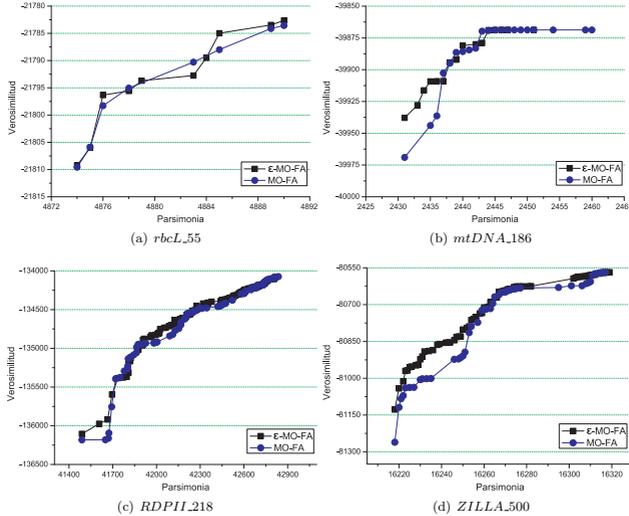


Fig. 2: Rendimiento multiobjetivo - Frentes de Pareto

originalmente por MO-FA. Por su parte, la relación de cobertura sugiere una mejora en la propiedad de convergencia de los frentes de ϵ -MO-FA, conforme al hecho de que la nueva propuesta es capaz de dominar, en término medio, porcentajes por encima del 70 % de las soluciones generadas por MO-FA. Estas dos ideas son confirmadas por la representación de frentes en la Figura 2, donde puede observarse una mejora notable tanto en la calidad de las soluciones obtenidas como en su distribución a lo largo del fren-

te. En conclusión, ϵ -MO-FA da lugar a una mejora exitosa en la calidad multiobjetivo de las soluciones, solventando los problemas de la versión original.

Tras analizar los resultados multiobjetivo, ahora nos centraremos en la evaluación de la calidad biológica de las soluciones inferidas mediante comparativas con otras herramientas del estado del arte. Para ello, la Tabla V proporciona los valores de parsimonia y verosimilitud de los puntos extremos de nuestros frentes de Pareto medianos y presenta una

comparación con los métodos TNT [18] (herramienta de parsimonia), RAxML (según el modelo GTR+ Γ) y PhyloMOEA (según el modelo HKY85+ Γ). Conforme a los resultados observados para parsimonia, podemos afirmar que ϵ -MO-FA iguala la calidad de TNT en todas las instancias analizadas, dominando a su vez a PhyloMOEA en mtDNA.186, RDPIL.218, y ZILLA.500. En lo que respecta a la verosimilitud, nuestra propuesta es capaz de generar resultados significativos atendiendo también a esta función filogenética, mejorando los valores de RAxML en tres conjuntos de datos y a PhyloMOEA en todos ellos. Por tanto, nuestra experimentación confirma la relevancia de las nuevas estrategias introducidas en ϵ -MO-FA para dar impulso adicional a sus capacidades de búsqueda, dando lugar a frentes de Pareto de calidad significativa tanto multiobjetivo como biológica.

V. CONCLUSIONES

En este trabajo, hemos afrontado el problema de la reconstrucción filogenética mediante una novedosa adaptación multiobjetivo del algoritmo de las luciérnagas, ϵ -MO-FA. El principal objetivo radicaba en mejorar las capacidades de búsqueda aplicando el concepto de ϵ -dominancia para dirigir los mecanismos de aprendizaje del método, junto con otras estrategias multiobjetivo para distinguir la calidad de las soluciones de una manera más precisa. Dada la dificultad del problema abordado, hemos propuesto un esquema paralelo basado en OpenMP para aprovechar las capacidades de cómputo paralelo de los sistemas multicore actuales. Los experimentos efectuados sobre cuatro bases de datos biológicas reales han arrojado luz sobre la bondad de la propuesta. Desde una perspectiva paralela, se ha conseguido una explotación significativa de una configuración hardware de 32 cores según las métricas de aceleración y eficiencia, mostrando escalabilidad mejorada con respecto a otras herramientas filogenéticas paralelas. Además, la evaluación de los frentes de Pareto inferidos según diversos indicadores de calidad multiobjetivo ha confirmado que las estrategias introducidas en ϵ -MO-FA permitan superar los problemas mostrados por la adaptación original del algoritmo, MO-FA. Finalmente, se ha confirmado la calidad a nivel biológico de las soluciones inferidas mediante comparativas con otros métodos del estado del arte.

Nuestras líneas de trabajo futuro tienen por objeto explorar de una manera más detallada la relación entre paralelismo y metaheurísticas multiobjetivo a la hora de resolver problemas NP-completos como la reconstrucción filogenética. En particular, llevaremos a cabo el análisis de múltiples diseños algorítmicos (incluyendo algoritmos evolutivos e inteligencia de enjambre), evaluando el impacto en rendimiento paralelo que implica el uso de diversas alternativas y estrategias multiobjetivo. Por otro lado, se estudiarán diseños paralelos avanzados para mejorar la calidad multiobjetivo y la escalabilidad a la hora de explotar sistemas hardware a gran escala.

AGRADECIMIENTOS

Los autores agradecen a la Universidad de Extremadura el apoyo económico ofrecido a Sergio Santander-Jiménez dentro del Plan de Iniciación a la Investigación, Desarrollo Tecnológico e Innovación 2015 (ACCION-III-04). Gracias a la Junta de Extremadura por la ayuda GRI5011 concedida al grupo de investigación TIC015.

REFERENCIAS

- [1] J. Handl, D. B. Kell, and J. D. Knowles, "Multiobjective Optimization in Bioinformatics and Computational Biology," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, no. 2, pp. 279–292, 2007.
- [2] A. Y. Zomaya, *Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies*, John Wiley & Sons Inc., Hoboken, New Jersey, 2006.
- [3] P. Lemey, M. Salemi, and A.-M. Vandamme, *The Phylogenetic Handbook: a Practical Approach to Phylogenetic Analysis and Hypothesis Testing*, Cambridge Univ. Press, Cambridge, 2009.
- [4] L. Poladian and L. Jermin, "Multi-Objective Evolutionary Algorithms and Phylogenetic Inference with Multiple Data Sets," *Soft Computing*, vol. 10, no. 4, pp. 359–368, 2006.
- [5] G. P. Coelho, A. E. A. Silva, and F. J. V. Zuben, "An Immune-Inspired Multi-Objective Approach to the Reconstruction of Phylogenetic Trees," *Neural Computing and Applications*, vol. 19, no. 8, pp. 1103–1132, 2010.
- [6] W. Cancino and A. C. B. Delbem, "A Multi-Criterion Evolutionary Approach Applied to Phylogenetic Reconstruction," in *New Achievements in Evol. Comp.*, pp. 135–156. InTech, 2010.
- [7] W. Cancino, L. Jourdan, E. G. Talbi, and A. C. B. Delbem, "Parallel Multi-Objective Approaches for Inferring Phylogenies," in *Evolutionary Computation, Machine Learning and Data Mining in Computational Biology*, 2010, vol. 6023 of *LNC3*, pp. 26–37, Springer-Verlag.
- [8] S. Santander-Jiménez and M. A. Vega-Rodríguez, "Parallel Multiobjective Metaheuristics for Inferring Phylogenies on Multicore Clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1678–1692, 2015.
- [9] C. Coello, C. Dhaenens, and L. Jourdan, *Advances in Multi-Objective Nature Inspired Computing*, Springer-Verlag, Berlin Heidelberg, 2010.
- [10] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, "Combining Convergence and Diversity in Evolutionary Multi-Objective Optimization," *Evolutionary Computation*, vol. 10, no. 3, pp. 263–282, 2002.
- [11] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, The MIT Press, Cambridge, Massachusetts, 2007.
- [12] J. R. Macey, "Plethodontid salamander mitochondrial genomics: A parsimony evaluation of character conflict and implications for historical biogeography," *Cladistics*, vol. 21, no. 2, pp. 194–202, 2005.
- [13] X.-S. Yang, "Firefly algorithm, Stochastic Test Functions and Design Optimisation," *International Journal of Bio-Inspired Computation*, vol. 2, no. 2, pp. 78–84, 2010.
- [14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.
- [15] J. Duthell et al., "Bio++: a set of C++ libraries for sequence analysis, phylogenetics, molecular evolution and population genetics," *BMC Bioinformatics*, vol. 7, pp. 188–193, 2006.
- [16] L. Poladian, "A GA for maximum likelihood phylogenetic inference using neighbour-joining as a genotype to phenotype mapping," in *Genetic and Evolutionary Computation Conference*, 2005, pp. 415–422.
- [17] A. Stamatakis, "RAxML Version 8: A Tool for Phylogenetic Analysis and Post-Analysis of Large Phylogenies," *Bioinformatics*, vol. 30, no. 9, pp. 1312–1313, 2014.
- [18] P. A. Goloboff, J. S. Farris, and K. C. Nixon, "TNT, a free program for phylogenetic analysis," *Cladistics*, vol. 24, no. 5, pp. 774–786, 2008.

PFIRE: Hacia un simulador de propagación de incendios forestales multiesquema paralelo

Nicolás Chiaraviglio¹, Àngel Farguella¹, Lucas Pelegrin¹, Ana Cortés¹, Tomàs Margalef¹

Resumen— Los incendios forestales producen anualmente la destrucción de miles de hectáreas de bosque y provocan importantes pérdidas ecológicas, económicas e incluso la pérdida de vidas humanas. Para mitigar los daños causados por los mismos resulta indispensable disponer de predicciones de la propagación de los incendios que resulten fiables y estén disponibles en el menor tiempo posible. En este contexto los simuladores de propagación de incendio juegan un rol fundamental y son ampliamente estudiados. En este trabajo se presenta el estado actual del desarrollo de PFIRE, un simulador paralelo multiesquema concebido para reducir el tiempo de ejecución y ofrecer una buena escalabilidad, y facilitar su acoplamiento con modelos meteorológicos.

Palabras clave— PFIRE, Simulador de incendios, FARSITE, WRF-SFIRE

I. INTRODUCCIÓN

LOS incendios forestales causan cada año la destrucción de miles de hectáreas de bosque y generan grandes pérdidas ecológicas, económicas y sociales. Por este motivo, la propagación de incendios forestales es estudiada desde los años 20 del siglo pasado y se han desarrollado desde hace más de 50 años diferentes modelos de propagación [1], [2]. A lo largo de los años, se han ido desarrollando otros modelos, pero el problema es muy complejo y, en él intervienen parámetros que en ciertos casos son difíciles de conocer, o tan siquiera estimar, de modo que no se han conseguido llevar a sistemas operacionales. Así, en la actualidad, el modelo de propagación más utilizado como núcleo de simulación sigue siendo el modelo propuesto por Richard Rothermel en 1972 [2], el cual es un modelo semi-empírico.

Este modelo fue implementado en distintos simuladores utilizando diferentes métodos de resolución entre los que pueden citarse principalmente los autómatas celulares [3], [4], método de la curva de nivel [5] y método de Huygens [6]. Cada uno de estos esquemas resolutivos presenta ciertas ventajas respecto a los anteriores y ciertos inconvenientes. En la bibliografía pueden encontrarse revisiones exhaustivas respecto a los distintos simuladores existentes y al modo de resolución utilizado por cada uno [7], [8], [9], [10], [11].

La principal complejidad del problema de predecir la propagación de incendios forestales radica en que se trata de un problema multifísica (combustión-meteorología) con parámetros de entrada de difícil obtención, como son los mapas de combustible, uso del suelo, condiciones de humedad del terreno, con-

diciones meteorológicas locales, condiciones de la vegetación, etc. Es por esto que se han utilizado distintos métodos de calibración de parámetros de entrada [12], [13], [14] para mejorar los resultados obtenidos con estos simuladores.

El grupo HPCA4SE del departamento de Arquitectura de Computadores y Sistemas Operativos de la Universidad Autónoma de Barcelona cuenta con una larga trayectoria en la utilización y adaptación de diferentes simuladores de propagación de incendios. Sobre esta base se ha comenzado el desarrollo de PFIRE, un simulador diseñado para utilizar diferentes esquemas de resolución, de fácil paralelización y que permita un fácil acoplamiento con modelos meteorológicos y métodos de calibración de parámetros y reducción de incertidumbres. A su vez, contar con una herramienta propia permite evaluar de forma más ágil posibles mejoras a incorporar en los algoritmos resolutivos.

Este trabajo se encuentra estructurado de la siguiente forma: En la sección II se describe el modelo de Rothermel y se presentan dos de los simuladores más utilizados: FARSITE y WRF-SFIRE. En la sección III se describe el simulador PFIRE, su modelo de propagación, los algoritmos resolutivos utilizados, la información requerida de entrada por el simulador y las salidas generadas por el mismo. Finalmente se discuten los métodos de optimización utilizados para acelerar el simulador.

II. SIMULADORES DE PROPAGACIÓN

A continuación se describe brevemente el modelo de propagación de Rothermel, así como también dos simuladores ampliamente utilizados en la comunidad, como son FARSITE [15] y WRF-SFIRE [16]. FARSITE implementa como método de propagación del frente el método de Huygens, mientras que WRF-SFIRE implementa el método de la curva de nivel (*Level Set Method*). No se analizan simuladores que utilizan autómatas celulares, ya que los mismos tienden a generar perímetros poco realistas y su uso se está discontinuando dentro de la comunidad.

A. Modelo de propagación

El modelo de Rothermel postula que la velocidad de propagación de un incendio responde a la ecuación:

$$R = R_0(\vec{n} + \vec{\phi}_w + \vec{\phi}_s) \quad (1)$$

donde R_0 es la velocidad de propagación en ausencia de viento y pendiente, \vec{n} es la dirección normal al

¹Dpto. de Arquitectura de Computadores y Sistemas Operativos, Univ. Autónoma de Barcelona. e-mail: nicolas.chiaraviglio@uab.cat.

perímetro del incendio, $\vec{\phi}_w$ es el factor de viento y ϕ_w es el factor de pendiente.

Este modelo requiere el conocimiento previo de distintas propiedades del combustible y del ambiente en el cual se propaga el incendio. El conjunto de las variables necesarias se muestra en la tabla I. A su vez, en la tabla II se muestran todas las ecuaciones del modelo, necesarias para el cálculo de la velocidad de propagación del frente. Una deducción de las mismas puede encontrarse en [2].

Puede verse como es necesario aportar información de difícil obtención como la relación área-volumen del combustible, su profundidad, etc. Para subsanar esto se generaron 13 modelos de vegetación [17], los cuales se distribuyen con los simuladores.

B. FARSITE

FARSITE es el simulador más utilizado en la comunidad. Resuelve las ecuaciones de Rothermel y expande un perímetro inicial, utilizando el método de Huygens. A su vez incorpora modelos de Spotting [18], consumo de combustible fuera del frente [19], [20], humedad del combustible [21], inicio de fuego de copas [22] y propagación de fuego de copas [23], y permite incorporar información meteorológica (viento, precipitaciones, humedad, temperatura) pero no realiza ningún tipo de realimentación. Es decir, no contempla la variación del campo de viento debida a las irregularidades del terreno ni realimenta las condiciones atmosféricas debido al fuego.

Para lograr un campo de vientos más realistas Potter et. al [24] utilizan WINDNinja [25], logrando contemplar las variaciones debidas a la orografía del terreno, pero sin captar el comportamiento dinámico debido a las alteraciones producidas por el mismo incendio.

Para mantener coherencia en los perímetros FARSITE utiliza un algoritmo costoso, publicado por Richards [26], el cual consume un 60% del tiempo de ejecución del programa [27]. Esto resulta una limitación cuando se quiere utilizar este simulador para hacer predicciones en tiempo real. Diversas técnicas de paralelización han sido probadas [27], pero la mejora del rendimiento no es lineal con el aumento de los recursos utilizados para su ejecución.

A pesar de las limitaciones mencionadas FARSITE es ampliamente utilizado en la comunidad y los resultados generados por el mismo son utilizados para evaluar la evolución de incendios forestales.

Como se mencionó FARSITE utiliza el método de propagación de Huygens. Para la implementación del mismo se debe expandir cada punto del perímetro en función de la velocidad de propagación encontrada en dicho punto. La dirección de propagación es una composición de la dirección normal a la curva sumada con los coeficientes de viento y pendiente.

Estos vectores de propagación para cada punto son compuestos en elipses con foco en el punto que se está propagando. Luego, los puntos extremos de cada elipse son los nuevos puntos del perímetro de incendio; esto se muestra esquemáticamente en la figura

1. A medida que el perímetro se expande los puntos propagados comienzan a alejarse entre sí por lo que debe realizarse una rediscritización del perímetro incrementando la cantidad de puntos de propagación en el frente. Por este motivo, para incendios grandes, el tiempo de resolución utilizando este método se incrementa considerablemente.

En el método de Huygens, la representación computacional de un perímetro viene dada por una serie de puntos ordenados que se conectan entre sí. Al utilizar este método para expandir estos puntos, surge el problema de que las elipses utilizadas para propagar los mismos se cruzan y al generar el perímetro resultante pueden formarse curvas sin sentido si no se reacomodan los puntos de forma adecuada. Un ejemplo de esto se muestra en la figura 2.

Para evitar esto, deben ejecutarse algoritmos de consistencia del perímetro que reordenen los puntos, o en caso que sea necesario, los eliminen. En FARSITE, esto se resuelve utilizando el algoritmo propuesto por Richards [26], el cual es muy costoso computacionalmente (N^2 , donde N es la cantidad de puntos sobre el perímetro).

C. WRF-SFIRE

WRF-SFIRE es un simulador que utiliza el método de la curva de nivel para determinar la propagación de incendios. Es decir, se define una función de nivel $\psi = \psi(\vec{x}, t)$ la cual define la región quemada según:

$$\begin{cases} \psi(\vec{x}, t) \leq 0 & \text{Área quemada} \\ \psi(\vec{x}, t) = 0 & \text{Frontera del fuego} \\ \psi(\vec{x}, t) > 0 & \text{Área sin fuego} \end{cases} \quad (2)$$

La evolución de esta función de nivel responde a la ecuación diferencial en derivadas parciales:

$$\frac{\partial \psi}{\partial t} + R|\nabla \psi| = 0 \quad (3)$$

la cual, en WRF-SFIRE, es resuelta por diferencias finitas. A su vez se incluye un modelo meteorológico, como el utilizado en el WRF [28] y realimentaciones por los flujos de calor generados por el incendio. Todo esto se especifica en detalle en [16].

Los resultados obtenidos mediante WRF-SFIRE son más realistas que los obtenidos con FARSITE, pero al utilizar un modelo meteorológico resulta mucho más costoso su cálculo incrementando considerablemente los tiempos de ejecución.

A su vez, la ejecución se vuelve más costosa al incrementar el tamaño del mapa base, ya que la solución debe calcularse para cada paso de tiempo sobre todos los puntos del espacio. Esto no ocurre al utilizar el método de Huygens dado que se expanden puntos sobre el perímetro y no se resuelve simultáneamente para todo el terreno.

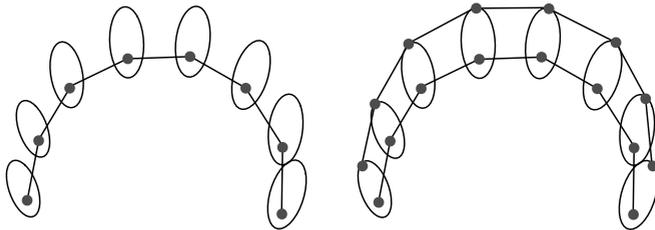
La versión disponible de WRF-SFIRE se encuentra paralelizada con MPI y OpenMP para reducir el impacto de estos factores. Sin embargo, estas limitaciones hacen que hoy en día este código no haya sido utilizado para realizar predicciones en tiempo real al momento de detectar un incendio.

TABLA I: Parámetros de entrada del modelo de Rothermel.

Variable	Descripción
w_0	Carga de combustible seco [lb/ft]
δ	Profundidad del combustible [ft]
σ	Relación área-volumen del combustible [ft]
h	Energía contenida en el combustible [B.t.u./lb]
ρ_P	Densidad del combustible seco
M_f	Humedad del combustible [lb humedad / lb comb. seco]
S_T	Contenido mineral del combustible [lb de mineral / lb comb. seco]
S_e	Contenido mineral efectivo del comb. [lb de mineral sin sílica / lb comb. seco]
U	Velocidad del viento a 6.1 m [ft/min]
$\tan\phi$	Pendiente
M_x	Humedad de extinción [%]

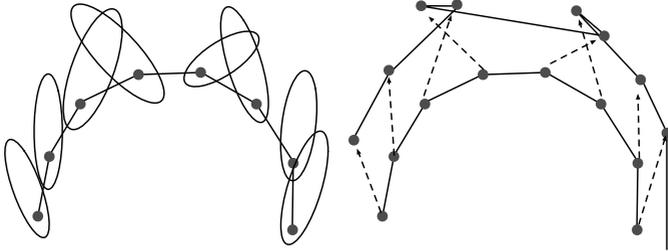
TABLA II: Ecuaciones del modelo de Rothermel

Ecuación	Descripción
$R = \frac{I_R \xi (1 + \Phi_w + \Phi_s)}{\rho_b Q_{1g}}$	Velocidad de propagación [ft/min]
$I_R = \Gamma' w_n h \eta_M \eta_S$	Intensidad de la reacción [B.t.u./ft ² min]
$\Gamma' = \Gamma'_{Max} (\beta / \beta_{op})^A e^{A(1 - \beta / \beta_{op})}$	Velocidad óptima de la reacción [1/min]
$\Gamma'_{Max} = \sigma^{1.5} (495 + 0.594 \sigma^{1.5})^{-1}$	Velocidad máxima de la reacción [1/min]
$\beta_{op} = 3,348 \sigma^{-0.8189}$	Relación de empaque óptimo
$A = 1 / (4,774 \sigma^{0.1} - 7,27)$	
$\eta_M = 1 - 2,59 \frac{M_f}{M_x} + 5,11 (\frac{M_f}{M_x})^2 - 3,52 + (\frac{M_f}{M_x})^3$	Coefficiente de amortiguación de humedad
$\eta_S = 0,174 S_e^{-0.19}$	Coefficiente de amortiguación mineral
$\xi = (192 + 0,2595 \sigma)^{-1} e^{(0,792 + 0,681 \sqrt{\sigma})(\beta + 0,1)}$	Relación de propagación
$\Phi_w = C U^B (\beta / \beta_{op})^{-E}$	Coefficiente de viento
$C = 7,47 e^{-0,133 \sigma^{0.25}}$	
$B = 0,02526 \sigma^{0.54}$	
$E = 0,715 e^{-3,59 \times 10^{-4} \sigma}$	
$w_n = w_0 / (1 + S_T)$	Carga neta de combustible [lb/ft ²]
$\Phi_S = 5,275 \beta^{-0.3} (\tan\phi)^2$	Coefficiente de pendiente
$\rho_b = w_0 / \delta$	Densidad aparente del combustible [lb/ft ³]
$\epsilon = e^{-138 / \sigma}$	Número de calentamiento efectivo
$Q_{1g} = 250 + 1,116 M_f$	Calor de preignición [B.t.u. / lb]
$\beta = \rho_b / \rho_P$	Relación de empaque



(a) Perímetro previo a la expansión y elipses formadas (b) Perímetro resultante luego de combinar la expansión por el método de Huygens.

Fig. 1: Método de expansión de perímetros de Huygens.



(a) Elipses generadas por el método de Huygens. Puede verse como 4 elipses se cruzan entre sí. (b) Perímetro resultante de unir los puntos propagados en donde se perciben los cruzamientos.

Fig. 2: Cruzamientos que se presentan al utilizar el método de expansión de perímetros de Huygens.

C.1 Método de propagación del frente en WRF-SFIRE

Este método, a diferencia del de Huygens, consiste en la resolución de una ecuación diferencial para cada punto de la grilla. De modo que el área quemada se obtiene como aquel conjunto de elementos de la malla en los cuales la curva de nivel toma un valor negativo. Esta información puede ser fácilmente integrada en un archivo raster. El método de la curva de nivel no requiere algoritmos de consistencia de perímetro sino que el mismo surge de considerar aquellos puntos donde $\psi(\vec{x}, t) = 0$.

III. PFIRE

PFIRE nace tras 18 años de trabajo en el grupo de investigación HPCA4SE del departamento de Arquitectura de Computadores y Sistemas Operativos de la UAB. La experiencia obtenida indica que un simulador versátil, de fácil utilización y codificado de forma modular, resulta fundamental para desarrollar nuevos algoritmos.

Por otro lado se busca simplificar considerablemente la generación de los archivos necesarios para ejecutar el simulador.

En la figura 3 se muestra el diseño de PFIRE, en la que se puede observar que es completamente modular y multiesquema, implementando los métodos de Huygens y de la curva de nivel. A continuación se detallan las implementaciones de cada uno de los métodos.

A. Métodos de propagación del frente en PFIRE

Como se mencionó anteriormente PFIRE implementa tanto el método de Huygens como el de la curva de nivel. Para agilizar el primero en PFIRE se propone un método más sencillo de resolución de perímetros que el que incorpora FARSITE, basado en la detección de variaciones bruscas del perímetro computando los ángulos α_i entre segmentos sucesivos como se muestra en la Figura 4b. De esta forma,

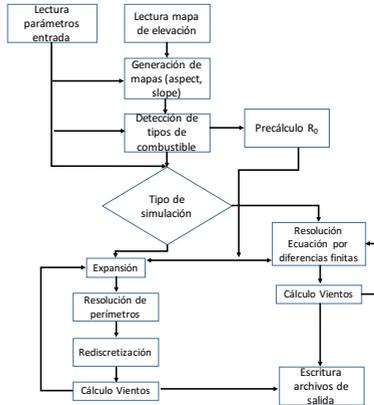


Fig. 3: Esquema de ejecución de PFIRE.

se asume que existe un cruzamiento para aquellos segmentos cuyos ángulos cumplan que:

$$\cos(\alpha_i) > \text{umbral} \quad (4)$$

donde el valor umbral dependerá de las variaciones que se permitan al perímetro.

Al detectar un cruzamiento se intercambian las posiciones que ocupan dentro del perímetro los puntos detectados y se verifica que los nuevos segmentos definidos por estos puntos no cumplan la condición que se muestra en la ecuación 4.

En PFIRE se utiliza una lista enlazada circular para representar el perímetro de incendio. Esta representación facilita la rediscretización del perímetro e intercambio de nodos respecto a la utilización de vectores para representar las coordenadas.

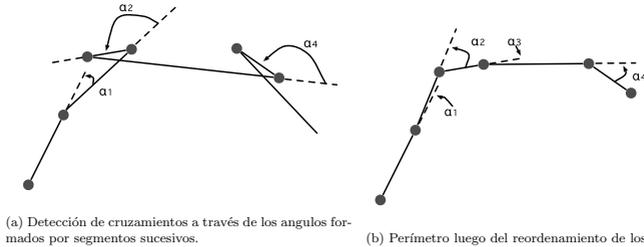


Fig. 4: Perímetro antes y después del reordenamiento. Puede verse como luego de reordenar los ángulos entre segmentos sucesivos serán siempre menores a 90° .

Por lo que se refiere al método de propagación basado en curvas de nivel, en PFIRE este algoritmo se implementará de forma análoga a como se implementó en WRF-SFIRE, utilizando diferencias finitas.

B. Datos de entrada

Tanto para FARSITE, como para WRF-SFIRE, la preparación de los archivos de entrada requiere recopilar tanto información topográfica (elevación, pendiente, aspecto de la pendiente, mapa de combustibles), como meteorológica. En el caso de FARSITE, además, deben combinarse estos datos en un archivo de landscape cuya edición no resulta posible por ser un binario de formato no abierto.

PFIRE se diseñó de forma que el proceso de creación de los inputs sea reducido y que la información requerida no dependa del método de resolución utilizado. PFIRE será distribuido con un mapa de elevaciones y vegetación de toda Europa con una resolución de 100 m^2 . De esta forma puede realizarse una simulación simplemente ingresando un perímetro de ignición el cual puede ser ingresado en formato ESRI Shapefile o bien mediante un archivo KML.

Sin embargo este no es el único modo de ejecución sino que puede especificarse el mapa totalmente por el usuario (por ejemplo si se quisiera trabajar a una resolución diferente). En este caso PFIRE requiere sólo el mapa de elevación en formato GEOTIFF o ASC y a partir del mismo se calculan los datos de pendiente y aspecto.

Por otro lado, usualmente se utilizan mapas homogéneos de vegetación debido a la escasa información. Es por esto que en PFIRE se puede especificar un tipo de combustible sin referirlo a posiciones espaciales, reduciendo de esta forma la cantidad de archivos necesaria. Sin embargo, en caso de que se quiera utilizar información heterogénea puede especificarse, al igual que en FARSITE, mediante un archivo ASC o bien utilizando un GEOTIFF.

Los modelos de combustibles aceptados por defecto por el simulador serán los 13 modelos mencionados y se podrán utilizar modelos definidos por el usuario.

La información meteorológica podrá ser introduci-

da o bien mediante archivos grib o netcdf, los cuales son los formatos de salida estándar de la mayoría de los simuladores meteorológicos.

C. Salidas

Típicamente los parámetros de interés en cualquier simulador son el área quemada en función del tiempo, el tiempo de llegada de la llama a una cierta posición y la intensidad de la reacción.

La salida de PFIRE será:

- Perímetro del área quemada en función del tiempo (en formato ESRI Shapefile).
- Tiempo de arribo (archivo raster).
- Intensidad de la reacción en función del tiempo (raster multicapa).
- Campo de vientos: Esto surge de la composición del campo de vientos introducido como input del programa más la realimentación por el fuego.

En FARSITE los perímetros de salida se representan en un archivo de formato ESRI Shapefile, mientras que en WRF-SFIRE, al resolverse una ecuación sobre una grilla, no se tiene información directa de los perímetros de incendio sino que se tiene un fichero netcdf en el cual se encuentra compilada la información en la misma resolución que la grilla espacial en la que se resuelve el problema. Esto implica la generación de perímetros escalonados y menos suaves que los producidos por FARSITE.

D. Métodos de optimización

Para acelerar los tiempos de cálculo involucrados en PFIRE se planearon diferentes estrategias para los esquemas de resolución por el método de Huygens y el de la curva de nivel. Sin embargo, en ambos casos debe calcularse el ritmo de propagación utilizando la ecuación 1. Si se observa la tabla II, puede verse que R_0 es independiente de los puntos individuales de la grilla (sólo dependerá del tipo de combustible utilizado) al igual que el coeficiente Φ_w , mientras que el término $(\bar{n} + \bar{\Phi}_s)$ dependerá de condiciones locales del punto donde se realice la propagación (dirección normal a la curva y valor y dirección de la pendiente).

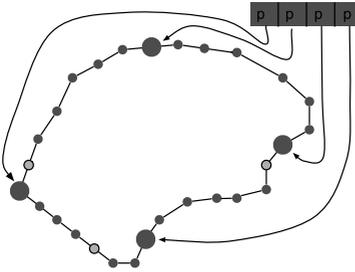


Fig. 5: Esquema de entrada a un perímetro para una arquitectura con más de un procesador. Los nodos más grandes se consideran nodos de entrada. Cada nodo de entrada propagará los nodos entre este y el siguiente nodo de entrada.

De esta forma, antes de comenzar el ciclo principal de la simulación, se calcula y tabula para cada tipo de combustible presente en el mapa los factores R_0 y Φ_w . De este modo, se reduce considerablemente la cantidad de operaciones que debe realizarse en cada paso de propagación.

E. Paralelización

La paralelización de los distintos esquemas resolutivos debe realizarse siguiendo diferentes estrategias. Para el caso de la curva de nivel se aprovechará que el cálculo puede vectorizarse y se utilizará una GPU como acelerador.

El método de Huygens puede paralelizarse con pequeñas modificaciones. Como se mencionó anteriormente, el perímetro se encuentra representado como una lista enlazada circular. Normalmente se considera un nodo de entrada como cabeza de la lista. Si, en lugar de esto, se generan tantos nodos de accesos como procesadores disponibles para realizar el cálculo, cada procesador puede trabajar sobre una sección del perímetro. Esto se muestra esquemáticamente en la figura 5. Cada uno de los nodos representados por círculos más grandes son considerados nodos de entrada, a partir de los cuales se comenzará una propagación. Cada hilo de ejecución trabajará desde un nodo de entrada hasta el punto inmediatamente anterior al siguiente nodo de entrada.

Cabe resaltar que para que este esquema no presente rápidamente desbalances de carga, luego de una redisección del perímetro deben volver a reasignarse los nodos de entrada debido a la adición y/o eliminación de nodos en diferentes regiones del perímetro. Para lograr esto de forma eficiente PFIRE lleva control sobre la cantidad de nodos existente para cada nodo de entrada de modo de evitar recorrer todo el perímetro luego de cada redisección.

IV. ESTADO ACTUAL

En la actualidad los siguientes módulos de PFIRE se encuentran codificados y testeados:

- Montaje de los mapas de vegetación, combustible, pendiente y aspect a partir de ficheros ASC.
- Conversión de ficheros GEOTIFF para su utilización como entrada.
- Generación de la grilla espacial para realimentar el viento de acuerdo al comportamiento del fuego.
- Modelo de Rothermel.
- Núcleo de propagación utilizando el método de Huygens.

A su vez el algoritmo propuesto para la resolución de los perímetros está siendo testeado.

V. TRABAJO FUTURO

Se espera que la primer versión completa de PFIRE esté disponible para la comunidad a principios del 2017. Para ello los siguientes pasos a implementar son los siguientes:

- Implementación del método de la curva de nivel.
- Implementación de un modelo meteorológico y su realimentación con el fuego.

VI. DISCUSIÓN

En este trabajo se describió PFIRE, un simulador de propagación de incendios forestales en desarrollo por el grupo HPC4SE del Departamento de Arquitectura de Computadores y Sistemas Operativos de la Universidad Autónoma de Barcelona, el cual cuenta con casi 20 años de experiencia en la utilización y customización de simuladores de incendios.

PFIRE está pensado para ser un simulador multi-esquema y de fácil paralelización, que a su vez permita el fácil acople de modelos meteorológicos.

A nivel de usuario el valor agregado de PFIRE radica en la versatilidad y simplicidad de la creación de los ficheros de entrada sin limitar sus capacidades de simular casos complejos.

A su vez el grupo HPC4SE publica activamente nuevos métodos para mejorar las predicciones de las propagaciones de incendios forestales. Contar con una herramienta propia sobre la cual implementar nuevos algoritmos, métodos de optimización y resolución resulta invaluable para acelerar los estudios que se realizan.

PFIRE será de código abierto y estará disponible para la comunidad a principios de 2017.

AGRADECIMIENTOS

El presente trabajo ha sido financiado mediante el proyecto del Ministerio de Economía y Competitividad TIN2014-53234-C2-1-R.

REFERENCIAS

- [1] Alan Grant McArthur, "Fire behaviour in eucalypt forests," 1967.
- [2] Richard C. Rothermel, "A mathematical model for predicting fire spread in wild land fuels," 1972.

- [3] Ioannis Karafyllidis and Adonios Thanailakis, "A model for predicting forest fire spreading using cellular automata," *Ecological Modelling*, vol. 99, no. 1, pp. 87–97, jun 1997.
- [4] A. Alexandridis, D. Vekalis, C.I. Siettos, and G.V. Bafas, "A cellular automata model for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1960," *Applied Mathematics and Computation*, vol. 204, no. 1, pp. 191–201, oct 2008.
- [5] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts.," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, feb 1996.
- [6] I Knight and J Coleman, "A Fire Perimeter Expansion Algorithm-Based on Huygens Wavelet Propagation," *International Journal of Wildland Fire*, vol. 3, no. 2, pp. 73, 1993.
- [7] A L Sullivan, "Wildland surface fire spread modelling, 1990–2007. 2: Empirical and quasi-empirical models," *International Journal of Wildland Fire*, vol. 18, no. 4, pp. 369–386, 2009.
- [8] A L Sullivan, "Wildland surface fire spread modelling, 1990–2007. 1: Physical and quasi-physical models," *International Journal of Wildland Fire*, vol. 18, no. 4, pp. 349–368, 2009.
- [9] A L Sullivan, "Wildland surface fire spread modelling, 1990–2007. 3: Simulation and mathematical analogue models," *International Journal of Wildland Fire*, vol. 18, no. 4, pp. 387–403, 2009.
- [10] George D Papadopoulos and Fotini-niovi Pavlidou, "A Comparative Review on Wild fire Simulators," vol. 5, no. 2, pp. 233–243, 2011.
- [11] T Opperman, J Gould, M Finney, and C Tynstra, "Applying Fire Spread Simulators in New Zealand and Australia: Results from an International Seminar," *USDA Forest Service Proceedings RMRS-P-41. 2006.*, vol. 2007, no. Thursday, 27 August 2007, pp. Workshop Proceedings, 2006.
- [12] Baker Abdalhaq, Ana Cortés, Tomàs Margalef, and Emilio Luque, "Enhancing wildland fire prediction on cluster systems applying evolutionary optimization techniques," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 61–67, 2005.
- [13] Andrés Cencerrado, Ana Cortés, and Tomàs Margalef, "Genetic algorithm characterization for the quality assessment of forest fire spread prediction," *Procedia Computer Science*, vol. 9, pp. 312–320, 2012.
- [14] Davide Ascoli, Giorgio Vacchiano, Renzo Motta, and Giovanni Bovio, "Building Rothermel fire behaviour fuel models by genetic algorithm optimisation," *International Journal of Wildland Fire*, vol. 24, no. 3, pp. 317–328, 2015.
- [15] Mark A Finney, "FARSITE : Fire Area Simulator — Model Development and Evaluation," *USDA Forest Service Research Paper*, , no. February, pp. 47, 1998.
- [16] Edward G. Patton and Janice L. Coen, "WRF-Fire : A Coupled Atmosphere-Fire Module for WRF," *Preprints of Joint MM5/Weather Research and Forecasting Model Users' Workshop, Boulder, CO, 22–25 June, NCAR, 221–223*, p. 3, 2004.
- [17] Joe H Scott and Robert E Burgan, "Standard fire behavior fuel models: A comprehensive set for use with Rothermel's surface fire spread model," , no. June, pp. 72, 2005.
- [18] Frank A Albin, "Spot fire distance from burning trees," 1979.
- [19] F A Albin and E D Reinhardt, "Modeling Ignition and Burning Rate of Large Woody Natural Fuels," *International Journal of Wildland Fire*, vol. 5, no. 2, pp. 81–91, 1995.
- [20] Frank A Albinil, Brown, James K., Reinhardt Elizabeth D., and Roger D. Ottmar, "Calibration of a Large Fuel Burnout Model," vol. 5, no. 3, pp. 173–192, 1995.
- [21] Ralph M Nelson Jr., "Prediction of diurnal change in 10-h fuel stick moisture content," *Canadian Journal of Forest Research*, vol. 30, no. Deeming 1983, pp. 1071–1087, 2000.
- [22] CE Van Wagner, "Conditions for the start and spread of crown fire," *Canadian Journal of Forest Research*, vol. 7, no. 1, pp. 23–34, 1977.
- [23] Richard C Rothermel, "Predicting behavior and size of crown fires in the northern Rocky Mountains," *USDA Forest Service, Intermountain Research Station, Research Paper*, , no. January, pp. 46, 1991.
- [24] Brian Potter and Bret Butler, "Using Wind Models To More Effectively Manage Wildfire," *Fire Management Today*, vol. 69, no. 2, pp. 40–46, 2009.
- [25] Jason M. Forthofer, "Modeling wind in complex terrain for use in fire spread prediction," 2007.
- [26] G D Richards and RW Bryce, "A Computer Algorithm for Simulating the Spread of Wildland Fire Perimeters for Heterogeneous Fuel and Meteorological Conditions," *International Journal of Wildland Fire*, vol. 5, no. 2, pp. 73, 1995.
- [27] Tomàs Artés, Andrés Cencerrado, Ana Cortés, and Tomàs Margalef, "Relieving the effects of uncertainty in forest fire spread prediction by hybrid MPI-openMP parallel strategies," *Procedia Computer Science*, vol. 18, pp. 2278–2287, 2013.
- [28] J. Michalakes, J. Dudhia, D.O. Gill, T.B. Henderson, J.B. Klemp, W. Skamarock, and W. Wang, "The weather research and forecast model: software architecture and performance," in *11th Workshop on the Use of High Performance Computing in Meteorology*, 2004.

FARSITE vs. WRF-SFIRE: simulación de la propagación de incendios forestales, una cuestión de precisión y de tiempo

Angel Farguell¹, Nicolas Chiaraviglio¹, Ana Cortés¹, Tomàs Margalef¹ y Josep Ramón Miró²

Resumen— Los incendios forestales afectan gravemente a la flora y fauna de los bosques causando anualmente que miles de hectáreas de bosque sean reducidas a cenizas. Predecir la propagación del incendio es indispensable para que los cuerpos de bomberos puedan actuar con mayor fiabilidad y rapidez. Para eso se necesitan simuladores capaces de predecir el incendio lo más fiable y rápido posible. Actualmente, los dos principales simuladores de incendios forestales son FARSITE y WRF-SFIRE. Por este motivo, en este trabajo se presenta una comparativa de la velocidad de ejecución y de la coherencia en los resultados desde el punto de vista físico de ambos simuladores. Trata de la funcionalidad y resultados que se obtienen al ejecutar ambos simuladores en casos sintéticos diseñados para analizar su comportamiento delante diferentes situaciones concretas.

Palabras clave— Simuladores, Propagación de incendios, FARSITE, WRF-SFIRE.

I. INTRODUCCIÓN

LOS incendios forestales son una de las principales causas de destrucción medioambiental. Cualquier medida que pueda ayudar a controlar la evolución del fuego es necesaria para ayudar a los bomberos a detener su propagación. Por ese motivo se usan simuladores que propagan el incendio y dan información valiosa para los cuerpos de bomberos que así pueden atacar con más fiabilidad el incendio.

Actualmente la mayoría de los simuladores de incendios calculan la propagación del incendio a partir del modelo semiempírico de Rothermel [1] en el cual:

$$R = R_0(\vec{n} + \vec{\phi}_w + \vec{\phi}_s) \quad (1)$$

donde R_0 es la velocidad de propagación sin viento ni pendiente, \vec{n} es la dirección normal al perímetro del incendio, $\vec{\phi}_w$ es el factor de viento y $\vec{\phi}_s$ el factor de pendiente.

El modelo de Rothermel requiere conocer la elevación del terreno (para obtener $\vec{\phi}_s$), la meteorología local (para obtener $\vec{\phi}_w$) y las propiedades de combustión de la vegetación (para obtener R_0) en el dominio donde se propaga el incendio. Al largo de los años se han propuesto diferentes estrategias para propagar el frente del incendio a partir del modelo puntual anterior.

En este artículo se propone analizar dos de los simuladores más usados y estudiados dentro de la co-

munidad, FARSITE [2] y WRF-SFIRE [3]. Ambos simuladores usan el modelo de Rothermel pero tienen diferencias muy notorias.

FARSITE es un simulador que resuelve el modelo de Rothermel y expande el perímetro a partir del principio de propagación de ondas de Huygens [2]. También ha sido mejorado incluyendo modelos de Spotting [4], consumo de combustible fuera del frente [5], [6], humedad del combustible [7], inicio de fuego de copas [8] y propagación de fuego de copas [9]. Además, permite incorporar información topográfica y meteorológica (viento, precipitaciones, humedad, temperatura). No obstante, no contempla variaciones del campo de viento por irregularidades del terreno o debidas al mismo incendio. Para mejorar el campo de viento se usa WindNinja [10] el cual utiliza conservación de la masa para considerar las variaciones producidas por la orografía. A pesar de todas estas limitaciones y de las que se verán más adelante, FARSITE es muy usado y los resultados que genera son aceptables dentro de ciertos márgenes.

Por otro lado, WRF-SFIRE es un simulador que acopla un modelo meteorológico (WRF-ARW [11]) y uno que propaga el incendio a partir de resolver el modelo de Rothermel y expande el perímetro a partir del método de la curva de nivel (SFIRE). En cada paso de tiempo del simulador WRF, se propaga el incendio y a continuación, se recalcula la meteorología a partir de los flujos de calor que transmite el fuego a la atmósfera. Para conseguirlo, es necesario el uso de una malla meteorológica y dentro de esta, una malla más fina en la cual se calcule la propagación del incendio. De esta forma, el simulador WRF-SFIRE adopta un realismo que no posee FARSITE ya que en WRF-SFIRE el fuego afecta a la meteorología y la meteorología afecta al fuego. No obstante, WRF-SFIRE genera los cálculos para todos los puntos de la malla y esto hace muy costosa la obtención de resultados a alta resolución. Además, al usar el método de la curva de nivel en el cual se requiere resolver numéricamente ecuaciones en derivadas parciales, según el paso de tiempo y la resolución de la malla que se use, aparecen problemas de estabilidad que conllevan a la no convergencia del modelo.

Este trabajo empieza analizando las diferencias más significativas entre FARSITE y WRF-SFIRE (sección II). A continuación, en la sección III, se explican los diferentes experimentos, desde su diseño hasta el resultado que se espera en cada uno. En la

¹Dpto. de Arquitectura de Computadores y Sistemas Operativos, Univ. Autónoma de Barcelona, e-mail: angel.farguell@uab.cat.

²Servei Meteorològic de Catalunya (SMC), e-mail: jrmiró@meteo.cat.

sección IV se podrán apreciar los resultados obtenidos y analizar su significado. Para dar paso a la sección IV, en la que se resumirán las observaciones encontradas. Finalmente, se expondrán los trabajos futuros en la sección .

II. DIFERENCIAS SIGNIFICATIVAS ENTRE FARSITE Y WRF-SFIRE

A. Método de propagación del frente

Tal y como se ha podido ver, FARSITE y WRF-SFIRE usan diferentes métodos de propagación del frente del incendio. En el caso de FARSITE, se usa el principio de propagación de ondas de Huygens. La idea es propagar el incendio en forma de ondas elípticas donde el foco de cada elipse sea un punto de la frontera y el extremo opuesto de la elipse sea el punto del nuevo perímetro expandido (figura 1a). De esta manera, se expanden todos los puntos y se genera un nuevo perímetro usando información proveniente del modelo de la sección I (Ecuación 1), donde se usa información que proviene de la vegetación, del viento y de la pendiente.

Este método solo procesa los puntos que están en la frontera y hace que el cómputo sea mucho más ligero que en el caso de WRF-SFIRE. No obstante, a medida que se expande el perímetro, se añaden dos complicaciones:

- Se puede incrementar la distancia entre los puntos del frente y por este motivo, se debe discretizar el perímetro generando nuevos puntos (para mantener una resolución mínima en la frontera). Por eso, en casos en los que el fuego es muy grande se incrementa el tiempo de resolución considerablemente.
- Se generan cruzamientos entre los nuevos puntos surgidos de la expansión del perímetro (figura 1b). Estos cruzamientos deben resolverse con reordenaciones para mantener la coherencia espacial de los puntos ya que en FARSITE, el perímetro esta formado por una sucesión de puntos conectados entre sí. Se requiere ejecutar algoritmos para solventar el problema de los cruzamientos. En el caso de FARSITE, se usa un algoritmo diseñado por Richards [12] que soluciona perímetros muy complejos pero es muy costoso (algoritmo de orden N^2 donde N es el número de puntos en el frente que acaba significando un 60% del tiempo de ejecución del programa [13]). Aunque se han probado diferentes técnicas de paralelización en [13], el programa no escala al aumentar los recursos.

En el caso de WRF-SFIRE, se usa el método de la curva de nivel para propagar el perímetro del incendio. En este método, se define la región quemada usando una función de nivel $\psi = \psi(\vec{x}, t)$ y según si:

$$\begin{cases} \psi(\vec{x}, t) \leq 0 & \text{Área quemada} \\ \psi(\vec{x}, t) = 0 & \text{Frontera del fuego} \\ \psi(\vec{x}, t) > 0 & \text{Área sin fuego} \end{cases} \quad (2)$$

Esta función de nivel evoluciona a partir de la ecuación en derivadas parciales:

$$\frac{\partial \psi}{\partial t} + R|\nabla \psi| = 0 \quad (3)$$

donde R es la velocidad de propagación del incendio resuelto en la ecuación 1. La ecuación es resuelta numéricamente en WRF-SFIRE usando diferentes métodos de integración numérica en tiempo y en espacio (explicados en [3]). Por lo tanto, al ser la resolución numérica de una ecuación en derivadas parciales tiene una primera limitación de elegir adecuadamente la resolución espacial y temporal acorde al método de integración numérico usado. Además, tiene la complicación añadida de tener que hacer todos los cálculos en cada paso de tiempo (según que resolución espacial con pasos de tiempo muy bajos) en todos los puntos de la malla definida dentro del dominio. Por lo tanto, WRF-SFIRE se vuelve más costoso cuanto más grande es el terreno y cuanta más resolución espacial queramos en él. Estas limitaciones hacen que hoy en día no se haya conseguido ejecutar el simulador para realizar una predicción sobre un incendio en el momento de su detección, incluso usando programación en paralelo híbrida de MPI y OpenMP (disponible para el caso real). Además, se podrá observar en la sección IV que en el caso de FARSITE, los perímetros son más suaves que en WRF-SFIRE. Este hecho es debido a que en WRF-SFIRE los perímetros se generan a partir de observar en cada punto el valor de la curva de nivel $\psi(\vec{x}, t)$. Si tiene un valor menor o igual a 0, tal y como acabamos de ver, la celda que contiene este punto está quemada y de esta forma se generan perímetros escalonados. Contrariamente, FARSITE expande los puntos del perímetro y por este motivo genera perímetros más suaves.

B. Uso de la meteorología

Una de las diferencias más notorias entre los simuladores mencionados es el tratamiento que realizan de la información meteorológica. En el caso de FARSITE, esta información es estática para un determinado período de tiempo sin que exista una realimentación entre el fuego y las condiciones atmosféricas. Es decir, la propagación del fuego no afecta a la modelización de la atmósfera y así el propio incendio no determina el porvenir de sus condiciones meteorológicas. En la figura 2a se puede observar un esquema del uso de la meteorología en el caso de FARSITE.

En cambio WRF-SFIRE considera condiciones iniciales y a partir de las mismas ejecuta un modelo meteorológico considerando que los flujos de calor generados por el fuego afectan las condiciones atmosféricas y viceversa[3]. Con este cambio, se gana realismo y se consigue reproducir las dependencias reales entre el estado del fuego y el estado de la atmósfera. Se puede observar el uso de la meteorología en WRF-SFIRE en la figura 2b.

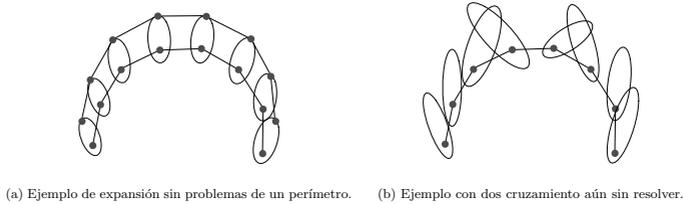


Fig. 1: Método de expansión de perímetros de Huygens.

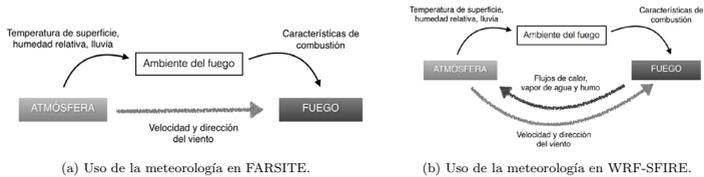


Fig. 2: Diferencias entre los distintos usos de la meteorología en ambos simuladores.

C. Datos de salida

Finalmente, las diferencias existentes entre los datos de salida de ambos simuladores. En el caso de FARSITE, la información de la propagación del incendio al venir dada por puntos enlazados viene en shapefiles. También genera rasters que contienen información diversa relacionada con el fuego. Por otro lado, en el caso de WRF-SFIRE, se puede elegir para ser en NetCDF (por defecto), en GRIB 1 o en GRIB 2. Se usan este tipo de ficheros de salida porque se requiere guardar muchas variables de muchas dimensiones diferentes y a muchos niveles de altitud diferentes. Esto permite visualizaciones en las que se pueden observar muchas otras variables además de la área incendiada.

III. EXPERIMENTOS

En esta sección se pretende ilustrar los 6 casos ideales que han sido diseñados para comparar los simuladores FARSITE y WRF-SFIRE. La idea principal era diseñar casos que fueran diversos, se pudieran ejecutar de la forma más similar posible y fueran interesantes de observar.

En primer lugar, se requiere decidir los parámetros que van a afectar a todos los casos. Por eso se decidió:

- **Tiempo de simulación:** 10 horas. Tiempo suficiente para poder visualizar las diferentes tendencias de cada simulador.
- **Domínio:** Cuadrado de 30x30 km.
- **Vegetación:** Vegetación homogénea del tipo 3 en los 13 modelos de Rothermel. Esto equivale a hierba larga de casi 1 metro de altitud.
- **Resolución de la malla:** Una resolución de 40 metros en la malla de fuego del modelo WRF-

SFIRE que sería el equivalente a la resolución de los datos de entrada de FARSITE. En la malla meteorológica, se pone una resolución de 600 metros (sino tendríamos que poner un paso de tiempo demasiado pequeño). Además, en el caso de FARSITE, también se elige:

1. Resolución del perímetro: 15 m.
 2. Resolución de distancia (distancia máxima antes de volver a buscar información al raster): 30 m.
- **Condiciones de contorno:** Usando WRF-SFIRE, se requiere elegir una condición de contorno. En un caso ideal se recomienda usar la condición de contorno *Open Boundary Conditions* en las cuatro caras del dominio.
 - **Variables meteorológicas fijas:** Para todos los casos se pone una temperatura inicial constante de 15 °C.

Así se construyeron los diferentes casos modificando el terreno, el viento y el punto de ignición como se muestra en la tabla I. Se puede apreciar que en los casos 1 y 2, se tiene un terreno plano, sin y con viento de noreste de 15 km/h y con el punto de ignición en el centro del dominio (desde el vértice izquierdo inferior, el punto (15,15) km). En los casos 3 y 4 se utiliza una montaña de 2000 m de altitud y 10000 m de diámetro (figura 3a), sin y con el mismo viento y con el punto de ignición un poco desviado hacia el noreste ((15.5,15.5) km). En los casos 5 y 6 se utiliza una pendiente de oeste a este que va de 0 a 2000 metros de altitud (figura 3b), nuevamente sin y con el mismo viento y el punto de ignición situado en (12,15) km y (12,22) km respectivamente. Finalmente, se procesan los datos del campo de viento

usando WindNinja y se aplican al caso 4. No se aplica a ningún otro caso porque usando WindNinja, el campo de viento no se modifica en el caso del plano y se modifica levemente en el caso de la pendiente sin afectar la propagación del incendio. Es decir, en los casos del plano y de la pendiente, la diferencia en los resultados no es significativa.

Para inicializar WRF-SFIRE, se usa un sondaje real del viento a diferentes niveles verticales. Como FARSITE requiere como dato de entrada el viento a 10 metros, se hace un promedio de los primeros niveles para tener las condiciones iniciales lo más similares posibles en ambos simuladores. Así la velocidad del viento acaba siendo de 26 km/h y la dirección proviene del nordeste.

A continuación, se discute el resultado esperado para cada caso:

- Caso 1: Al ser una superficie plana y sin viento, se espera obtener circunferencias concéntricas.
- Caso 2: Se espera obtener elipses bajando del centro hacia el sudoeste del mapa (viento del nordeste de 26 km/h).
- Caso 3: En este caso es más difícil decir qué tendría que pasar al tener la montaña de la figura 3a. No obstante, el fuego debería ir hacia la pendiente y subir la montaña. Pero a medida que se va aproximando a la cima, el incendio debería bajar su velocidad y le debería costar bajar la pendiente.
- Caso 4: Se tiene el caso 3 pero incluyendo viento a favor de la pendiente. Por lo tanto, es importante distinguir entre lo que se espera que pase en cada simulador. En el caso de FARSITE, como solo usa la información del terreno y de la meteorología de forma independiente y el viento en este caso va a favor de la pendiente, se espera que sea similar al caso anterior pero propagando con mayor velocidad. Por lo contrario, en el simulador WRF-SFIRE se espera que el incendio avance rodeando la montaña a causa de la modelización atmosférica que depende del terreno y de los flujos de calor del fuego. Finalmente, se ejecuta el mismo caso pero procesando antes los campos de viento usando WindNinja. Los resultados de FARSITE deberían mejorar obteniendo un comportamiento del fuego más real. Sin embargo, hay que recordar que el fuego no modifica las condiciones atmosféricas.
- Caso 5: En este caso, se tiene una pendiente y sin viento. Por lo tanto, el fuego debería escalar la pendiente.
- Caso 6: Se añade viento al caso de la pendiente y se debería ver una lucha entre la pendiente y el viento. No obstante, debería ganar el viento ya que es violento (26 km/h) y la pendiente no es excesivamente brusca (2000 m de altitud en 15000 m de largo).

Además, se espera más recorrido y más irregularidad en los perímetros usando WRF-SFIRE en los casos ya que el fuego genera flujos de calor que

pueden modificar la propagación del propio incendio.

IV. RESULTADOS

En este apartado, se verán los resultados obtenidos en cada caso diseñado (resumidos en la tabla I). En todos los gráficos que contengan altitudes diferentes de 0 m, se podrá visualizar la altitud del terreno (el fondo). Además, en cada gráfico, aparecen 4 conjuntos de perímetros con representaciones en escala de grises diferentes. En el caso del color negro punteado, representa los diferentes perímetros cada 30 minutos de FARSITE con factor de propagación 1. El factor de propagación de FARSITE es un factor de ajuste que se usa para incrementar o decrementar la capacidad de propagación del incendio. De esta forma, aumentando el factor de propagación, FARSITE simula la propagación del incendio con una mayor velocidad de propagación. De color negro y gris liso, el mismo caso anterior pero con factor de propagación 2 y 3 respectivamente (FARSITE 2 y FARSITE 3). Finalmente, rayado de color gris y blanco, se representan los diferentes perímetros cada 20 minutos de WRF-SFIRE. En todos los gráficos, se amplía el dominio en la zona que permita observar mejor la diferencia existente entre los diferentes conjuntos de perímetros.

A. Caso 1: Flat

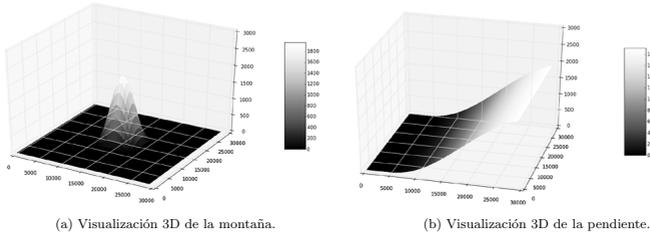
En este caso, la altitud es siempre 0 m y no se tiene viento inicial. Es decir, tal y como se puede observar en la figura 5, los perímetros son circunferencias concéntricas con centro el punto de ignición. Se puede ver también, que WRF-SFIRE quema más rápido que FARSITE con factor de propagación 1. En lugar de eso, FARSITE con factor de propagación 2, quema más que WRF-SFIRE. Por lo tanto, calculando el factor de propagación exacto para que FARSITE quemara de la misma forma que WRF-SFIRE, sale un factor de propagación de 1.7. Esta mayor propagación tiene sentido desde el punto de vista de que el propio fuego genera una meteorología que propicia una mayor velocidad de propagación.

B. Caso 2: Flat wind

El caso 2 es como el caso 1, pero añadiendo viento de nordeste de 26 km/h. En la figura 6, se puede observar como afecta el viento en todos los casos comparando los resultados con la figura 5. En este caso, se puede observar que WRF-SFIRE, expande el perímetro con elipses mucho más cerradas y deformadas que FARSITE. Es decir, el viento hace que el fuego avance hacia delante y también provoca que se expanda menos hacia los lados. Esto causa una diferencia notoria entre FARSITE y WRF-SFIRE. Además, se puede calcular que para que FARSITE queme igual de rápido que WRF-SFIRE, se necesitaría un factor de propagación de 2.9.

C. Caso 3: Hill

El caso 3 contiene una montaña de 2000 m y 10000 m de diámetro situada en el medio del dominio. Se



(a) Visualización 3D de la montaña. (b) Visualización 3D de la pendiente.

Fig. 3: Visualización 3D de los terrenos usados en las simulaciones.

TABLA I: Resumen de los experimentos realizados con ambos simuladores.

Casos	Punto de ignición	Terreno	Viento
1	(15000,15000)	Plano	Sin viento
2	(15000,15000)	Plano	N-E de 26 km/h
3	(15500,15500)	Montaña	Sin viento
4	(15500,15500)	Montaña	N-E de 26 km/h
5	(12000,15000)	Pendiente	Sin viento
6	(12000,22000)	Pendiente	N-E de 26 km/h

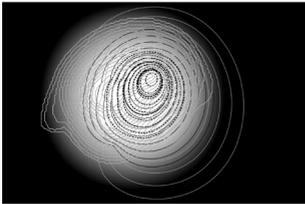


Fig. 4: Resultado del caso 4 usando FARSITE con factor de propagación 1, 2 y 3 y WRF-SFIRE.

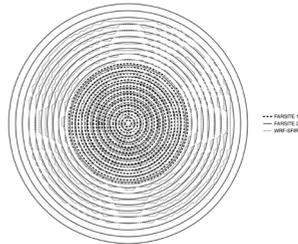


Fig. 5: Resultado del caso 1 usando FARSITE con factor de propagación 1, 2 y 3 y WRF-SFIRE.

empieza con un punto de ignición situado un poco desviado del punto central en dirección noreste ((15.5,15.5) km). Los resultados de la figura 7 muestran que WRF-SFIRE se expande en forma de elipses subiendo la montaña hasta sobrepasarla. En el caso de FARSITE, al acercarse a la cima, la velocidad de propagación disminuye. Incluso subiendo hasta 3 el factor de propagación, en sentido sudoeste WRF-SFIRE llega más lejos que FARSITE. En contraposición, en sentido noreste, FARSITE necesitar un factor de 2 para tener una velocidad de propagación similar al WRF-SFIRE. De todas formas, los resultados indican diferencias significativas en la resolución de la pendiente de una montaña. En este caso, se puede observar lo importante que es realimentar la meteorología a partir de los resultados del fuego ya que se puede apreciar que el efecto de la montaña en el fuego es mucho más real en WRF-SFIRE

por el hecho de que ejecuta un modelo meteorológico (WRF) el cual modeliza la atmósfera a partir de las condiciones iniciales.

D. Caso 4: Hill wind

El caso 4 añade al caso 3 el mismo viento visto anteriormente (figura 8). Este efecto del viento hace que FARSITE quemé más hacia el sudoeste de lo que se había podido ver en el caso 3. Además, comparando ambos casos, se puede observar que con viento se quema más que sin viento (tiene sentido por el hecho de tener viento a favor de la pendiente). No obstante, FARSITE y WRF-SFIRE se propagan alrededor de la montaña en direcciones distintas. En el caso de FARSITE, ataca con más velocidad hacia el este y en el caso de WRF-SFIRE, ataca más hacia el oeste. Así que se puede observar como cada simulador coge diferentes sentidos. Es una diferencia significativa

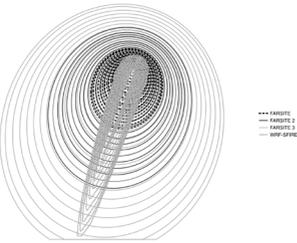


Fig. 6: Resultado del caso 2 usando FARSITE con factor de propagación 1, 2 y 3 y WRF-SFIRE.

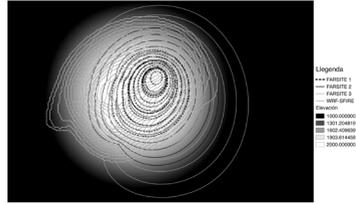


Fig. 8: Resultado del caso 4 usando FARSITE con factor de propagación 1, 2 y 3 y WRF-SFIRE.

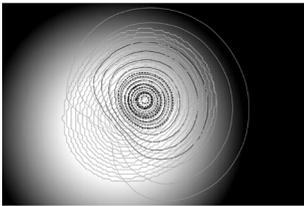


Fig. 7: Resultado del caso 3 usando FARSITE con factor de propagación 1, 2 y 3 y WRF-SFIRE.

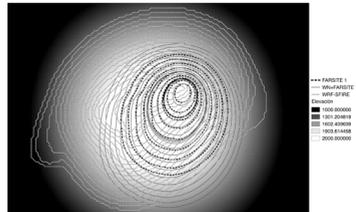


Fig. 9: Resultado del caso 4 usando FARSITE con factor de propagación 1 sin y con campo de viento procesado por WindNinja y WRF-SFIRE.

pero vista la complejidad de propagar un incendio encima de una montaña con un viento a favor de la pendiente de la montaña, poniendo un factor de propagación entre 2 y 3, ambos simuladores pueden parecerse bastante. Finalmente, se ejecuta el mismo caso pero añadiendo el campo de viento procesado por WindNinja. En la figura 9 se muestra el resultado de esta propagación en donde se puede observar un mayor avance del frente de incendio. Es decir, los perímetros se acercan más al resultado de WRF-SFIRE usando WindNinja en FARSITE sin embargo el fuego no rodea la montaña de la misma forma que WRF-SFIRE.

E. Caso 5: Slope

En este caso 5, se tiene una pendiente de oeste a este que va de 0 a 2000 m y un punto de ignición en el (12,15) km. Los resultados de la figura 10 muestran que es, en diferencia, el caso que conlleva menos similitud entre FARSITE y WRF-SFIRE. Queda evidente que en el caso de FARSITE, la pendiente casi no afecta a la propagación del incendio. Contrariamente, en el caso de WRF-SFIRE, la pendiente afecta mucho a la propagación del incendio hasta al punto de dejar de generar elipses y propagarse con mucha velocidad en dirección y. Parece ser un comportamiento sorprendente en ambos casos. En el caso de FARSITE,

uno esperaría que le afectara más la pendiente y en el caso de WRF-SFIRE, uno podía pensar que saldrían elipses subiendo cada vez más. Pero se puede ver que el propio incendio genera un viento lateral que causa este patrón de propagación. En este caso, ni subiendo a 3 el factor de propagación no se gana igualdad por el hecho de que el comportamiento de propagación es muy diferente.

F. Caso 6: Slope wind

Finalmente, el caso 6 es el caso 5 pero añadiendo el mismo viento visto anteriormente y poniendo el punto de ignición más hacia el norte. En la figura 11 se puede observar la misma conclusión que en el caso 5. Es decir, que la pendiente no afecta casi al FARSITE y contrariamente, afecta mucho al WRF-SFIRE. Además, se puede ver que WRF-SFIRE quema mucho más terreno que FARSITE incluso usando un factor de propagación de 3. También se puede observar de forma parecida a los otros casos que FARSITE quema más terreno en dirección contraria al viento que WRF-SFIRE. Resumiendo, se puede ver que hay mucha diferencia en la dirección de propagación (WRF-SFIRE siempre tiene tendencia a subir la pendiente) y la velocidad de propagación (WRF-SFIRE llega a la frontera del dominio por la acción del viento

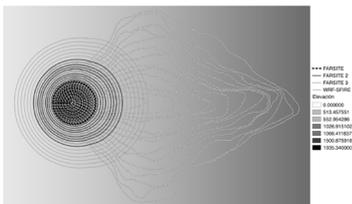


Fig. 10: Resultado del caso 5 usando FARSITE con factor de propagación 1, 2 y 3 y WRF-SFIRE.

generado por el propio fuego) y por eso al final ambos simuladores generan perímetros muy diferentes. En este caso, al igual que el anterior, WRF-SFIRE tiene un comportamiento físicamente más real.

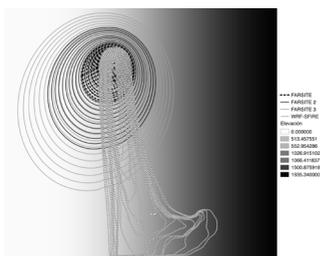


Fig. 11: Resultado del caso 6 usando FARSITE con factor de propagación 1, 2 y 3 y WRF-SFIRE.

G. Tiempos de ejecución

Además de analizar los resultados, también es interesante estudiar los tiempos de ejecución en ambos simuladores. En la tabla II se pueden observar los tiempos de ejecución de cada caso para cada simulador y así poder interpretar las grandes diferencias existentes entre ellos. Las diferencias entre FARSITE, FARSITE 2 y FARSITE 3, son causadas por tener que aumentar el número de puntos del perímetro para mantener la resolución. Al tener un algoritmo de resolución de cruces costoso[12], si se aumentan los puntos, el tiempo de ejecución en FARSITE aumenta significativamente. Así, se puede observar que los casos en los que el fuego acaba ocupando mayor espacio son los casos en los que FARSITE tarda más tiempo. Cuando se usa WindNinja, al tiempo de ejecución del simulador se le debe agregar el tiempo de generación del campo de viento (2 min y 14s para una grilla de 750x750). Con el campo de viento proveniente de WindNinja, el tiempo de ejecución del FARSITE aumenta considerablemente.

En el caso de WRF-SFIRE, se espera que los tiempos de ejecución, al tener dominios y mallas iguales, no varíen demasiado entre los diferentes casos ya que siempre se calcula lo mismo para todos los puntos de la malla independientemente del terreno, viento o punto de ignición. Se puede observar que en todos los casos el tiempo de ejecución está alrededor de 4 horas.

Cabe comentar que se han ejecutado tanto FARSITE como WRF-SFIRE en serie y por lo tanto, estos tiempos de ejecución se podrían llegar a mejorar usando computación de altas prestaciones. Por lo tanto, en todos los casos, se analizan ambos simuladores de forma serial.

V. DISCUSIÓN

En este artículo, se han podido observar las diferencias más significativas entre FARSITE y WRF-SFIRE. De esta manera, se puede concluir que FARSITE tiene una velocidad de propagación menor a la que posee WRF-SFIRE por el acoplamiento meteorología-combustión que posee WRF-SFIRE. Este acoplamiento genera una meteorología propia que causa una mayor propagación del incendio. Además, en el caso de FARSITE, la propagación de los perímetros es elíptica a diferencia de los perímetros irregulares que genera WRF-SFIRE. FARSITE contiene un factor de propagación para poder incrementar o decrementar esa propagación, no obstante no existen reglas definidas por las cuales dada una situación concreta, pueda determinarse un factor de propagación. Además, en el caso de FARSITE, la pendiente afecta muy ligeramente a la propagación del fuego. En contraposición, en el caso de WRF-SFIRE la pendiente llega a competir con un viento de 26 km/h. No obstante, en la figura 7 se puede observar que el hecho de que el fuego baje la pendiente no retrasa el avance del incendio. Esto es debido al viento generado por el fuego que va en dirección descendente a la montaña y que ayuda a que el fuego avance. También se observó que al utilizar un generador de campo de viento como WindNinja se mejoran los resultados obtenidos con FARSITE pero los mismos aún se apartan considerablemente de aquellos obtenidos con WRF-SFIRE.

Se puede concluir que WRF-SFIRE genera resultados que se acercan más al comportamiento físico esperado en todos los casos. Sin embargo esto debe determinarse posteriormente utilizando un incendio real como punto de referencia. No obstante, el tiempo de ejecución aumenta considerablemente y, si se quisiera más resolución, se debería bajar el paso de tiempo (para obtener convergencia a causa de los métodos de integración numérica usados) que aún aumentaría mucho más el tiempo de ejecución. En este sentido, FARSITE consigue resultados con un tiempo de ejecución más corto y así permite una mayor resolución con facilidad.

TABLA II: Tiempos de ejecución de los 6 casos ideales en FARSITE y en WRF-SFIRE.

Simulador	Flat	Flat wind	Hill	Hill wind	Slope	Slope wind
FARSITE	1 s	44 s	2 s	6 s	7 s	2 min 59 s
FARSITE 2	4 s	5 min 13 s	9 s	27 s	25 s	19 min 23 s
FARSITE 3	10 s	17 min 13 s	33 s	77 s	63 s	40 min 39 s
WN + FARSITE	-	-	-	2 min 14 s + 3 min 30 s	-	-
WRF-SFIRE	3 h 55 min	3 h 50 min	3 h 53 min	3 h 54 min	3 h 57 min	3 h 48 min

AGRADECIMIENTOS

El presente trabajo ha sido financiado mediante el proyecto del Ministerio de Economía y Competitividad TIN2014-53234-C2-1-R.

REFERENCIAS

- [1] Richard C. Rothermel, "A mathematical model for predicting fire spread in wild land fuels," 1972.
- [2] Mark A Finney, "FARSITE: Fire Area Simulator — Model Development and Evaluation," *USDA Forest Service Research Paper*, no. February, pp. 47, 1998.
- [3] J. Mandel, J. D. Beezley, and a. K. Kochanski, "Coupled atmosphere-wildland fire modeling with WRF 3.3 and SFIRE 2011," *Geoscientific Model Development*, vol. 4, no. 3, pp. 591–610, 2011.
- [4] Frank A Albini, "Spot fire distance from burning trees," 1979.
- [5] F A Albini and E D Reinhardt, "Modeling Ignition and Burning Rate of Large Woody Natural Fuels," *International Journal of Wildland Fire*, vol. 5, no. 2, pp. 81–91, 1995.
- [6] Frank A Albinil, Brown, James K., Reinhardt Elizabeth D., and Roger D. Ottmar, "Calibration of a Large Fuel Burnout Model," vol. 5, no. 3, pp. 173–192, 1995.
- [7] Ralph M Nelson Jr., "Prediction of diurnal change in 10-h fuel stick moisture content," *Canadian Journal of Forest Research*, vol. 30, no. Deeming 1983, pp. 1071–1087, 2000.
- [8] CE Van Wagner, "Conditions for the start and spread of crown fire," *Canadian Journal of Forest Research*, vol. 7, no. 1, pp. 23–34, 1977.
- [9] Richard C Rothermel, "Predicting behavior and size of crown fires in the northern Rocky Mountains," *USDA Forest Service, Intermountain Research Station, Research Paper*, no. January, pp. 46, 1991.
- [10] Jason M. Forthofer, "Modeling wind in complex terrain for use in fire spread prediction," 2007.
- [11] J. Michalakes, J. Dudhia, D.O. Gill, T.B. Henderson, J.B. Klemp, W. Skamarock, and W. Wang, "The weather research and forecast model: software architecture and performance," in *11th Workshop on the Use of High Performance Computing in Meteorology*, 2004.
- [12] G D Richards and RW Bryce, "A Computer Algorithm for Simulating the Spread of Wildland Fire Perimeters for Heterogeneous Fuel and Meteorological Conditions," *International Journal of Wildland Fire*, vol. 5, no. 2, pp. 73, 1995.
- [13] Tomàs Artés, Andrés Cencerrado, Ana Cortés, and Tomàs Margalef, "Relieving the effects of uncertainty in forest fire spread prediction by hybrid MPI-openMP parallel strategies," *Procedia Computer Science*, vol. 18, pp. 2278–2287, 2013.

Algoritmo de pre-análisis para la estimación de movimiento en HEVC

Gabriel Cebrián-Márquez¹, José Luis Martínez¹, Pedro Cuenca¹

Resumen—La creciente demanda de contenido multimedia de alta calidad y la llegada del formato Ultra High Definition (UHD) ha motivado el desarrollo del estándar High Efficiency Video Coding (HEVC), el cual es capaz de superar a estándares previos en un 50% en términos de eficiencia de codificación. Esta mejora, sin embargo, conlleva una alta complejidad computacional del codificador, lo cual hace que sea necesario adoptar algoritmos rápidos para conseguir la codificación en tiempo real. En este sentido, este artículo propone un algoritmo de pre-análisis diseñado para proveer información de codificación a la etapa de estimación de movimiento del codificador. Este algoritmo, en cuyo diseño se han tenido en consideración las particularidades del estándar, hace uso de la información estimada para reducir el número de cuadros de referencia, así como el tamaño del área de búsqueda. Como resultado, la evaluación experimental del algoritmo muestra que es capaz de reducir el tiempo de codificación en un 16.10% sin derivar en pérdidas apreciables de eficiencia de codificación.

Palabras clave— HEVC, H.265, pre-análisis, estimación de movimiento

I. INTRODUCCIÓN

DURANTE más de una década, el estándar de codificación de vídeo H.264/Advanced Video Coding (AVC) [1] ha sido el más ampliamente utilizado en todo tipo de escenarios y aplicaciones multimedia. Sin embargo, las continuas demandas del mercado por una mayor calidad de experiencia junto con la llegada de nuevos formatos y resoluciones como Ultra High Definition (UHD) motivaron el desarrollo del estándar High Efficiency Video Coding (HEVC) [2]. Diseñado y establecido por la Joint Collaborative Team on Video Coding (JCT-VC) a principios del año 2013, este estándar es capaz de alcanzar hasta el doble de eficiencia de codificación que su predecesor, H.264/AVC. No obstante, esta mejora viene acompañada de un elevado incremento de la complejidad computacional de la codificación que es necesario acometer [3].

Con el objetivo de reducir el tiempo de codificación, HEVC define algunas herramientas a alto nivel como Tiles o Wavefronts, permitiendo con ello la paralelización del codificador desde el propio estándar. Sin embargo, estas herramientas suelen ser a menudo insuficientes para alcanzar el tiempo real en la codificación. Por ello, resulta esencial encontrar vías complementarias de aceleración del codificador de HEVC, no sólo desde el punto de vista del paralelismo, sino también algorítmico, reduciendo la

¹Grupo de Redes y Arquitecturas de Altas Prestaciones (RAAP). Instituto de Investigación en Informática de Albacete (I3A). Universidad de Castilla-La Mancha, Albacete, España. E-mail: Gabriel.Cebrian@uclm.es, JoseLuis.Martinez@uclm.es, Pedro.Cuenca@uclm.es

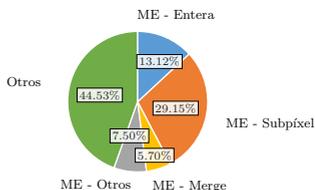


Fig. 1. Perfil de tiempos para la secuencia Kimono (1920×1080) bajo la configuración Random Access

cantidad de trabajo secuencial que cada hilo habría de ejecutar.

La Figura 1 muestra el perfil de tiempos obtenido a partir del codificador base de HEVC para la secuencia Kimono (1920×1080), usando la configuración Random Access y estableciendo el valor del parámetro de cuantización (QP) a 32. Como puede observarse, más de la mitad del tiempo total de codificación se dedica al algoritmo de estimación de movimiento (ME), convirtiéndose en la operación más costosa de todo el codificador. El 44.53% restante del tiempo se dedica al resto de módulos de codificación (etiquetado en el gráfico como “Otros”), incluyendo los procesos de predicción tipo intra, transformada, cuantización, codificación de entropía y los filtros in-loop. Esta notable complejidad del módulo de estimación de movimiento viene justificada por la gran cantidad de operaciones de carácter repetitivo que el codificador ha de realizar sobre las mismas muestras de la imagen, pero con diferentes particiones de bloque, es decir, distintos Prediction Units (PU) para el mismo Coding Tree Unit (CTU).

Por su parte, es claro que algunas aplicaciones multimedia requieren codificadores en tiempo real, tales como aquellas destinadas a la transmisión de contenidos en vivo. Por esta razón, este tipo de codificadores suelen implementar algoritmos que proporcionan soporte al proceso de codificación. En concreto, la mayor parte de estos codificadores implementan los denominados algoritmos de pre-análisis, los cuales son capaces de proveer al codificador información preliminar en relación a la secuencia de entrada, la cual puede usarse en el proceso de codificación con diversas finalidades como reducir el proceso de codificación u optimizar el patrón de codificación. En este artículo, proponemos una técnica de reducción de la complejidad computacional del módulo ME basada en una etapa de pre-análisis, teniendo en cuenta que se trata de aquel módulo cuyas

operaciones suponen el mayor tiempo de cómputo para el codificador. Con este objetivo en mente, esta etapa de pre-análisis se ocupa de realizar una estimación rápida de movimiento para calcular la información que más tarde es usada en el módulo ME del codificador para reducir el número de cuadros de referencia empleados y el tamaño del área de búsqueda. Como resultado, el tiempo de codificación se ve notablemente reducido, suponiendo un impacto apenas apreciable en la eficiencia de codificación. Ha de tenerse en cuenta, en cualquier caso, que este algoritmo también puede ser aplicado a cualquier otro estándar de codificación que defina esquemas de particionamiento complejos como HEVC.

El resto del artículo queda organizado de la manera que se enuncia a continuación. En primer lugar, la Sección II cubre algunos de los trabajos más relevantes relacionados con la temática. A continuación, la Sección III describe el algoritmo de pre-análisis propuesto y su integración con el módulo ME del codificador. La evaluación experimental del algoritmo se lleva a cabo en la Sección IV, mostrando los correspondientes resultados en términos de reducción de tiempo y eficiencia de codificación. Por último, la Sección V incluye las principales extraídas de los resultados, así como las posibles líneas de trabajo futuro.

II. TRABAJOS RELACIONADOS

A pesar de que el estándar define herramientas de paralelización para dar soporte al multiprocesamiento, existen numerosos trabajos que tratan de explotar todavía más el paralelismo de HEVC. En particular, los autores de [4] analizan el paralelismo de dichas herramientas y proponen el algoritmo denominado Overlapped Wavefront (OWF), el cual se trata de una mejora del algoritmo Wavefront Parallel Processing (WPP) definido en el estándar, logrando evitar la disminución paralelismo existente al inicio y al final de un cuadro. De manera alternativa, en [5] se analizan las dependencias existentes en el módulo ME del codificador, proponiendo una arquitectura altamente paralela diseñada para plataformas many-core. Otros trabajos hacen uso de arquitecturas heterogéneas, especialmente aquellas basadas en GPU [6]. Sin embargo, tal y como se mencionó en la introducción de este artículo, estas técnicas no son suficiente para superar la complejidad computacional del codificador, siendo todavía necesario desarrollar algoritmos rápidos de codificación.

De manera complementaria a los trabajos anteriores, existen otros que centran sus esfuerzos en reducir el tiempo de codificación desde un punto de vista algorítmico. El planteamiento seguido por muchos de ellos es limitar la profundidad del árbol de decisión con el objetivo de no procesar aquellas particiones en las cuales es poco probable encontrar una mejor predicción a la actual [7]. Otros, en cambio, buscan reducir la complejidad computacional del módulo ME. Por ejemplo, los autores de [8] hacen uso de

variables espacio-temporales para reducir el número de particiones a comprobar. Un estudio de algoritmos combinando ambas técnicas puede encontrarse en [9], donde se presentan resultados de más del 50% de reducción de tiempo a costa de pequeñas penalizaciones en eficiencia de codificación. En lo que se refiere al algoritmo propuesto en este artículo, ha de tenerse en cuenta que puede ser utilizado junto con todos los algoritmos anteriormente mencionados, dado que sus decisiones se toman a un nivel superior y, por tanto, todavía han de llevar a cabo la estimación de movimiento. En consecuencia, podría llegar a conseguirse mayores aceleraciones.

Con respecto a algoritmos de pre-análisis, merece la pena mencionar el módulo look-ahead del conocido codificador de H.264/AVC denominado x264 [10, 11]. Este módulo toma una versión submuestreada de los cuadros de entrada y los divide en bloques cuadrados de tamaño fijo, calculando sus correspondientes costes inter e intra. Con esta información, este codificador es capaz de determinar el patrón de codificación ideal, acelerar algunas operaciones de codificación o ajustar los parámetros de control de tasa. Sin embargo, este módulo look-ahead está diseñado a partir de H.264/AVC, el cual implica muchas menos opciones de particionamiento que HEVC, y aunque ha sido asimilado en x265 [12] (la versión de x264 para HEVC), no se ajusta completamente a las necesidades de este nuevo estándar. Otros trabajos hacen uso de algoritmos de pre-análisis con el objetivo de, por ejemplo, ajustar el algoritmo de control de tasa [13] o analizar la textura de la imagen en codificaciones tipo intra [14]. En ambos casos, dado que el ámbito de estos trabajos no está relacionado con la estimación de movimiento, el algoritmo propuesto en este artículo sería totalmente compatible.

III. ESTIMACIÓN DE MOVIMIENTO BASADA EN UNA ETAPA DE PRE-ANÁLISIS

El objetivo principal de la técnica propuesta es el de reducir la complejidad computacional del módulo ME, el cual, como ya se ha visto previamente, es el que más tiempo consume de todo el codificador. En este sentido, la manera en que se consigue esta reducción es mediante el cálculo de información preliminar de la secuencia de entrada en la etapa de pre-análisis, la cual es usada posteriormente en la etapa de codificación tal y como se muestra en la Figura 2. Como puede observarse, la etapa de pre-análisis lleva a cabo una estimación rápida de movimiento, mientras que el módulo ME hace uso de la información predicha en dicha etapa para limitar el número de cuadros de referencia y reducir el tamaño del área de búsqueda. Ambos pasos serán descritos en más detalle en las subsecciones que siguen a continuación.

A. Etapa de pre-análisis

La meta de la etapa de pre-análisis es realizar una estimación del movimiento existente en un cuadro de manera rápida. Esto se lleva a cabo a través de un proceso similar a aquél realizado en el módulo ME.

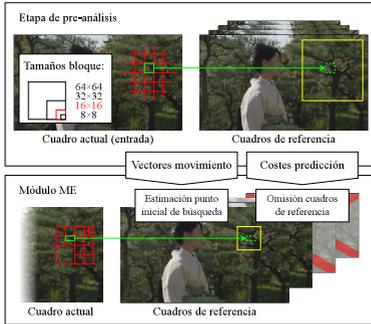


Fig. 2. Interacción entre la etapa de pre-análisis y el módulo ME según la técnica propuesta

Este último, sin embargo, supone una gran carga computacional para el codificador, dado que necesita comprobar cada posible PU por cada nivel de profundidad en el árbol de particionamiento. Por este motivo, esta operación se ha simplificado en la etapa de pre-análisis mediante el uso de estructuras como bloques de tamaño fijo [15], lo cual genera como resultado una matriz de vectores de movimiento (MV) y unos costes estimados por cada cuadro de referencia relativo al cuadro actual. No obstante, diversas pruebas experimentales demostraron que la estimación obtenida a través de este patrón fijo no es del todo precisa dado el amplio rango de posibilidades y tamaños de PU que ofrece el estándar HEVC. En su lugar, se demostró que realizar la estimación para todos los posibles tamaños de PU $2N \times 2N$ (64×64 , 32×32 , 16×16 and 8×8) proporciona una información mucho más útil para el codificador, dado que estos bloques pueden adaptarse a los PU de manera mucho más sencilla y flexible. Así pues, la técnica propuesta divide el cuadro actual en bloques de todos los tamaños anteriormente mencionados, y realiza una estimación rápida de movimiento en cada cuadro de referencia. Por tanto, el resultado de esta etapa es la estimación de los MVs y sus costes por cada bloque y cuadro de referencia.

Con tal de obtener el esquema de MVs más fiable posible, se han tenido en cuenta las siguientes consideraciones:

- El algoritmo de estimación de movimiento es el mismo que el utilizado en la etapa de codificación, si bien su precisión se limita únicamente a muestras enteras, descartando la búsqueda subpíxel, ya que tan sólo se pretende realizar una estimación de los MVs.
- Los MVs vecinos al bloque actual son considerados predictores de la misma manera que el estándar define Advanced Motion Vector Prediction (AMVP) para el codificador [16].
- Al estimar el movimiento de bloques 8×8 , la búsqueda utiliza un patrón de tamaño 16×16 en

su lugar, dado que se ha observado experimentalmente que capturar el contexto del bloque ofrece mejores estimaciones.

- Se hace uso del siguiente modelo lagrangiano de Rate-Distortion (R-D):

$$J_{MV} = Distortion_{MV} + \lambda \cdot Rate_{MV} \quad (1)$$

donde J_{MV} representa la función de coste, $Distortion_{MV}$ es la función de distorsión en términos de suma de diferencias absolutas (SAD), λ es el multiplicador de Lagrange para la ME, y $Rate_{MV}$ representa los bits necesarios para codificar el MV.

B. Etapa de estimación de movimiento

Una vez que la etapa de pre-análisis se ha llevado a cabo, el módulo ME tiene a su disposición toda la información de movimiento del cuadro actual. Si bien esta información puede ser utilizada de muchas maneras, el propósito de la técnica propuesta es el de reducir la complejidad computacional de la ME. En este sentido, los valores estimados serán usados tanto para reducir el número de cuadros de referencia comprobados, como para predecir el punto inicial de la búsqueda.

En primer lugar, resulta necesario determinar cómo adaptar cada bloque de pre-análisis y su información estimada con cada PU. Si el PU tiene un tamaño $2N \times 2N$, entonces existe un bloque de pre-análisis que se corresponde exactamente con dicho PU. Para el resto de tamaños, el PU correspondiente puede formarse a partir de los dos bloques de pre-análisis de menor tamaño que ocupan la misma porción de la imagen, excepto para el caso de 8×4 y 4×8 , en cuyo lugar se emplea el bloque 8×8 correspondiente, que es el más pequeño calculado en la etapa de pre-análisis. En caso de que la información se infiera de dos bloques, se calcula el coste asociado a ambos MVs, tomando sólo aquel que suponga el menor coste. En consecuencia, el resultado de este primer paso para cada PU es un MV y su correspondiente coste de predicción por cada cuadro de referencia.

Tal y como se anticipó anteriormente, los costes de predicción estimados permiten al codificador seleccionar aquellos cuadros de referencia que deberían ser comprobados para cada PU. No obstante, esta decisión cobra mayor complejidad cuando se considera la bipredicción, ya que en este caso se emplean dos cuadros de referencia en lugar de uno. En este sentido, es estadísticamente conocido que los cuadros de referencia empleados en PUs de tamaños distintos a $2N \times 2N$ son generalmente aquellos utilizados en el PU $2N \times 2N$ del mismo nivel de profundidad. Por este motivo, nuestra propuesta comprueba todos los cuadros de referencia en PUs de tamaño $2N \times 2N$, de manera que si el mejor cuadro de referencia predicho por la etapa de pre-análisis para un bloque que no sea $2N \times 2N$ coincide, se usará únicamente ese cuadro, omitiendo el resto. En el caso de bipredicción, si el cuadro de referencia escogido por la etapa de pre-

TABLA I
 RESULTADOS DEL ALGORITMO PROPUESTO AL SELECCIONAR
 CUADROS DE REFERENCIA

	Secuencia	BD-rate (%)	Reducción de tiempo (%)	Cuadros ref. omitidos (%)
A	Traffic	0.13	9.48	26.81
	PeopleOnStreet	0.24	7.17	29.56
	NebutaFestival	0.24	5.99	25.91
	SteamLocomotive	0.29	7.69	31.37
	Kimono	0.10	7.28	25.78
B	ParkScene	0.12	8.19	25.24
	Cactus	0.13	9.88	32.27
	BasketballDrive	0.18	6.96	27.06
	BQTerrace	0.40	8.91	25.40
C	BasketballDrill	0.19	8.87	32.67
	BQMall	0.29	8.49	28.11
	PartyScene	0.35	7.64	27.37
	RaceHorses	0.44	6.86	30.09
D	BasketballPass	0.18	7.43	30.37
	BQSquare	0.32	6.63	25.26
	BlowingBubbles	0.22	8.82	29.25
	RaceHorses	0.36	7.18	29.46
Valores medios		0.25	7.80	28.35

TABLA II
 RESULTADOS DEL ALGORITMO PROPUESTO AL REDUCIR EL
 ÁREA DE BÚSQUEDA EN LA ETAPA ME

	Secuencia	BD-rate (%)	Reducción de tiempo (%)	Reducción SAD (%)
A	Traffic	0.07	5.02	65.53
	PeopleOnStreet	0.04	4.50	86.57
	NebutaFestival	0.04	5.13	77.34
	SteamLocomotive	-0.86	6.42	86.08
	Kimono	0.04	6.58	81.67
B	ParkScene	0.07	5.80	71.02
	Cactus	0.13	5.16	80.10
	BasketballDrive	0.01	7.58	86.68
	BQTerrace	0.21	5.06	66.60
C	BasketballDrill	-0.06	5.78	83.74
	BQMall	0.07	5.19	77.57
	PartyScene	0.32	5.66	75.49
	RaceHorses	0.53	7.29	89.18
D	BasketballPass	-0.02	6.24	85.91
	BQSquare	0.37	4.77	59.92
	BlowingBubbles	0.16	4.58	67.99
	RaceHorses	0.24	6.61	86.42
Valores medios		0.08	5.73	78.11

análisis coincide con alguno de los dos utilizados en el bloque $2N \times 2N$ de la misma profundidad, entonces ambos cuadros de referencia son utilizados, omitiendo el resto. Si en ninguno de los dos casos los índices de los cuadros de referencia coinciden, entonces todos ellos se comprueban para ese PU con el objetivo de prevenir decisiones incorrectas.

A la vez que resulta posible escoger los cuadros de referencia a comprobar, los MVs predichos en la etapa de pre-análisis pueden ser utilizados para reducir la complejidad computacional del módulo ME. Más específicamente, los MVs obtenidos de dicha etapa pueden usarse como punto inicial de búsqueda en la ME, reduciendo de manera notable el tamaño del área de búsqueda por defecto de 128×128 a, por ejemplo, 8×8 . En consecuencia, en algoritmo de búsqueda TZSearch [17] implementado por defecto puede ser reemplazado por otro que realice una búsqueda de carácter local. En el caso propuesto por este artículo, se ha empleado un patrón de búsqueda hexagonal modificado capaz de reducir el número de operaciones SAD realizadas por el módulo ME. Además, con el objetivo de asegurar que el MV estimado es un buen candidato para la búsqueda, se comprueba si dicho MV es similar a alguno de los candidatos del algoritmo AMVP del codificador. Si el MV predicho se encuentra en un radio de 3 píxeles, dicho vector se escoge como punto inicial de la búsqueda y el candidato AMVP más cercano se toma como predictor. Si no se encuentra en ese radio, se calcula los costes de todos estos vectores, escogiendo el punto inicial de búsqueda y el correspondiente predictor en función de dichos costes.

IV. EVALUACIÓN EXPERIMENTAL

Los experimentos han sido llevados a cabo basándose en el software de referencia HEVC Test Model (HM-16.6) [18], siguiendo las indicaciones y las condiciones de codificación provistas por el documento elaborado por la propia JCT-VC [19]. En

este sentido, se ha empleado la configuración Random Access y los valores de QP 22, 27, 32 y 37. Las codificaciones se han realizado bajo el perfil Main y con una profundidad de 8 bits por píxel. El conjunto de prueba incluye secuencias de clases A a la D conforme se clasifican en el documento anteriormente mencionado, incluyendo las siguientes resoluciones: 2560×1600 (A), 1920×1080 (B), 832×480 (C) y 416×240 (D). El tamaño de área de búsqueda de la etapa de pre-análisis se ha establecido a 128×128 , que es el equivalente al valor por defecto utilizado en el módulo ME del codificador de referencia.

La plataforma hardware utilizada para los experimentos se compone de procesadores Intel® Xeon® E5-2630L v3 a una frecuencia de 1.80 GHz, y memoria principal de 16 GB. El codificador ha sido compilado con GCC 4.8.5-4 y ejecutado en CentOS 7 (Linux 3.10.0-327). La tecnología Turbo Boost fue desactivada al inicio de las pruebas para conseguir la reproducibilidad de los resultados.

Los resultados serán presentados en términos de reducción de tiempo y Bjøntegaard Delta Rate (BD-rate). El BD-rate permite medir la eficiencia de codificación, ya que representa el porcentaje de variación en la tasa de bits entre dos secuencias que tienen la misma calidad objetiva [20]. Un valor positivo, por tanto, implicaría un incremento en la tasa de bits, es decir, una reducción de la eficiencia de codificación. Adicionalmente, también se indicará el porcentaje de cuadros de referencia omitidos y el porcentaje de reducción de operaciones SAD en el módulo ME en los casos que corresponda.

En primer lugar, la Tabla I muestra los resultados obtenidos cuando el algoritmo de pre-análisis propuesto predice únicamente qué cuadros de referencia ha de utilizar el módulo ME. En otras palabras, el área de búsqueda se mantiene a su tamaño por defecto de 128×128 , se emplea el algoritmo TZSearch y el punto inicial de búsqueda lo decide el codificador en lugar de la etapa de pre-análisis. Como puede

TABLA III
 RESULTADOS DE APLICAR LAS TÉCNICAS DE SELECCIÓN DE CUADROS DE REFERENCIA Y REDUCCIÓN DEL ÁREA DE BÚSQUEDA

Secuencia	BD-rate (%)	Reducción de tiempo (%)	Cuadros de referencia omitidos (%)	Reducción SAD (%)	
A	Traffic	0.19	16.44	73.31	
	PeopleOnStreet	0.22	16.39	90.09	
	NebutaFestival	0.27	12.73	82.52	
	SteamLocomotive	-0.54	18.17	89.72	
B	Kimono	0.06	16.96	85.84	
	ParkScene	0.16	15.76	77.18	
	Cactus	0.29	17.46	85.48	
	BasketballDrive	0.23	17.81	89.89	
C	BQTerrace	0.57	14.81	73.65	
	BasketballDrill	0.10	17.32	88.23	
	BQMall	0.33	16.05	82.94	
	PartyScene	0.58	14.72	80.99	
D	RaceHorses	0.92	17.57	92.14	
	BasketballPass	0.16	16.75	89.67	
	BQSquare	0.70	13.63	67.84	
	BlowingBubbles	0.54	14.63	75.82	
	RaceHorses	0.62	16.56	29.89	90.04
Valores medios		0.32	16.10	28.84	83.25

observarse, prácticamente el 30% de los cuadros de referencia son descartados al llevar a cabo la ME en el codificador, siendo un resultado recurrente en todas las secuencias. Como resultado, el tiempo total de codificación se ve reducido en un 7.80% de media, afectando a la eficiencia de codificación en tan sólo un 0.25%.

De manera similar, la Tabla II también muestra los resultados ofrecidos por el algoritmo de pre-análisis, pero en esta ocasión sin omitir ningún cuadro de referencia. En su lugar, los MVs obtenidos en la etapa de pre-análisis son empleados para reducir el área de búsqueda tal y como fue descrito en la Sección III-B. Consecuentemente, para este experimento se emplea un área de búsqueda de 8×8 píxeles. En este caso, puede observarse como la reducción de tiempo es algo más moderada que en el anterior, ya que esta técnica engloba una parte más reducida del codificador. No obstante, es capaz de ahorrar el 78.11% de las operaciones SAD realizadas en la búsqueda unipredictiva de precisión entera del módulo ME, lo que supone un 5.73% de reducción en el tiempo de codificación. Por su parte, la eficiencia de codificación tan sólo se ve afectada en un 0.08%.

Por último, la Tabla III muestra los resultados obtenidos por el algoritmo de pre-análisis propuesto al combinar las dos alternativas antes descritas. Al igual que en el caso anterior, el área de búsqueda se ha establecido a 8×8 píxeles para el módulo ME. Los resultados muestran que el número de cuadros de referencia omitidos es aproximadamente el mismo que en la Tabla I, lo cual indica que el algoritmo de selección de cuadros de referencia no se ve afectado por las decisiones del algoritmo de reducción del área de búsqueda. En relación al número de operaciones SAD realizadas en el módulo ME, puede observarse que ha podido ahorrarse más del 80% de éstas, reduciendo en gran medida el tiempo dedicado a este paso. Este porcentaje es mayor al mostrado en la Tabla II debido a la influencia del algoritmo de selección de cuadros de referencia. Como resul-

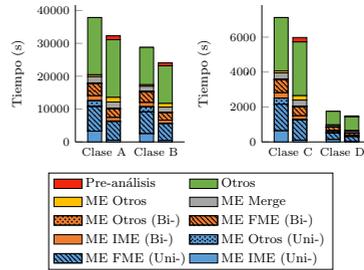


Fig. 3. Perfil de tiempos del codificador de referencia (barra de la izquierda) comparado con la propuesta (barra de la derecha)

tado de ambos algoritmos, puede observarse que el tiempo total de codificación puede reducirse en un 16.10% de media a costa de un insignificante incremento de 0.32% en eficiencia de codificación, lo cual es prácticamente imperceptible en la mayor parte de los casos. También es importante observar que esta reducción de tiempo de codificación es superior a la suma de la reducción de ambas técnicas por separado, ya que la información de pre-análisis se calcula una sola vez para ambas.

Como consecuencia del algoritmo de pre-análisis, el tiempo de codificación sufre una reestructuración como la mostrada en la Figura 3. Estos valores de tiempo han sido extraídos de los valores medios de los perfiles de tiempo de todas las secuencias codificadas y todos los valores de QP. Como puede observarse, una parte sustancial del tiempo dedicado al módulo ME se ve reducida. Por un lado, esto se debe al hecho de que algunos cuadros de referencia son omitidos. Por otro lado, el número de operaciones SAD realizadas en la ME unipredictiva de precisión entera es tan reducido que su fracción

de tiempo correspondiente apenas se aprecia en la figura. También puede observarse que, sin embargo, el tiempo del bloque "ME Otros" se ve incrementado, ya que es donde se realiza el cálculo de costes y las decisiones a partir de la información de la etapa de pre-análisis, así como el algoritmo AMVP, que antes tenía lugar en "ME Otros (Uni-)" en el codificador de referencia. La etapa de pre-análisis, a su vez, introduce una pequeña sobrecarga menor al 4% total. Sin embargo, esto no supone latencia alguna al codificador, ya que la reducción de tiempo es superior al tiempo dedicado a esta etapa.

V. CONCLUSIONES Y TRABAJO FUTURO

La llegada de HEVC ha abierto la puerta a multitud de nuevas aplicaciones y formatos de video tales como UHD con el objetivo de satisfacer las demandas de calidad de experiencia del mercado. Sin embargo, la mejora en eficiencia de codificación de HEVC ha llevado a su vez a un gran incremento de la complejidad computacional del codificador. Como consecuencia, los codificadores basados en este estándar requieren algoritmos rápidos y eficientes para poder alcanzar la codificación en tiempo real. Por este motivo, este artículo propone un algoritmo de pre-análisis diseñado para extraer la información de movimiento de una imagen, la cual es utilizada posteriormente en el módulo ME para reducir el tiempo de codificación. Por un lado, los costes estimados se utilizan para seleccionar los cuadros de referencia que serán empleados en la búsqueda. Por otro lado, los MVs predichos permiten realizar una búsqueda local y reducir el área de referencia correspondiente. Además, este artículo también detalla las consideraciones que han sido tenidas en cuenta en el diseño del algoritmo propuesto dadas las particularidades del estándar HEVC.

La evaluación experimental del algoritmo ha mostrado que pueden llegar a omitirse hasta el 30% de los cuadros de referencia, y que el número de operaciones SAD puede verse reducido, a su vez, en más del 80% en el módulo ME. Como consecuencia, el tiempo total de codificación puede reducirse un 16.10% con un incremento inapreciable del BD-rate del 0.32%.

Entre las líneas de trabajo futuro se encuentran calcular los costes y el modo de predicción intra en la etapa de pre-análisis para asistir al módulo de codificación intra. Con ello, sería posible diseñar un algoritmo de selección de modo inter/intra. Adicionalmente, se podría considerar extender esa línea a un algoritmo de particionamiento CTU, entre otras muchas aplicaciones.

AGRADECIMIENTOS

Este trabajo ha sido cofinanciado por el Ministerio de Economía y Competitividad y la Comisión Europea bajo el proyecto TIN2015-66972-C5-2-R (MINECO/FEDER), y por el Ministerio de Educación, Cultura y Deporte mediante la beca FPU 13/04601.

REFERENCIAS

- [1] ISO/IEC and ITU-T, "Advanced Video Coding for Generic Audiovisual Services. ITU-T Recommendation H.264 and ISO/IEC 14496-10 (version 10)," Feb. 2016.
- [2] ISO/IEC and ITU-T, "High Efficiency Video Coding (HEVC). ITU-T Recommendation H.265 and ISO/IEC 23008-2 (version 3)," Apr. 2015.
- [3] J.-R. Ohm, G. J. Sullivan, H. Schwarz, Thiew Keng Tan, and T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards - Including High Efficiency Video Coding (HEVC)," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1669-1684, Dec. 2012.
- [4] Chi Ching Chi, M. Álvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel Scalability and Efficiency of HEVC Parallelization Approaches," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1827-1838, Dec. 2012.
- [5] Chenggang Yan, Yongdong Zhang, Jizheng Xu, Feng Dai, Jun Zhang, Qionghai Dai, and Feng Wu, "Efficient Parallel Framework for HEVC Motion Estimation on Many-Core Processors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 12, pp. 2077-2089, Dec. 2014.
- [6] G. Cebrián-Márquez, J. L. Hernández-Losada, J. L. Martínez, P. Cuenca, Minhao Tang, and Jiangtao Wen, "Accelerating HEVC using heterogeneous platforms," *J. Supercomput.*, vol. 71, no. 2, pp. 613-628, Feb. 2015.
- [7] Seunghyun Cho and Munchul Kim, "Fast CU Splitting and Pruning for Suboptimal CU Partitioning in HEVC Intra Coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 9, pp. 1555-1564, Sept. 2013.
- [8] Liqun Shen, Zhaoyang Zhang, and Zhi Liu, "Adaptive Inter-Mode Decision for HEVC Jointly Utilizing Inter-Level and Spatiotemporal Correlations," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 10, pp. 1709-1722, Oct. 2014.
- [9] Chae Eun Rhee, Kyujoong Lee, Tae Sung Kim, and Hyuk-Jae Lee, "A Survey of Fast Mode Decision Algorithms for Inter-Prediction and Their Applications to High Efficiency Video Coding," *IEEE Trans. Consum. Electron.*, vol. 58, no. 4, pp. 1375-1383, Nov. 2012.
- [10] "x264 Source Code Repository," <http://git.videolan.org/git/x264.git>.
- [11] J. Garrett-Glaser, "A Novel Macroblock-Tree Algorithm for High-Performance Optimization of Dependent Video Coding in H.264/AVC," Tech. Rep., Department of Computer Science, Harvey Mudd College, 2009.
- [12] "x265 Source Code Repository," <http://hg.videolan.org/x265>.
- [13] Lin Sun, O. C. Au, Cong Zhao, and F. H. Huang, "Rate distortion modeling and adaptive rate control scheme for high efficiency video coding (HEVC)," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, June 2014, pp. 1933-1936.
- [14] T. Mallikarachi, A. Fernando, and H. K. Arachchi, "Efficient coding unit size selection based on texture analysis for HEVC intra prediction," in *IEEE International Conference on Multimedia and Expo (ICME)*, July 2014, pp. 1-6.
- [15] G. Cebrián-Márquez, Chi Ching Chi, J. L. Martínez, P. Cuenca, M. Álvarez-Mesa, S. Sanz-Rodríguez, and B. Juurlink, "Reducing HEVC Encoding Complexity Using Two-Stage Motion Estimation," in *IEEE International Conference on Visual Communications and Image Processing (VCIP)*, Dec. 2015.
- [16] G. J. Sullivan, J.-R. Ohm, Woo-Jin Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649-1668, Dec. 2012.
- [17] H. Kibeya, F. Belghith, H. Loukil, M. A. Ben Ayed, and N. Masmoudi, "TZSearch pattern search improvement for HEVC motion estimation modules," in *1st International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, Mar. 2014, pp. 95-99.
- [18] "HEVC Test Model (HM) Reference Software," <https://hevc.hhi.fraunhofer.de/>.
- [19] F. Bossen, "Common Test Conditions and Software Reference Configurations," Tech. Rep. JCTVC-L1100, Jan. 2013.
- [20] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," Tech. Rep. VCEG-M33, ITU-T Video Coding Experts Group (VCEG), 2001.

Robust High-speed Eye Pupil Tracking System on GPUs

Juan Mompeán^{1,2}, Juan L. Aragón¹, Pedro M. Prieto² y Pablo Artal²

Resumen— Pupil tracking under infrared illumination is an important tool for many researchers in physiological visual optics and ophthalmology. It is also a relevant topic for gaze tracking which is used in psychological and medical research, marketing, human-computer interaction, virtual reality and other areas. A typical setup can be either a low-cost webcam with some infrared LEDs or glasses with mounted cameras and infrared illumination. In this work, we evaluate and parallelize several pupil tracking algorithms with the aim of estimating the pupil's position and size with high accuracy in order to develop a high-speed pupil tracking system. To achieve high processing speed the original non-parallel algorithms have been parallelized by using CUDA and OpenMP. Graphics cards are designed to process images at very high frequencies and resolutions, and CUDA enables them to be used for general purpose computing. Our experimental results show that pupil tracking can be efficiently performed at high speeds with high-resolution images (up to 530 Hz with images of 1280x1024 pixels) using a state-of-the-art GP-GPU.

I. INTRODUCTION

Pupil tracking is an important approach to know key information about the eye and also for supporting a lot of different applications on both the research and commercial arena, specially when used for gaze tracking. There are many application cases of the later, being an interesting one the remote control of computers with the subject's gaze: it can be used instead of the mouse, for moving the camera in 3D virtual reality scenarios. But it is also a powerful tool for disabled people with reduced movement capabilities who can control a computer with their gaze.

Pupil tracking is very often performed by using infrared illumination because it does not perturb the subject, neither annoys him nor it affects the eye's properties since the subject cannot see the infrared light. In particular, the pupil size does not change with infrared illumination, which could be useful in some experiments. Pupil tracking under infrared illumination is commonly used in visual optics experiments and in ophthalmic instruments. In some cases it is a key component of the system because the actions that are performed need to be done in the center of the pupil, which in turn needs to be calculated with high accuracy and at a high speed. While for eye gazing a low precision on the pupil determination does not affect much the results [1], in some visual optics applications a low precision can lead to important errors in the final results.

In this paper we parallelize and evaluate the aforementioned pupil tracking algorithms in differ-

ent computing systems (from general purpose multicore CPUs to state-of-the-art GP-GPUs) by using both OpenMP [2] and CUDA [3] programming environments, with the final goal of designing a high performance (up to 85.5x speedup compared to their corresponding sequential implementations) that allows for high speed processing (up to 530 frames per second) while not sacrificing a high detection accuracy (more than 90% of the pupils in the dataset are detected with an error in their ratio lower than 5%).

II. BACKGROUND AND RELATED WORK

Different algorithms have been developed to track the pupil position and size under different conditions. A well known approach is the Starburst algorithm [4] which is based on a series of rays that extend from an initial position to the edges of the image. A change over a threshold is searched in the intensity derivative of the pixels in each of the rays. Then new rays are created, that go from the newly found points to the initial position. By doing this, points on the other side of the pupil are found. Since only the pixels that belong to the rays are analyzed, a small amount of pixels of the image are accessed, which reduces the computation time. While it is an iterative algorithm, experimental results have shown that in most cases it converges after two or three iterations.

Another simpler approach is to use the Hough Transform on the edges detected with the Canny algorithm [5]. This method is also used by Masek [6] with a specific pre-processing for removing the eyelids. Although it was initially proposed for segmenting the iris, it can be used for pupil tracking as well. It is a simple approach, however, its accuracy are very dependent of the input set of pictures. In addition, this algorithm is very sensitive to noise added by eyelashes or even other smaller features of the eye. Furthermore, it is a very slow approach when used in large images with a wide range of pupil's radii since it will search sequentially for all the targeted pupil sizes in all the pixels of the image.

GPU processors have been used for pupil tracking with color images [7]. Borovikov proposed a custom blob detector; he assumes that the pupil will be near the center of the image and searches iteratively a dark blob. In each iteration the center of mass of a circle around the current estimated center is calculated providing higher values to the darker pixels. After each iteration the initial radio is decreased and the algorithm stops when the center position does not change between two consecutive iterations. Another GPU approach for pupil tracking was proposed by Mulligan in [8]. It is based on searching an area

¹Dept. de Ingeniería y Tecnología de Computadores, Univ. de Murcia, e-mail: {juan.mompean, jlaragon}@um.es.

²Laboratorio de Óptica, IUIoYN, Univ. de Murcia, e-mail: {pgrito, pablo}@um.es.

within a threshold in the image. This approach may have problems to find the pupil when the eyelashes are partially hiding it or when there are corneal reflections inside the pupil. Under good experimental conditions it achieves a processing speed of up to 250 Hz but for low resolution images.

III. GPU-BASED REAL TIME PUPIL TRACKING

The main goal of this work is to parallelize several state-of-the-art pupil detection algorithms to develop a high speed pupil tracking system capable of processing high quality images with an accurate pupil determination. A deep exploration and a throughout evaluation must be carried out, by varying the input parameters of these algorithms to determine how they behave with different tuning values in order to characterize their run-time performance and the achieved accuracy, in terms of relative error compared to the correct pupil size and position.

A. Preprocessing

In order to perform a fair comparison of the algorithms, the same preprocessing is applied to all the images. The set of images used for the experiments (refer to Section IV for additional details) are sharp enough and they do not have significant noise. However, as it will be also explained in Section IV, the illumination setup added corneal reflections. Every image taken with direct infrared illumination will have similar reflections, so an algorithm for removing them is used.

The algorithm used for this purpose was proposed by Aydi *et al.* [9]. It is a simple method that applies a top-hat transform, followed by an image thresholding, an image dilation and, finally, an interpolation of the detected points.

After the corneal reflection removal step, a Gaussian filter is applied (with a kernel size of 5 and a sigma of 2). Similarly to the approach used with the erosion and dilation two separable kernels are used. These CUDA optimizations are further described in Section V. A median filter has been also tested as a replacement for the Gaussian filter.

B. Threshold and Labelling Algorithm

A similar approach to the proposed by Rankin *et al.* [10] is used. Although instead of selecting the central blob we have modified the original algorithm to select the biggest blob. For selecting the threshold several values are tested using an iterative algorithm which tests values ranging from 20 to 100. After applying the threshold to the image, as described before, the labelling process is performed. A state-of-the-art labelling algorithm by Chen [11] was used.

After labelling the image, the histogram of the image is calculated. By calculating the histogram of the labelled image the biggest labelled area can easily be found by searching the maximum value within the histogram. Once the maximum label is found the image is scanned to remove the rest of labels while

keeping the biggest labelled area. Then the image is scanned for searching the border of the labelled area.

With the list of found border pixels a RANSAC (Random Sample Consensus) circle fit is performed [12] to determine the pupil. Every time that RANSAC is performed 1024 fits using the method developed by Taubin [13] are calculated, using 1024 sets of 5 points. Afterwards, the votes are computed and a reduction is performed to add the votes of each circle. Finally, the circle with the maximum number of votes is selected as the best fit.

C. Starburst Algorithm

The second algorithm is called Starburst, developed by Dongheng and Parkhurst [4]. It is based on the fact that the pupil border is usually the place with the biggest gradient values. This algorithm is very sensitive to the corneal reflections, so the preprocessing step for removing them is crucial.

Starburst needs a starting central point of the pupil. When no previous information is available the center of the image is typically selected, otherwise the previous center is used. A variable number of rays, we have used 20, are "launched" from the center to the limits of the image. Each ray is projected in a different direction, dividing a circle in equal portions. The ray uses the pixels that are in its direction to calculate its gradient which is later inspected to search the first value bigger than a threshold. The result of this stage are the points found in the rays.

In the second stage of the Starburst algorithm those points are used to create new rays from them to the center. The new rays are limited to an angle of $\pm 50^\circ$. The purpose is to find points on the other side of the pupil and avoid going out of it.

Finally, the points from both stages are joint in the same array and a RANSAC circle fit is performed. The distance between the new center and the old center is calculated and, if it is smaller than 10 pixels, the algorithm finishes. Otherwise, another iteration is executed until it converges or ten iterations have been performed.

D. Canny Edge Detector and Hough Transform Algorithm

The third evaluated algorithm uses the Canny edge detector [14] and the Hough Transform [15]. For performing the Canny edge detection, the image is smoothed horizontally and vertically with a 1-D Gauss kernel and with the first derivative of a Gauss kernel. Four images are generated as the result of the process. Then the gradient value and direction are calculated using a naïve CUDA implementation.

With the result of the Canny edge detector the Hough Transform is applied. The Hough Transform will check for each pixel every possible radio in the range selected. The final result is the circle which obtains the maximum number of votes.

IV. EXPERIMENTAL FRAMEWORK

While many different setups can be used for performing pupil tracking, our approach consists of a set of infrared LEDs and a camera focusing at the eye. In order to track the pupil the subject sits in front of the camera and stays there while the tracking is running. The tracking can be done in real-time at the same time that the subject is standing in front of the camera or later if a video is captured.

The GPU used for testing the CUDA programs is a high-end NVIDIA GeForce GTX 980, with 2048 CUDA cores and 4GB of memory. This GPU has one of the biggest amount of cores currently available. That enables our programs to exploit all the parallelism in the algorithms. For the CPU and OpenMP experiments an Intel i7-2600K processor with 4 physical cores and hyperthreading has been used.

Pupil Images Dataset. The experiments were performed using 964 pictures of 1280x1024 pixels taken with the described optical setup from 51 different subjects. A subset of this image collection was used in [16]. The dataset contains pupils with a wide variety of radii, ranging from 1.66mm to 4.28mm, with a mean of 3.21mm.

V. ACCELERATING PUPIL TRACKING

The following paragraphs explain the different CUDA optimizations that have been used for the pupil tracking algorithms. Note that while some optimizations apply to all of the algorithms others only apply to Starburst. Figure 1 summarizes the speedup obtained by the Starburst algorithm for each CUDA optimization (and also an OpenMP implementation running in a multicore CPU) over a sequential implementation in a high-end CPU. The CUDA optimizations are cumulative and while each of them separately would not make a big performance impact, together, they are able to significantly reduce the global execution time.

Naïve CUDA Implementation. A naïve CUDA implementation directly maps the C code into CUDA. While the code is not optimized some good practices are followed: unnecessary memory copies are avoided, intrinsic functions are preferred (high precision is not necessary) and optimal block sizes are selected. The naïve implementation achieves a tracking speed of 38.6 frames per second and a speedup of 6.3x.

Pre-calculating random numbers. Random numbers are used to select the points used in the RANSAC method. They are generated with the *cu-RAND* library [17] and then are scaled to match the range of the number of points previously found. Generating random numbers is an expensive task, so it should be avoided whenever possible. In the naïve implementation, the numbers are generated for each iteration of the Starburst, but that is inefficient. Since the points found will be different for each image and for each iteration the same random numbers can be used for all the images. It is still necessary to scale the numbers for each set of points to generate

random numbers between 0 and $number\ of\ points - 1$. After removing the generation of random numbers off the main loop and pre-calculating them at the beginning, the tracking speed is increased to 62.4 frames per second and the speedup comparing with the previous version is 1.6x (or 10x respect to the CPU).

Separable preprocessing kernels. The erosion, dilation and gaussian filter operations are performed with a $k \times k$ filter in the naïve implementation. As a result, they generate a lot of memory loads saturating the memory system while not fully utilizing the CUDA cores. Although new CUDA cards have big caches that partially mitigate the problem it is still possible to highly reduce the necessary bandwidth by dividing the kernel in two 1-dimensional kernels. The three operations can be divided in two kernels reducing the memory loads of each thread from $k \times k$ to $2 \times k$. And only an intermediate buffer, which can be pre-allocated, is needed. After adding the separable kernels the frames per second are increased dramatically up to 203.2 and a huge speedup of 32.7x is obtained.

Preprocessing kernels with shared memory. Although the memory bandwidth used by the preprocessing operations has been greatly reduced with the *separable kernels* optimization there is still room for improvement. Most of the pixels loaded by each thread are also loaded by others threads in the same block, so it is a good idea to preload the data once to shared memory [18] and do the calculations accessing the much faster shared memory. After adding the shared memory to the three preprocessing operations the frames per second are increased to 257.9 and the speedup over a CPU raises to 41.6x.

Preallocate memory. Allocating the memory in the GPU is relatively expensive, so allocating memory on a loop should be avoided. In the naïve implementation the memory is allocated for each image or even for each iteration of the Starburst for some variables that may slightly change its size through iterations. Since the maximum size of each dynamically allocated variable is known, they can be pre-allocated at the beginning and reused through the whole execution. After applying this optimization the obtained number of frames per second is 378.8 and the speedup increases to 61.1x.

Pinned memory. Copying the memory from the CPU to the GPU is an expensive task. Since the amount of time spent processing an image after applying the previous optimizations is very low, the overhead due to the memory copies is increasing its weight. It takes over 25% of the processing time to copy the images to the GPU. Each image is only copied once, so it cannot be reduced. However, it is possible to reduce the memory copy time by using pinned memory. After switching to pinned memory the number of frames per second obtained is 428.6 and the speedup over a CPU raises to 69.1x.

Reduction with shuffle instructions. The RANSAC is a key part of the Starburst algorithm, in order to find the best fitting circle it needs to sum

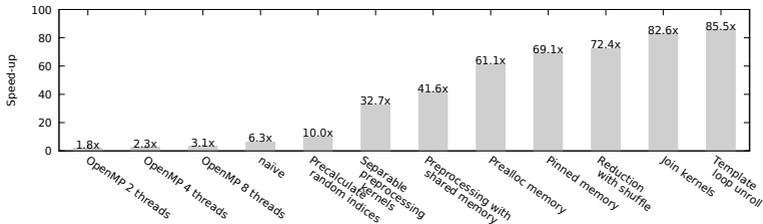


Fig. 1: Speedup of the parallelized StarBurst using OpenMP and the CUDA optimizations.

the votes from every point from all the circles. It is a reduction which is performed in parallel for over 100 points for the 1024 circles. The naïve approach with an `atomicAdd` is very slow, in the naïve implementation we are using a typical approach with shared memory [19]. However, using *shuffle instructions* provides a big improvement, although the total time spent doing the reduction is low and the achieved speedup is modest. The new number of frames per second raises to 449.4 and the resulting speedup increases to 72.4x.

Join kernels While separating the functionality may simplify the code, it can lead to reduced performance. There are three functions where separating the functionality has resulted in repeated accesses to memory and, as a consequence, wasted computing time. These three functions are: the last part of the top-hat functions (second step dilation and subtraction) and the thresholding; generating gradients and searching the points in Starburst; and the RANSAC with the reduction. All of these functions shared the same pattern, they were accessing the data generated from the previous function. After joining them they avoid storing data into global memory that would be later loaded. After the optimization 512.6 frames per second and a speedup of 72.4x are achieved.

Template unroll. CUDA is capable of using C++ templates, which are useful for reusing the same functions with different types but are also useful for compiling a function several times for different values of a variable. After adding templates to the gaussian, the erosion and the dilation filters, the radio of these convolutions has been parameterized enabling the compiler to unroll the loop for each of these functions. To do so a `switch` with different radio sizes has been added to keep a small amount of flexibility in the radio size. Unrolling the loops avoid the tests that must be performed in each iteration of a loop to check if it must stop or not, therefore, reducing the execution time. The final frames per second raises to 530.3 and the final achieved speedup is 85.5x.

VI. EXPERIMENTAL RESULTS EVALUATION

A. Starburst Results

Figure 2a shows the accuracy of the Starburst algorithm applying six different image preprocessing

configurations. Two of them use a Gaussian filter with a kernel size of 5 and a sigma of 2. Whereas four of them use a median filter with a kernel size of 7 or 11. The size of the corneal reflections removal kernel is also specified with values between 13-25.

The accuracy is measured as the percentage of images where the center and radio have been found within a given margin of error with respect to the correct position and size. E.g., if the error in the calculated radio is $0.1mm$, whereas the error in the center is $0.05mm$, and assuming a pupil radio of $2mm$ the relative margin of error are 5% and 2.5% for the pupil radio and the center, respectively. The biggest error is used, in this example a 5% corresponding to the pupil radio.

In Figure 2a it can be observed that there are some configurations with more than 85% of the images detected with an error under 5% and more than 95% detected with an error smaller than 10%. For the average pupil, which has a radio of $3.21mm$, a relative error of 5% would mean an absolute error of $0.16mm$ in the radio. But the error of the radio is on average 1% less than the error of the center: the same amount of pupils are detected within a margin of error of 5% for the center and within a margin of error of 4% for the radio. A relative error of 4% for the average radio would be an absolute error of $0.13mm$, and a relative error of 3% would be an absolute error of $0.096mm$.

In addition to that, a performance comparison is shown in Figure 2b. Clearly, the configurations using the median filter are slower than the configurations using the Gaussian filter due to the different performance of both methods.

B. Thresholding and Labelling Results

The accuracy of the Thresholding and Labelling algorithm (referred as T&L in Figure 3a) is quite good taking into account that this algorithm is very sensitive to overlappings of the eyelids and eyelashes. It is able to achieve an accuracy over 90% for small steps within a reasonable margin of error. Still its accuracy is decreased as the size of the steps is increased because it misses some good thresholds. The performance (Figure 3b) is directly related to the size of the steps used. Increasing the size of the steps decreases the execution time because it reduces

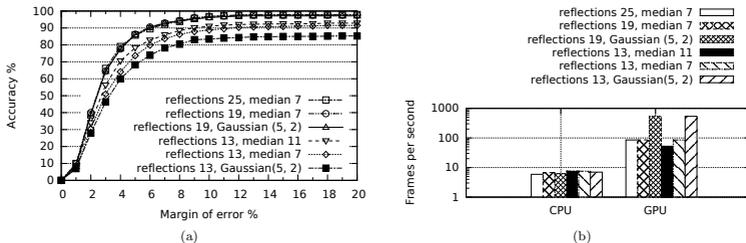


Fig. 2: (a) Starburst's accuracy with different preprocessing configurations. (b) Starburst's performance for the same configurations.

the number of iterations of the algorithm. Relatively high speeds are achieved with step of size 20 and 40. After examining the accuracy and the performance of the algorithm it is clear that there is a trade-off between both. Increasing one of them will decrease the other, so a balance must be found.

C. Hough Results

The Hough Transform turns out to obtain very accurate results (Figure 3a). It is able to detect the pupil correctly despite the iris, eyelashes and eyelids adding edges to the image. However, it is a very slow method (Figure 3b) since it is only capable of processing 4.7 frames per second in the GPU. The Hough Transform requires a huge computing power because it is checking all the possible radii for all the pixels.

D. Side-by-side comparison of the three approaches

A comparison of the accuracy (Figure 4a) and the performance (Figure 4b) helps to understand how well the algorithms behave. For the sake of simplicity only the best configuration for both the Starburst and for Thresholding and Labelling is shown. For Starburst a corneal reflections removal kernel of size 19 and a Gaussian filter of size 5 and sigma 2 are used. For Thresholding and Labelling a step of size 20 is chosen.

When considering both metrics, performance and accuracy, it stands out that the best overall algorithm is Starburst. Its accuracy has been measured to be really high, with 96.2% of the pupils detected within a margin of error of 10%. Although the Hough Transform is just slightly better (96.38% of the pupils detected within a margin of error of 10%) it exhibits a significant lower performance, 4.7 frames per second, making it unusable for real time applications. Contrarily, the Starburst algorithm achieves 530 frames per second for the mentioned accuracy. On the other hand, the Threshold and Labelling algorithm is much worse in terms of accuracy with 78.5% of the pupils detected within a margin of error of 10% (although other configurations of the T&L

have shown a higher accuracy, again, at the cost of degrading too much the performance) and also worse in terms of performance when compared to the Starburst, achieving only 61.9 frames per second. Summarizing, the GPU-accelerated version of Starburst, after applying all the CUDA optimizations discussed in the paper, is by far the fastest approach (able to achieve more than 500 frames per second) while at the same time exhibiting a high accuracy.

VII. CONCLUSIONS

Looking at the previous side-by-side comparison it is clear that highly accurate and fast pupil tracking have been achieved which enables high speed tracking of the pupil with small errors. This has been possible with the Starburst algorithm that has been parallelized by using CUDA. The speedup obtained with CUDA is 85.5x which is much more than the 3.1x obtained with OpenMP. Indeed, the speed of the algorithm is so high that it is possible to perform high frequency tracking of fast movements of the eye, such as saccades, with high quality images. High-speed cameras capable of capturing hundreds of frames per second are widely available and they can be used in combination with our parallelized algorithm to do real-time tracking with high frame rates. Furthermore, in real-time environments usually some time must be spent on communication with the cameras and the actuators. So the GPU cannot be processing all the time, therefore, having an algorithm faster than what it is theoretically needed enables real-time processing when considering the overhead of the communication and synchronization. The parallel implementation we have developed is a powerful tool that will help to improve visual optics applications which rely on pupil tracking, enabling speeds and a pupil tracking accuracy that were not possible previously. Furthermore, the stability requirements for the subject could be relaxed in some experiments.

VIII. ACKNOWLEDGMENTS

This research has been supported by the European Research Council Advanced Grant ERC-2013-AdG-

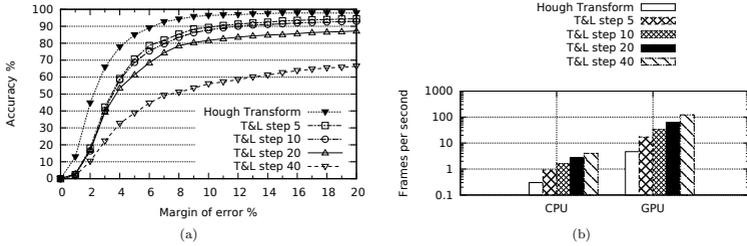


Fig. 3: (a) Hough and Thresholding and labelling accuracy with different step sizes. (b) Hough and Thresholding and labelling performance with different preprocessing configurations.

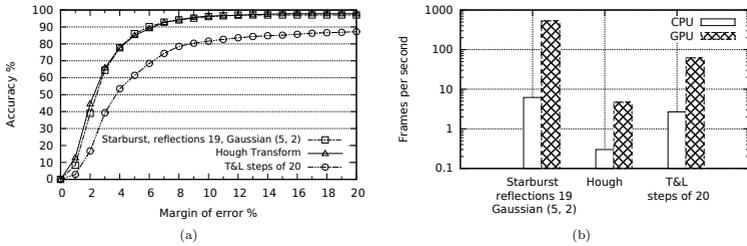


Fig. 4: (a) Accuracy of the algorithms. (b) Performance of the algorithms.

339228 (SEECAT), the Spanish SEIDI under grant FIS2013-41237-R, and the Spanish MINECO under grant TIN2015-66972-C5-3-R.

REFERENCIAS

- [1] C. Hennessey, B. Nouruddin, and P. Lawrence, "Fixation precision in high-speed noncontact eye-gaze tracking," *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 2, pp. 289–298, 2008.
- [2] OpenMP Architecture Review Board, "OpenMP application program interface version 3.1," 2011.
- [3] *CUDA C Programming guide*, Nvidia Corporation, 2015.
- [4] D. Li, D. Winfield, and D. J. Parkhurst, "Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) - Workshops*, 2005.
- [5] M. Soltany, S. T. Zadeh, and H.-R. Pourreza, "Fast and accurate pupil positioning algorithm using circular hough transform and gray projection," in *Proc. of the Int'l Conf. on Computer Communication and Management (CCMT)*, vol. 5, Sydney, Australia, 2011, pp. 556–561.
- [6] L. Masek et al., "Recognition of human iris patterns for biometric identification," Master's thesis, University of Western Australia, 2003.
- [7] I. Borovikov, "Gpu-acceleration for surgical eye imaging," in *Proc. of the 4th SIAM Conference on Mathematics for Industry (MI09)*, San Francisco, USA, 2009.
- [8] J. B. Mulligan, "A GPU-accelerated software eye tracking system," in *Proc. of the ACM Symp. on Eye Tracking Research and Applications*, Santa Barbara, CA, USA, 2012, pp. 265–268.
- [9] W. Aydi, N. Masmoudi, and L. Kamoun, "New corneal reflection removal method used in iris recognition system," *World Academy of Science, Engineering and Technology*, vol. 5, no. 5, pp. 898–902, 2011.
- [10] D. M. Rankin, B. W. Scotney, P. J. Morrow, D. R. McDowell, and B. K. Pierscionek, "Dynamic iris biometry: a technique for enhanced identification," *BMC research notes*, vol. 3, no. 1, p. 182, 2010.
- [11] P. Chen, H. Zhao, C. Tao, and H. Sang, "Block-run-based connected component labelling algorithm for gpgpu using shared memory," *IET Electronics Letters*, vol. 47, no. 24, pp. 1309–1311, 2011.
- [12] M. A. Fischer and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
- [13] G. Taubin, "Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 11, pp. 1115–1138, 1991.
- [14] J. Canny, "A computational approach to edge detection," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.
- [15] D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111–122, 1981.
- [16] S. Manzanera, P. M. Prieto, A. Benito, J. Tabernero, and P. Artal, "Location of achromatizing pupil position and first purkinje reflection in a normal population," *Investigative Ophthalmology & Visual Science*, vol. 56, no. 2, pp. 962–966, 2015.
- [17] "cuRAND library, CUDA 7," *NVIDIA Corp., Santa Clara, CA*, 2010.
- [18] V. Psozhlozhnyuk, "Image convolution with CUDA," *NVIDIA Corp. White Paper*, vol. 2007, no. 3, June, 2007.
- [19] M. Harris et al., "Optimizing parallel reduction in CUDA," *NVIDIA Developer Tech.*, vol. 2, no. 4, 2007.

Stereo Matching using SGM on the GPU

D. Hernandez-Juarez, A. Chacón, A. Espinosa, D. Vázquez,
J. C. Moure, A. M. López^{1 2}

Resumen—La extracción de información de profundidad densa, robusta y en tiempo real de cámaras estéreo es una tarea con una alta complejidad computacional, pero necesaria para diversos ámbitos como robótica, advanced driver assistance systems (ADAS) y vehículos autónomos. Semi-Global Matching (SGM) es un algoritmo ampliamente conocido que suaviza los resultados de profundidad en diferentes direcciones a través de la imagen. Este trabajo presenta un sistema de tiempo real que produce resultados de disparidad fiables en las nuevas GPUs embebidas y eficientes energéticamente. Nuestra propuesta alcanza 42 frames por segundo (fps) para una imagen de 640×480 pixels, 128 niveles de disparidad, y usando 4 direcciones de SGM.

Palabras clave— Stereo, Semi-global Matching, CUDA, GPU, Computer Vision, Drive PX, Embedded Systems, Autonomous Driving

I. INTRODUCTION

DENSE, robust and real-time computation of depth information from stereo-camera systems is a requirement in many industrial applications such as advanced driver assistance systems (ADAS), robotics navigation and autonomous vehicles. An efficient stereo algorithm has been a research topic for decades [1]. It has multiple applications, for example, [2] uses stereo information to filter candidate windows for pedestrian detection and provides better accuracy and performance.

Fig. 1 illustrates how to infer the depth of a given real-world point from its projection points on the left and right images. Assuming a simple translation between the cameras (otherwise, images must be rectified using multiple extrinsic and intrinsic camera parameters), the corresponding points must be in the same row of both images, along the epipolar lines. A similarity measure correlates matching pixels and the *disparity* (d) is the similarity distance between both points.

Disparity estimation is a difficult task because of the high level of ambiguity that often appears in real situations. For those, a large variety of proposals have been extensively presented [3]. Most of the high-accuracy stereo vision pipelines [4] include the semi-global matching (SGM) consistency-constraining algorithm [5]. The combination of SGM with different kinds of local similarity metrics is insensitive to various types of noise and interferences (like lighting), efficiently deals with large untextured areas and is capable of retaining edges.

The high computational load and memory bandwidth requirements of SGM pose hard challenges for

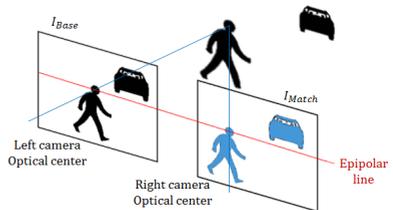


Fig. 1. Each pixel of I_{Base} corresponds to one pixel of I_{Match} , and the epipolar geometry of the two cameras limits the search to a one dimensional line. The distance z between the 3D point and the baseline of the camera is computed from the disparity d using triangulation, where f is the focal length and T is the baseline of the camera pair.

fast and low energy-consumption implementations. Dedicated hardware solutions (e.g. FPGA or ASIC) [6][7] achieve these goals, but they are very inflexible regarding changes in the algorithms. Implementations on desktop GPUs can assure real-time constraints [8], but their high power consumption and the need to attach a desktop computer makes them less suitable for embedded systems.

Recently, with the appearance of embedded GPU-accelerated systems like the NVIDIA Jetson TX1 and the DrivePX platforms (incorporating, respectively, one and two Tegra X1 ARM processors), low-cost and low-consumption real-time stereo computation is becoming attainable. The objective of this work is to implement and evaluate a complete disparity estimation pipeline on this embedded GPU-accelerated device.

We present simple, but well-designed, baseline massively parallel schemes and data layouts of each of the algorithms required for disparity estimation, and then optimize the baseline code with specific strategies, like vectorization or *CTA-to-Warp* conversion, to boost performance around 3 times. The optimized implementation runs on a single Tegra X1 at 42 frames per second (fps) for an image size of 640×480 pixels, 128 disparity levels, and using 4 path directions for the SGM method, providing high-quality real-time operation. While a high-end desktop GPU improves around 10 times the performance of the embedded GPU, the performance per watt ratio is 2.2 times worse.

The rest of the paper is organized as follows. Section II presents the algorithms composing the disparity estimation pipeline, overviews the GPU architecture and programming model and discusses related work. In section III we describe each algorithm and

¹Computer Architecture and Operating Systems Dpt. UAB, e-mail: {daniel.hernandez.juarez, alejandro.chacon, antoniomiguel.espinosa, juancarlos.moure}@uab.es

²Computer Vision Center and Computer Science Dpt. UAB, e-mail: {antonio, dvazquez}@cvc.uab.es

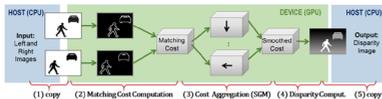


Fig. 2. Stages of the GPU-accelerated Disparity Estimation Pipeline

then propose and discuss a parallel scheme and data layout. Finally, section IV provides the obtained results and section V summarizes the work.

II. DISPARITY ESTIMATION PIPELINE

Fig. 2 shows the stages of the disparity computation pipeline: (1) the captured images are copied from the Host memory space to the GPU Device; (2) features are extracted from each image and used for similarity comparison to generate a local matching cost for each pixel and potential disparity; (3) a smoothing cost is aggregated to reduce errors (SGM); (4) disparity is computed and a 3×3 median filter is applied to remove outliers; and (5) the resulting disparity image is copied to the Host memory.

A. Local Matching Cost and Semi-Global Matching (SGM)

Different similarity metrics or cost functions have been proposed in the literature. The less computationally-demanding, and modest quality providers, are Sum of Absolute Differences, ZSAD and Rank Transform. According to [9], Hierarchical Mutual Information and the Census Transform (CT) features [10] provide similar higher quality, being CT substantially less time-consuming. Recently, costs based on neural networks have outperformed CT [4], but at the expense of a higher computational load.

A CT feature encodes the comparisons between the values of the pixels in a window around a central pixel. After empirically evaluating different variants we selected a Center-Symmetric Census Transform (CSCT) configuration with a 9×7 window, which provides a more compact representation with similar accuracy [11]. The similarity of two pixels is defined as the Hamming distance of their CSCT bit-vector features. Two properties provide robustness for outdoor environments with uncontrolled lighting and in front of calibration errors: the invariance to local intensity changes (neighboring pixels are compared to each other) and the tolerance to outliers (an incorrect value modifies a single bit).

In order to deal with non-unique or wrong correspondences due to low texture and ambiguity, consistency constraints can be included in the form of a global two-dimensional energy minimization problem. Semi-global matching (SGM) approximates the global solution by solving a one-dimensional minimization problem along several (typically 4 or 8) independent paths across the image. For each path di-

rection, image point and disparity, SGM aggregates a cost that considers the cost of neighboring points and disparities. The number of paths affects both the quality and the performance of the results.

B. Overview of GPU architecture and performance

GPUs are massively parallel devices containing tens of throughput-oriented processing units called *streaming multiprocessors* (SMs). Memory and compute operations are executed as vector instructions and are highly pipelined in order to save energy and transistor budget. SMs can execute several vector instructions per cycle, selected from multiple independent execution flows: the higher the available parallelism the better the pipeline utilization.

The CUDA programming model allows defining a massive number of potentially concurrent execution instances (called *threads*) of the same program code. A unique two-level identifier $\langle ThrId, CT Aid \rangle$ is used to specialize each thread for a particular data and/or function. A CTA (*Cooperative Thread Array*) comprises all the threads with the same $CT Aid$, which run simultaneously and until completion in the same SM, and can share a fast but limited memory space. *Warps* are groups of threads with consecutive $ThrIds$ in the same CTA that are mapped by the compiler to vector instructions and, therefore, advance their execution in a lockstep synchronous way. The warps belonging to the same CTA can synchronize using an explicit barrier instruction. Each thread has its own private local memory space (commonly assigned to registers by the compiler), while a large space of global memory is public to all execution instances (mapped into a large-capacity but long-latency device memory, which is accelerated using a two-level hierarchy of cache memories).

The parallelization scheme of an algorithm and the data layout determine the available parallelism at the instruction and thread level (required for achieving full resource usage) and the memory access pattern. GPUs achieve efficient memory performance when the set of addresses generated by a warp refer to consecutive positions that can be *coalesced* into a single, wider memory transaction. Since the bandwidth of the device memory can be a performance bottleneck, an efficient CUDA code should promote data reuse on shared memory and registers.

C. Related work

A reference implementation of SGM on CPU [12] reached 5.43 frames per second (fps) with 640×480 image resolution and 128 disparity levels. They applied SGM with 8 path directions and an additional left-right consistency check and sub-pixel interpolation. A modified version with reduced disparity computation (rSGM) was able to reach 12 fps.

Early GPU implementations [13] and [14] present OpenGL/Cg SGM implementations with very similar performance results peaking at 8 fps on 320×240 resolution images. Versions designed for early CUDA systems and proposed specific modifications of the

SGM algorithm. Haller and Nedeveschi [15] modified the original cost aggregation formula removing the P1 penalty and using 4 path directions for cost aggregation. In this way, they reduced computation and memory usage, but also reduced accuracy. Their implementation reached 53 fps on a Nvidia GTX 280 with images of 512×383.

The most recent implementation [8] stated very fast results: 27 fps on 1024×768 images using a NVIDIA Tesla C2050, with 128 disparity levels. By using Rank Transform [10] as matching cost function, their proposal provides lower accuracy [9]. We will notice some differences in the parallel scheme on the following discussion.

As far as we know this is the first evaluation of disparity estimation in a Nvidia GPU-accelerated embedded system, as well as in the last Maxwell architecture. We propose better parallelization schemes to take advantage of the hardware features available in current systems.

III. ALGORITHM DESCRIPTION AND MASSIVE PARALLELIZATION

This section describes the algorithms used for disparity computation and discusses the alternative parallelization schemes and data layouts. We present the baseline pseudocode for the proposed massively parallel algorithms and explain additional optimizations.

A. Matching Cost Computation

A 9×7-window, Center-Symmetric Census Transform (CSCT) concatenates the comparisons of 31 pairs of pixels into a bit-vector feature. Equation 1 defines the CSCT, where \otimes is bit-wise concatenation, $I(x, y)$ is the value of pixel (x, y) in the input image, and $s(u, v)$ is 1 if $u \geq v$, or 0 otherwise. The matching cost $MC(x, y, d)$ between a pixel (x, y) in the base image and each potentially corresponding pixel in the match image at disparity d is defined by equation 2, where \oplus is bit-wise exclusive-or and $bitcount$ counts the number of bits set to 1.

$$CSCT_{9,7}(I, x, y) = \otimes \left(\begin{array}{l} \bigotimes_{i=1}^4 \bigotimes_{j=-3}^3 s(I(x+i, y+j), I(x-i, y-j)) \\ \bigotimes_{j=1}^3 s(I(x, y+j), I(x, y-j)) \end{array} \right) \quad (1)$$

$$MC(x, y, d) = bitcount(CSCT_{9,7}(I_{base}, x, y) \oplus CSCT_{9,7}(I_{match}, x-d, y)) \quad (2)$$

The data access patterns inherent in both equations exhibit different data reuse schemes, which prevent both algorithms to be fused. The 2D-tiled parallel scheme shown in Fig. 3 and Table I matches the 2D-stencil computation pattern of CSCT, and maximizes data reuse: the attached table shows how a tiled scheme using shared memory reduces the total global data accesses by $(62 + 4)/(1.5 + 4) = 12$ times with respect to a straightforward, naïve, em-

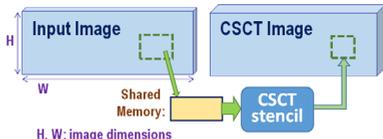


Fig. 3. CSCT: 2D-tiled CTA-parallel scheme and computational analysis

TABLE I
 CSCT: KERNEL DETAILS. EVERYTHING IS PER THREAD EXCEPT THREAD PARALLELISM (PER IMAGE).

Parallelization Scheme	Naive	2D-tiled
Thread Parallelism	$W \times H$	$W \times H$
Compute Work	$\odot(1)$	$\odot(1)$
Global Data Read	62 B	≈ 1.5 B
Global Data Written	4 B	4 B
Shared Data Read	0	62 B
Shared Data Written	0	≈ 1.5 B

barrasingly parallel design, where each thread reads its input values directly from global memory.

The 1D-tiled parallel scheme for computing matching cost (MC) exploits data reuse on the x-dimension (see Fig. 4 and Table II). As proposed in [8], we can represent matching cost using a single byte without losing accuracy, which reduces 4 times the memory bandwidth requirements in comparison to using 32-bit integers. The attached table shows that the read-cooperative scheme, compared to the naïve design, sacrifices parallelism (divides the number of threads by D , the maximum disparity considered) by higher data reuse (around 8 times less global memory accesses). The low arithmetic intensity of the algorithm (2 main compute operations every 9-Byte memory accesses) advises for this kind of optimization.

Algorithms 1 and 2 show the pseudocode of both parallel algorithms, not including special code for corner cases handling image and CTA boundaries. In both cases, threads in the same CTA cooperate to read an input data tile into shared memory, then synchronize, and finally perform the assigned task reading the input data from shared memory. The first algorithm assumes a CTA size of

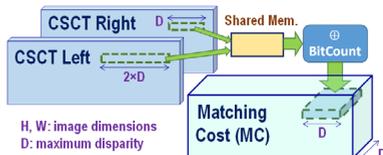


Fig. 4. Matching cost: 1D-tiled CTA-parallel scheme and computational analysis

TABLE II
MATCHING COST: KERNEL DETAILS

Parallelization Scheme	Naive	1D-tiled
Thread Parallelism	$W \times H \times D$	$W \times H$
Compute Work	$\odot(1)$	$\odot(D)$
Global Data Read	8 B	12 B
Global Data Written	1 B	D B
Shared Data Read	0	$8 \times D$ B
Shared Data Written	0	12 B

$WarpSize \times WarpSize$ threads and the second algorithm a CTA of D threads. They are both scalable designs that use a small constant amount of shared memory per thread (1.5 and 12 Bytes, respectively).

There are two memory-efficient layout alternatives for algorithm 2. Each CTA generates a $D \times D$ slice in the y -plane of the MC matrix, and threads can generate together the cost for (1) all the disparity levels for the same pixel or (2) all the pixels in the block for the same disparity level. We chose the first option, and adapt the data layout so that the indexes of disparity levels vary faster on the MC cube and global write instructions are coalesced. The second solution, used in [8], provides similar performance on this algorithm but compromises the available parallelism and the performance of the following SGM algorithm.

Algorithm 1: CSCT: 2D-tiled, read-cooperative parallel scheme

```

input : I[H][W], H, W
output: CSCT[H][W]
1 parallel for  $y=0$  to  $H$  step  $WarpSize$  do
2   parallel for  $x=0$  to  $W$  step  $WarpSize$  do
3     CTA parallel for  $yCTA, xCTA=(0,0)$ 
4       to  $(WarpSize, WarpSize)$  do
5       copy
6        $(WarpSize + 8) \times (WarpSize + 6)$  tile
7       of I[][] into SharedI[][];
8       CTA Barrier Synchronization;
9        $CSCT[y+yCTA][x+xCTA] =$ 
10       $CSCT_{9,7}(SharedI, xCTA, yCTA);$ 

```

B. Smoothing Cost Aggregation (SGM) and Disparity Computation

The SGM method solves a one-dimensional minimization problem along different paths $r=(r_x, r_y)$ using the recurrence defined by equation 3 and a dynamic programming algorithmic pattern. Matrix L_r contains the smoothing aggregated costs for path r . The first term of equation 3 is the original matching cost, and the second term adds the minimum cost of the disparities corresponding to the previous pixel $(x-r_x, y-r_y)$, including penalties for small disparity changes (P_1) and for larger disparity discontinuities and (P_2). P_1 is intended to detect slanted and curved

Algorithm 2: Matching Cost computation: 1D-tiled, read-cooperative parallel scheme; Data layout: $MC[y][x][d]$ (d indexes vary faster)

```

input : CSCTbase[H][W], CSCTmatch[H][W],
        H, W, D
output: MC[H][W][D]
1 parallel for  $y=0$  to  $H$  do
2   parallel for  $x=0$  to  $W$  step  $D$  do
3     CTA parallel for  $ThrId=0$  to  $D$  do
4       SharedM[ThrId] =
5       CSCTmatch[y][x+ThrId-D];
6       SharedM[D+ThrId] =
7       CSCTmatch[y][x+ThrId];
8       SharedB[ThrId] =
9       CSCTbase[y][x+ThrId];
10      CTA Barrier Synchronization;
11      for  $i=0$  to  $D$  do
12         $MC[y][x+i][ThrId] =$  BitCount (
13          SharedB[i]  $\oplus$ 
14          SharedM[ThrId+1+i] );

```

surfaces, while P_2 smooths the results and makes abrupt changes difficult. The last term ensures that aggregated costs are bounded. For a detailed discussion refer to [5]. The different L_r matrices must be added together to generate a final cost and then select the disparity corresponding to the minimum (*winner-takes-all* strategy), as shown by equation 4.

$$L_r(x, y, d) = MC(x, y, d) + \min \begin{cases} L_r(x-r_x, y-r_y, d) \\ L_r(x-r_x, y-r_y, d-1) + P_1 \\ L_r(x-r_x, y-r_y, d+1) + P_1 \\ \min_i L_r(x-r_x, y-r_y, i) + P_2 \end{cases} - \min_k L_r(x-r_x, y-r_y, k) \quad (3)$$

$$D(x, y) = \min_d \sum_r L_r(x, y, d) \quad (4)$$

Equation 3 determines a recurrent dependence that prevents the parallel processing of pixels in the same path direction. Parallelism can be exploited, though, in the direction perpendicular to the path, in the disparity dimension, and for each of the computed path directions. Our proposal exploits all the available parallelism by creating a CTA for each slice in the aggregated cost matrix along each particular path direction.

Fig. 5 and Table III illustrates the case of the top-to-bottom path direction and algorithm 3 shows the pseudocode. Each of the W slices is computed by a different CTA of D threads, with each thread executing a recurrent loop (line 4) to generate H cost values along the path. Computing the cost for the current pixel and disparity level requires the cost of the previous pixel on neighboring disparity levels: one value can be reused in a private thread register but the neighboring costs must be communicated among threads (lines 7,8 and 12). Finally, all threads in the CTA must collaborate to compute the minimum cost for all disparity levels (line 11).

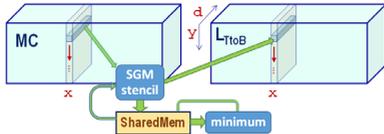


Fig. 5. Aggregated cost, Top-to-Bottom: CTA-parallel scheme with recurrence in the y-dimension and computational analysis

TABLE III
AGGREGATED COST: KERNEL DETAILS

Thread Parallelism	$W \times D$
Compute Work	$\odot (H)$
Global Data Read	H B
Global Data Written	H B
Shared Data Read	$2H + H B$
Shared Data Written	$2H + H B$

The case for horizontal paths is very similar, with H slices computed in parallel. Diagonal path directions are a little more complex: W independent CTAs process the diagonal slices moving in a vertical direction (assuming $W \geq H$). When a CTA reaches a boundary, it continues on the other boundary. For example, a top-to-bottom and right-to-left diagonal slice starting at $(x,y) = (100,0)$ will successively process pixels $(99,1)$, $(98,2) \dots (0, 100)$, and then will reset the costs corresponding to the previous pixel and continue with pixels $(W-1,101)$, $(W-2,102) \dots$

Algorithm 3: Aggregated Cost computation: top-to-bottom path direction

```

input : MC[H][W][D], H, W, D
output: L[H][W][D]
1 parallel for  $x=0$  to  $W$  do
2   CTA parallel for  $ThrId=0$  to  $D$  do
3     Initialize aggr, min and SharedAggr[]
       with MAX_VALUE;
4     for  $y=0$  to  $H$  do
5        $cost = MC[y][x][ThrId]$ ;
6       CTA Barrier Synchronization;
7        $left = SharedAggr[ThrId]$ ;
8        $right = SharedAggr[ThrId+2]$ ;
9        $aggr = cost + \text{minimum}(aggr,$ 
        $left+P1, right+P1, min+P2) -$ 
        $min$ ;
10       $L[y][x][ThrId] = aggr$ ;
11       $min = \text{CTA.Minimum.Reduce}(aggr)$ ;
       *** includes Barrier Sync.
        $SharedAggr[ThrId+1] = aggr$ ;

```

The cost aggregation and disparity computation defined by equation 4 have been fused in Algorithm 4 in order to reduce the amount of memory accesses (avoids writing and then reading the final cost matrix). A CTA-based parallel scheme is proposed so

that each CTA produces the disparity of a single pixel (line 7): first, each CTA thread adds the costs corresponding to a given disparity level for all path directions (line 4), and then CTA threads cooperate to find the disparity level with minimum cost (line 5).

Algorithm 4: Summation of all path costs and Disparity Computation

```

input :  $L_0[W][H][D]$ ,  $L_1[W][H][D]$ ,
        $L_2[W][H][D]$  ...  $W, H, D$ 
output:  $Disp[W][H]$ 
1 parallel for  $x=0$  to  $W$  do
2   parallel for  $y=0$  to  $H$  do
3     CTA parallel for  $ThrId=0$  to  $D$  do
4        $cost =$ 
        $L_0[x][y][ThrId] + L_1[x][y][ThrId] + \dots$ ;
5        $MinIndex =$ 
        $\text{CTA.Minimum.Reduce}(cost, ThrId)$ ;
6       if  $ThrId == 0$  then
7          $Disp[x][y] = MinIndex$ ;

```

C. Additional Optimizations

We have applied three types of optimizations to the baseline algorithms that provided a combined performance improvement of almost $3\times$. We have vectorized the inner loop of algorithm 3 (lines 4-12) to process a vector of 4 cost values (4 bytes) per instruction (requiring a special byte-wise SIMD instructions for computing the minimum operation). We have also modified the parallel scheme so that a single warp performs the task previously assigned to a CTA, which we call *CTA-to-warp* conversion. It (1) avoids expensive synchronization operations, (2) allows using fast register-to-register communication (using special shuffle instructions) instead of shared-memory communications, and (3) reduces instruction count and increases instruction-level parallelism. A drawback of both strategies is a reduction of thread-level parallelism, as shown in [16]. This is not a severe problem in the embedded Tegra X1 device, with a maximum occupancy of ≈ 4 thousand threads.

Finally, to reduce the amount of data accessed from memory, the computation of the aggregated cost for the last path direction (Alg. 3 Bottom-to-Top) is fused with the final cost summation and disparity computation (Alg. 4), providing a 1.35x performance speedup on the Tegra X1. Also, fusing the computation of the initial matching cost (Alg. 2) with the aggregate cost computation for the horizontal path directions (Alg. 3) improves performance by 1.13x.

IV. RESULTS

We have measured execution time and disparity estimation accuracy for multiple images, 128 disparity levels, and 2, 4 and 8 path directions. Apart from executing on a NVIDIA Tegra X1, which inte-

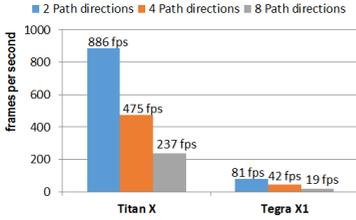


Fig. 6. Performance (fps) for 640×480px images, 128 disparity levels, and 2, 4 and 8 SGM path directions

grates 8 ARM cores and 2 Maxwell SMs with a TDP of 10W, and for comparison purposes, we have also executed on a high-end NVIDIA Titan X, with 24 Maxwell SMs and a TDP of 250W. We ignore the time for CPU-GPU data transfers (less than 0.5% of the total elapsed time) since it can be overlapped with computation. Since performance scales proportional to the number of image pixels, we will restrict our explanation to 640×480 images.

Fig. 8 indicates the disparity estimation accuracy, measured using the KITTI benchmark-suite [17], when using different SGM configurations, and not considering occluded pixels and treating more than 3 pixel differences as errors. Using 4 path directions (excluding diagonals) reduces accuracy very slightly, while using only the left-to-right and top-to-bottom directions reduces accuracy more noticeably.

Fig. 6 show, the performance throughput (frames per second, or fps) and Fig. 7 shows the performance per watt (fps/W) on both GPU systems and also for different SGM configurations. The high-end GPU always provides more than 10 times the performance of the embedded GPU (as expected by the difference in number of SMs), but the latter offers around 2 times more performance per Watt. It is remarkable that real-time rates (42 fps) with high accuracy are achieved by the Tegra X1 when using 4 path directions.

Finally, an example of the disparity computed by our proposed algorithm can be seen in Fig. 10 and the input left image in Fig. 9.

V. CONCLUSIONS

The results obtained show that our implementation of depth computation for stereo-camera systems is able to reach real-time performance on a Tegra X1. This fact indicates that low-consumption embedded GPU systems, like the Tegra X1, are well capable of attaining real-time processing demands. Hence, their low-power envelope and remarkable performance make them good target platforms for real-time video processing, paving the way for more complex algorithms and applications.

We have proposed baseline parallel schemes and data layouts for the disparity estimation algorithms

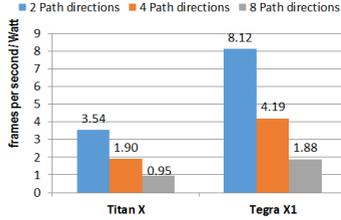


Fig. 7. Performance per Watt results for 640×480px images, 128 disparity levels, and 2, 4 and 8 SGM path directions

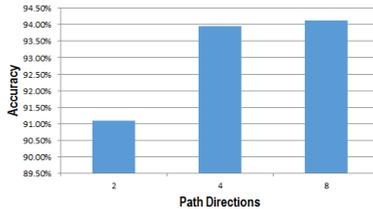


Fig. 8. Accuracy results for 640×480px images, 128 disparity levels, and 2, 4 and 8 SGM path directions

that follow general optimization rules based on a simple GPU performance model. They are designed to gracefully scale on the forthcoming GPU architectures, like NVIDIA Pascal. Then, we have optimized the baseline code and improved performance around 3 times with different specific strategies, like vectorization or *CTA-to-Warp* conversion, that are also expected to be valid for forthcoming architectures.

We plan to prove the higher performance potential of the new embedded NVIDIA Pascal GPUs to enable real-time implementations with larger images and a higher number of disparity levels, and more complex algorithms that provide better estimation results. In this sense, we are going to include post-filtering steps such as Left-Right Consistency Check, subpixel calculation, and adaptive P2, which are well-known methods of increasing accuracy.

ACKNOWLEDGEMENTS

This research has been supported by the MICINN under contract number TIN2014-53234-C2-1-R. By



Fig. 9. Image obtained from the left camera of the car



Fig. 10. Disparity computed with SGM described here

the MEC under contract number TRA2014-57088-C2-1-R, the spanish DGT project SPIP2014-01352, and the Generalitat de Catalunya projects 2014-SGR-1506 and 2014-SGR-1562. We thank Nvidia for the donation of the systems used in this work.

REFERENCIAS

- [1] Hamid R Arabnia, "A distributed stereocorrelation algorithm," in *Computer Communications and Networks, 1995. Proceedings., Fourth International Conference on*. IEEE, 1995, pp. 479–482.
- [2] Alejandro González et al., "3D-Guided Multiscale Sliding Window for Pedestrian Detection," in *Iberian Conf. on Pattern Recognition and Image Analysis*, 2015, vol. 9117, pp. 560–568.
- [3] Daniel Scharstein and Richard Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.
- [4] Jure Zbontar and Yann LeCun, "Computing the Stereo Matching Cost with a Convolutional Neural Network," *arXiv preprint arXiv:1409.4326*, 2014.
- [5] Heiko Hirschmüller, "Stereo Processing by Semiglobal Matching and Mutual Information," *Pattern Analysis and Machine Intelligence, IEEE Trans.*, vol. 30, no. 2, pp. 328–341, 2008.
- [6] Christian Banz et al., "Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation," *Embedded Computer Systems*, pp. 93–101, 2010.
- [7] Payá-Vayá et al., "VLIW architecture optimization for an efficient computation of stereoscopic video applications," in *Green Circuits and Systems (ICGCS)*. IEEE, 2010, pp. 457–462.
- [8] Christian Banz, Holger Blume, and Peter Pirsch, "Real-Time Semi-Global Matching Disparity Estimation on the GPU," in *Computer Vision Workshops (ICCV Workshops)*. IEEE, 2011, pp. 514–521.
- [9] Heiko Hirschmüller et al., "Evaluation of Stereo Matching Costs on Images with Radiometric Differences," *Pattern Analysis and Machine Intelligence, IEEE Trans.*, vol. 31, no. 9, pp. 1582–1599, 2009.
- [10] Ramin Zabih and John Woodfill, "Non-parametric local transforms for computing visual correspondence," in *Computer Vision-ECCV'94*, pp. 151–158. Springer, 1994.
- [11] Robert Spangenberg et al., "Weighted Semi-Global Matching and Center-Symmetric Census Transform for Robust Driver Assistance," *Computer Analysis of Images and Patterns*, pp. 34–41, 2013.
- [12] Robert Spangenberg, Tobias Langner, Sven Adfeldt, and Renan Rojas, "Large scale Semi-Global Matching on the CPU," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE, 2014, pp. 195–201.
- [13] Ines Ernst and Heiko Hirschmüller, "Mutual Information Based Semi-Global Stereo Matching on the GPU," in *Advances in Visual Computing*, pp. 228–239. Springer, 2008.
- [14] Ilya D Rosenberg et al., "Real-time stereo vision using semi-global matching on programmable graphics hardware," in *ACM SIGGRAPH 2006 Sketches*. ACM, 2006, p. 89.
- [15] Istvan Haller and Sergiu Nedevschi, "GPU optimization of the SGM stereo algorithm," in *Intelligent Computer Communication and Processing (ICCP)*. IEEE, 2010, pp. 197–202.
- [16] Alejandro Chacón, Santiago Marco-Sola, Antonio Espinosa, Paolo Ribeca, and Juan Carlos Moure, "Thread-cooperative, bit-parallel computation of levenshtein distance on GPU," in *Proceedings of the 28th ACM international conference on Supercomputing*. ACM, 2014, pp. 103–112.
- [17] Andreas Geiger, Philip Lenz, and Raquel Urtasun, "Are we ready for Autonomous Driving? the KITTI Vision Benchmark Suite," in *Conference on Computer Vision and Pattern Recognition*, 2012.
- [18] Joel Gibson and Oge Marques, "Stereo depth with a Unified Architecture GPU," in *Computer Vision and Pattern Recognition Workshops*. IEEE, 2008, pp. 1–6.

Acceso eficiente a memoria para selección de características multi-objetivo en GPUs

Juan José Escobar¹, Julio Ortega¹, Jesús González¹, Miguel Damas¹, Consolación Gil²

Resumen—Muchas aplicaciones de enorme interés implican problemas de optimización en espacios de elevada dimensionalidad con bases de datos de gran tamaño. En este trabajo se describe el uso de GPUs para acelerar la selección de características con optimización multi-objetivo en la clasificación de electroencefalogramas (EEG) para interfaces cerebro-computador (BCI). A partir del análisis de los resultados experimentales de la ejecución de diversos códigos implementados en *OpenCL*, se muestra la importancia del aprovechamiento eficiente de la jerarquía de memoria de las GPUs para alcanzar, en este tipo de problemas, resultados competitivos con el paralelismo en núcleos superescalares.

Palabras clave— Algoritmos evolutivos paralelos, BCI, clasificación de EEG, clustering multi-objetivo, GPU, *OpenCL*, selección de características.

I. INTRODUCCIÓN

LA tecnología de almacenamiento y las plataformas distribuidas, incluyendo redes, sensores y otros dispositivos de captura de datos, han hecho posible disponer de bases de datos de gran tamaño, a partir de las que están surgiendo aplicaciones de enorme interés socio-económico. Entre ellas están las relacionadas con el análisis de datos de dimensión elevada en bioinformática e ingeniería biomédica, que precisan el uso eficiente, en cuanto a prestaciones y consumo energético, de arquitecturas de computador paralelas y distribuidas heterogéneas incluyendo aceleradores, tales como GPUs, y recursos de almacenamiento gestionados por sistemas de ficheros distribuidos. En este trabajo nos centraremos en la selección de características para la clasificación de electroencefalogramas (EEG) para BCI [1] como ejemplo de este tipo de aplicaciones. Así, analizaremos el aprovechamiento del paralelismo en GPUs poniendo de manifiesto la importancia que la gestión de la memoria tiene en este tipo de arquitecturas. Esta circunstancia, que se ha puesto de manifiesto en numerosos trabajos [2,3], se analiza aquí en el contexto de la optimización multi-objetivo para clasificación de EEGs. Para ello, se proponen códigos *OpenCL* [4] paralelos para tareas de optimización evolutiva y clasificación, capaces de aprovechar eficientemente arquitecturas

paralelas heterogéneas incluyendo núcleos superescalares y GPUs. Estas arquitecturas constituyen la tendencia dominante en el desarrollo de arquitecturas que implementen las mejoras tecnológicas [5] y su uso se ha propuesto en bastantes trabajos previos relacionados con los algoritmos de optimización evolutiva [6-8]. No obstante, la paralelización de aplicaciones completas, con las características de la que aquí se considera, es menos frecuente en la literatura.

El artículo se ha estructurado en cinco secciones. Tras esta primera sección dedicada a presentar los ámbitos del trabajo, la Sección II introduce la aproximación multi-objetivo a la selección de características en problemas de clasificación de EEGs. La Sección III describe las características de los códigos propuestos y los detalles de su implementación en *OpenCL*. Los resultados de su evaluación experimental se proporcionan en la Sección IV y, finalmente, las conclusiones del trabajo se dan en la Sección V.

II. SELECCIÓN MULTI-OBJETIVO DE CARACTERÍSTICAS

En este trabajo estudiamos las posibilidades que ofrecen las arquitecturas paralelas heterogéneas en la clasificación de electroencefalogramas (EEGs) para tareas de interfaz cerebro-computador (BCI, *Brain-Computer Interfaces*). Para caracterizar un EEG suele utilizarse un número muy elevado de características (componentes) debido a la baja relación señal/ruido, la necesidad de representar la evolución temporal de las señales de los electrodos y el carácter no estacionario de las señales. Por otra parte, dado que el número de EEGs disponibles para ajustar el clasificador es usualmente menor que el de características debido al coste experimental asociado a su obtención, tenemos un problema de clasificación con pocos patrones de dimensión elevada y nos enfrentamos a la denominada “maldición de la dimensionalidad” (*curse of dimensionality*) [9]. La selección de un conjunto de características que no sean ruidosas, redundantes, o irrelevantes permitiría tanto evitar dicho problema, como disminuir el coste asociado al entrenamiento del clasificador.

¹Departamento de Arquitectura y Tecnología de Computadores, CITIC, Universidad de Granada.
{jjesobar,jortega,jesusgonzalez,mdamas}@ugr.es

²Departamento de Informática, Universidad de Almería.
cgilm@ual.es

La Figura 1 muestra un esquema del procedimiento evolutivo de selección multi-objetivo de características que se ha paralelizado. En un problema de optimización multiobjetivo se busca el vector de variables de decisión $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x} = [x_1, x_2, \dots, x_n]$, que satisface un conjunto de restricciones, $g(\mathbf{x}) \leq 0$, $h(\mathbf{x}) = 0$, y optimiza el vector $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x}))$ cuyos componentes representan las funciones objetivo a optimizar. Dado que los objetivos están usualmente en conflicto, en lugar de obtener una única solución, la optimización multi-objetivo busca un conjunto de soluciones conocidas con el nombre de soluciones óptimas de Pareto, entre las que el usuario puede escoger la más adecuada en cada caso. Las soluciones óptimas de Pareto definen el denominado frente de Pareto, en el que no existe ninguna solución peor que otra cuando se tienen en cuenta todos los objetivos: las denominadas soluciones *no-dominadas*.

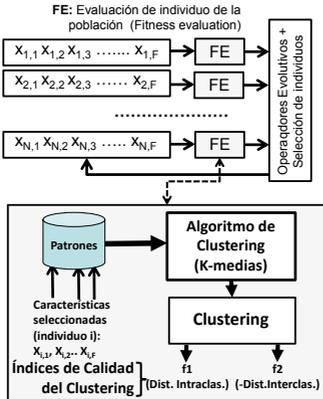


Fig. 1. Procedimiento de tipo *wrapper* para selección multiobjetivo

El procedimiento de selección de la Figura 1 es de tipo *wrapper* [10], dado que los objetivos a optimizar se definen considerando las prestaciones de la agrupación (*clustering*) que se obtiene con la selección de características codificada por cada individuo de la población del procedimiento evolutivo. La agrupación se obtiene aplicando el algoritmo K-medias al conjunto de patrones de entrenamiento y se evalúa utilizando como índices de calidad [11] de los clusters obtenidos las funciones f_1 y f_2 , relacionadas, respectivamente,

con las distancias intraclase e interclases de la agrupación:

$$f_1 = \sum_{j=1}^W \frac{1}{|C(j)|} \left(\sum_{P_i \in C(j)} \|P_i - K(j)\| \right) \quad (1)$$

$$f_2 = - \sum_{j=1}^{W-1} \left(\sum_{i>j} \|K(i) - K(j)\| \right) \quad (2)$$

En las ecuaciones (1) y (2), $|C(j)|$ es el número de patrones en la clase, o *cluster*, $C(j)$ ($j=1, \dots, W$) con centroide $K(j)$, y $\|P_i - K(j)\|$ es la distancia euclídea entre el patrón P_i y el centroide $K(j)$. Como se ha implementado un problema de minimización, y la agrupación es mejor cuanto mayor es la distancia entre clases, la función de coste f_2 es igual a menos la distancia interclases. El algoritmo K-medias consta de los siguientes pasos:

1. Generar W centroides iniciales (tantos como *clusters* o clases).
2. Asignar cada patrón al *cluster* correspondiente al centroide más cercano.
3. Calcular los nuevos centroides.
4. Terminar si la condición de final se cumple (no se han producido cambios en los centroides o se ha llegado a un número máximo de iteraciones). Si no se cumple ir a 2.

La utilidad de la optimización multi-objetivo en problemas de análisis de datos, entre los que están la selección de características y la clasificación, se ha descrito en [12]. Las ventajas de una formulación multi-objetivo del problema de selección de características varían según la clasificación sea supervisada o no supervisada [10]. En los problemas de clasificación no supervisada, donde las medidas de calidad del *clustering* de patrones obtenido suelen estar sesgadas hacia la maximización o hacia la minimización de características, una formulación multi-objetivo puede contrarrestar este efecto. Entre los trabajos que se han propuesto en esta línea destaca la revisión realizada en [10] y las propuestas presentadas en [13, 14]. Con respecto a las implementaciones de procedimientos evolutivos de optimización multi-objetivo, cabe citar los trabajos [15,16]. En muchos artículos se ha abordado la paralelización del algoritmo K-medias en GPUs [17-19]. De hecho, el uso de K-medias en grandes bases de datos se aborda en [18], y en [19] se utiliza con patrones de elevada dimensionalidad. Aquí, consideramos la paralelización del algoritmo K-medias para patrones de dimensionalidad elevada, junto con el procedimiento de optimización multi-objetivo en el que se usa.

III. CÓDIGOS PARALELOS

En esta sección describiremos los códigos paralelos que se han implementado en *OpenCL* [20] para aprovechar las arquitecturas heterogéneas

incluyendo tanto núcleos superescalares como GPUs. La GPU actúa como un coprocesador conectado a través de un bus a un host con varios núcleos superescalares que comparten memoria. Los núcleos de la GPU (denominados *Stream Processors*, SP, en las GPUs de NVIDIA) disponen de unidades escalares de ejecución y se encuentran distribuidas entre los distintos multiprocesadores (*Streaming Multiprocessors*, SMX, en las GPUs de NVIDIA) que incluye la GPU. Los SP de un mismo SMX comparten una o varias unidades de instrucción y el banco de registros. Al disponer de varios SMXs, la GPU puede ejecutar programas paralelos según un modelo SPMD. Las hebras se agrupan en bloques, de forma que todas las hebras de un mismo bloque se ejecutan en un mismo SMX. Además, los bloques están constituidos por varios *warps*, cada uno de los cuales contiene las hebras que, con números de identificación consecutivos, empiezan juntas en la misma dirección de programa. Mientras que las hebras de un mismo bloque pueden cooperar y comparten la misma unidad de instrucciones, banco de registros y ciertas unidades de memoria de baja latencia, las hebras de bloques diferentes solo pueden comunicarse a través de la denominada memoria global.

En *OpenCL*, uno o varios programas se ejecutan en un host y llaman a funciones, denominadas *kernels*, que se ejecutan en los dispositivos (*OpenCL devices*), que pueden ser tanto GPUs como núcleos superescalares (CPUs), FPGAs u otros tipos de aceleradores. Un dispositivo *OpenCL* es un array de unidades independientes de computación (*computing units*) que contienen elementos de procesamiento (*processing elements*). Por ejemplo, en una GPU de NVIDIA, los SMXs son las unidades de computación (*computing units*) y los SPs, los elementos de proceso (*processing elements*). Las unidades de ejecución concurrente se denominan *work-items* y se asignan a los *processing elements*. El modelo de memoria de *OpenCL* define espacios de memoria que se ajustan bastante a las jerarquías de memoria usuales. Así, la memoria global (*global memory*) es visible a todas las unidades de computación del dispositivo, lo mismo que la memoria constante (*constant memory*), que está incluida en la memoria global y contiene las variables que no cambian en el programa. Todos los elementos de proceso de una unidad de computación comparten la memoria local (*local memory*), mientras que solo un elemento de procesamiento (o un *work-item*) tiene acceso a la memoria privada (*private memory*).

A partir de lo explicado en la Sección II, y de la Figura 1, se pone de manifiesto que la aplicación implica ejecutar un algoritmo evolutivo multi-objetivo, y ejecutar un algoritmo de *clustering* y determinar los índices de calidad de los clusters obtenidos para cada uno de los individuos de la

población del algoritmo evolutivo. En [21] hemos propuesto distintas alternativas para ejecutar en paralelo el algoritmo evolutivo, pero en ninguno de esos casos se ha paralelizado la evaluación de la solución que codifica el individuo: ni el algoritmo de *clustering*, ni la evaluación de la calidad de los *clusters*. En [22] proponemos una primera versión en *OpenCL* que permite aprovechar, en una GPU, el paralelismo de datos implícito en el algoritmo K-medias. Tal y como se describirá más adelante, en este artículo proponemos una mejora del algoritmo de [22] basada en la optimización de los accesos a memoria que generan los algoritmos.

El algoritmo evolutivo de optimización multi-objetivo implementado se basa en el algoritmo NSGA-II [23]. Un primer nivel de paralelización implementado consiste en distribuir los individuos entre los distintos SMX de la GPU, en el caso del kernel implementado para la GPU (*Kernel de GPU*), o entre los núcleos superescalares de los nodos del host (*Kernel de CPU*). De esta manera, el trabajo de evaluación de los individuos de la población se divide entre los procesadores utilizados según el tipo de *Kernel*. Se trata de una implementación de tipo maestro-trabajador del algoritmo evolutivo multi-objetivo en la que los operadores evolutivos se ejecutan en el host, sin aprovechar paralelismo dado que su coste es muy pequeño respecto a la evaluación de todos los individuos de la población. No obstante, el *Kernel de GPU* implementa un segundo nivel de paralelismo al ejecutar en paralelo el algoritmo k-medias para cada individuo de la población, aprovechando el paralelismo de datos que ofrece la GPU.

El aprovechamiento de la GPU, no obstante, no solo requiere transferir, al comienzo del programa, los patrones de entrenamiento entre la memoria del host y la de la GPU, sino que además, en cada iteración, también hay que enviar los nuevos individuos de la población y recoger los resultados de su evaluación. Hay que tener en cuenta que el bus a través del que se accede a la GPU tiene un ancho de banda y una latencia peores que la de los buses de memoria del host y de la CPU y que en la aplicación que consideramos el tamaño de la base de datos con patrones de dimensión elevada, es considerable. Podemos estar ante un importante cuello de botella, y es esencial gestionar eficientemente la jerarquía de memoria de la GPU, como se muestra a continuación.

La Figura 2 muestra el pseudocódigo del *Kernel de GPU* que implementa el algoritmo K-medias y calcula las funciones de costo para cada los individuos de la población. Un individuo es evaluado por un *work-group* como se muestra en la línea 02 de la Figura 2. El primer nivel de paralelismo al que nos hemos referido se consigue a través de varios *work-groups* que se ejecutan en

paralelo en los SMX. El segundo nivel se obtiene a través de los warps que componen cada *work-group* (en las GPUs de NVIDIA, cada warp incluye 32 *work-items*). La ejecución del algoritmo K-medias (líneas 04-14 de la Figura 2) y la computación de las distancias intraclase e interclase (líneas 16 y 17 de la Figura 2) constituyen el segundo nivel de paralelismo. Así, la expresión `<<Work-groupID,Work-itemID>>` hace referencia a la distribución de *work-items* en cada *work-group*. Para comparar el aprovechamiento de la GPU respecto a los núcleos superescalares del host (Sección IV) se ha implementado un Kernel de CPU que aprovecha el nivel de paralelismo correspondiente al esquema maestro-trabajador pero no el paralelismo de datos (en este *kernel* un *work-group* está constituido solo por un *work-item*).

```

Kernel función evaluar(S(i), DS, K, DS)
Input : Un conjunto de características seleccionadas, S(i)
Input : Conjunto de datos DS = {DS(i), i=1, ..., P} (P patrones de F componentes)
Input : Conjunto K de W centroides elegidos aleatoriamente del conjunto DS
Input : El conjunto DS es el DS ordenado por columnas (column-major order)
Output: f(S(i)), estimación de las distancias intraclase para la selección S(i)
Output: f(S(i)), estimación de las distancias interclase para la selección S(i)

01 << Todas las work-groups, todos los work-items >>
02 for i = 1 to N individuos
03 << Work-groupID, todos los work-items del work-groupID >>
04 K ← Copiar los centroides de la memoria global a la memoria local
05 I ← Copiar el individuo S(i) de la memoria global a la memoria local
06 Inicializar la tabla de mapeo (Mapping Table), MT ← 0
07 do
08 << Work-groupID, Work-itemID >>
09 MT ← Se asigna cada patrón al cluster más cercano en DS;
10 D ← Se almacena la distancia más cercana de cada patrón
11 Comprobar si el patrón se ha asignado a otro centroide
12 << Work-groupID, todos los work-items del work-groupID >>
13 K ← Actualizar los centroides utilizando DS
14 while no se alcanza el criterio de parada
15 << Work-groupID, Work-item número 0 >>
16 f(S(i)) ← intraclase(K, DS)
17 f(S(i)) ← interclase(K, DS)
18 end
19 return (f(S(i)), f(S(i)))
End
    
```

Fig. 2. Pseudocódigo para el kernel de GPU de evaluación de las funciones de coste.

A continuación se describen las optimizaciones, fundamentalmente en relación con el uso de la jerarquía de memoria, que se han llevado a cabo en el Kernel de GPU que proponemos aquí, con respecto al descrito en [22]:

1. Los *Kernels* de GPU y de CPU reciben los parámetros desde el código de host: los individuos de la población, el conjunto de datos y los centroides iniciales para el algoritmo K-medias. Cada uno de los individuos de la población, $S(i)$, es un array unidimensional de unos y ceros consecutivos (según se seleccione o no la característica correspondiente a la posición del uno o el cero) y se almacenarán en la memoria global. Posteriormente, como se muestra en la línea 5 de la Figura 2, el individuo se almacena en la memoria local, más rápida. Así, para almacenar la población se utilizan $S_{pop} = N \times F$ bytes de memoria global (N es el número de individuos de la población y F el número de características entre las que se hace la

selección). Los arrays DS y DS_c corresponden ambos al conjunto de P patrones de entrenamiento (cada uno con F características) y se almacenan en la memoria global como arrays unidimensionales de $P \times F$ elementos normalizados por el host (Figura 3). El array DS está ordenado según las filas de la matriz de P filas y F columnas correspondiente al fichero de P patrones de F características (*row-major order*) mientras que DS_c lo está según la columnas de dicha matriz (*column-major order*). Cada uno de estos arrays, DS y DS_c , ocupa $S_{DB} = 4 \times P \times F$ bytes. En lugar de copiar los W centroides que se eligen, inicialmente de manera aleatoria, entre los patrones del conjunto de datos, solo se copian los índices de dichos patrones desde la memoria del host a la memoria constante (*constant memory*) de la GPU. Ocupan $S_w = 4 \times W$ bytes.

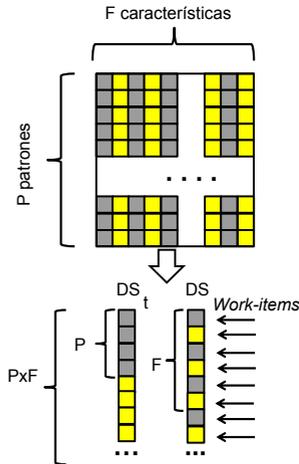


Fig. 3. Distribución de características de los patrones en los arrays DS y DS_c .

2. Dado que las posiciones de los centroides se modifican a lo largo de las iteraciones del algoritmo K-medias (si no cambiasen el algoritmo habría terminado), cada vez que se va a evaluar un individuo hay que copiar los centroides desde la memoria constante (la memoria global en realidad) a la memoria local (línea 04). Las operaciones de las líneas 04 y 05 de la Figura 2 se realizan en paralelo por los *work-items* del *work-group* correspondiente al individuo que se está evaluando. Para estas transferencias podemos aprovechar la *coalescencia*, una técnica que permite acelerar el acceso a memoria de las hebras consecutivas de un *warp*

cuando hacen referencia a datos almacenados en direcciones consecutivas en la memoria global. De esta forma se aprovecha el ancho del bus de memoria para acceder a varios datos en una única transacción. Por otra parte, y como muestra la Figura 4, también se minimizan los conflictos entre bancos de memoria de la memoria local (shared memory en las GPUs de NVIDIA) al almacenar adecuadamente los datos. Una vez que los *WT work-items* del *work-group* terminan de procesar *WT* datos, se solicitan y procesan los siguientes *WT* datos, hasta finalizar. En el Kernel de CPU el único *work-item* del *work-group* realiza secuencialmente la copia de los individuos de la población y de los centroides. Los centroides necesitan $S_{Ki} = 4 \times W \times F$ bytes de memoria y cada individuo $S_{ind} = F$ bytes (W centroides, F características y 4 bytes por dato en coma flotante).

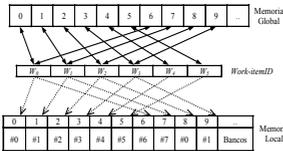


Fig. 4. Copias desde memoria global a memoria local por parte de los *Work-items* implementando coalescencia y minimizando los conflictos entre los bancos de memoria.

3. La tabla de mapeo *MT* necesita $S_{MT} = P$ bytes de memoria local (P es el número de patrones de entrenamiento del conjunto de datos *DS*). Esta tabla contiene el centroide asignado a cada uno de los patrones tras cada iteración del algoritmo K-medias. En la inicialización (línea 06) participan los *work-items* como lo hacen en la inicialización de los centroides y los individuos. Solo hay que almacenar el índice del centroide correspondiente a cada patrón, K_i . Por otra parte, a través de la tabla *MT*, es más fácil determinar si ha terminado el algoritmo K-medias al comprobar si ha habido algún patrón para el que ha cambiado su centroide (línea 11) en lugar de hacerlo al final de la iteración (si ningún patrón cambia su centroide el algoritmo K-medias ha finalizado). En el *Kernel de GPU* que se presenta en [22] y que tomamos como referencia se utilizan tres tablas en la memoria local. Una tabla almacena el centroide asignado a cada patrón en la iteración i , y otra almacena la asignación que se hizo en la iteración $i-1$. Cuando termina la iteración i se comparan ambas tablas para comprobar si ha habido algún cambio y comprobar la condición de final del algoritmo K-medias. La tercera tabla se utiliza como almacenamiento auxiliar de valores intermedios en el intercambio de las otras dos tablas al finalizar cada iteración. Así, en el *Kernel de GPU* de [22] se

realizan $3 \times W \times P$ operaciones para intercambiar las tablas y se necesitan $S_{MT} = 3 \times W \times P$ bytes de memoria local, dado que cada tabla tiene $W \times P$ bytes para guardar una etiqueta (un byte) que indica si un determinado centroide está asignado a un patrón o no. En la versión que presentamos aquí, no se utilizan estas etiquetas sino que para cada patrón se guarda el índice del centroide al que pertenece ($S_{MT} = P$). Hay que tener en cuenta que en el *Kernel de CPU* implementado en [22] solo se utilizaban dos de las tablas de $W \times P$ bytes dado que el intercambio se puede hacer, en un tiempo independiente del tamaño de las tablas, a través del intercambio de punteros a dichas tablas dado que en el *Kernel de CPU* no había diferencia real entre memoria global o local.

4. Para encontrar el centroide más cercano a un patrón se utiliza un *work-item*, que tiene que usar las distancias euclídeas entre el patrón y los centroides. El array *DS*, se utiliza precisamente para facilitar esa tarea (no son más que los datos del array *DS* pero almacenados en un orden distinto). Así, las P primeras direcciones de *DS*, almacenan los valores de la primera característica de los P patrones, y así sucesivamente. Puesto que cada *work-item* trabaja con un patrón diferente, *work-items* consecutivos pueden acceder a la misma característica de patrones distintos que están almacenados en direcciones lógicas consecutivas. De esta forma se puede aprovechar la coalescencia de memoria. Además, cuando se han determinado los centroides más cercanos a cada patrón, y las distancias correspondientes, esta información se puede escribir en *MT* y *D*, respectivamente, con el mínimo número de conflictos en los bancos de memoria. El array *D* contiene las distancias euclídeas entre cada patrón y su centroide (un total de $S_D = 4 \times P$ bytes) y se encuentra almacenado en la memoria local, como *MT*.

5. La actualización de los centroides (línea 13) es, posiblemente, la etapa más difícil de paralelizar del algoritmo K-medias. De hecho, aproximaciones como las descritas en [24,25] no paralelizan este paso, aunque el coste de transferir, en cada iteración, los centroides entre el host y la GPU y viceversa puede ser demasiado costoso, especialmente en aplicaciones con patrones de dimensionalidad elevada. En nuestro *Kernel de GPU* cada *work-item* suma la misma característica de todos los patrones que pertenecen al centroide en cuestión. Ahora el conjunto *DS_i* no contiene la ordenación adecuada dado que *work-items* consecutivos deben procesar características consecutivas de un patrón. Para acceder a esta información de forma eficiente mediante la técnica de coalescencia y también reducir los conflictos en los bancos de memoria se utiliza *DS*, dado que sus F primeras direcciones contienen las características del primer patrón, y así sucesivamente para el resto

de patrones. En el *Kernel de GPU* de referencia [22] solo se utiliza el array *DS* y no se hace uso de la coalescencia en este caso.

6. Los *Kernels de GPU* y de *CPU* devuelven los valores de las funciones de coste determinados a partir de las distancias intraclase e interclase obtenidas según las ecuaciones (1) y (2) de la Sección II (líneas 16 y 17).

IV. RESULTADOS EXPERIMENTALES

A continuación se analizan las prestaciones de los códigos *OpenCL* desarrollados a partir de los resultados experimentales obtenidos tras su ejecución en una plataforma constituida por dos nodos NUMA conectados con Gigabit Ethernet. Cada nodo tiene 32 GBytes de memoria DDR3 y dos procesadores Intel Xeon E5-2620 con seis núcleos cada uno y dos hebras por nodo, y frecuencia de reloj de 2.1 GHz. Uno de los nodos incluye una GPU Tesla K20c con 5 GBytes de memoria global, ancho de banda máximo de memoria de 208 GBytes/s, 2496 núcleos CUDA (SP) a 705.5 MHz distribuidos entre 13 SMX multiprocesadores (192 núcleos CUDA por SMX). En los experimentos se han utilizado dos conjuntos de patrones de entrenamiento extraídos de las bases de datos del *BCI Laboratory* de la Universidad de Essex descritas en [26]. El conjunto b480a consta de 178 patrones (es decir 178 EEGs) con 480 características cada uno correspondientes al individuo 110 de la base de datos mencionada. También se ha utilizado otro conjunto, el b3600a, con 178 patrones de 3600 características cada uno y también correspondiendo al individuo 110. Cada experimento se ha repetido 10 veces y se ha aplicado el test de Kolmogorov-Smirnov para determinar si los resultados siguen una distribución normal. Según los resultados de este test, se ha aplicado un test ANOVA o de Kruskal-Wallis para analizar si las diferencias observadas son estadísticamente significativas o no.

Como se ha indicado más arriba, el algoritmo evolutivo multi-objetivo utilizado ha sido NSGA-II [23] implementando un operador de cruce de dos puntos que se aplica con una probabilidad de 0.9, operador de mutación por inversión del bit seleccionado, y operador de selección de torneo binario. La aproximación al frente de Pareto obtenida al finalizar el algoritmo se ha evaluado a través del hipervolumen [27], tomando como referencia el punto (1,1).

La Tabla I compara las necesidades de almacenamiento de los *Kernels de GPU* comparados (el descrito en [22] y el que se propone en este trabajo) en los distintos tipos de memoria. Se puede ver que aprovechar la coalescencia de memoria supone incrementar las necesidades de

almacenamiento para los patrones (*Datos* en la Tabla I). No obstante, solo se transfieren una vez al inicio del programa y además, para el resto de estructuras se han reducido las necesidades de almacenamiento.

TABLA I Memoria (bytes) necesaria en el código de [22] (Ref) y en el propuesto en este trabajo (Opt)

Memoria		Global		Constante	
Descripción		Población	Datos	Centroide	
Array		S_{Pop}	S_{DB}	S_W	
Tam.	Ref	$N \times F$	$4 \times P \times F$	$4 \times W \times F$	
	Opt	$N \times F$	$8 \times P \times F$	$4 \times W$	
Tam.	Ref	$N \times F + 4 \times P \times F$		$4 \times W \times F$	
	Total	$N \times F + 8 \times P \times F$		$4 \times W$	
Memoria		Local			
Descripción		Centroide	Indiv.	Tablas	Distanc.
Array		S_{Ki}	S_{ind}	S_{Mrr}	S_D
Tam.	Ref	$4 \times W \times F$	F	$3 \times W \times P$	$4 \times W \times P$
	Opt	$4 \times W \times F$	F	P	$4 \times P$
Tam.	Ref	$4 \times W \times F + 7 \times W \times P + F$			
Total	Opt	$4 \times W \times F + 5 \times P + F$			

La Figura 5 muestra los hipervolumenes obtenidos después de diez repeticiones de experimentos correspondientes a diferentes códigos y configuraciones de núcleos, tamaños de población, e iteraciones del algoritmo evolutivo. El análisis realizado muestra que las diferencias observadas en los códigos paralelos no son estadísticamente significativas con respecto a los resultados de los códigos secuenciales en las mismas condiciones de población e iteraciones. Esto era de esperar dado que el algoritmo evolutivo paralelo implementado es de tipo maestro-trabajador y tiene el mismo comportamiento que la versión secuencial.

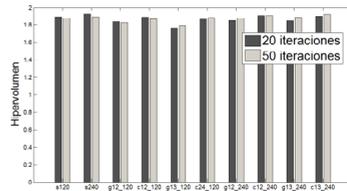


Fig. 5. Medias del hipervolumen para 20 y 50 iteraciones y diferentes condiciones experimentales (sY=secuencial; cX_Y kernel de CPU; gX_Y kernel de GPU; X=hebras en el host o SMXs en la GPU; Y=tamaño de la población).

A continuación se analizan las prestaciones de los códigos en relación al aprovechamiento del paralelismo en la GPU, y respecto al

aprovechamiento de los núcleos superescalares del host. En todos los experimentos realizados, los tiempos de ejecución del *Kernel de GPU* en la versión que proponemos en este trabajo son menores que los de la versión de referencia [22] considerando las mismas condiciones de uso de la GPU. Para ilustrar esta situación, la Figura 6 compara los tiempos de ejecución medios del *Kernel de GPU* para 20 y 50 iteraciones del algoritmo evolutivo NSGA-II, utilizando los 13 SMXs multiprocesadores de la GPU y 192, 256, 512, y 1024 work-items en el fichero de datos b480a. Se puede ver que no solo se mejoran los tiempos cuando se compara la versión de referencia (Ref) [22] con la que proponemos aquí (Opt), sino que, además la reducción de tiempos aumenta a medida que se incrementa el número de *work-items*, debido al mejor aprovechamiento de la coalescencia. Esto se pone de manifiesto, a partir de 256 *work-items*, en la Figura 7, que muestra las diferencias de tiempo para distinto número de SMXs e iteraciones. Es más, para el fichero de datos b3600a no ha sido posible comparar en las mismas condiciones los códigos de referencia (Ref) y los aquí propuestos (Opt) debido a los requisitos de memoria del mencionado código de referencia.

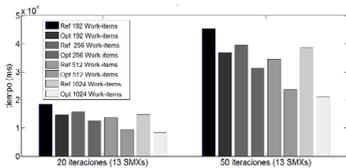


Fig. 6. Tiempos medios de ejecución para 13 SMXs y el fichero de datos b480a (población= 1000 individuos; Ref= códigos de [22]; Opt=códigos propuestos).

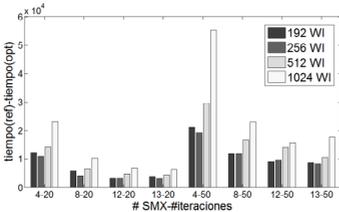


Fig. 7. Reducción del tiempo (en ms) de los Kernels de GPU optimizados propuestos respecto a los descritos en [22] con el fichero de datos b480a (población=1000 individuos)

La Figura 8 compara las ganancias de velocidad obtenidas por el *Kernel de CPU* con las obtenidas por los *Kernels de GPU* que se proponen aquí. Se

consideran poblaciones de 120 y 240 individuos y 20 y 50 iteraciones utilizando los patrones de 3600 características (fichero de datos b3600a). Por un lado se ha considerado la ejecución en tantos SMXs como núcleos superescalares en el host (12 SMXs frente a 12 núcleos), y por otro lado se proporcionan las ganancias para el máximo número de SMXs en la GPU (13 SMXs) y de núcleos en el host (24 núcleos). Todas las medidas de ganancia se calculan respecto al mismo tiempo medio de ejecución de los códigos propuestos para un núcleo de CPU (la comparación de ganancias también permite comparar tiempos medios de ejecución). Como se puede ver, si se utiliza el mismo número de SMXs y núcleos superescalares, se obtienen mejores prestaciones con la GPU. Esto no sucedía en la versión de referencia de los *Kernels de GPU* descrita en [22]. Cuando se compara toda la GPU con todos los núcleos superescalares disponibles en la plataforma, la GPU queda por detrás, pero estamos comparando 13 SMXs con 24 núcleos en arquitecturas que, además funcionan con frecuencias de reloj bastante distintas: 2.1 GHz el host frente a 706 MHz la GPU.

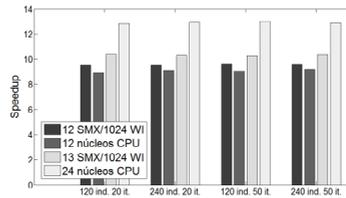


Fig. 8. Ganancias de velocidad media para los kernels de CPU y GPU optimizados propuestos con el fichero de datos b3600a (población=1000 individuos)

V. CONCLUSIONES

Un número considerable de trabajos han abordado la paralelización de algoritmos evolutivos en GPUs proporcionando resultados de ganancias de velocidad bastante significativos. Sin embargo, existe menos información acerca de los beneficios de este tipo de arquitecturas (dotadas de un número considerable de núcleos capaces de aprovechar el paralelismo de datos) en aplicaciones de minería de datos con irregularidades en los códigos o en los accesos a los datos, y patrones de alta dimensionalidad y/o grandes volúmenes de datos. En este artículo se comparan distintos códigos paralelos, ejecutados en plataformas heterogéneas incluyendo núcleos superescalares y GPUs, para la selección de características multi-objetivo en problemas de clasificación de EEGs para tareas de BCI. Los códigos se han implementado en *OpenCL* y nos han permitido analizar algunos de los aspectos

del comportamiento de las diversas aproximaciones paralelas a través de los correspondientes *Kernels para GPU* y *para CPU* desarrollados.

Más concretamente, la aplicación paralelizada se basa en un algoritmo de optimización multi-objetivo con dos funciones de costo. Evaluar estas funciones para un individuo de la población del algoritmo evolutivo (una selección de características) implica calcular los valores de dos índices de validación (la distancia intraclase y la distancia interclase) en los *clusters* obtenidos tras aplicar el algoritmo de *clustering* K-medias al conjunto de patrones utilizado (los patrones tienen por componentes las características de la selección codificada a través del individuo de la población que se está evaluando).

Los *Kernels de GPU* que se han implementado aprovechan dos niveles de paralelismo. Por un lado el algoritmo evolutivo multi-objetivo se ha paralelizado mediante la distribución de los individuos de la población entre los multiprocesadores SMX de la GPU, de forma que cada uno de ellos se encarga de evaluar las funciones de coste de los individuos que les han correspondido. Dado que el tiempo necesario para la ejecución de los operadores evolutivos de selección, cruce, y mutación del algoritmo evolutivo es mucho menor que la evaluación de los individuos, esos operadores no se han paralelizado y se ejecutan en el host. Esta implementación paralela de tipo maestro-trabajador que se ha hecho del algoritmo evolutivo multi-objetivo es la que se ha llevado a cabo en el *Kernel de CPU*. El segundo nivel de paralelismo solo se aprovecha en el *Kernel de GPU* y consiste en paralelizar la propia evaluación de las funciones de costo de cada individuo. Así, la ejecución del algoritmo K-medias para cada individuo se distribuye entre los *work-items* del multiprocesador SMX al que se haya asignado dicho individuo. Para alcanzar un buen nivel de aprovechamiento del paralelismo de datos que nos proporciona la arquitectura de la GPU se han estudiado con detalle las posibilidades de uso eficiente de la jerarquía de memoria de la GPU que permite la aplicación considerada, y se han aprovechado técnicas entre las que la coalescencia de los accesos a memoria y la minimización de los conflictos entre los bancos de memoria han sido las más eficientes.

Los resultados experimentales ponen de manifiesto una reducción considerable del tiempo de ejecución en las implementaciones optimizadas del *Kernel de GPU* que proponemos en este trabajo con respecto a una primera versión de referencia descrita en [22]. Además, en el caso de uso del mismo número de multiprocesadores SMX en la GPU y núcleos superescalares en el host se obtienen mejores tiempos en la GPU, a pesar de la diferencia entre las frecuencias de reloj en uno y otro caso. Se pone de

manifiesto, por tanto, el efecto del paralelismo de datos que se ha implementado solo en el *Kernel de GPU*. En el *Kernel de GPU* mejorado que describimos aquí también se observa un uso más eficiente de los *work-items* a medida que se incrementa el número de ellos que se utiliza. No obstante, a pesar de los relativos buenos resultados que se describen en este artículo, se deberían explorar más alternativas de paralelización para mejorar el aprovechamiento de las arquitecturas heterogéneas en nuestro ámbito de aplicación. Entre esas alternativas estarían la implementación del algoritmo evolutivo multi-objetivo mediante modelos paralelos de islas, que podrían poner de manifiesto nuevas posibilidades de aprovechamiento de las arquitecturas de las GPUs conjuntamente con los núcleos superescalares del host.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad y los fondos FEDER a través del proyecto TIN2015-67020-P. Agradecemos al laboratorio de BCI de la Universidad de Essex, y muy especialmente al Prof. John Q. Gan, la posibilidad de utilizar su base de datos.

REFERENCIAS

- [1] Rupp R, Kleih SC, Leeb R, Millán JR, Kübler A, Müller-Putz GR. Brain-Computer Interfaces and Assistive Technology. Brain-Computer Interfaces in Their Ethical, Social and Cultural Contexts. The International Library of Ethics, Law and Technology, Grübler and Hildt (eds.), Springer Science-Business Media Dordrecht, 2014, 12: 7-38. DOI: 10.1007/978-94-017-8996-7_2.
- [2] Jang B, Schaa D, Mistry P, Kaeli D. Exploiting Memory Access Patterns to Improve Memory Performance in Data-Parallel Architectures. IEEE Trans. On Parallel and Distributed Systems, 22(1), pp.105-118, 2013.
- [3] Fauzia, N, Pouchet L-N, Sadayappan P. Characterizing and Enhancing Global Memory Data Coalescing on GPUs. 13th IEEE/ACM Intl. Symp. On Code Generation and Optimization, 2015.
- [4] OpenCL registry: www.khronos.org/registry/cl/
- [5] Collet P. Why GPGPUs for Evolutionary Computation?. In Tsutsui, S.; Collet, P. (Eds.), Massively Parallel Evolutionary Computation on GPGPUs, pp. 3-14, Springer, 2013.
- [6] Luong T V, Melab N, Talhi E-G. GPU-based island model for evolutionary algorithms. GECCO'10 Proceedings of the 12th annual conference on Genetic and evolutionary computation, pp. 1089-1096, 2010.
- [7] Alba E, Laque G, Nesmachnow, S. Parallel Metaheuristics: recent advances and new trends. Intl. Trans. in Op. Research, 20, pp.1-48, 2013.
- [8] Prospichal P, Jaros J, Schwarz. Parallel Genetic Algorithms on the CUDA Architecture. In Di Chio C et al. (Eds): Evoapplications 2012, Part I, LNCS 6024, pp.442-451, Springer, 2010.
- [9] Bellman, G.A. Adaptive Control Processes: A Guided Tour. Princeton University Press, 1961.
- [10] Handl J, Knowles J. Feature subset selection in unsupervised learning via multiobjective optimization. International Journal of Computational Intelligence Research 2006, 2(3): 217-238. DOI: 10.5019/ijicir.2006.64.

- [11] Arbelaitz O, Gurrutxaga I, Muguerza J, Pérez JM, Perona I. An extensive comparative study of cluster validity indices. *Pattern Recognition* 2013; 46 (1): 243-256.
- [12] Mukhopadhyay A, et al. A Survey of Multiobjective Evolutionary Algorithms for Data Mining: Part I and Part II. *IEEE Trans. on Evolutionary Computation*, 18(1)4-35, 2014.
- [13] Emmanouilidis C, Hunter A, MacIntyre J. A multiobjective evolutionary setting for feature selection and a commonality-based crossover operator. *Proceedings of the 2000 Congress on Evolutionary Computation*, IEEE Press, New York, NY, 2000; 309-316. DOI: 10.1109/CEC.2000.870311.
- [14] Morita M, Sabourin R, Bortolozzi F, Suen CY. Unsupervised feature selection using multi-objective genetic algorithms for handwritten word recognition. *Proceedings of the 7th International Conference on Document Analysis and Recognition*, IEEE Press, New York, NY, 2003; 666-670. DOI: 10.1109/ICDAR.2003.1227746.
- [15] Sharma D, Collet, P. Implementation techniques for massively parallel Multi-objective optimization. In Tsutsui, S.; Collet, P. (Eds.), *Massively Parallel Evolutionary Computation on GPGPUs*, pp. 267-286, Springer, 2013.
- [16] Wong M L, Cui G. Data mining using parallel multi-objective evolutionary algorithms on graphics processing units. In Tsutsui, S.; Collet, P. (Eds.), *Massively Parallel Evolutionary Computation on GPGPUs*. pp. 287-307, Springer, 2013.
- [17] Baramkar P P, Kulkarni D B. Review for K-means on graphics processing units (GPU). *Intl. J. Research & Technology (IJERT)*, Vol. 3, 6, pp.1911-1914, June, 2014.
- [18] Wu R, Zhang B, Hsu M. Clustering billions of data points using GPUs. *Proc. UCHPC-MAW'09*, 2009, doi: 10.1145/1531666.1531668.
- [19] Zechner M, Granitzer M. Accelerating K-Means on the Graphics Processor via CUDA. *Proc. First Intl. Conference on Intensive Applications and Services*, pp. 7-15, 2009.
- [20] Gaster BR, Howes L, Kaeli D, Mistri P, Schan D. *Heterogeneous Computing with OpenCL*. Morgan Kaufmann, 2012.
- [21] Kimovski D, Ortega J, Ortiz A, Baños R. Leveraging cooperation for parallel multi-objective feature selection in high-dimensional EEG data. *Concurrency: Practice and Experience*, 2015.
- [22] Escobar JJ, Ortega J, González J, Damas M. Assessing Parallel Heterogeneous Computer Architectures for Multiobjective Feature Selection on EEG Classification. In *Proceedings of 4th International Conference, IWBBIO 2016*, Granada, Spain, April 20-22, pp.277-289, 2016.
- [23] Deb K, Agrawal S, Pratap A, Meyarivan T. A fast elitist Non-dominated Sorting Genetic Algorithms for multi-objective optimisation: NSGA-II. *Proc. Of the 6th Int. Conference on Parallel Problem Solving from Nature (PPSN VI)*, LNCS 1917, pp.849-858, Springer-Verlag, 2000.
- [24] Gunarathe T, Salpitiikorala B, Chauhan A, Fox G. Optimizing OpenCL Kernels for Iterative Statistical Algorithms on GPUs. In *Proceedings of the Second International Workshop on GPUs and Scientific Applications (GPUScA)*, PACT 2011. Galveston, Tx, pp.33-44.
- [25] Dhanasekaran B, Rubin N. A new method for GPU based irregular reductions and its application to k-means clustering. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units. Proc. 4th Workshop on General Purpose Processing on GPUs (GPGPU-4)*, ACM, New York, NY, USA, Article No.2, 2011. DOI: 10.1145/1964179.1964182.
- [26] Asensio-Cubero J, Gan JQ, Palaniappan R. Multiresolution analysis over simple graphs for brain computer interfaces. *Journal of Neural Engineering*, 10(4), 2013, doi: 10.1088/1741-2560/10/4/046014.
- [27] Fonseca C M, López-Ibáñez M, Paquete L, Guerrero A P. *Computation of the Hypervolume indicator: http://iriidia.ulb.ac.be/~manuel/hypervolume*, 2014.

Nueva Implementación Paralela en GPUs para la Extracción de Firmas Espectrales Puras con Información Espacial y Espectral

Luis Ignacio Jiménez, Javier Plaza, Antonio Plaza¹, Sergio Sánchez² y Gabriel Martín³

Resumen—

La identificación de firmas espectrales puras o *endmembers* en imágenes hiperespectrales ha estado tradicionalmente sujeta exclusivamente al aprovechamiento de la información espectral. Recientemente, algunas técnicas tales como *spatial-spectral endmember extraction* (SSEE) incorporan tanto la información espectral como la información espacial disponible en la imagen. Debido a que la imágenes hiperespectrales contienen abundante información en ambos dominios, la integración de ambas fuentes de información normalmente supone un incremento de la complejidad y del coste computacional. En este trabajo, hemos desarrollado una nueva y eficiente implementación para el algoritmo SSEE utilizando tarjetas gráficas programables (GPUs), dispositivos compactos y de bajo coste que nos permiten obtener un factor de aceleración significativo si los recursos de la arquitectura son usados apropiadamente. La validación experimental se centra en la evaluación de los píxeles candidatos seleccionados por el algoritmo SSEE y en el rendimiento computacional de la implementación paralela. Dicha validación confirma que se puede aprovechar el algoritmo SSEE de una forma computacionalmente eficiente.

Palabras clave— Análisis hiperespectral, *spatial-spectral endmember extraction* (SSEE), tarjetas gráficas programables (GPUs).

I. INTRODUCCIÓN

EL desmezclado espectral [1] es una técnica muy importante para el aprovechamiento de las imágenes hiperespectrales en teledetección. En los últimos años, muchos métodos han sido desarrollados con el objetivo de identificar las firmas espectralmente puras que constituyen la imagen (llamadas *endmembers*) y su correspondiente fracción de abundancia a nivel de píxel [2]. Un problema recurrente es cómo identificar automáticamente los *endmembers* que son representativos de la información espacial y espectral contenida en la imagen. Por ejemplo, normalmente es complicado obtener píxeles espacialmente representativos, teniendo en cuenta que normalmente los algoritmos de identificación de *endmembers* se centran únicamente en la información espectral, lo que conlleva sensibilidad al ruido, a los valores atípicos y a *endmembers* anómalos [3].

Para abordar este asunto, varias estrategias han sido propuestas con la intención de guiar la identificación de *endmembers* hacia áreas espacialmente

homogéneas, de las que se espera contengan las firmas espectrales más puras en la imagen [4], [5], [6]. Con este objetivo, han sido varios los métodos espaciales/espectrales desarrollados para la identificación de *endmembers* en datos hiperespectrales.

Uno de los primeros algoritmos diseñados con este propósito en la literatura fue *automatic morphological endmember extraction* (AMEE) [4], el cual usa las operaciones morfológicas de erosión y dilatación para discriminar los *endmembers* que son suficientemente puros, espectralmente hablando, cubriendo un área más amplia en términos espaciales. Otro método comúnmente utilizado es el *spatial-spectral endmember extraction* (SSEE) [5], que utiliza restricciones espaciales para mejorar el contraste espectral relativo entre *endmembers* con información espectral mínima y única mejorando así el potencial de estos elementos potencialmente importantes. Con el SSEE, la información espacial de la imagen es utilizada para incrementar el contraste espectral entre *endmembers* espectralmente similares pero espacialmente indistinguibles.

Por último, otros algoritmos de preprocesado espacial han sido usados para la identificación de *endmembers* [7], [8], [9]. Estos métodos fueron desarrollados con la idea de ser combinados con otros algoritmos basados en información espectral para la extracción de *endmembers*. *Spatial preprocessing* (SPP) [7] es un algoritmo que introduce la información espacial en el proceso de extracción de los *endmembers*, de tal manera que se puede acoplar con los métodos clásicos de identificación de *endmembers* [10]. La idea principal de este preprocesado es la de estimar, para cada píxel de la imagen de entrada, un valor escalar relativo a la similaridad espacial entre dicho píxel y su vecindario, definido por una ventana espacial alrededor del píxel. Una extensión de este concepto fue presentada en [8], donde el uso de vecindarios espaciales fijos en el SPP fue sustituido por la incorporación de regiones pensadas para una mejor caracterización del contexto espacial. Sin embargo, el RBSP depende enormemente del algoritmo de crecimiento de regiones que realiza antes que el procedimiento haciéndolo especialmente sensible a la técnica seleccionada. Otra técnica más reciente es el *spatial and spectral preprocessing* (SSPP) desarrollado en [9], donde se integran ambos tipos de información, espacial y espectral, a nivel de preprocesado. El proceso se divide en cuatro etapas: 1) filtrado gaussiano multiescalar, donde la imagen origi-

¹Dpto. de Arquitectura de Tecnología de los Computadores y las Comunicaciones, Univ. de Extremadura, Cáceres, España. e-mail: {lujimenez, jplaza, aplaza}@unex.es

²Masdar Institute of Science and Technology, Abu Dhabi, UAE. e-mail: smartinez@masdar.ac.ae

³Instituto de Telecomunicações, Lisboa, Portugal. e-mail: salmeron@per.edu

inal es filtrada para remarcar el contexto espacial, 2) el cálculo de la homogeneidad espacial, donde la imagen filtrada es procesada para obtener un índice de homogeneidad espacial para píxel de la imagen original, 3) el cálculo de un índice de pureza espectral, que en primer lugar reduce la imagen hiperespectral usando el análisis de componentes principales (PCA) [11] y crea un índice de pureza espectral usando un método parecido al adoptado por el algoritmo *pixel purity index* (PPI) [12], y 4) un proceso de fusión de ambos tipos de información donde los únicos píxeles seleccionados son aquellos que son al mismo tiempo espectralmente puros y espacialmente homogéneos.

Todas las técnicas anteriormente mencionadas para la identificación de endmembers basadas en información espacial y espectral se caracterizan por tener un coste computacional alto, debido a la necesidad de procesar ambos tipos de información contenidos en las imágenes hiperespectrales. Sin embargo, debido a que los cálculos son similares, muchas de estas técnicas han sido eficientemente implementadas utilizando tarjetas de procesamiento gráfico o GPUS. Por ejemplo el algoritmo SPP fue implementado para GPUS en [13]. El SSPP también ha sido implementado para GPUS en [14], mientras que el RBSP también es susceptible de ser paralelizado (con la principal dificultad del proceso de segmentación que generalmente resulta en una distribución irregular en versiones paralelas). El algoritmo AMEE ha sido también implementado para GPUS en [15]. Aunque, el SSEE (un método bastante representativo y exitoso para la extracción de endmembers utilizando información espacial y espectral) aA^n no ha sido implementado sobre aceleradores gráficos.

En este trabajo de ha desarrollado una implementación del algoritmo SSEE [5] para GPUS. En este método, la imagen hiperespectral es procesada con la intención de buscar un conjunto de firmas que se consideran candidatas a ser endmembers de entre las cuales que procederá a la extracción de las firmas espectrales puras finales. En consecuencia, el SSEE se puede considerar un algoritmo de preprocesado espacial. Reduciendo el número de candidatos a ser endmembers se puede reducir considerablemente el tiempo computacional el posterior algoritmo de extracción de endmembers, por lo que el SSEE puede ser combinado con algoritmos clásicos de extracción de endmembers como si fuera otro algoritmo de preprocesado tal y como el SPP, el SSPP o el RBSP. Aunque el algoritmo SSEE se caracteriza por una complejidad y unos tiempos de computación muy elevados, se han implementado una serie de optimizaciones que permiten una ejecución eficiente del método en GPUS utilizando *Arquitectura Unificada de Dispositivos de Cómputo* (CUDA)¹ de NVidia. La implementación propuesta ha sido probada en dos arquitecturas NVidia distintas usando imágenes reales. Se han realizado comparaciones de los nuevos métodos propuestos contra las implementaciones paralelas disponibles de los

¹<https://developer.nvidia.com/cuda-zone>

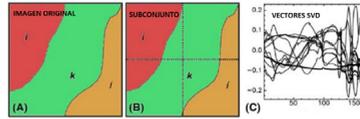


Fig. 1. La primera etapa del algoritmo SSEE (cálculo de autovectores). La imagen (A) contiene tres componentes diferentes (i, j, k) y es dividida en cuatro subconjuntos (B) del mismo tamaño y disjuntos. La SVD es utilizada para obtener los autovectores (C).

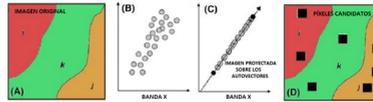


Fig. 2. El segundo paso del algoritmo SSEE (la proyección de los datos). La imagen original (A) representada en el espacio caracterizado como se indica en (B), es proyectada sobre los autovectores obtenidos en la Figura 1(C), identificando los valores de las proyecciones máxima y mínima para seleccionar los píxeles candidatos (D).

algoritmos SPP y SSPP, revelando que estos algoritmos pueden ser eficientemente aprovechados en GPUS al incluir la información espacial en el proceso de extracción de endmembers.

El resto del artículo se organiza como se detalla a continuación. La Sección II describe el algoritmo SSEE en general, así como las distintas versiones optimizadas. La Sección III presenta la implementación GPU del algoritmo SSEE. Los experimentos llevados a cabo para evaluar la precisión de las firmas extraídas y el rendimiento conseguido por las implementaciones paralelas propuestas se especifican en la Sección IV. Por último, la Sección V finaliza este trabajo con una puntualización y posibles líneas de investigación futuras.

II. SPATIAL-SPECTRAL ENDMEMBER EXTRACTION (SSEE) E IMPLEMENTACIONES

El algoritmo SSEE original puede resumirse en los siguiente pasos [5]:

1. En primer lugar, la imagen hiperespectral original es dividida en partes más pequeñas y se utiliza la descomposición de valores singulares (SVD) para obtener los autovectores que describen la varianza espectral de cada uno de subconjuntos. Esta etapa del algoritmo es la que más tiempo consume. La implementación original del algoritmo SSEE define que los subconjuntos de la imagen no pueden solaparse entre ellos (ver la Figura 1).
2. Toda la imagen es proyectada sobre los mencionados autovectores obteniendo los valores de la proyección máxima y mínima que determinan los píxeles candidatos iniciales (ver la Figura 2).
3. En la tercera etapa (la expansión y promedio de los píxeles candidatos), el algoritmo SSEE

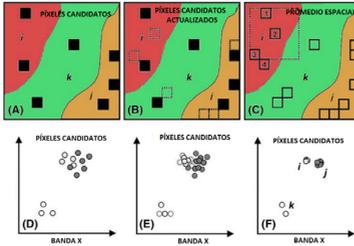


Fig. 3. Tercera etapa del algoritmo SSEE. Después de seleccionar los píxeles candidatos iniciales distribuidos espacialmente con respecto a las clases i , j y k (A), el proceso de expansión comienza alrededor de cada píxel que es espacialmente cercano y espectralmente similar a los seleccionados anteriormente en (B). Un proceso de promedio espacial (utilizando una ventana fija alrededor de cada píxel candidato) es realizado en (C). Por ejemplo en (D), los píxeles candidatos no están promediados. El promedio espacial-espectral comienza en (E) para cada clase. Esta etapa continúa por un número de iteraciones determinado por el usuario, agrupando los píxeles candidatos en clases espectrales separadas (F).

utiliza restricciones espaciales para combinar y promediar los píxeles candidatos que son espectralmente similares. En otras palabras, del conjunto de píxeles candidatos obtenidos en el paso anterior es extendido en base a la similitud espectral entre candidatos dentro de una ventana alrededor de cada píxel, seguido de un promedio en otra ventana del mismo tamaño que la primera. Este proceso separa los endmembers por clases en el espacio espectral, como se ilustra en la Figura 3. En este punto, el algoritmo SSEE ha obtenido un conjunto de píxeles candidatos que contendrá a los endmembers finales. En consecuencia, hasta este punto el algoritmo SSEE produce unos resultados del mismo tipo que otros algoritmos de preprocesado tales como el SPP o el SSPP.

4. En el último paso el algoritmo SSEE realiza un listado de los píxeles candidatos basado en la distancia de cada uno de ellos respecto al primero de la lista. A partir de ahí, como se menciona en el artículo original [5], se puede usar un procedimiento manual o técnicas de extracción de endmembers basadas en información espectral clásicas como *orthogonal subspace projection* (OSP) [16].

El algoritmo SSEE descrito anteriormente está caracterizado por un alto coste computacional. La mayoría del tiempo de computación (alrededor de un 99%) se gasta en la selección de candidatos, debido principalmente al coste computacional de la operación SVD y las posteriores proyecciones usando la imagen completa. Antes de explorar la propuesta de implementación para GPUs, se va a describir la optimización principal del método usada para reducir su complejidad. Esta modificación está centrada en

el método de obtención de los autovectores.

La SVD no es única técnica de proyección que se puede usar para determinar los autovectores necesarios para el algoritmo SSE. En este trabajo se estudia la posibilidad de usar una implementación rápida de la PCA [17] con esta intención. Esta técnica tiene la ventaja de reducir la complejidad del cálculo de las proyecciones. Además, la PCA es altamente paralelizable con muchas implementaciones disponibles.

III. IMPLEMENTACIÓN GPU

La implementación GPU para el algoritmo SSEE ha sido desarrollada usando el lenguaje de programación CUDA de NVidia, con algunas llamadas a funciones disponibles en las librerías cuBLAS² y BLAS/LAPACK³. A continuación se describe la arquitectura general de las GPUs y la implementación específica de nuestra propuesta.

A. Arquitectura GPU

Las GPUs se pueden considerar en términos de un modelo de flujo de datos. La Figura 4 nos muestra la arquitectura de una GPU, la cual puede ser considerada como un conjunto de multiprocesadores (MPs). Cada multiprocesador es definido por un módulo SIMD (*single instruction multiple data*). Cada procesador tiene acceso a la memoria local compartida y también a las memorias caché en el multiprocesador, mientras que el multiprocesador tiene acceso a la memoria global de la GPU. Los algoritmos están constituidos por funciones, llamadas *kernels* que operan con flujos de datos completos y son ejecutados por el multiprocesador, usando uno o más flujos como entradas y produciendo uno o más flujos como salidas. De este modo, el paralelismo a nivel de datos está expuesto al hardware, y los kernels pueden ser ejecutados concurrentemente sin ningún tipo de sincronización. Los kernels puede realizar un tipo de procesamiento por lotes dispuestos en forma de malla (*grid*) de bloques (*blocks*) (ver la Figura 5), donde cada bloque se compone de varios hilos (*threads*) que comparten los datos a través de la memoria compartida local y sincronizan su ejecución mediante coordenadas de acceso a la memoria. En consecuencia, hay diferentes niveles de memoria en la GPU para hilo, bloque y malla (ver la Figura 6). También hay un número máximo de hilos que un bloque puede contener pero el número de hilos que se ejecutan concurrentemente puede ser mayor (bloques ejecutados por el mismo kernel pueden ser manejados concurrentemente, a expensas de reducir la cooperación entre hilos ya que pertenecen a distintos bloques de la misma malla y no pueden sincronizar entre ellos).

B. Implementación GPU del Algoritmo SSEE

La implementación GPU para las distintas etapas del SSEE pueden resumirse en:

²<https://developer.nvidia.com/cublas>

³<http://www.netlib.org/lapack/>

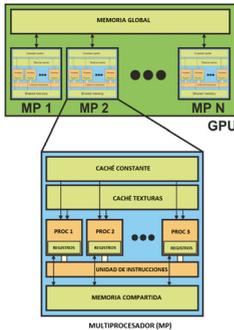


Fig. 4. Esquema general de una arquitectura GPU, vista como un conjunto de multiprocesadores (MPs).

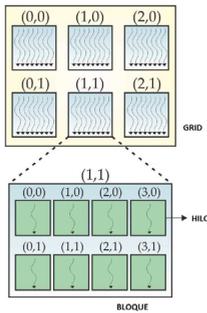


Fig. 5. Procesamiento por lotes en la GPU: grids de bloques de hilos, donde cada bloque está compuesto por un grupo de hilos.

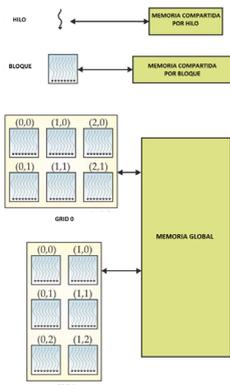


Fig. 6. Diferentes niveles de memoria en la GPU en relación con los conceptos de hilo, bloque y grid.

1. *Cálculo de autovectores.* Una parte significativa del tiempo de ejecución del algoritmo SSEE es utilizado para el cálculo de los autovectores. Este paso puede realizarse usando la SVD o la PCA. La SVD ha sido implementada llamando a la función *dgesvd* disponible en la librería BLAS/LAPACK, la cual ha sido evaluada experimentalmente para ser computacionalmente muy rápida. Para la implementación GPU de la PCA, se realizan los siguientes pasos. Primero, un kernel normaliza el subconjunto de la imagen restando el píxel medio de dicho subconjunto. Este kernel, llamado *avgXCUDA*, establece una malla unidimensional cuyo tamaño es igual al número de bandas de la imagen y el tamaño de bloque es igual al número de hilos disponible en la GPU. A continuación se llama a la función de cuBLAS *cublasDgemm* que realiza la multiplicación del subconjunto de la imagen por su transpuesta. Finalmente la transformación SVD es calculada en la CPU, ya que experimentalmente se ha comprobado que la paralelización no es necesaria para obtener unos resultados mejores a nivel de rendimiento.

2. *Proyección de datos.* Una vez los autovectores han sido calculados y transferidos a la memoria de la GPU, se utiliza la función *cublasDgemm* de nuevo para realizar la proyección de la imagen. Este paso es muy importante en términos de coste computacional mayormente por el incremento de complejidad al aumentar el tamaño de la ventana utilizada. Para este paso, se usa un kernel llamado *maxminProjection* que ha sido diseñado para seleccionar los valores de las proyecciones máxima y mínima de cada banda de la imagen, utilizando una operación de reducción correspondiente al número de bandas que representan el porcentaje del total definido por un valor umbral, *svd.th*, introducido como parámetro. Aquí el número de bloques se establece como el número de bandas y el número de hilos se fija en el máximo permitido por el dispositivo.

3. *Expansión y promedio de los píxeles candidatos.* Para realizar este paso lo primero es calcular la norma euclídea de cada píxel de la imagen utilizando un kernel llamado *euclideanNorm*. El número de hilos para este kernel se fija al máximo permitido por la GPU, y el número de bloques se establece como la relación entre el número de píxeles y el tamaño del bloque más uno. Un segundo kernel llamado *expandCandidatePixels* es desarrollado para realizar la expansión del conjunto de candidatos dentro del rango que marca el tamaño de ventana *ws*. Finalmente otro kernel llamado *averageStep* calcula el promedio espectral de los candidatos dentro del mismo rango definido por la ventana alrededor de cada uno de ellos. Ambos kernels definen una malla bidimensional en la cual el número de filas y columnas son definidos por el

TABLA I

PROMEDIO DE DISTANCIA ESPECTRAL (EN GRADOS) ENTRE EL CONJUNTO DE PÍXELS OBTENIDOS POR LOS DISTINTOS MÉTODOS (SSEE, SPP Y SSPP) Y EL CONJUNTO DE REFERENCIA DE FIRMAS SELECCIONADAS DE LA LIBRERÍA USGS EN LA IMAGEN AVIRIS CUPRITE. EN EL CASO DE LOS ALGORITMOS SSEE Y SSPP VARIOS TAMAÑOS DE VENTANA HAN SIDO CONSIDERADOS. LOS RESULTADOS CORRESPONDEN A LOS OBTENIDOS TRAS 100 EJECUCIONES DEL ALGORITMO DE EMPAREJAMIENTO. LOS MEJORES RESULTADOS (ÁNGULOS BAJOS) SE HAN RESALTADO EN NEGRITA.

	SSEE-SVD				SSEE-PCA				SPP				SSPP
Ventana	35x35	50x50	70x70	175x175	35x35	50x50	70x70	175x175	3x3	5x5	7x7	9x9	
Candidatos	2416	1795	1310	403	204	140	77	28	122500	122500	122500	122500	3653
<i>Alunite GDS84 Na03</i>	7.467	7.088	7.472	8.756	5.735	5.965	6.361	9.241	5.819	5.942	6.041	6.041	6.863
<i>Alunite GDS83 Na63</i>	7.660	6.676	8.165	9.398	6.886	6.660	7.796	10.204	6.876	6.955	7.033	7.035	9.449
<i>Alunite GDS82 Na82</i>	5.089	5.077	5.572	6.871	4.669	4.107	5.223	8.165	4.302	4.395	4.491	4.474	6.958
<i>Alunite AL706 Na</i>	5.713	5.847	6.644	7.848	4.268	5.229	6.286	8.328	4.794	4.820	4.852	4.844	7.320
<i>Alunite HS295.3B</i>	15.044	14.865	15.465	17.039	11.398	11.202	11.724	17.443	12.196	12.486	12.725	12.737	11.002
<i>Alunite SUSTDA-20</i>	6.153	6.195	6.748	7.916	5.427	5.352	6.385	8.810	5.434	5.501	5.562	5.562	7.763
<i>Buddingtonite GDS85 D-206</i>	3.793	3.932	3.904	4.854	3.924	3.918	4.049	5.553	3.781	3.778	3.777	3.777	3.863
<i>Calcite WS272</i>	7.237	7.440	8.341	9.543	5.052	7.035	7.948	10.066	5.252	5.307	5.383	5.380	8.647
<i>Calcite HS48.3B</i>	7.761	7.976	8.863	10.082	5.503	7.644	8.563	10.787	5.559	5.624	5.699	5.700	8.703
<i>Chalcodony CU91-6A</i>	4.242	4.425	5.194	6.275	2.918	4.666	5.019	7.139	2.993	3.035	3.061	3.057	5.717
<i>Chlorite HS179.3B</i>	15.833	16.155	17.032	22.978	15.545	15.192	17.528	22.276	14.796	15.001	15.129	15.146	13.868
<i>Chlorite SMR-13.a 104-150</i>	20.676	21.056	22.675	27.574	20.506	20.018	22.767	27.151	19.695	19.803	20.021	20.018	18.821
<i>Dickite NMNH106242</i>	7.221	7.272	7.769	8.819	6.563	6.664	7.490	9.592	6.618	6.652	6.704	6.699	8.664
<i>Halloysite NMNH106236</i>	12.106	11.733	12.097	13.587	9.376	9.548	9.823	13.546	10.026	10.237	10.388	10.385	10.817
<i>Jarosite GDS99 K-y 200C</i>	8.637	8.607	8.827	11.015	8.531	8.845	9.838	10.630	8.357	8.380	8.392	8.392	8.245
<i>Kaolinite KGa-1 (azyl)</i>	5.296	5.455	6.246	7.460	3.755	5.082	5.928	8.138	4.199	4.249	4.285	4.297	6.848
<i>Kaolinite KGa-2 (pyrl)</i>	4.393	4.573	5.360	6.489	3.052	4.466	5.110	7.005	3.386	3.435	3.455	3.461	5.611
<i>Kaolinite/Smect KLF506 95%K</i>	3.835	3.619	4.249	6.946	4.137	3.363	5.885	6.984	2.799	2.853	2.867	2.885	3.080
<i>Montmorillonite SWy-1</i>	5.922	6.113	6.937	8.045	4.234	6.006	6.681	8.812	4.507	4.550	4.594	4.597	7.232
<i>Montmorillonite SWa-1</i>	8.733	8.688	9.016	10.159	8.348	7.867	8.800	11.163	7.987	8.064	8.142	8.150	9.807
<i>Muscovite GDS107</i>	5.191	5.084	5.682	8.118	5.571	6.314	6.628	8.312	4.092	4.055	4.133	4.142	4.047
<i>Muscovite HS146.3B</i>	8.205	8.003	8.953	12.232	8.090	9.335	10.274	12.119	7.716	7.843	7.902	7.896	7.745
<i>Muscovite HS24.3</i>	6.213	6.026	7.039	10.765	6.138	7.422	8.304	10.275	5.788	5.818	5.833	5.831	5.365
<i>Nontronite GDS41</i>	14.366	14.833	16.429	21.576	14.205	13.372	16.609	21.054	13.365	13.583	13.741	13.727	12.362
<i>Pyrophyllite PYSIA fine g</i>	6.674	6.869	7.675	8.862	4.773	6.616	7.338	9.178	4.892	4.984	5.031	5.041	7.905
Promedio	8.138	8.184	8.930	10.928	7.144	7.756	8.734	11.279	7.009	7.098	7.170	7.171	8.255
S(%)	89.014	89.267	89.465	79.836	91.917	91.943	89.969	75.989	—	—	—	—	88.509

número de filas y columnas de la imagen original respectivamente, y el máximo número de hilos disponibles se fija como tamaño de bloque.

4. *listado*. El último paso es listar los píxeles candidatos en función de la distancia que los separa. En este trabajo, se usa un kernel llamado *BitonicSort* para calcular reordenar mediante un algoritmo de ordenamiento bitónico para GPUs. número de hilos se ha establecido empíricamente a 256 y el número de bloques se calcula según la siguiente expresión:

$$n_bloques = \exp(\log_2 \lfloor \frac{n_pixels}{n_hilos} \rfloor),$$

donde $n_bloques$, n_pixels and n_hilos denotan el número de bloques, número de píxeles y número de hilos respectivamente. Se quiere enfatizar que este último paso es puramente opcional y puede ser reemplazado por un método de extracción de endmembers. También se quiere remarcar que esta última etapa significa un tiempo despreciable con respecto al total, incluso cuando el conjunto de candidatos resulta muy grande.

IV. RESULTADOS EXPERIMENTALES

Los experimentos se han realizado utilizando un subconjunto de la imagen obtenida por *Airborne Visible Infra-Red Imaging Spectrometer* (AVIRIS), operado por *NASA's Jet Propulsion Laboratory* (AVIRIS), sobre la región minera de Cuprite en Nevada en el verano de 1997 (disponible online⁴). La porción utilizada en los experimentos (ver Figura 7) corresponde a un subconjunto de 350×350 píxeles, que comprende 188 bandas espectrales en el rango entre los 400 y los 2500 nm y un total de 50MB de tamaño. Los minerales presentes en esta imagen se conocen muy bien y las firmas espectrales de referencia están disponibles en una librería (*United States Geological Survey (USGS) library*⁵) que es usada en este trabajo para una evaluación de la precisión de los algoritmos.

Dos tipos de experimentos se llevan a cabo en este trabajo para evaluar la calidad de los píxeles candidatos extaridos y el rendimiento de la implementación en paralelo para GPUS del algoritmo SSEE. Primero, se analizarán la calidad de los candidatos seleccionados por distintos métodos (no solo

⁴<http://aviris.jpl.nasa.gov>

⁵<http://speclab.cr.usgs.gov>

TABLA II
 ÁNGULO ESPECTRAL (EN GRADOS) ENTRE EL CONJUNTO DE PÍXELES OBTENIDOS POR LOS DIFERENTES MÉTODOS DE
 PREPROCESADO ESPACIAL (SSEE, SPP Y SSPP) Y EL CONJUNTO DE FIRMAS DE REFERENCIA SELECCIONADO DE LA LIBRERÍA
 USGS PARA LA IMAGEN AVIRIS CÚPRITE. LOS RESULTADOS MOSTRADOS CORRESPONDEN A LOS EMPAREJAMIENTOS MÍNIMOS DE
 ÁNGULO ESPECTRAL. LOS MEJORES RESULTADOS (ÁNGULOS BAJOS) SE HAN RESULTADO EN NEGRITA.

Ventana	SSEE-SVD				SSEE-PCA				SPP				SSPP
	35x35	50x50	70x70	175x175	35x35	50x50	70x70	175x175	3x3	5x5	7x7	9x9	
Candidatos	2416	1795	1310	403	204	140	77	28	122500	122500	122500	122500	3653
<i>Alunite GDS84 Na03</i>	7,467	7,988	7,472	8,761	5,735	5,965	5,500	6,444	5,819	5,942	6,041	6,041	6,804
<i>Alunite GDS83 Na63</i>	7,662	7,678	8,207	9,405	6,886	6,663	8,227	9,203	6,939	7,015	7,070	7,070	9,518
<i>Alunite GDS82 Na82</i>	5,087	5,075	5,454	6,861	4,669	4,098	5,014	6,545	4,224	4,285	4,365	4,365	6,905
<i>Alunite AL706 Na</i>	5,720	5,847	6,666	7,842	4,268	5,228	6,031	7,392	4,836	4,877	4,911	4,911	7,270
<i>Alunite HS295.3B</i>	15,044	14,865	15,465	17,049	11,398	11,202	11,084	21,917	12,196	12,660	12,834	12,834	11,002
<i>Alunite SUSTA-20</i>	6,153	6,203	6,736	7,912	5,427	5,359	6,202	7,643	5,424	5,526	5,561	5,561	7,753
<i>Buddingtonite GDS85 D-206</i>	3,793	3,932	3,904	4,750	3,924	3,918	4,045	4,767	3,781	3,778	3,777	3,777	3,863
<i>Calcite WS272</i>	7,258	7,446	8,413	9,549	5,053	7,060	8,314	8,912	5,323	5,381	5,432	5,432	8,772
<i>Calcite HS48.3B</i>	7,785	7,983	8,970	10,090	5,503	7,665	9,074	9,569	5,678	5,753	5,827	5,827	8,900
<i>Chalcodan CU91-6A</i>	4,140	4,415	4,808	6,271	2,918	4,644	4,869	5,639	2,959	2,988	3,021	3,021	5,569
<i>Chlorite HSI79.3B</i>	15,896	16,562	17,964	22,946	15,547	15,192	20,597	24,182	14,805	14,951	15,134	15,134	14,126
<i>Chlorite SMR-13.a 104-150</i>	20,841	21,470	22,762	27,658	20,653	20,018	27,473	28,595	19,755	20,029	20,126	20,126	19,041
<i>Dicksite NMNH106242</i>	7,221	7,277	7,804	8,821	6,563	6,670	7,420	8,616	6,659	6,689	6,787	6,787	7,804
<i>Halloystite NMNH106236</i>	12,106	11,733	12,097	13,595	9,376	9,548	9,408	19,196	10,026	10,197	10,360	10,360	10,816
<i>Jarosite GDS99 K-y 200C</i>	8,589	8,607	8,796	10,992	8,959	8,797	9,806	10,405	8,357	8,380	8,392	8,392	8,245
<i>Kaolinite KGa-1 (wryl)</i>	5,299	5,446	6,230	7,455	3,755	5,066	5,736	6,949	4,192	4,277	4,308	4,308	6,639
<i>Kaolinite KGa-2 (wryl)</i>	4,290	4,562	5,307	6,483	3,052	4,440	4,950	6,079	3,335	3,346	3,363	3,363	5,461
<i>Kaolin/Smect KLF506 95%K</i>	3,821	3,619	4,200	6,701	4,137	5,363	5,509	5,192	2,637	2,721	2,763	2,763	3,080
<i>Montmorillonite SWy-1</i>	5,941	6,115	6,995	8,041	4,234	6,029	6,485	7,463	4,482	4,601	4,655	4,655	7,253
<i>Montmorillonite SA-1</i>	8,733	8,688	9,045	10,164	8,348	7,867	9,251	14,627	8,114	8,237	8,178	8,178	9,911
<i>Muscovite GDS107</i>	5,214	5,084	5,782	8,258	5,571	6,314	6,623	7,026	3,935	4,024	4,076	4,076	4,047
<i>Muscovite HSI146.3B</i>	8,206	7,984	8,919	12,373	8,073	10,066	10,412	11,964	7,867	7,901	7,946	7,946	7,423
<i>Muscovite HS24.3</i>	6,218	5,868	6,749	10,807	6,058	7,006	8,406	10,273	5,788	5,818	5,830	5,830	5,354
<i>Nontronite GDS41</i>	14,434	14,568	16,429	21,561	13,621	13,372	18,601	22,426	13,324	13,502	13,622	13,622	12,093
<i>Pyrophyllite PYSIA fine g</i>	6,685	6,871	7,781	8,865	4,773	6,640	7,315	8,294	4,872	4,942	5,021	5,021	7,987
Promedio	8,148	8,199	8,918	10,928	7,140	7,768	9,054	11,173	7,173	7,113	7,176	7,176	8,268
S(%)	89,014	89,267	89,465	79,836	91,917	89,943	89,969	81,989	—	—	—	—	88,509

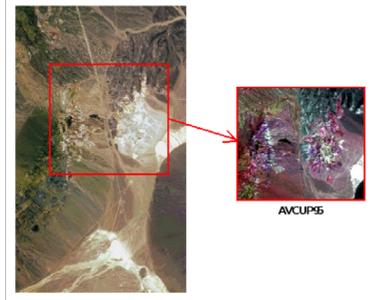


Fig. 7. Composición en falso color de la imagen hiperespectral recogida por el sensor AVIRIS sobre la región de Cuprite en Nevada.

las implementaciones del SSEE, sino también por parte del SPP y el SSPP) en comparación a un conjunto de firmas espectrales de la librería USGS, utilizando el ángulo espectral (SAD) como métrica cuantitativa. Se recuerda que el algoritmo SSEE puede ser visto como un preprocesado espacial, muy

parecido al SPP y al SSPP. En consecuencia, una comparación entre estos métodos es realizada para medir la calidad de los candidatos seleccionados por cada uno de ellos. Con este propósito se han seleccionado 25 firmas de minerales de la librería USGS, que se muestran en la Figura 8. A continuación, se analizará el rendimiento de las versiones paralelas del algoritmo utilizando dos arquitecturas diferentes.

A. Calidad de los píxeles candidatos

Par evaluar la calidad de los píxeles candidatos seleccionados por las implementaciones paralelas en relación al conjunto de firmas seleccionadas de la librería USGS, se consideraron dos algoritmos de emparejamiento distintos [9]. El primero de ellos, obtiene el promedio de los mejores emparejamientos entre los dos conjuntos reordenados aleatoriamente en cada iteración de un total de 100, que se pueden ver en la Tabla I. Esta tabla incluye los resultados tanto de las dos versiones del algoritmo SSEE como de los algoritmos SPP y SSPP. El segundo algoritmo empareja las firmas espectrales cogiendo el valor mínimo de ángulo espectral entre todas las posibles combinaciones (ver la Tabla II). En ambos casos las 25 firmas de minerales seleccionadas son usadas como conjunto de referencia para ambos algoritmos. Con ánimo de es-

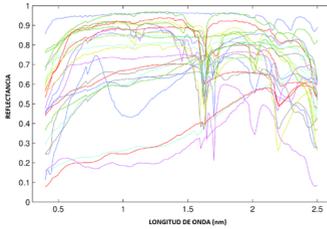


Fig. 8. Conjunto de 25 firmas espectrales seleccionadas para los experimentos de la librería USGS: Alunite GDS84 Na03, Alunite GDS83 Na63, Alunite GDS82 Na82, Alunite AL706 Na, Alunite HS295.3B, Alunite SUSTDA-20, Buddingtonite GDS85 D-206, Calcite WS272, Calcite HS48.3B, Chalcedony CU91-6A, Chlorite HS179.3B, Chlorite SMR-13.a 104-150, Dickite NMNH106242, Halloysite NMNH106236, Jarosite GDS89 K-y 200C, Kaolinite KCa-1 (weyl), Kaolinite KCa-2 (pywl), Kaolin/Smeect KLF506 95%K, Montmorillonite SWy-1, Montmorillonite SAz-1, Muscovite GDS107, Muscovite HS146.3B, Muscovite HS24.3, Nontronite GDS41, Pyrophyllite PYS1A fine g.

tabecer una comparación justa se requiere comparar un número de de candidatos que coincidan con las firmas de referencia. Aunque surge el problema de como decidir si un píxel candidato está bien emparejado con su respectiva firma de referencia. En nuestros experimentos se considera que una firma de referencia es encontrada cuando su valor de ángulo espectral esta por debajo de 10° (donde el peor caso entre dos firmas en un valor de 90°). En base a lo comentado anteriormente, se utiliza la siguiente métrica para mostrar el porcentaje de acierto S :

$$S(\%) = \frac{m}{P} + \frac{[1 - \frac{n}{N}]}{2} * 100,$$

donde n es el número de candidatos extraídos por el método, N es el total de píxeles de la imagen, m es el número de firmas satisfactoriamente emparejadas y P es el número de firmas de referencia. Las Tablas I y II incluyen los resultados de esta métrica. En el caso del algoritmo SPP, el número de píxeles candidatos es igual al tamaño de la imagen ya que este algoritmo no realiza ninguna reducción en el número de candidatos, pero tanto el SSPP como las versiones del SSEE si reducen esta cantidad.

Como se muestra en las Tablas I y II, varias de las 25 firmas de referencia no están bien emparejadas por algunos de los métodos, como por ejemplo *Alunite HS295.3B* o *Chlorite SMR-13.a 104-150*. Esto se debe a la dificultad de garantizar que esas mismas firmas se encuentren presentes en la imagen original, teniendo en cuenta que algunos de los minerales que están representados en la imagen pueden no estar en el conjunto de 25 firmas de referencia. Muchos de los algoritmos son capaces de proporcionar una serie de píxeles candidatos que pueden ser muy similares, espectralmente hablando, a las firmas de referencia de la librería USGS, incluso considerando que el umbral

superior es de 10° . En general podemos concluir en base a las Tablas I y II que el algoritmo SSEE proporciona buenos resultados en la mayoría de los casos, siendo levemente mejores en la versión que usa la PCA. Considerando todos los algoritmos, los mejores resultados son obtenidos por el algoritmo SPP pero se ha de tener en cuenta que el SSEE cuando usa la PCA obtiene un conjunto mucho más reducido de píxeles candidatos, lo cual tendrá un fuerte impacto en el rendimiento computacional del que se habla en el siguiente apartado.

B. Computational Performance

Antes de presentar los resultados de rendimiento obtenidos, se quiere señalar que todas las implementaciones GPU de los algoritmos SSEE, SPP y SSPP mencionadas en esta sección obtienen exactamente los mismos resultados tanto en serie como en paralelo. Con intención de evaluar el rendimiento computacional se han realizado los experimentos en dos arquitecturas diferentes:

- Arquitectura 1. Un ordenador de sobremesa (Intel core i7 920 CPU a 2.67 GHz y 6 GB de RAM) con una tarjeta Nvidia GTX 580, con 512 núcleos operando a 1.54 GHz (de ahora en adelante Arquitectura 1).
- Arquitectura 2. Un clúster⁶, con nodos Nvidia TESLA S2070 (2 M2075 por nodo) y un núcleo Intel Xeon CPU E5645 a 240GHz y 24GB DDR3, divididos en 12 módulos de 2GB cada uno (de ahora en adelante Arquitectura 2).

En ambos casos las versiones serie de los algoritmos fueron ejecutadas en unco de los núcleos disponibles, y los tiempos paralelos han sido medidos en función de los recursos de cada arquitectura. La aceleración media ha sido calculada entre cada par CPU/GPU tras 10 ejecuciones.

La Figura 9 muestra el rendimientos de las implementaciones paralelas del algoritmo SSEE considerando las dos arquitecturas propuestas (para los resultados de los algoritmos SPP y SSPP, nos remitiremos a los trabajos [13], [14] respectivamente donde se describen dichos métodos más en detalle). De la Figura 9, podemos concluir que la versión del algoritmo SSEE que utiliza la PCA en vez de la SVD exhibe un coste computacional más bajo, mayormente debido a la reducción del volumen de datos sumado a un rango de paralelización mayor. La versión con la SVD está limitada en este caso debido a que su paralelización en GPUs es rentable únicamente con volúmenes de datos muy grandes [18]. Si se tienen en cuenta los resultados de precisión de la subsección IV-A, podemos decir que la versión PCA mejora claramente los resultados obtenidos por la versión SVD.

V. CONCLUSIONES Y LÍNEAS FUTURAS

En este trabajo se ha presentado una nueva implementación del algoritmo *spatial-spectral endmem-*

⁶<http://www.ceta-ciemat.es/>

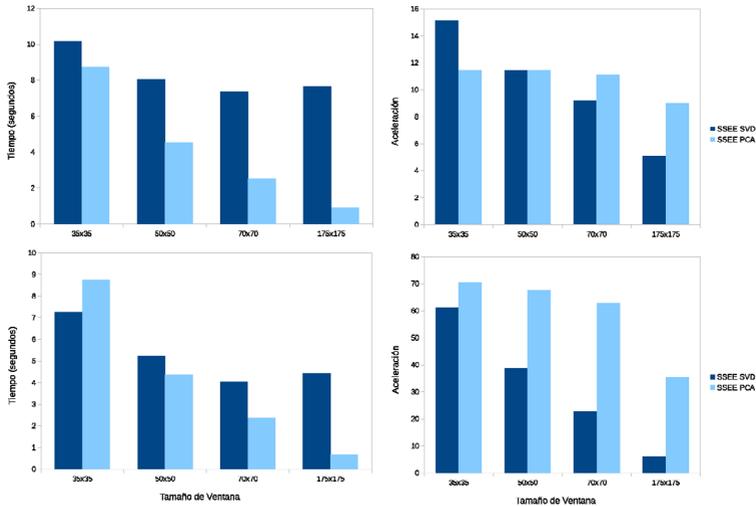


Fig. 9. Tiempos de ejecución (en segundos) y rendimiento (aceleración) para las implementaciones paralelas del algoritmo SSEE, considerando distintos tamaños de ventana, tras procesar la imagen AVIRIS Cuprite en la Arquitectura 1 (fila superior) y la Arquitectura 2 (fila inferior). Los valores obtenidos son los promedios tras 10 ejecuciones.

ber extraction (SSEE), el cual se puede considerar como un preprocesado espacial capaz de seleccionar píxeles candidatos para una posterior extracción de endmembers. Diferentes implementaciones han sido consideradas, basándonos en el método de cálculo de los autovectores. La implementación GPU propuesta ha sido comparada con otros métodos de preprocesado espacial conocidos, tales como, el algoritmo SPP y el SSPP. Los resultados obtenidos sugieren que la implementación GPU del algoritmo SSEE proporciona unos resultados competitivos ante los algoritmos mencionados anteriormente, y permiten una reducción de los tiempos computacionales a la vez que se mantiene una alta calidad del conjunto de píxeles seleccionados como candidatos a ser endmembers. Para futuros trabajos nos centraremos en la inclusión de nuevos algoritmos de preprocesado espacial en la comparación y el desarrollo de nuevas estrategias de paralelización en el preprocesado usando GPUs y otra arquitecturas de computación de alto rendimiento tales como las *field programmable gate-arrays* (FPGAs).

VI. AGRADECIMIENTOS

Este trabajo ha sido subvencionado por la Junta de Extremadura (a través del decreto 297/2014, ayudas para la realización de actividades de investigación y desarrollo tecnológico, de divulgación y de transferencia de conocimiento por los Grupos de Investi-

gación de Extremadura, Ref. GR15005). Además, el presente trabajo ha sido llevado a cabo haciendo uso de la infraestructura de computación facilitada por el Centro Extremeño de Tecnologías Avanzadas (CETA-CIEMAT), financiado por el Fondo Europeo de Desarrollo Regional (FEDER). El CETA-CIEMAT pertenece al CIEMAT y al Gobierno de España.

REFERENCIAS

- [1] N. Keshava and J. F. Mustard, "Spectral unmixing," *IEEE Signal Process. Mag.*, vol. 19, no. 1, pp. 44–57, 2002.
- [2] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical and sparse regression-based approaches," *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, vol. 5, no. 2, pp. 354–379, 2012.
- [3] A. Plaza, P. Martínez, R. Pérez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 3, pp. 650–663, 2004.
- [4] A. Plaza, P. Martínez, R. Pérez, and J. Plaza, "Spatial/spectral endmember extraction by multidimensional morphological operations," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 40, no. 9, pp. 2025–2041, 2002.
- [5] D. M. Rogge, B. Rivard, J. Zhang, A. Sanchez, J. Harris, and J. Feng, "Integration of spatial-spectral information for the improved extraction of endmembers," *Remote Sens. Environ.*, vol. 110, no. 3, pp. 287–303, 2007.
- [6] S. Lopez, J. F. Moure, A. Plaza, G. M. Callico, J. F. Lopez, and R. Sarmiento, "A new preprocessing technique for fast hyperspectral endmember extraction,"

- IEEE Geoscience and Remote Sensing Letters*, vol. 10, no. 5, pp. 1070–1074, September 2013.
- [7] M. Zortea and A. Plaza, "Spatial preprocessing for endmember extraction," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, pp. 2679–2693, 2009.
- [8] G. Martín and A. Plaza, "Region-based spatial preprocessing for endmember extraction and spectral unmixing," *IEEE Geosci. Remote Sens. Lett.*, vol. 8, no. 4, pp. 745–749, 2011.
- [9] G. Martín and A. Plaza, "Spatial-spectral preprocessing prior to endmember identification and unmixing of remotely sensed hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, vol. 5, no. 2, pp. 380–395, 2012.
- [10] J. Plaza, E. M. T. Hendrix, I. García, G. Martín, and A. Plaza, "On endmember identification in hyperspectral images without pure pixels: A comparison of algorithms," *Journal of Mathematical Imaging and Vision*, vol. 42, no. 2-3, pp. 163–175, 2012.
- [11] J. A. R. y X. Jia, "Remote Sensing Digital Image Analysis," in *Springer-Verlag*, 1999.
- [12] J. W. Boardman, F. A. Kruse, and R. O. Green, "Mapping Target Signatures Via Partial Unmixing of Aviris Data," *Proc. JPL Airborne Earth Sci. Workshop*, pp. 23–26, 1995.
- [13] J. Delgado, G. Martín, J. Plaza, L. I. Jiménez, and A. Plaza, "Fast spatial preprocessing for spectral unmixing of hyperspectral data on graphics processing units," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 2, pp. 952–961, 2016.
- [14] L. I. Jiménez, G. Martín, S. Sánchez, J. Plaza, and A. Plaza, "GPU implementation of spatial-spectral preprocessing for hyperspectral unmixing," *IEEE Geoscience and Remote Sensing Letters*, submitted, 2016.
- [15] J. Setoain, M. Prieto, C. Tenllado, A. Plaza, and F. Tirado, "Parallel morphological endmember extraction using commodity graphics hardware," *IEEE Geoscience and Remote Sensing Letters*, vol. 43, no. 3, pp. 441–445, 2007.
- [16] J. C. Harsanyi and C.-I. Chang, "Hyperspectral image classification and dimensionality reduction: An orthogonal subspace projection approach," *IEEE Trans. Geosci. Remote Sens.*, vol. 32, pp. 779–785, 1994.
- [17] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.
- [18] S. Lahabar and P. J. Narayanan, "Singular value decomposition on GPU using CUDA," *Proceedings of the IEEE Parallel and Distributed Processing Symposium*, pp. 1–10, 2009.

Clasificación de imágenes de teledetección mediante ELM kernel y perfiles morfológicos en GPU

Alberto S. Gare¹, Dora B. Heras¹, and Francisco Argüello²

Resumen— Hoy en día el uso de sensores hiperespectrales se ha extendido a una gran variedad de aplicaciones tales como la clasificación de imágenes de teledetección. Recientemente se ha presentado un esquema de clasificación espectral-espacial (ELM-EMP) basado en *Extreme Machine Learning* (ELM) y Perfiles Morfológicos Extendidos (EMP) obtenidos utilizando Análisis en Componentes Principales (PCA) y operaciones morfológicas. En este trabajo se han introducido varias mejoras para incrementar la precisión de la clasificación del esquema original (ELM-EMP). En particular, se presenta un esquema que utiliza un clasificador ELM basado en kernels (KELM-EMP) y se aplica una regularización espacial. Además, se ha realizado una implementación eficiente sobre Unidades de Procesamiento Gráfico (GPUs) de estos esquemas. En cuanto a esta proyección en GPU, se han aplicado diferentes técnicas como pueden ser el uso de librerías CUDA optimizadas y la ejecución de bloques asíncronos. Como resultado, la precisión obtenida por los dos esquemas (ELM-EMP-S y KELM-EMP-S) es mejor que para el esquema ELM-EMP original y el tiempo de ejecución se ha reducido significativamente.

Palabras clave— Teledetección; clasificación, datos hiperespectrales, *Extreme Machine Learning* (ELM), Análisis de Componentes Principales (PCA), Perfiles Morfológicos Extendidos (EMP).

I. INTRODUCCIÓN

LOS avances en la tecnología de sensores de imagen han hecho posible ampliar el rango del espectro electromagnético que puede ser capturado, pasando de unas pocas bandas en imágenes multi-espectrales a cientos de bandas en imágenes hiperespectrales [1]. Este amplio rango proporciona más información que puede ser utilizada para mejorar las tareas de reconocimiento y clasificación de materiales. Debido a la alta dimensionalidad de las imágenes hiperespectrales, se necesitan técnicas específicas para aprovechar toda esa información [2]. En el campo de las redes neuronales usamos el término *Extreme Machine Learning* (ELM) para describir un tipo de *Single-hidden Layer Feedforward Neural Network* (SLFN) con pesos aleatorios [3]. Se ha visto en [4], [5] que una SLFN (con N nodos en la capa oculta) donde los pesos y los sesgos se eligen aleatoriamente puede aprender exactamente N observaciones distintas [6]. El mismo autor también propone el uso de la pseudo-inversa en el algoritmo ELM para evitar la baja inferencia de los algoritmos de entre-

namiento con retropropagación. En el ELM, la asignación aleatoria de los pesos entre la capa de entrada y la capa oculta produce una amplia variación en la precisión de la clasificación en diferentes pruebas usando el mismo número de nodos ocultos. Para evitar esto, [7] propone, para el algoritmo ELM, el uso de funciones *kernel* en sustitución de la capa oculta.

Como valor añadido, el algoritmo ELM es muy adecuado para ser implementado en Unidades de Procesamiento Gráfico (GPUs) de uso común y otras arquitecturas paralelas debido a que la mayoría de sus operaciones están relacionadas con operaciones matriciales que pueden ser calculadas en bloques independientes, es decir, sin dependencia de datos entre ellos. Heeswijk et al. [8] desarrolló una implementación en GPU de parte de un sistema de clasificación basado en la ejecución de un centenar de instancias de ELM. Más tarde se publicó una implementación eficiente, completamente en GPU, del clasificador hiperespectral ELM [9]. Sin embargo, no se han publicado implementaciones en GPU para la versión *kernel* del ELM.

Las precisión de los resultados proporcionados por una clasificación píxel a píxel puede ser mejorada aportando información espacial al clasificador [10], [2], [11], [12]. Esto significa que la decisión de asignar un píxel a una clase determinada se basa tanto en la característica espectral, que es el valor del píxel, como en cierta información extraída del vecindario del píxel, que se puede considerar información espacial. Entre los métodos espaciales podemos incluir los basados en segmentación, como la transformada *watershed* [13], [14], [9] y la segmentación basada en autómatas celulares (ECAS-II) [15], los basados en técnicas de agrupamiento por particiones [16], los basados en los bosques de expansión mínima [2] o los basados en filtrado local [17].

La información espacial también puede ser extraída de los datos hiperespectrales mediante herramientas morfológicas. Los operadores morfológicos más usados son la apertura y el cierre, que están basados en las operaciones fundamentales de erosión y dilatación. A partir de esas operaciones básicas se pueden construir los llamados Perfiles Morfológicos (MP) [18], [19], [20], [12]. Un MP contiene información de las estructuras de la imagen a diferentes tamaños de resolución. En la teledetección, los MPs se calculan normalmente en base a los datos obtenidos del Análisis en Componentes Principales (PCA) realizado sobre los datos hiperespectrales [21], [2], [22]. Si se conservan varias componentes prin-

¹Centro Singular de Investigación en Tecnoloxías da Información (CITIUS), Universidade de Santiago de Compostela, Spain, {jorge.suarez.garea, dora.blanco}@usc.es.

²Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Spain, francisco.arguello@usc.es.

cipales, los MPs obtenidos para cada uno de esas componentes se pueden utilizar todos juntos en un Perfil Morfológico Extendido (EMP) [23], [24]. Esto se puede generalizar más con el fin de modular la información espacial de forma más precisa. Por ejemplo, se puede crear un Perfil de Atributos (AP) morfológicos del mismo modo que un MP [25]. En [11] se crea un Perfil de Atributos Morfológicos Extendido (EMAP) usando filtros de atributos morfológicos. En [18] y [21] se presentan esquemas de clasificación espectral-espacial para imágenes de teledetección basados en SVM e introduciendo información proporcionada por un EMP. En el caso de [26] se utilizan características Gabor, en lugar de EMP, sobre las componentes generadas por el PCA para, posteriormente, concatenar el resultado con la imagen original y realizar la clasificación mediante el ELM *kernel*. Este último artículo también presenta un clasificador basado en KELM con Predicciones Multihipótesis (MH).

En [27] se presenta un esquema de clasificación espectral-espacial para imágenes hiperespectrales de teledetección basado en ELM y que además integra información proporcionada por un EMP. El perfil se crea a partir de las componentes generadas por el PCA. El esquema espectral-espacial propuesto permite asignar diferentes pesos a las características espectrales y a las espaciales.

En este artículo se han abordado principalmente tres tareas. La primera tarea ha sido el desarrollo de un esquema similar al publicado en [27], basado en ELM con funciones *kernel*. Estas funciones *kernel* sustituyen a la generación de pesos aleatorios en el esquema ELM original. La segunda ha sido la mejora, en términos de precisión, del esquema presentado en [27]. La tercera tarea ha sido la implementación, sobre GPU, de los esquemas descritos previamente para reducir el tiempo de ejecución. El resto del artículo se organiza como sigue: La sección 2 describe los algoritmos utilizados. En la sección 3 se presenta la implementación del esquema de clasificación sobre GPU. La evaluación se realiza en la sección 4. Y, finalmente, la sección 5 presenta las conclusiones.

II. CLASIFICACIÓN ESPECTRAL-ESPACIAL BASADA EN ELM

En esta sección se explican las diferentes etapas del esquema de clasificación que llamaremos ELM-EMP, como en [27] (Ver Fig. 1). Este proceso comienza con la extracción de la información espacial. El primer paso es reducir la dimensionalidad de la imagen hiperespectral usando PCA. El segundo paso es la construcción del EMP mediante operaciones morfológicas. Tanto las contribuciones espectrales como las espaciales se ajustan usando un método de combinación de características llamado concatenación ponderada. Esto permite asignar los pesos k_w y k_s a los datos espectrales y espaciales respectivamente. Finalmente, utilizamos ELM para la clasificación.

A. Clasificación basada en KELM

ELM fue originalmente desarrollado como una técnica de entrenamiento para un tipo de SLFN con pesos aleatorios [28], [6]. En [7] se propuso el uso de una función *kernel* en lugar del uso de pesos aleatorios. El objetivo era evitar la gran variación, en términos de precisión de la clasificación, producida en diferentes entrenamientos y debida a la aleatoriedad de los pesos. Además, [7] sugirió sumar un valor positivo $\frac{1}{C}$ (donde C es definido por el usuario) para calcular los pesos de β como:

$$\beta = \mathbf{H}^T \left(\frac{\mathbf{I}}{C} + \mathbf{H}\mathbf{H}^T \right)^{-1} \mathbf{T}, \quad (1)$$

donde \mathbf{H} es la matriz de salida de la capa oculta y \mathbf{T} es la matriz objetivo para el conjunto de datos de entrenamiento. Con este valor positivo se tienden a obtener soluciones más estables, así como una mejora en el rendimiento. Por lo tanto, para $\mathbf{x} \in \mathbb{R}^d$, la función de salida del ELM es:

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\beta = \mathbf{h}(\mathbf{x})\mathbf{H}^T \left(\frac{\mathbf{I}}{C} + \mathbf{H}\mathbf{H}^T \right)^{-1} \mathbf{T}, \quad (2)$$

donde $\mathbf{h}(\mathbf{x})$ es un mapeo de características. La matriz del *kernel* para ELM, Ω , puede ser representada como:

$$\begin{aligned} \Omega_{\text{ELM}} &= \mathbf{H}\mathbf{H}^T : \Omega_{ELM_{i,j}} \\ &= [\mathbf{h}(\mathbf{x}_i) \times \mathbf{h}(\mathbf{x}_j)] = [K(\mathbf{x}_i, \mathbf{x}_j)], \end{aligned} \quad (3)$$

donde \mathbf{x}_i y \mathbf{x}_j son muestras de entrenamiento, $i = 1, \dots, N$, $j = 1, \dots, N$, y $K(\mathbf{x}_i, \mathbf{x}_j)$ es la función *kernel*. Finalmente, la función de salida se puede escribir como:

$$f(\mathbf{x}) = \underbrace{\begin{bmatrix} K(\mathbf{x}, \mathbf{x}_1) \\ \vdots \\ K(\mathbf{x}, \mathbf{x}_N) \end{bmatrix}}_{\text{Kernel}_1}^T \left(\frac{\mathbf{I}}{C} + \underbrace{\Omega_{\text{ELM}}}_{\text{Kernel}_2} \right)^{-1} \mathbf{T}. \quad (4)$$

Así, podemos resumir el ELM con *kernel* como:

Algoritmo ELM con *kernel*: Dado un conjunto de entrenamiento $\mathbf{J} = \{\mathbf{x}_i | \mathbf{x}_i \in \mathbb{R}^d, i = 1, \dots, N\}$, una matriz de entrenamiento \mathbf{T} , una función *kernel* $K(\mathbf{u}, \mathbf{v})$ (e.g., $K(\mathbf{u}, \mathbf{v}) = \exp(-\gamma\|\mathbf{u} - \mathbf{v}\|^2)$) y los parámetros definidos por el usuario C y γ ,

1. Calcular $\left(\frac{\mathbf{I}}{C} + \Omega_{\text{ELM}} \right)^{-1} \mathbf{T}$, con $\Omega_{\text{ELM}} = [K(\mathbf{x}_i, \mathbf{x}_j)]$, $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{J}$ y $K(\mathbf{u}, \mathbf{v}) = \exp(-\gamma\|\mathbf{u} - \mathbf{v}\|^2)$.
2. Calcular $\begin{bmatrix} K(\mathbf{x}, \mathbf{x}_1) \\ \vdots \\ K(\mathbf{x}, \mathbf{x}_N) \end{bmatrix}^T$ donde $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbf{J}$ y \mathbf{x} hace referencia a todos los puntos del conjunto de datos de test.
3. Calcular la función de salida del ELM como en la Ecuación (4).

En el caso de nuestra imagen hiperespectral, cada muestra de entrenamiento representa un píxel de la

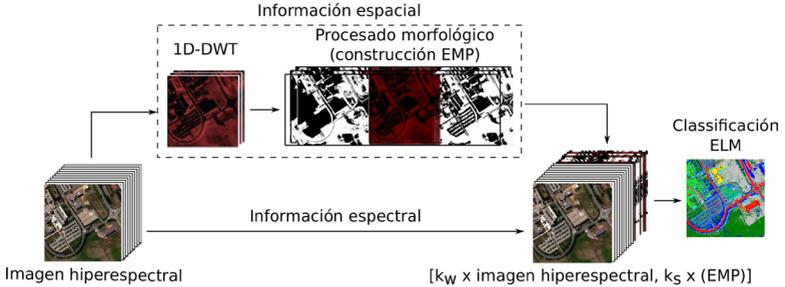


Fig. 1. Esquema de clasificación espectral-espacial basado en ELM y mapeo de características compuesto (ELM-EMP).

imagen seleccionado de forma aleatoria, que a su vez es un componente espectral y espacial ponderado del píxel. Los componentes espaciales proporcionados por el EMP son multiplicados por el factor k_s , mientras que los componentes de la imagen original son multiplicados por la constante k_w . La salida que se obtiene después de la fase de clasificación es la predicción de la clase a la que pertenece cada uno de los píxeles de la imagen.

B. Extracción de características

La principal función del PCA es reducir la dimensionalidad del conjunto de datos, constituido por un gran número de variables interrelacionadas, mientras se retiene, tanto como sea posible, la variación presente en el conjunto de datos.

Matemáticamente existen varios métodos para calcular el PCA. Se puede realizar usando *Single Value Decomposition* (SVD) sobre el conjunto de datos o mediante *Eigenvalue Decomposition* (EVD) sobre la matriz de covarianza del conjunto de datos, siendo este último el método utilizado en este documento. Ambos métodos necesitan que el conjunto de datos esté centrado [29].

Vamos a asumir que la matriz de datos \mathbf{X} está centrada, es decir, a cada elemento de una columna de \mathbf{X} se le ha restado la media aritmética de dicha columna, quedando ahora dicha media igual a cero. Por lo tanto, la matriz de covarianza $\mathbf{A} = \mathbf{X}^T \mathbf{X} / n$, donde n es el número de características, puede ser diagonalizada:

$$\mathbf{A} = \mathbf{USV}^T, \quad (5)$$

donde los elementos de la diagonal de \mathbf{S} son los valores singulares de \mathbf{A} y las primeras columnas de \mathbf{U} y \mathbf{V} son los vectores singulares derecho e izquierdo de \mathbf{A} respectivamente. La proyección de los datos en el eje principal se conoce como Componentes Principales (PCs).

C. EMP

Las transformaciones morfológicas fueron propuestas para utilizar la información espacial en la clasificación de teledetección [30], [21], [23]. El MP fue

introducido en [21] y extendido a imágenes multidimensionales en [23] para extraer información sobre el contraste y el tamaño de las estructuras presentes en la imagen. El MP de orden n de la imagen I se puede expresar como:

$$\text{MP}^{(n)}(I) = \{\gamma_r^{(n)}(I), \dots, \gamma_c^{(n)}(I), \phi_r^{(n)}(I), \dots, \phi_c^{(n)}(I)\}, \quad (6)$$

siendo $\gamma_r^{(i)}$ y $\phi_r^{(i)}$ los operadores de apertura y cierre por reconstrucción respectivamente, con un elemento estructural i desde 1 a n , cuyo tamaño incrementa normalmente en pasos de 1 o 2.

Cuando se usa la aproximación de MP sobre los datos hiperespectrales, se usan los m PCs más significativos como imagen base. El resultado es un EMP,

$$\text{EMP}_m^{(n)}(I) = \{(MP)_1^{(n)}(I), \dots, (MP)_m^{(n)}(I)\}, \quad (7)$$

con $m(2n + 1)$ componentes.

Una vez que se ha extraído de la imagen la información espacial, esta debe ser integrada en el clasificador.

III. CLASIFICACIÓN ESPECTRAL-ESPACIAL DE UNA IMAGEN HIPERESPECTRAL SOBRE GPU

En esta sección introduciremos algunos de los principios de la programación *Compute Unified Device Architecture* (CUDA), así como la implementación de los algoritmos propuestos en la sección II. Los *kernels* ejecutados en GPU se encuentran entre los símbolos $\langle \rangle$. Los pseudocódigos también incluyen los acrónimos GM y SM para indicar que los *kernels* usan memoria global o memoria compartida respectivamente.

A. Fundamentos de programación CUDA

CUDA permite a las GPUs de Nvidia ejecutar programas a través de la invocación de funciones paralelas llamadas *kernels* [31]. Los *kernels* son ejecutados por hilos que a su vez se organizan en bloques. Los bloques se disponen en una rejilla donde se mapean a una jerarquía de procesadores CUDA en la GPU. Los

hilos tienen acceso a múltiples espacios de memoria, como pueden ser la memoria local y los registros. Cada bloque de hilos tiene un espacio de memoria compartida que es visible únicamente a los hilos del mismo bloque y cuyo tiempo de vida es igual al del bloque. Por último, todos los hilos pueden acceder al mismo espacio de memoria global.

El tiempo de vida de la memoria compartida dificulta el poder compartir datos entre bloques de hilos, lo que obliga a usar la memoria global que es más lenta. La arquitectura Kepler [31] incluye una jerarquía con dos niveles de caché. En este trabajo se han seguido diferentes estrategias para optimizar el rendimiento:

1. **Maximizar la ejecución paralela.** Organizando los algoritmos en bloques computacionalmente independientes.
2. **Mejorar la eficiencia en el uso de la jerarquía de memoria.** Para realizar el máximo número de cálculos con los datos ya almacenados en la memoria compartida.
3. **Reducir el número de sincronizaciones globales mediante el cómputo con bloques asíncronos.** En el cálculo del EMP, cada bloque se actualiza varias veces a través de una sincronización local antes de realizar una sincronización global.
4. **Añadir un borde a las regiones de datos.** Dado que en la fase de construcción del EMP cada píxel requiere datos de sus vecinos, cada región de datos se amplía con una frontera con el fin de minimizar las dependencias entre los bloques.
5. **Superposición de operaciones en la CPU y la GPU.** A veces es posible solapar la ejecución de operaciones independientes en la CPU y la GPU, reduciendo el tiempo de ejecución. En particular, en el cálculo del ELM, la generación de los pesos aleatorios para los datos de entrada y las neuronas ocultas se solapa con la generación de la matriz \mathbf{X}_{test} .
6. **Explotar las librerías disponibles.** Dado que la mayoría de las operaciones de los algoritmos utilizados en este trabajo son operaciones con matrices, se han utilizado diferentes bibliotecas CUDA optimizadas para álgebra lineal y procesamiento de imágenes. En particular, MAGMA [32], que es una librería de álgebra lineal para arquitecturas heterogéneas usada para calcular la matriz \mathbf{X}_{test} y la pseudoinversa de la matriz \mathbf{H} , y CULA [33], que es un conjunto de librerías algebraicas usadas para calcular los EVD en el algoritmo PCA. Por último, CUBLAS [34], que es una versión en GPU de la librería estándar *Basic Linear Algebra Subprograms* (BLAS) y que se utilizó para obtener la matriz de correlación en el cálculo de la PCA y para calcular la matriz de pesos en la fase de entrenamiento, tanto para el algoritmo ELM como para el KEML.

Algoritmo 1 Algoritmo KEML en GPU

Entrada: El conjunto de datos de la imagen hiperespectral \mathbf{X} , el conjunto de etiquetas \mathbf{T} y los parámetros C y γ definidos por el usuario

	▷ Fase de preprocesado	
1:	<Normalización del conjunto de datos>	▷ SM + GM
2:	Selección de los puntos de entrenamiento ($\mathbf{X}_{\text{train}}$)	▷ SM + GM
3:	<Procesado de la matriz de objetivos ($\mathbf{T}_{\text{train}}$)>	▷ GM
	▷ Cálculo de la función <i>kernel</i>	
4:	<Calcular <i>Kernel</i> _2 (Eq. 4)>	▷ GM
5:	<Calcular $\alpha = (1/C + \text{Kernel}_2)^{-1} \mathbf{T}_{\text{train}}$ >	▷ GM
6:	<Preparar los datos de test (\mathbf{X}_{test})>	▷ GM
7:	<Calcular <i>Kernel</i> _1 (Eq. 4)>	▷ GM
	▷ Clasificación.	
8:	<Calcular α <i>Kernel</i> _1>	▷ GM

B. Clasificación basada en KEML sobre GPU

El algoritmo KEML se compone de tres fases principales: preprocesado, cálculo de las funciones *kernel* y clasificación. El pseudocódigo del Algoritmo 1 nos muestra la implementación en GPU tal y como se explica en la sección II-A.

En nuestro caso, el algoritmo KEML es parte del esquema KEML-EMP, siendo los datos de entrada el resultado de la unión y la normalización del conjunto de datos hiperespectrales y el EMP. En la fase de unión, los datos del perfil obtenidos a través del algoritmo EMP son multiplicados por un peso determinado. Después, en la normalización (línea 1 del pseudocódigo), a todos los puntos del conjunto de datos hiperespectrales se le resta el mínimo de dicho conjunto mientras que en el conjunto de datos del EMP la normalización se realiza por bandas, restando el mínimo de cada banda a los elementos de dicha banda. El *kernel* utilizado para el cálculo del mínimo utiliza memoria compartida y evita el conflicto de bancos, reduciendo el tiempo de ejecución.

Dado que el ELM y el KEML son algoritmos de aprendizaje supervisado, la selección de los píxeles de entrenamiento en el mapa de referencia se hace de forma aleatoria, escalando posteriormente los valores correspondientes del conjunto de datos entre $[0:1]$ y almacenándolos en la matriz $\mathbf{X}_{\text{train}}$ (línea 2 del pseudocódigo). Finalmente se procesa la matriz de entrenamiento, donde cada fila representa una muestra y cada columna una clase, y para la que el valor 1 indica que pertenece a la clase y el valor 0 que no pertenece a la clase (línea 3 del pseudocódigo).

La segunda fase (ejecución de las funciones *kernel*) comienza con la primera función de *kernel*, donde $K(\mathbf{u}, \mathbf{v}) = \exp(-\gamma \|\mathbf{u} - \mathbf{v}\|^2)$ y \mathbf{u}, \mathbf{v} pertenece al conjunto de datos de entrenamiento. En primer lugar calculamos la matriz $\mathbf{X}_{\text{train}} \mathbf{X}_{\text{train}}^T$, que contiene todos los productos entre dos elementos del conjunto de entrenamiento. Con la matriz anterior y el parámetro γ , definido por el usuario, calculamos la matriz *Kernel*_2. El siguiente paso es generar una matriz identidad donde todos sus elementos son divididos por el parámetro C , también definido por el usuario. A continuación sumamos la matriz identidad modificada y la matriz *Kernel*_2. Por último usamos la función de MAGMA llamada *magma_dgesv_gpu* para resolver el sistema de ecuaciones lineales y obtener α , donde:

$$\alpha = (\mathbf{I}/C + \text{Kernel_2})^{-1} \mathbf{T}_{\text{train}}.$$

La segunda fase continua con el cálculo de la otra función *kernel* del mismo modo que la anterior, pero esta vez \mathbf{u} pertenece al conjunto de datos de test y \mathbf{v} al de entrenamiento. Primero escalamos entre [0:1] el conjunto de datos de test y lo almacenamos en la matriz \mathbf{X}_{test} . A continuación calculamos la función *kernel* y creamos una matriz para almacenar $\mathbf{X}_{\text{test}} \mathbf{X}_{\text{train}}$, un vector para almacenar la diagonal de $\mathbf{X}_{\text{test}} \mathbf{X}_{\text{test}}$ y otro vector para almacenar la diagonal de $\mathbf{X}_{\text{train}} \mathbf{X}_{\text{train}}$. Por último calculamos la función *kernel* con todos los elementos de \mathbf{X}_{test} y todos los elementos de $\mathbf{X}_{\text{train}}$.

En la última fase del algoritmo KELM únicamente necesitamos calcular el producto de *Kernel_1* por α para obtener la clasificación final.

C. Extracción de características sobre GPU

Para reducir la dimensionalidad del conjunto de datos se utiliza el algoritmo EVD (EVD-PCA) explicado en la sección II-B.

El algoritmo 2 incluye el pseudocódigo del algoritmo EVD-PCA en GPU. Suponemos que la matriz \mathbf{X} contiene el conjunto de datos y que este se encuentra almacenado en la memoria global de la GPU. Es necesario preprocesar el conjunto de datos, es decir, deben estar centrados. Para ello restamos a cada píxel la suma de todos los píxeles de su banda dividida por el número de píxeles de dicha banda (líneas 1-3 del Algoritmo 2). En este paso, además del uso de memoria compartida para la eficiencia en el acceso a los datos, también se utiliza la función de CUBLAS *cublasSgemv*.

Una vez que la matriz \mathbf{X} contiene los datos centrados, comienza la fase del PCA. En primer lugar se calcula la matriz de correlación de la matriz \mathbf{X} usando la función de CUBLAS *cublasSsyrc* (línea 4). Esta función realiza la operación $\mathbf{X}\mathbf{X}^T$, pero debido a que el resultado es simétrico, *cublasSsyrc* solo devuelve la matriz superior. El siguiente paso consistirá en completar la matriz inferior del cálculo anterior con los datos de la matriz superior (línea 5).

El algoritmo PCA continua con el cálculo de

$$\mathbf{X}\mathbf{X}^T = \mathbf{U}\mathbf{S}\mathbf{V}^T,$$

usando la función de CULA *culaDeviceSgesvd*, donde los elementos de la diagonal de \mathbf{S} son los valores singulares de $\mathbf{X}\mathbf{X}^T$ y las primeras columnas de \mathbf{U} y

\mathbf{V} son los vectores singulares derechos e izquierdos de $\mathbf{X}\mathbf{X}^T$ respectivamente (línea 6). El último paso es la obtención de los PCs mediante la función de CUBLAS *cublasSgemm*, que realiza el producto entre \mathbf{X} y \mathbf{V} (línea 7).

D. EMP sobre GPU

El EMP es el conjunto de MPs creado por medio de operaciones de apertura y cierre por reconstrucción sobre cada una de las bandas obtenidas en la fase del PCA.

La implementación realizada se basa en la propagación de bloques asíncronos [35], donde se realizan múltiples barridos en ambas direcciones al mismo tiempo. La principal ventaja de esta implementación asíncrona en GPU [36], para la reconstrucción morfológica (que consisten en actualizaciones intra e inter-bloque), es que se realizan tantas actualizaciones como sean posibles con los datos disponibles en la memoria compartida antes de realizar una sincronización entre los hilos de los bloques. Los datos actualizados dentro de un bloque de memoria compartida se pueden reutilizar, lo cual es mucho más rápido que las actualizaciones de memoria global [37]. En esta versión asíncrona, el número de sincronizaciones globales se reduce en comparación con la versión síncrona del algoritmo [35].

IV. RESULTADOS

Esta sección muestra los resultados experimentales obtenidos por los clasificadores. Los algoritmos propuestos han sido evaluados en un PC con un quad-core Intel Xeon E5-2609v2 a 2.5 GHz y 15 GB de RAM. El código se ha compilado usando gcc versión 4.8.4 con soporte OpenMP (OMP) 3.0 bajo Linux y 4 hilos. Se ha utilizado la librería OPENBLAS [38] para acelerar las operaciones de los distintos algoritmos. En cuanto a la implementación en GPU, se ha utilizado una tarjeta NVIDIA GrForce GTX Titan con 14 SMXs de 192 procesadores CUDA cada uno. Se usó la versión 7.5 del conjunto de herramientas CUDA para Linux.

La precisión de los resultados está expresada en porcentaje en términos de precisión total (OA), que es el porcentaje de píxeles clasificados correctamente, precisión media (AA), que se calcula como la media de las precisiones de las diferentes clases, y el coeficiente kappa [39], que es el porcentaje de acierto corregido por la cantidad de acierto que sería de esperar debido al azar.

Los resultados de rendimiento se expresan en términos de tiempo de ejecución (en segundos) y aceleración. Los resultados son la media de 100 ejecuciones. Los resultados se muestran como valor medio y desviación estándar de las 100 ejecuciones. El tiempo de ejecución para los códigos en GPU no incluye la transferencia de datos CPU-GPU, solo los tiempos de computación.

Los algoritmos han sido probados sobre tres imágenes de teledetección: Una imagen obtenida por el sensor ROSIS de 103 bandas de la Universidad de

Algoritmo 2 Algoritmo EVD-PCA sobre GPU

Entrada: El conjunto de datos \mathbf{X} se almacena inicialmente en la memoria global

- | | | |
|----|---|-----------|
| 1: | <Crear un vector de unos> | ▷ GM |
| 2: | Obtener la suma de todos los píxeles en cada banda del conjunto de datos | ▷ SM + GM |
| 3: | <Centrado del conjunto de datos> | ▷ SM |
| 4: | Calcular la matriz de covarianza $\mathbf{X}\mathbf{X}^T$ del conjunto de datos centrados | ▷ SM + GM |
| 5: | <Completar la matriz triangular inferior de $\mathbf{X}\mathbf{X}^T$ > | ▷ GM |
| 6: | Calcular la matriz de auto-vectores \mathbf{V} | ▷ SM + GM |
| 7: | Obtener los PCs mediante $\mathbf{X}\mathbf{V}$ | ▷ SM + GM |

Pavia (Pavia Univ.), con una dimensión espacial de 610×340 píxeles, una imagen obtenida por el sensor AVIRIS de 220 bandas y 145×145 píxeles tomada sobre el Noroeste de Indiana (Indian Pines) y una imagen obtenida mediante AVIRIS de 204 bandas y 512×217 píxeles del valle de Salinas, California (Salinas).

Para poder realizar la comparación, el número de muestras de entrenamiento es el mismo que en el trabajo de referencia [27]. La tabla I muestra información de las imágenes utilizadas, incluyendo sus dimensiones y el número y porcentaje de muestras de entrenamiento. La selección de las muestras de entrenamiento se ha realizado de forma aleatoria, además de que dichas muestras no son tenidas en cuenta cuando se evalúa la precisión. El número de neuronas de la capa oculta empleadas en el ELM es de 1000 para Pavia Univ., 300 para Indian Pines y 350 para Salinas en todos los casos [27].

Para el cálculo del EMP, los mejores resultados fueron los obtenidos considerando 7 componentes principales que se extrajeron previamente mediante análisis en componentes principales. Se consideran además 7 operaciones de apertura y cierre por reconstrucción, usando discos de radio incremental (el tamaño de los elementos estructurales es de 3, 5, 9, 13, 17, 21 y 25, dando lugar a un total de 105 componentes). El mapeo de características compuestas es de tipo concatenación ponderada, es decir, se le da un peso a la información espacial y otro a la espectral. El peso para la característica espectral (k_w) es 1, mientras que el peso para la característica espacial (k_s) se ha ajustado para cada imagen por prueba y error en valores desde 0.5 hasta 10. Los mejores resultados se obtuvieron con k_s 1, 5 y 3 para la Universidad de Pavia, Indian Pines y Salinas respectivamente. Tal y como se menciona en la Sección II-A, con el fin de maximizar la capacidad de discriminación del clasificador, cada conjunto de datos χ fue ajustado al rango $[0, \max(\chi) - \min(\chi)]$. Este proceso se llevó a cabo de forma independiente para los datos hiperspectrales y para cada uno de los componentes individuales del EMP. Finalmente, después de la concatenación de las dos características, todo el conjunto de datos se escaló dentro del rango de $[0, 1]$.

En primer lugar hemos realizado una comparación entre el esquema original basado en ELM (ELM-EMP) y la versión que incluye una post-regularización espacial (ELM-EMP-S). Esta post-regularización consiste en asignar a cada píxel la clase mayoritaria de entre sus vecinos. En nuestro caso se han tenido en cuenta a los 8 vecinos de cada píxel. La tabla II muestra que esta técnica mejora los resultados de precisión de la clasificación con un bajo coste en el tiempo de ejecución, pasando de un valor para OA de 92.81% a 95.05% en el caso de la imagen de Indian Pines y de un 99.65% a 99.82% en el caso de la imagen de Pavia Univ.. Por lo tanto, a partir de ahora los esquemas aquí estudiados incluyen dicha post-regularización.

En la tabla III se comparan, en términos de

precisión de la clasificación y tiempo de ejecución en GPU, los esquemas propuestos, ELM-EMP-S y KELM-EMP-S (KELM-EMP con regularización), con otros esquemas proyectados sobre GPU y disponibles en la literatura. Todos los experimentos han sido realizados en las mismas condiciones experimentales. Algunos de esos esquemas, como ELM+wat [9], usan ELM como clasificador mientras que otros, como SVM+wat [14] y WT-EMP [18], utilizan SVM, que es un clasificador bastante utilizado en la literatura. Todos los experimentos mostrados en la tabla se han realizado bajo las mismas condiciones experimentales descritas al inicio de la sección. En la tabla, los mejores resultados se muestran en negrita. El esquema ELM implementado en [9] realiza una regularización después de la clasificación, del mismo modo que los esquemas propuestos ELM-EMP-S y KELM-EMP-S. El esquema ELM+wat [9] combina la información proporcionada por el esquema ELM con información espacial obtenida a través de un algoritmo watershed. Los esquemas SVM [14] y SVM+wat [14] usan SVM como clasificador, pero el segundo añade información espacial usando un algoritmo de segmentación watershed. Por último, en el esquema WT-EMP [18] se crea un EMP a partir de las características extraídas mediante wavelets que se combina con la imagen original, sin ruido, en un vector que es procesado por el clasificador SVM.

En la tabla III podemos ver que los esquemas propuestos en este artículo (ELM-EMP-S y KELM-EMP-S) obtienen resultados muy cercanos en términos de precisión de la clasificación, siendo el esquema KELM-EMP-S ligeramente superior en las imágenes de Pavia Univ. e Indian Pines.

En las tablas IV y V se muestran los tiempos de ejecución y aceleraciones de las versiones GPU sobre las versiones OpenMP optimizadas para los esquemas propuestos (ELM-EMP-S y KELM-EMP-S). Ambos esquemas incluyen post-regularización. En el caso del esquema KELM-EMP-S, los valores para los parámetros definidos por el usuario C y γ para las tres imágenes hiperspectrales son: $C = 10^8$ y $\gamma = 10$ para Pavia Univ., $C = 10^6$ y $\gamma = 10$ para Indian Pines y $C = 10^8$ y $\gamma = 12$ para Salinas. Tal y como se mostraba en la tabla III, las precisiones alcanzadas por el esquema KELM-EMP-S para las dos primeras imágenes (Pavia Univ. e Indian Pines) son mejores que las obtenidas por el esquema ELM-EMP-S, sin embargo, podemos observar que requiere más tiempo para obtener el resultado. En cuanto a las aceleraciones, se han alcanzado valores de $4.92 \times$ para la imagen de Pavia Univ., mientras que para la imagen de Indian Pines el valor obtenido es una aceleración de $1.28 \times$. Esto se debe a que la imagen de Indian Pines es 4.5 veces más pequeña que las otras dos utilizadas en este artículo y, por lo tanto, el número de hilos necesarios es también menor, lo que dificulta ocultar el coste en las transferencias de datos y, por tanto, aumenta el tiempo de ejecución.

Basada en los resultados de la clasificación

TABLA I
 INFORMACIÓN SOBRE LAS IMÁGENES DE TELEDETECCIÓN UTILIZADAS PARA EL TEST.

	sensor	# clases	dimensiones	# muestras	# muestras de entrenamiento
Pavia Univ.	ROSIS	9	610×340×103	42776	3921 (9.17%)
Indian Pines	AVIRIS	16	145×145×220	10249	625 (6.10%)
Salinas	AVIRIS	16	512×217×204	54129	1076 (1.99%)

TABLA II
 COMPARATIVA, EN TÉRMINOS DE PRECISIÓN DE LA CLASIFICACIÓN Y TIEMPOS DE EJECUCIÓN EN GPU, ENTRE EL ESQUEMA ELM-EMP Y ELM-EMP-S.

Esquema	Pavia Univ.				Indian Pines			
	OA(%)	AA(%)	Kappa(%)	t(s)	OA(%)	AA(%)	Kappa(%)	t(s)
ELM-EMP	99.65 ±0.08	99.60 ±0.06	99.52 ±0.11	2.29 ±0.08	92.81 ±0.76	95.02 ±0.65	91.77 ±0.86	0.72 ±0.13
ELM-EMP-S	99.82 ±0.09	99.76 ±0.05	99.75 ±0.13	2.30 ±0.10	95.05 ±0.74	96.44 ±0.56	94.32 ±0.85	0.72 ±0.12

TABLA III
 PRECISIONES Y TIEMPOS DE EJECUCIÓN EN GPU PARA LAS DISTINTAS IMÁGENES. LAS DESVIACIONES ESTÁNDAR VAN SEGUIDAS DEL SÍMBOLO ±. LAS MEJORES PRECISIONES SE MUESTRAN EN NEGRITA.

Esquema	Pavia Univ.			Indian Pines			Salinas		
	OA(%)	AA(%)	Kappa(%)	OA(%)	AA(%)	Kappa(%)	OA(%)	AA(%)	Kappa(%)
KELM-EMP-S	99.83 ±0.07	99.79 ±0.04	99.77 ±0.09	95.39 ±0.80	96.82 ±0.63	94.72 ±0.92	99.16 ±0.18	99.05 ±0.20	99.06 ±0.20
ELM-EMP-S	99.82 ±0.09	99.76 ±0.05	99.75 ±0.13	95.05 ±0.74	96.44 ±0.56	94.32 ±0.85	99.21 ±0.25	99.09 ±0.18	99.12 ±0.28
ELM	96.94 ±0.31	96.13 ±0.34	95.83 ±0.43	82.25 ±1.57	89.85 ±0.98	79.93 ±1.73	93.63 ±0.29	96.38 ±0.25	92.89 ±0.32
ELM+wat	97.20 ±0.33	96.42 ±0.43	96.30 ±0.44	80.64 ±2.46	79.90 ±3.04	78.40 ±2.73	93.60 ±0.32	96.41 ±0.27	92.86 ±0.36
SVM	79.43 ±0.96	86.62 ±0.86	76.70 ±1.05	79.43 ±0.96	86.62 ±0.86	76.70 ±1.05	88.45 ±0.47	92.37 ±0.50	87.08 ±0.54
SVM+wat	96.18 ±0.52	96.58 ±0.25	94.81 ±0.70	87.65 ±1.26	91.41 ±1.78	85.96 ±1.42	-	-	-
WT-EMP	98.70 ±0.18	98.72 ±0.13	98.22 ±0.24	89.73 ±1.14	94.12 ±0.67	88.28 ±0.073	-	-	-

TABLA IV
 TIEMPOS DE EJECUCIÓN Y ACELERACIÓN PARA EL ESQUEMA ELM-EMP-S.

		PCA (s)	EMP (s)	ELM (s)	Total (s)
Pavia Univ.	CPU	0.43	14.93	14.97	30.33
	OMP	0.28	4.35	6.70	11.33
	GPU	0.05	1.08	1.17	2.30
	Speedup GPU vs. OMP: 4.92×				
Indian Pines	CPU	0.13	1.52	0.82	2.47
	OMP	0.07	0.44	0.4	0.91
	GPU	0.06	0.58	0.07	0.71
	Speedup GPU vs. OMP: 1.28×				
Salinas	CPU	0.60	8	3.96	12.56
	OMP	0.31	2.30	2.06	4.67
	GPU	0.07	0.70	0.22	0.99
	Speedup GPU vs. OMP: 4.71×				

obtenidos por cada esquema de clasificación también realizamos la prueba estándar de McNemar [40], [26]. Se emplea para verificar la relevancia estadística de las diferencias de precisión entre pares de esquemas.

La tabla VI presenta los resultados del test, donde cada valor Z compara el clasificador KELM-EMP-S con cada uno de los métodos mostrados en la tabla. La diferencia en precisión entre cada par de clasifi-

TABLA V
 TIEMPOS DE EJECUCIÓN Y ACELERACIÓN PARA EL ESQUEMA KELM-EMP-S.

		PCA (s)	EMP (s)	KELM (s)	Total (s)
Pavia Univ.	CPU	0.43	14.93	51.61	66.97
	OMP	0.28	4.35	14.92	19.55
	GPU	0.05	1.08	2.80	3.93
	Speedup GPU vs. OMP: 4.97×				
Indian Pines	CPU	0.13	1.52	1.22	2.87
	OMP	0.07	0.44	0.56	1.07
	GPU	0.06	0.58	0.15	0.79
	Speedup GPU vs. OMP: 1.35×				
Salinas	CPU	0.60	8.00	8.92	17.52
	OMP	0.31	2.30	2.81	5.42
	GPU	0.07	0.70	0.62	1.39
	Speedup GPU vs. OMP: 3.90×				

cadore es vista como significativa con un nivel de confianza del 95% si $|Z| > 1.96$, y con un nivel de confianza del 99 % si $|Z| > 2.58$. Un valor positivo de Z indica que el primer clasificador supera al segundo ($Z > 0$). Podemos observar que sólo en un caso, cuando comparamos los esquemas KELM-EMP-S y ELM-EMP-S, las pequeñas diferencias en la precisión no son estadísticamente relevantes, lo que indica que ambos sistemas de clasificación son similares en términos de precisión.

V. CONCLUSION

En este artículo se presenta un esquema de clasificación para imágenes hiperspectrales de teledetección basado en *kernels* llamado KELM-EMP desarrollado a partir de un esquema publicado anteriormente (ELM-EMP) por los autores[27]. Estos esquemas realizan una reducción de la dimensionalidad de la imagen hiperspectral mediante análisis en componentes principales, seguida de la construcción de un perfil morfológico extendido. A continuación se utiliza una concatenación ponderada para combinar los datos del perfil con los datos de la imagen hiperspectral sobre los que se ejecuta el clasificador. En este artículo se propone también añadir un proceso de regularización espacial al final de la clasificación, de modo que se eliminen clasificaciones erróneas de píxeles aislados. Una última aportación de este artículo es proponer proyecciones de los algoritmos ELM-EMP-S y KELM-EMP-S sobre GPU, utilizando CUDA y aplicando técnicas de mejora como la ejecución asíncrona de algunas secciones del algoritmo.

Los resultados de precisión de la clasificación muestran que el esquema basado en KELM con regularización espacial (KELM-EMP-S) obtiene precisiones altas alcanzando un valor de 99.83% de precisión media para la imagen de Pavia Univ. En cuanto a la ejecución en GPU se consiguen aceleraciones altas en comparación con la ejecución de las versiones OpenMP alcanzando un speedup de 4.97× para la imagen de Pavia Univ.

AGRADECIMIENTOS

Agradecemos al profesor Chen Chen, de la Universidad de Texas, por su colaboración compartiendo su código MATLAB con nosotros. Este trabajo fue financiado en parte por el Ministerio de Ciencia e Innovación, Gobierno de España, cofinanciado con fondos FEDER a través de la Unión Europea bajo el contrato TIN 2013-41129-P, y por la Xunta de Galicia, mediante el programa de consolidación y estructuración de unidades de investigación competitivas ref. 2014/008.

REFERENCIAS

- [1] David Landgrebe, "Hyperspectral image data analysis," *Signal Processing Magazine, IEEE*, vol. 19, no. 1, pp. 17–28, 2002.
- [2] Mathieu Fauvel, Yuliya Tarabalka, Jón Atli Benediktsson, Jocelyn Chaussoot, and James C Tilton, "Advances in spectral–spatial classification of hyperspectral images," *Proceedings of the IEEE*, vol. 101, no. 3, pp. 652–675, 2013.
- [3] Guang-Bin Huang, "An insight into extreme learning machines: random neurons, random features and kernels," *Cognitive Computation*, vol. 6, no. 3, pp. 376–390, 2014.
- [4] Shinichi Tamura and Masshiko Tateishi, "Capabilities of a four-layered feedforward neural network: four layers versus three," *Neural Networks, IEEE Transactions on*, vol. 8, no. 2, pp. 251–255, 1997.
- [5] Guang-Bin Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *Neural Networks, IEEE Transactions on*, vol. 14, no. 2, pp. 274–281, 2003.
- [6] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.
- [7] Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, and Rui Zhang, "Extreme learning machine for regression and multiclass classification," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 42, no. 2, pp. 513–529, 2012.
- [8] Mark Van Heeswijk, Yoan Miche, Erkki Oja, and Amaury Lendasse, "Gpu-accelerated and parallelized elm ensembles for large-scale regression," *Neurocomputing*, vol. 74, no. 16, pp. 2430–2437, 2011.
- [9] Javier López-Fandiño, Pablo Quesada-Barrisuo, Dora B Heras, and Francisco Argüello, "Efficient elm-based techniques for the classification of hyperspectral remote sensing images on commodity gpus," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 8, no. 6, pp. 2884–2893, 2015.
- [10] Antonio Plaza, Jón Atli Benediktsson, Joseph W. Boardman, Jason Brazile, Lorenzo Bruzzone, Gustavo Camps-Valls, Jocelyn Chaussoot, Mathieu Fauvel, Paolo Gamba, Anthony Gualtieri, et al., "Recent advances in techniques

TABLA VI
 RELEVANCIA ESTADÍSTICA DE LAS DIFERENCIAS EN LA PRECISIÓN DE CLASIFICACIÓN ENTRE KELM-EMP-S Y LOS ESQUEMAS PROPUESTOS EN LA TABLA.

		ELM-EMP-S	ELM	ELM+wat	SVM	SVM+wat	WT-EMP
Z	Pavia Univ.	0.615	33.94	32.31	64.30	38.25	19.72
	Indian Pines	1.55	31.27	26.86	89.76	89.52	88.76
	Salinas	-1.48	50.99	74.32	51.09	-	-

for hyperspectral image processing,” *Remote sensing of environment*, vol. 113, pp. S110–S122, 2009.

[11] Mauro Dalla Mura, Alberto Villa, Jón Atli Benediktsson, Jocelyn Chanussot, and Lorenzo Bruzzone, “Classification of hyperspectral images by using extended morphological attribute profiles and independent component analysis,” *Geoscience and Remote Sensing Letters, IEEE*, vol. 8, no. 3, pp. 542–546, 2011.

[12] J.A. Palmason, J.A. Benediktsson, J.R. Sveinsson, and J. Chanussot, “Classification of hyperspectral data from urban areas using morphological preprocessing and independent component analysis,” in *International Geoscience And Remote Sensing Symposium*, 2005, vol. 1, pp. 176–179.

[13] Yun Zhang, Xuezhi Feng, and Xinghua Le, “Segmentation on multispectral remote sensing image using watershed transformation,” in *Image and Signal Processing, 2008. CISP’08. Congress on*. IEEE, 2008, vol. 4, pp. 773–777.

[14] Yuliya Tarabalka, Jocelyn Chanussot, and Jón Atli Benediktsson, “Segmentation and classification of hyperspectral images using watershed transformation,” *Pattern Recognition*, vol. 43, no. 7, pp. 2367–2379, 2010.

[15] Blanca Priego, Francisco Bellas, and Richard J. Duro, “FCAS-II: A hybrid algorithm for the construction of multidimensional image segmenters,” in *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE, 2015, pp. 1–8.

[16] Yuliya Tarabalka, Jón Atli Benediktsson, and Jocelyn Chanussot, “Spectral–spatial classification of hyperspectral imagery based on partitional clustering techniques,” *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 47, no. 8, pp. 2973–2987, 2009.

[17] Xudong Kang, Shutao Li, and Jón Atli Benediktsson, “Spectral–spatial hyperspectral image classification with edge-preserving filtering,” *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 52, no. 5, pp. 2666–2677, 2014.

[18] Pablo Quesada-Barriso, Francisco Argüello, and Dora B. Heras, “Spectral–spatial classification of hyperspectral images using wavelets and extended morphological profiles,” *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 7, no. 4, pp. 1177–1185, 2014.

[19] Martino Pesaresi and Jón Atli Benediktsson, “A new approach for the morphological segmentation of high-resolution satellite imagery,” *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 39, no. 2, pp. 309–320, 2001.

[20] Pierre Soille and Martino Pesaresi, “Advances in mathematical morphology applied to geoscience and remote sensing,” *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 40, no. 9, pp. 2042–2055, 2002.

[21] Jón Atli Benediktsson, Martino Pesaresi, and Kolbeinn Amason, “Classification and feature extraction for remote sensing images from urban areas based on morphological transformations,” *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 41, no. 9, pp. 1940–1949, 2003.

[22] Prashanth Reddy Marpu, Mattia Pedergnana, Mauro Dalla Mura, Stijn Peeters, Jón Atli Benediktsson, and Lorenzo Bruzzone, “Classification of hyperspectral data using extended attribute profiles based on supervised and unsupervised feature extraction techniques,” *International Journal of Image and Data Fusion*, vol. 3, no. 3, pp. 269–298, 2012.

[23] Jón Atli Benediktsson, Jón Aevar Palmason, and Johannes R. Sveinsson, “Classification of hyperspectral data from urban areas based on extended morphological profiles,” *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 43, no. 3, pp. 480–491, 2005.

[24] Giorgio Licciardi, Prashanth Reddy Marpu, Jón Atli Benediktsson, and Jocelyn Chanussot, “Extended morphological profiles using auto-associative neural networks for hyperspectral data classification,” in *Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), 2011 3rd Workshop on*. IEEE, 2011, pp. 1–4.

[25] Mauro Dalla Mura, Jón Atli Benediktsson, Björn Waske, and Lorenzo Bruzzone, “Morphological attribute profiles for the analysis of very high resolution images,” *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 48, no. 10, pp. 3747–3762, 2010.

[26] Chen Chen, Wei Li, Hongjun Su, and Kui Liu, “Spectral–spatial classification of hyperspectral image based on kernel extreme learning machine,” *Remote Sensing*, vol. 6, no. 6, pp. 5795–5814, 2014.

[27] Francisco Argüello and Dora B. Heras, “ELM-based spectral–spatial classification of hyperspectral images using extended morphological profiles and composite feature mappings,” *International Journal of Remote Sensing*, vol. 36, no. 2, pp. 645–664, 2015.

[28] Guang-Bin Huang, Dian Hui Wang, and Yuan Lan, “Extreme learning machines: a survey,” *International Journal of Machine Learning and Cybernetics*, vol. 2, no. 2, pp. 107–122, 2011.

[29] Hervé Abdi and Lynne J. Williams, “Principal component analysis,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[30] Martino Pesaresi and Ioannis Kanelopoulos, “Detection of urban features using morphological based segmentation and very high resolution remotely sensed data,” in *Machine Vision and Advanced Image Processing in Remote Sensing*, pp. 271–284. Springer, 1999.

[31] Nvidia, “Nvidia kepler gk110 architecture,” 2012.

[32] MAGMA, “Matrix algebra on gpu and multicore architectures,” 2015.

[33] Nvidia, “Cuda tools,” 2015.

[34] Nvidia, “Cuda toolkit documentation: Cublas,” 2015.

[35] Pablo Quesada-Barriso, Dora B. Heras, and Francisco Argüello, “Efficient 2D and 3D watershed on graphics processing unit: block-asynchronous approaches based on cellular automata,” *Computers & Electrical Engineering*, vol. 39, no. 8, pp. 2638–2655, 2013.

[36] Pablo Quesada-Barriso, Francisco Argüello, Dora B. Heras, and Jón Atli Benediktsson, “Wavelet–based classification of hyperspectral images using extended morphological profiles on graphics processing units,” 2014.

[37] D. Kirk and W. Hwu Wen-mei, *Programming massively parallel processors: a hands-on approach*, vol. 1. Elsevier Science, Applications of GPU Computing Series, 2010.

[38] OpenBLAS, “An optimized blas library,” 2015.

[39] John A. Richards, *Remote sensing digital image analysis*, vol. 3. Springer, 1999.

[40] Giles M Foody, “Thematic map comparison,” *Photogrammetric Engineering & Remote Sensing*, vol. 70, no. 5, pp. 627–633, 2004.

Análisis del efecto del particionado en Tiles sobre una versión paralela del codificador HEVC

Otoniel López, Pablo Piñol, Héctor Migallón, Vicente Galiano, Manuel P. Malumbres¹

Resumen.— El nuevo estándar de codificación de vídeo HEVC introduce un nuevo concepto llamado 'Tiles'. Los Tiles son regiones rectangulares de un frame de vídeo que pueden ser codificadas de manera independiente. Con el fin de reducir el tiempo necesario para codificar una secuencia de vídeo con HEVC, hemos utilizado un enfoque paralelo basado en Tiles y se han evaluado los beneficios y desventajas de esta aproximación. En las pruebas realizadas se obtienen aceleraciones de hasta 9.3x utilizando 10 procesos con una pérdida en R/D despreciable.

Palabras clave.— HEVC, openmp, tiles, paralelismo, codificación de vídeo.

EL Joint Collaborative Team on Video Coding (JCT-VC) ha desarrollado un nuevo estándar de codificación de vídeo llamado High Efficiency Video Coding (HEVC)[1]. El JCT-VC está formado por miembros de la norma ISO/IEC Moving Picture Experts Group (MPEG) y la UIT-T Video Coding Experts Group (VCEG). El nuevo estándar alcanza casi un 50% de ahorro de tasa de bits en comparación con el anterior estándar de codificación de vídeo H.264/AVC (Advanced Video Coding) [2]. El aumento de la eficiencia está ligada a un aumento de la complejidad computacional. Para hacer frente al aumento de la complejidad se utilizan técnicas de paralelización para aprovechar las arquitecturas de computación paralela disponibles.

HEVC incluye nuevas características que permiten la computación paralela de alto nivel, como el Wavefront Parallel Processing (WPP) o los Tiles y también incluye nuevas características que permiten la computación paralela de bajo nivel (dentro del proceso de codificación), como el Parallel Method [3] que permite la estimación de movimiento de forma paralela.

Los tiles son regiones rectangulares de un frame de vídeo que pueden ser codificadas de forma independiente. En este trabajo se analiza el comportamiento paralelo cuando se usan tiles y cómo el diseño de particionado afecta en el rendimiento tanto en la aceleración como en el Rate/Distortion (R/D).

Algunos trabajos como [4] se centran en la paralelización del decodificador HEVC, ya que se busca una decodificación rápida de contenidos multimedia pre-codificada, como el cine digital y de vídeo bajo demanda. Sin embargo, pensamos que se deben realizar más esfuerzos en el lado del codificador, donde se encuentra la mayor complejidad computacional. La paralelización de la parte de codificación de vídeo

puede ser útil para aplicaciones como la grabación de vídeo o el streaming de eventos en vivo.

Otros trabajos analizan y proponen técnicas paralelas de bajo nivel para la codificación de secuencias de vídeo con HEVC, como la paralelización del módulo de estimación de movimiento [5] o la paralelización del módulo de predicción intra [6]. Nuestro trabajo se basa en aplicar técnicas paralelas de alto nivel, mediante el uso de tiles sobre arquitecturas de memoria compartida.

En [7], los autores comparan slices y tiles sobre HEVC. Los resultados se muestran en términos de porcentaje de incremento o decremento de la tasa de bits. En nuestro estudio evaluamos el rendimiento de los tiles, pero nos centraremos tanto en la reducción de la complejidad relacionada con la codificación como en el R/D obtenido. Además, se evalúa el impacto del particionado de los tiles.

El resto del trabajo se organiza de la siguiente manera. En la Sección I presentaremos los principales aspectos relacionados con los tiles en HEVC. En la Sección II se presentan y analizan los resultados de las pruebas realizadas. Finalmente, en la Sección III se presentan las conclusiones obtenidas.

I. PARTICIONADO EN TILES

La división de un frame de vídeo en tiles es una técnica nueva incluida en HEVC que no existía en los estándares anteriores de codificación de vídeo. Los tiles son regiones rectangulares de un frame de vídeo que puede ser codificadas de forma independiente (y posteriormente decodificadas), con el uso de algunos datos comunes para todo el bitstream. Esta independencia permite la paralelización de los procesos de codificación y decodificación a nivel 'sub-picture'. Los tiles contienen un número entero de Coding Tree Units (CTUs). Cada región rectangular resulta de la división de un frame en varias columnas y filas. El ancho de cada columna (en unidades CTU) y también el alto de cada fila se puede ajustar individualmente. En el ejemplo que se muestra en la Figura 1, un frame de resolución Full HD (1920x1080) se divide en 10 tiles utilizando un esquema de particionado de 5 columnas con un ancho de 6 CTUs y 2 filas con una altura de 8 y 9 CTUs, respectivamente.

La independencia de los tiles permite la paralelización del proceso de codificación, pero tiene un inconveniente principal: la eficiencia de la codificación respecto al rendimiento en R/D, se ve reducida. Esto sucede porque los tiles no pueden

¹Dpto. de Física y Arquitectura de Computadores, Universidad Miguel Hernández, e-mail: otoniel, pablomp, hmgallon, vgaliano, mels@umh.es.

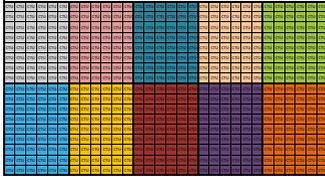


Fig. 1. División de un frame full-HD (1920x1080 pixels) en 10 tiles (5 columnas de 6 CTUs de ancho; 2 filas con 8 and 9 CTUs de alto cada una).

utilizar la información de los CTUs perteneciente a los otros tiles vecinos para realizar cualquier tipo de predicción, y por lo tanto, la redundancia existente entre los CTUs vecinos que pertenecen a diferentes tiles no puede ser explotada.

En este trabajo se ha implementado una paralelización a nivel de tile en el codificador de vídeo HEVC para plataformas de memoria compartida. La codificación de cada tile se asigna a un núcleo diferente. Hemos evaluado el rendimiento de la versión paralela para 2, 4, 6, 8, 9, y 10 procesos y se han comparado los resultados obtenidos con los proporcionados por la versión secuencial, tanto con respecto al rendimiento en R/D, como en tiempo de cómputo. Como se ha comentado anteriormente, se mapean los tiles de cada frame en el mismo número de procesos. Para un cierto número de tiles por frame, podemos encontrar diferentes particionados del frame. Por ejemplo, si queremos dividir el frame en 9 tiles podemos elegir tres diferentes distribuciones: 9x1 (9 columnas por fila 1), 1x9 (1 columna por 9 filas) y 3x3 (3 columnas por filas 3). Cada una de estas distribuciones se podría, además, haber diseñado de manera diferente si cambiamos el ancho y el alto de cada una de las columnas y las filas. Con el fin de obtener la máxima eficiencia de computación en paralelo vamos a utilizar los diseños que proporcionan la distribución de la carga más equilibrada, es decir, los tiles que producen un número similar de CTUs. Una distribución de carga equilibrada no siempre garantiza una distribución equilibrada de trabajo debido a que los recursos necesarios para codificar un solo CTU pueden variar, pero una distribución desequilibrada de la carga probablemente implicará una baja eficiencia computacional. Como una medida del equilibrio en la distribución de la carga, hemos calculado la eficiencia máxima teórica que se podría obtener en la versión paralela, teniendo en cuenta la misma complejidad computacional para cada CTU. Un valor del 100% significa que todos los procesos codificarán el mismo número de CTUs. En la Tabla I, se enumeran los diferentes particionados de tiles que utilizaremos en nuestras pruebas con diferente número de núcleos y dos formatos de vídeo.

La eficiencia óptima paralela teórica sería proporcionada por la columna 'balance %' en la Tabla I. Si dividimos, por ejemplo, un frame de 2560x1600 píxeles en 10 tiles, y elegimos el particionado 10x1,

entonces tendremos 10 tiles con 100 CTUs cada uno, lo que significa un 100% de balanceo de carga (todos los tiles tienen el mismo número de CTUs). Si nosotros, por lo contrario, seleccionamos el particionado de 1x10, entonces vamos a tener 5 tiles con 80 CTUs cada uno, y 5 tiles con 120 CTUs cada uno. El escenario más probable es que los 5 procesos que gestionan los tiles más 'pequeños' permanezcan a la espera de los procesos responsables de los tiles más 'grandes'. En este caso, el índice de balanceo de carga máxima sería el 83%. Por lo tanto, para un número específico de procesos, el diseño seleccionado puede afectar a la eficiencia de la versión paralela. Se ha de tener también en cuenta que un solo diseño puede proporcionar diferentes porcentajes de equilibrio de carga en función de la resolución de la secuencia de vídeo. Por ejemplo, la disposición 4x1 obtiene el 100% y el 94% de equilibrio de carga para resoluciones de vídeo de 2560x1600 y 1920x1080 respectivamente.

En la siguiente sección se presentarán los resultados para las secuencias de vídeo seleccionadas, usando todos los particionados que se indican en la Tabla I.

II. EXPERIMENTOS

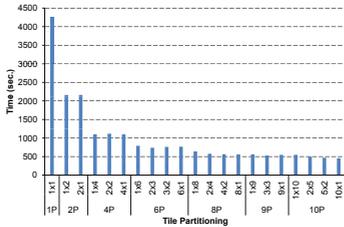
El algoritmo paralelo propuesto se ha lanzado sobre una plataforma de memoria compartida que consiste en dos hexacores Intel Xeon X5660 con frecuencias de hasta 2.8GHz, una caché de 12 MB por procesador y 48 GB de RAM. El sistema operativo utilizado es CentOS Linux 5.6 para x86 de 64 bits. El entorno paralelo usado es OpenMP [8]. El compilador utilizado es *g++* compilador v.4.1.2. La versión del software de codificación de referencia utilizado es HM 16.3 [9].

Las secuencias de vídeo de test utilizadas en nuestros experimentos son Traffic (2560x1600), People on Street (2560x1600), Tennis (1920x1080) y Park Scene (1920x1080). Los resultados obtenidos son para los modos de codificación Low-delay B (LB) y All Intra (AI), codificando 150 frames las secuencias Traffic y People y 240 frames para ParkScene y Tennis con diferentes parámetros de cuantificación (QPs) (22, 27, 32, 37).

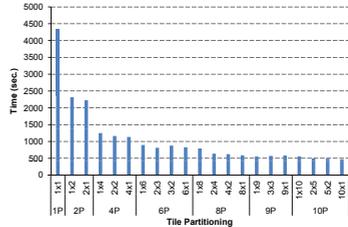
En la Figura 2, se presenta la evolución del tiempo de codificación para las secuencias Traffic y People para los modos AI y LB en función del número de procesos según el particionado de tiles propuesto. Como se puede ver, el tiempo de codificación puede verse reducido hasta 9.3 veces cuando usamos 10 procesos. Como puede verse en la Figura 3, el algoritmo de paralelización a nivel de tiles obtiene un buen rendimiento y también buenos resultados en cuanto a escalabilidad. Si nos fijamos en la Figura 3, para 10 procesos, existen diferencias en el rendimiento paralelo cuando usamos diferentes particionados de tiles. En concreto, en el modo de codificación AI podemos encontrar diferencias de hasta 1.58x de un esquema de partición de tiles de 1x10 con respecto al esquema de partición de tiles de 10x1. Por lo gen-

TABLA I
 PARTICIONADO Y PORCENTAJE DE BALANCEO DE CARGA

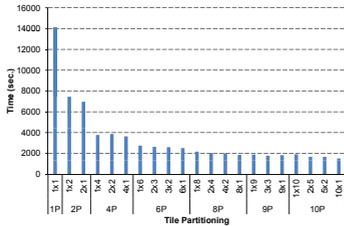
(a) 2560x1600 (40x25 CTUs)					(b) 1920x1080 (30x17 CTUs)					
#Proc	Particionado	AvgCTU	MaxCTU	Bal %	#Proc	Particionado	AvgCTU	MaxCTU	Bal %	
1P	1x1	1000	1000	100%	1P	1x1	510	510	100%	
	2P	1x2	500	520		96%	2P	1x2	255	270
2P	2x1	500	500	100%	2P	2x1		255	255	100%
	4P	1x4	250	280		89%	4P	1x4	127.5	150
2x2		250	260	96%	4P	2x2		127.5	135	94%
4x1		250	250	100%		4P		4x1	127.5	136
6P	1x6	166.7	200	83%	6P		1x6	85	90	94%
	2x3	166.7	180	93%		6P	2x3	85	90	94%
	3x2	166.7	182	92%			6P	3x2	85	90
	6x1	166.7	175	95%		6P		6x1	85	85
8P	1x8	125	160	78%	8P		1x8	63.8	90	71%
	2x4	125	140	89%		8P	2x4	63.8	75	85%
	4x2	125	130	96%			8P	4x2	63.8	72
	8x1	125	125	100%		8P		8x1	63.8	68
9P	1x9	111.1	120	93%	9P		1x9	56.7	60	94%
	3x3	111.1	126	88%		9P	3x3	56.7	60	94%
	9x1	111.1	125	89%			9P	9x1	56.7	68
10P	1x10	100	120	83%	10P	1x10		51	60	85%
	2x5	100	100	100%		10P	2x5	51	60	85%
	5x2	100	104	96%			10P	5x2	51	54
	10x1	100	100	100%		10P		10x1	51	51



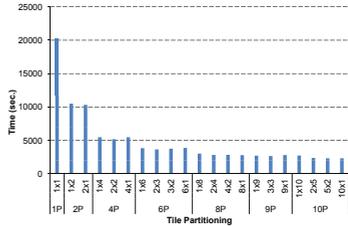
(a) Traffic AI mode.



(b) People AI mode.



(c) Traffic LB mode.



(d) People LB mode.

Fig. 2. Evolución del tiempo de codificación para todas las secuencias de vídeo con diferente número de procesos y particionado de tiles para un QP=37.

eral, los diseños de particionado de tiles basados en columnas de CTUs o los tiles cuadrados obtienen un mejor rendimiento paralelo. Como es de esperar, este

efecto depende de la resolución de vídeo. Siguiendo con el ejemplo anterior, para una resolución de vídeo de 2560x1600, y un tamaño de CTU de 64x64, el

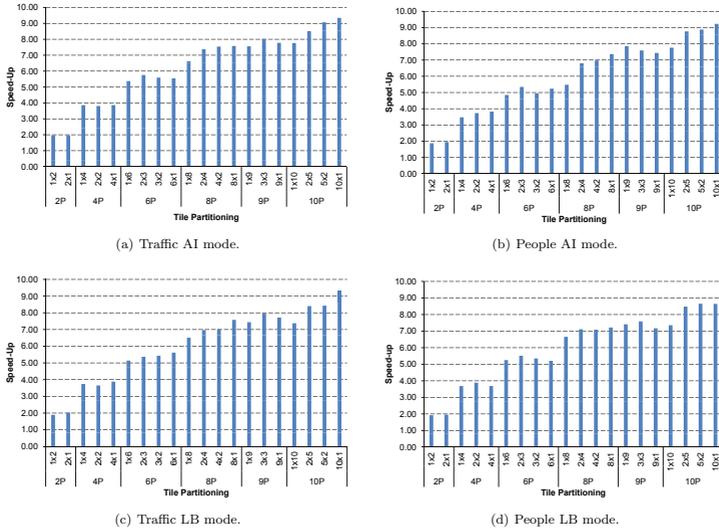


Fig. 3. Evolucion del Speed-Up para todas las secuencias de video con diferente numero de procesos y particionados de tiles para un QP=37.

número de CTUs en un frame son 40x25. Si dividimos el frame con el particionado 1x10, tenemos 5 procesos con CTUs de 40x2 y 5 procesos con 40x3 CTUs. Por otra parte, si dividimos el frame con la disposición 10x1, contamos con 10 procesos de 4x25 CTUs. En el primer caso (1x10), 5 procesos tienen que realizar un 50% más trabajo que los otros 5 procesos. Por lo general, cuanto más equilibrada esté la carga computacional, mejor rendimiento paralelo se consigue, a excepción de algunas secuencias en las que esto no se logra. En estas excepciones, incluso cuando cada proceso tiene el mismo número de CTUs, la complejidad computacional inherente a cada CTU difiere, produciendo que algunos procesos terminen antes que los otros.

En la Figura 4 se muestra la eficiencia media de la versión paralela para todos los valores de QP de las diferentes secuencias codificadas para ambos modos LB y AI. Como se puede ver, se obtienen buenas eficiencias para los dos modos de codificación LB y AI, siendo la eficiencia media del 87%. Sin embargo, si nos centramos en los diseños de particionado de tiles cuadrados, la eficiencia media se eleva hasta el 91%.

En cuanto al rendimiento en R/D, en la Figura 5 se presenta el % de BD-rate para las secuencias de video de resolución 4K en función del número de procesos y del esquema de particionado en tiles. Como se puede ver, el % de BD-rate aumenta cuando aumenta el número de procesos. Este es un comportamiento

esperado porque los tiles son estructuras independientes y por lo tanto el codificador aritmético funciona de una manera independiente en cada tile, no disponiendo de información de los tiles previamente codificados. Por otra parte, la partición en tiles cuadrados obtiene ligeramente mejor resultado tanto en el modo LB como en AI, debido a que hay más información de los CTUs vecinos para la predicción inter e intra.

Según los resultados obtenidos, podemos asegurar que la paralelización del codificador HEVC en tiles es interesante, ya que reduce drásticamente el tiempo de codificación (hasta 9.3x para 10 procesos) con un ligero incremento en R/D, especialmente si se utilizan esquemas de particionado cuadrados (BD-rate de 0.75% para el modo AI y 1.2% para el modo LB, en promedio). El incremento máximo debido al esquema de particionado del tile es de un 5% para la secuencia Tennis en el modo LB utilizando 10 procesos. Por lo tanto, el principal aspecto que afectará al rendimiento de la versión paralela es la carga computacional y es por eso que debemos dividir los tiles de tal manera que todos los procesos tengan el mismo número de CTUs. En el caso de las secuencias de video analizadas en este trabajo, un particionado de tiles cuadrados o con más columnas que filas de CTUs, tienen una mejor distribución de la carga computacional.

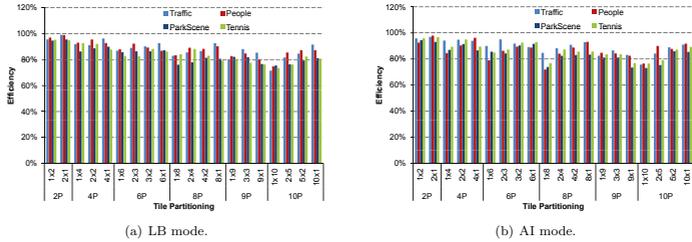


Fig. 4. Eficiencia paralela media para todas las secuencias de video con diferente número de procesos y particionado de tiles para todos los QPs.

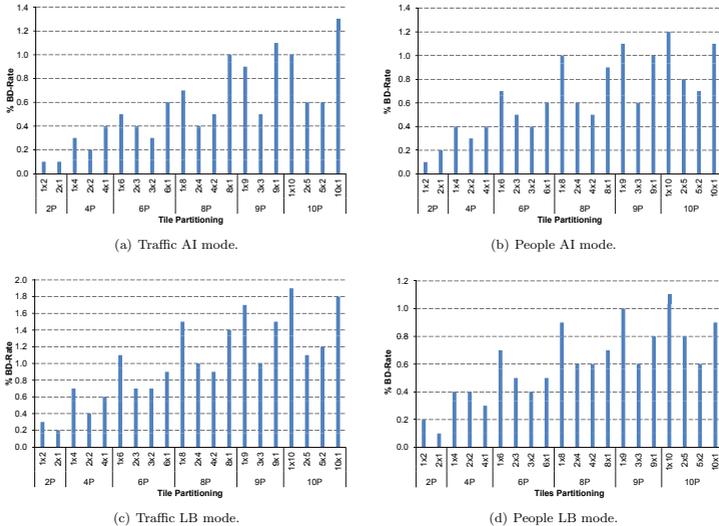


Fig. 5. % BD-Rate medio para todas las secuencias de video con diferente numero de procesos y particionado de tiles para todos los QPs.

III. CONCLUSIONES

En este artículo se presenta un estudio de la paralelización a nivel de tiles del codificador HEVC, usando diferentes diseños de particionado de frame. Los resultados muestran que tanto los particionados de tiles cuadrados como aquellos con más columnas que filas obtienen los mejores resultados (hasta 9.3x para 10 procesos) en las secuencias de video utilizadas. Aunque en algunos experimentos los particionados de tiles basados en columnas puedan obtener una mejor eficiencia, en promedio, los particionados de tiles cuadrados presentan un mejor comportamiento tanto en speed-up como en R/D. Además, el incremento en el porcentaje de BD-rate

es baja en todos los casos, especialmente cuando se aplican particionados de tiles cuadrados, debido a la mayor información disponible de los CTUs vecinos para los procesos de predicción inter e intra.

Por último, debemos tener en cuenta la resolución de la secuencia de video con el fin de realizar el particionado de los frames, de tal manera que el número de CTUs en cada tile sea casi el mismo y por lo tanto la carga computacional esté más equilibrada.

AGRADECIMIENTOS

Esta investigación ha sido financiada gracias al Ministerio de Economía y Competitividad TIN2015-66972-C5-4-R co-financiado con fondos FEDER.

REFERENCIAS

- [1] Benjamin Bross, Woo-Jin Han, Jens-Rainer Ohm, Gary J. Sullivan, Ye-Kui Wang, and Thomas Wiegand, "High Efficiency Video Coding (HEVC) text specification draft 10," Tech. Rep. JCTVC-L1003, Joint Collaborative Team on Video Coding (JCT-VC), Geneva (Switzerland), January 2013.
- [2] ITU-T and ISO/IEC JTC 1, "Advanced video coding for generic audiovisual services," *ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC) version 16*, 2012, 2012.
- [3] Minhua Zhou, "AHG10: Configurable and CU-group level parallel merge/skip," Tech. Rep., Joint Collaborative Team on Video Coding-H0082, 2012.
- [4] M. Alvarez-Mesa, C.C. Chi, B. Juurlink, V. George, and T. Schierl, "Parallel video decoding in the emerging HEVC standard," in *International Conference on Acoustics, Speech, and Signal Processing, Kyoto*, March 2012, pp. 1-17.
- [5] Qin Yu, Liang Zhao, and Siwei Ma, "Parallel AMVP candidate list construction for HEVC," in *VCIP'12*, 2012, pp. 1-6.
- [6] Jie Jiang, Baolong Guo, Wei Mo, and Kefeng Fan, "Block-based parallel intra prediction scheme for HEVC," *Journal of Multimedia*, vol. 7, no. 4, pp. 289 -294, August 2012.
- [7] K. Misra, A. Segall, M. Horowitz, Shilin Xu, A. Fuldseth, and Minhua Zhou, "An overview of tiles in HEVC," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 7, no. 6, pp. 969-977, Dec 2013.
- [8] "Openmp application program interface, version 3.1," *OpenMP Architecture Review Board*, <http://www.openmp.org>, 2011.
- [9] HEVC Reference Software, <https://hevc.hhi.fraunhofer.de/svn/svn.HEVCSoftware/tags/HM-16.3/>, " .

MARL-Ped+Hitmap: Aumentando la productividad de simulaciones basadas en agentes con una herramienta de arrays distribuidos

Eduardo Rodríguez-Gutierrez ¹, Francisco Martínez-Gil ², Juan Manuel Orduña-Huertas ³, Arturo González-Escribano ⁴

Resumen— Los sistemas Multi-agente están constituidos por piezas software llamadas agentes que son capaces de percibir el entorno y actuar en él de manera autónoma. MARL-Ped es un modelo Multi-agente de peatones donde cada agente (peatón) aprende el comportamiento adecuado para la simulación de diferentes situaciones (aglomeraciones, cruces, evacuaciones de recintos cerrados,...). MARL-Ped utiliza el estándar de paso de mensajes MPI para su explotación en sistemas distribuidos de forma portable. Programar utilizando directamente sistemas de paso de mensajes requiere un gran esfuerzo si se desean introducir políticas de reparto de carga flexibles y que se adapten a la plataforma de destino. Hitmap es una biblioteca de funciones para facilitar la programación de aplicaciones paralelas, basada en arrays distribuidos. Introduce abstracciones para la partición y mapeo transparente de arrays, así como para construir patrones de comunicación flexibles que se adaptan a la partición de forma automática.

En este trabajo presentamos la metodología y técnicas de Hitmap aplicadas a la simulación de agentes, utilizando MARL-Ped como caso de estudio. Mostramos conceptual y experimentalmente las ventajas de aplicar Hitmap para aumentar la productividad de este tipo de aplicaciones, tanto en adaptabilidad como en rendimiento, permitiendo agrupar agentes en procesos y reduciendo los costes de comunicación y sobrecargas de forma transparente.

Palabras clave— Agentes, simulación de multitudes, paso de mensajes, herramientas de programación, arrays distribuidos

I. INTRODUCCIÓN

Los sistemas Multi-agente [1], [2] están basados en la construcción de procesos computacionales independientes denominados agentes que tienen la capacidad de percibir el entorno y, de manera autónoma, tomar decisiones acerca de las actividades a llevar a cabo para conseguir sus objetivos. Este tipo de software es especialmente interesante para estudiar sistemas complejos, como la dinámica de los peatones, donde la interacción autónoma de los individuos genera comportamientos globales del sistema. MARL-Ped [3] es un modelo de peatones Multi-agente distribuido donde cada agente (peatón) aprende un comportamiento propio, que permite

simular a grupos de peatones (desde unos pocos hasta multitudes) en diferentes escenarios como congestiones, cruces, avance en colas etc. La cantidad de agentes necesaria para simular muchedumbres, así como la gran carga computacional que implica tanto el proceso de aprendizaje de los agentes como la simulación de su comportamiento y del entorno, hacen de MARL-Ped un buen ejemplo de aplicación paralela apropiada para entornos de alto rendimiento. La versión actual de MARL-Ped utiliza el estándar de paso de mensajes MPI para su explotación en sistemas distribuidos de forma portable. Programar utilizando directamente sistemas de paso de mensajes requiere un gran esfuerzo si se desean introducir políticas de reparto de carga flexibles y que se adapten a la plataforma de destino.

Por otra parte, Hitmap [4] es una biblioteca de funciones diseñada para facilitar la programación de aplicaciones paralelas, basada en arrays distribuidos. Introduce abstracciones para la partición y mapeo automático de arrays, así como para construir patrones de comunicación flexibles que se adaptan a la partición de forma automática.

En este trabajo presentamos la metodología y técnicas de Hitmap aplicadas a la simulación de agentes, utilizando MARL-Ped como caso de estudio. Mostramos las ventajas de aplicar Hitmap para aumentar la productividad de este tipo de aplicaciones, tanto en esfuerzo de desarrollo, como en adaptabilidad y rendimiento. Los resultados muestran que la versión de MARL-Ped utilizando Hitmap es más simple en términos de código, y a la vez más flexible. Hitmap permite de forma transparente agrupar agentes en procesos para reducir los costes de comunicación y evitar sobrecargas, permitiendo una mayor escalabilidad y una explotación más eficiente de los recursos de cómputo.

El resto del artículo se estructura de la siguiente forma. La sección II presenta trabajo relacionado. La sección III describe los fundamentos de MARL-Ped y Hitmap. La sección IV describe la aplicación de Hitmap al programa original. La sección V presenta un estudio experimental en términos de esfuerzo de desarrollo y escalabilidad de la solución conseguida. Finalmente la sección VI presenta las conclusiones y trabajo futuro.

¹Dpto. de Informática, Univ. Valladolid, e-mail: eduardo@infor.uva.es

²Dpto. de Informática, Univ. Valencia, e-mail: francisco.martinez-gil@uv.es

³Dpto. de Informática, Univ. Valencia, e-mail: juan.orduna@uv.es

⁴Dpto. de Informática, Univ. Valladolid, e-mail: arturo@infor.uva.es

II. TRABAJO RELACIONADO

Los estudios de dinámicas de peatones se han desarrollado a lo largo de los últimos 75 años. Aunque la aparición de los primeros modelos de peatones basados en modelos físicos de fluidos y las ecuaciones cinéticas de gases son de los años 60 del siglo XX, el verdadero desarrollo de modelos aconteció con el auge del uso de computadores de bajo coste a partir de la década de los 80 del pasado siglo. Entre los modelos de peatones que más éxito han tenido entre la comunidad científica debido a su versatilidad y simplicidad se encuentra el modelo de fuerzas sociales [5], los modelos basados en autómatas celulares [6], los modelos continuos basados en ecuaciones de cinética de gases [7] y los basados en agentes [8]. Dentro de este último grupo, en los últimos años se han realizado trabajos para incorporar técnicas de aprendizaje máquina (machine learning) a este campo [9]. Los sistemas de aprendizaje aplicado al modelado de peatones tiene características muy interesantes que los señalan como una alternativa a los sistemas más clásicos indicados anteriormente. Entre ellas la más destacada consiste en que es el propio agente o peatón el que aprende su comportamiento, liberando al programador de esta tarea. Siendo la dinámica de grupos de peatones un problema complejo, esta tarea se ha convertido en la dificultad principal de todo modelo de peatones.

Por otra parte, a nivel computacional la simulación microscópica de peatones en escenarios de grandes densidades (multitudes) constituyen un desafío en el que son necesarias nuevas estrategias de procesamiento paralelo y organización y paso de información. En este sentido, en el trabajo de Lozano et al. [10] se propone una arquitectura escalable basada en un sistema cliente-servidor jerárquico que soporta del orden de miles de agentes. En los trabajos posteriores [11], [12], [13] se propone una arquitectura paralela para simulación de muchedumbres en la que los servidores interconectados comparten la carga computacional proveniente de dos cuellos de botella: el servidor de acciones y la base de datos que representa el terreno. En el trabajo de Yilmaz et al. [14] se propone una arquitectura basada en CUDA con lógica difusa que se usó para simular un maratón con más de un millón de corredores.

Hitmap ofrece una capa de abstracción intermedia entre el manejo manual en paso de mensajes de estructuras distribuidas y los lenguajes PGAS (Partitioned Global Address Space), como Chapel [15], o UPC [16]. Estos modelos no proporcionan suficientes herramientas para permitir el paralelismo de procesos jerárquicos en entornos híbridos como Hitmap. Hitmap permite además construir patrones de comunicación reutilizables en tiempo de ejecución que se adaptan a la partición de datos, generando un número mínimo de comunicaciones agregadas. Esto permite conseguir, por ejemplo, una eficiencia comparable a UPC reduciendo incluso la complejidad de programación [4]. Hitmap se utiliza como capa de ejecución en el sistema de programación paralela

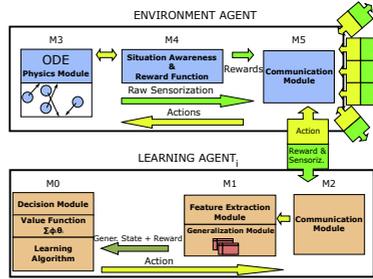


Fig. 1. Esquema de los tipos de agentes y su relación en MARL-Ped.

Trasgo [17], que ofrece una aproximación similar a los lenguajes PGAS. Hitmap extiende las funcionalidades de creación de jerarquías y de reparto de datos de otras bibliotecas o modelos de arrays distribuidos, por ejemplo HTAs [18] o Parray [19]. Entre otras cosas permite la utilización de políticas de partición transparentes e intercambiables, regulares o irregulares, definidas como módulos con un interfaz común. Esto elimina la necesidad de tomar decisiones sobre la granularidad y sincronización en diferentes niveles jerárquicos. Hitmap ha sido también extendido para soportar estructuras de datos como matrices dispersas o grafos, usando la misma metodología e interfaz [20].

III. MARL-PED Y HITMAP

A. MARL-Ped

MARL-Ped es un sistema multiagente para modelado y simulación de peatones que utiliza aprendizaje por refuerzo (reinforcement learning, RL) para aprender el comportamiento individual de cada peatón (agente) en un determinado escenario (congestión, evacuación de un recinto cerrado, circulación por un pasillo con flujos de movimiento contrarios,...). El objetivo del algoritmo de aprendizaje RL consiste en calcular una función de control (llamada función de valor en el campo de RL) que indicará al peatón qué acción tomar en cada momento en función del estado local percibido. Existen en MARL-Ped dos clases de agentes: los agentes peatón, que ejecutan los algoritmos de aprendizaje por refuerzo y almacenan la función de control aprendida, y el agente entorno que ejecuta el sistema físico que simula a los peatones en el escenario y que sensoriza el estado en el que se encuentra cada peatón dentro de este escenario. El escenario es un mundo virtual 3D en el que el motor físico Open Dynamic Engine (ODE) se encarga de simular las colisiones y las fuerzas que mueven a los peatones. En la Figura 1 aparece una descripción gráfica del sistema.

Es posible observar en la Figura 1 que tanto los agentes peatones (en la figura "Learning Agents") como el agente entorno (en la figura "Environment

agent”) están compuestos de diferentes módulos funcionales. Existen dos modos de funcionamiento en MARL-Ped: modo de aprendizaje y modo de simulación. Ambos modos establecen el mismo tipo de comunicación entre los agentes y el entorno. La única diferencia es que en el modo de aprendizaje los algoritmos de RL están activos y la función de control se va calculando incrementalmente.

El modo de funcionamiento es síncrono y consiste en un ciclo clásico compuesto de observación-acción-recompensa. Concretamente el ciclo se compone de los siguientes pasos:

1. El agente entorno consulta a ODE la situación dinámica de cada peatón que consiste en posición, velocidad, distancia a los n peatones más cercanos y distancia a los n objetos más cercanos. Si estamos en el modo aprendizaje, el agente entorno prepara una recompensa para cada uno de los agentes peatones en función de varios hechos: que el peatón haya llegado a su objetivo, que se haya chocado contra otro peatón u objeto, etc.
2. El agente entorno transmite la información del estado y la recompensa de cada agente al sistema para que sea recogida por los agentes.
3. Cada agente recibe la información del entorno y prepara con ella las características específicas que definen el estado para el algoritmo de aprendizaje y la señal de recompensa. Ambas informaciones son usadas por el algoritmo de RL para modificar la función de control que también está gestionada por el agente peatón (“Learning agent” en la Figura 1). En el modo de simulación, la información del estado sirve para consultar directamente a la función de control mientras que los algoritmos de RL están desactivados.
4. El agente consulta a la función de control la nueva acción que, en el estado actual, se debe de ejecutar. Las acciones consisten en una modificación de la dirección y rapidez con la que se mueve el peatón.
5. Los agentes transmiten las acciones al agente entorno que es el encargado de traducirlas a acciones físicas que son ejecutadas por ODE en el entorno virtual.

La comunicación de datos en MARL-Ped se establece entre el agente entorno y los agentes peatones, no habiendo comunicación entre los agentes peatones. Existe un doble flujo de datos, que se puede observar en la Figura 1. Un flujo comunica los datos de sensorización del estado de cada agente y la correspondiente recompensa desde el entorno hasta cada uno de los agentes (flechas de color verde). Otro flujo comunica la siguiente acción a realizar por cada uno de los agentes al entorno (flechas de color amarillo). Este ciclo descrito se ejecuta un número de veces que es a su vez un parámetro de configuración del sistema. En el modo aprendizaje con decenas de agentes, este ciclo puede repetirse desde

cientos de miles de veces hasta millones de veces.

B. Hitmap

Hitmap [4] es una biblioteca de funciones para la gestión y distribución jerárquica de estructuras de datos en tiempo de ejecución. Trabaja sobre arrays densos y ha sido también extendida para soportar estructuras de datos como matrices dispersas o grafos, usando la misma metodología e interfaz [20]. Se basa en un modelo SPMD (Single Program Multiple Data) y en el paradigma de paso de mensajes. Hitmap define varias abstracciones para escribir programas paralelos que manejan estructuras de datos distribuidas. La biblioteca se puede dividir en tres partes:

B.1 Métodos de tiling

Definición y manipulación de arrays y tiles. Estos métodos se pueden usar de manera independiente al resto de funcionalidades de Hitmap, para mejorar la localidad en el código secuencial, así como para generar distribuciones de datos manualmente para la ejecución en paralelo. Se definen objetos para representar los dominios de índices de las estructuras de datos de forma compacta. Se define una clase de objetos denominada *Tile* que representa la asociación entre elementos del espacio de índices y los datos, y permiten el acceso o modificación de los mismos. Los tiles pueden crearse de forma que en un proceso sólo se dispone localmente de una parte del espacio de índices obteniéndose la estructura distribuida.

B.2 Métodos de mapping

Incluye módulos intercambiables que implementan políticas para particionar dominios automáticamente, dependiendo de una topología virtual seleccionada en tiempo de ejecución. Permiten crear objetos denominados *Layouts* que pueden ser consultados respecto a la parte de un dominio de índices que es asignada tanto localmente, como o a cualquier otro proceso remoto. Además, se establecen relaciones de vecindad en términos de las reglas de la topología virtual escogida.

B.3 Métodos de comunicaciones

Estas funciones son una abstracción del modelo de paso de mensajes para comunicar tiles o partes de tiles entre procesadores virtuales. Permite crear objetos que almacenan la información necesaria para empaquetar e intercambiar los datos seleccionados entre procesadores. Hitmap provee de múltiples interfaces de creación que implementan diversos tipos de comunicaciones punto-a-punto, colectivas, o patrones más complejos compuestos de múltiples comunicaciones de diversa naturaleza o sobre diversos tiles. Las interfaces de creación incluyen un objeto de tipo *Layout*, que es consultado internamente para decidir qué tiene que comunicarse y a quién, generándose automáticamente la información interna necesaria. Por tanto, estos objetos se adaptan transparentemente en el momento de su creación a la plataforma de ejecución y la distribución de datos

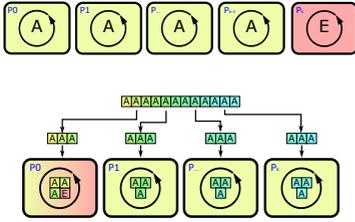


Fig. 2. Estructura general de la simulación y el reparto de tareas entre procesadores en MARL-Ped original (arriba) y tras aplicar Hitmap (abajo).

utilizada. Los objetos de comunicación son reutilizables. El método que realmente ejecuta la comunicación se puede invocar en cualquier momento después de su creación, tantas veces como sea necesario a lo largo de la aplicación. Internamente se basan en el estándar MPI, utilizando técnicas eficientes como la creación de tipos de datos derivados, comunicaciones asíncronas, etc.

IV. APLICACIÓN DE LA METODOLOGÍA Y TÉCNICAS DE HITMAP

En esta sección se describe cómo aplicar la metodología y técnicas propias de Hitmap a aplicaciones de simulación basadas en agentes, utilizando MARL-Ped como caso de estudio.

A. Cambios estructurales

La estructura de la aplicación MARL-Ped ha sido rediseñada. La versión de Hitmap aplica el concepto de arrays distribuidos para agrupar agentes en procesos, en lugar de utilizar un proceso MPI para cada agente, más otro para la simulación del entorno. En la parte superior de la figura 2 se aprecia la distribución conceptual de la computación en MARL-Ped original. Cada proceso ejecuta el código de un agente. El último proceso ejecuta el cómputo de la simulación del entorno. Los objetos de clase *RLAgent* y *REnvironment* tienen un método que internamente ejecuta repetitivamente el bucle de simulación.

Una primera decisión de diseño para la nueva versión es repartir los agentes entre los procesos disponibles sin reservar un proceso para el entorno. El código del entorno lo ejecutará uno de los procesos que también tiene agentes asignados ya que la computación principal de los agentes y del entorno se alternan sin solaparse en el tiempo.

Hitmap provee de las herramientas necesarias para distribuir equitativamente los agentes entre los procesos disponibles (ver la parte de abajo en la figura 2). Cada proceso tiene que poder ejecutar en cada iteración de simulación el código de varios agentes, y en el caso del proceso designado para ello, además ejecutar el código del entorno. Por tanto, el bucle de simulación no puede estar dentro de los ob-

jetos de agente o entorno. Es necesario rediseñar la aplicación para que el bucle de simulación lo ejecute el programa principal, que debe iterar a su vez dependiendo del número de agentes asignados al proceso. Para ello se elimina el bucle de simulación dentro del correspondiente método de la clase de agentes y entorno. Se transforman los métodos a los que se invocaba dentro de ese bucle en métodos públicos. Y se traslada la lógica de control de dichas invocaciones al nuevo bucle de iteraciones de simulación en el programa principal. Se realiza lo mismo con el entorno, rodeando la lógica de control del mismo, ahora en el programa principal, con un condicional para que sólo sea ejecutado por un proceso. Hitmap distingue para cada grupo de procesos a uno de ellos como el líder del grupo. Este proceso puede identificarse a sí mismo a través de una llamada a una función, y por tanto es el seleccionado para ejecutar la lógica del entorno.

B. Arrays distribuidos y patrones de comunicación

Las comunicaciones basadas en MPI de MARL-Ped original se han substituido por el manejo de arrays distribuidos con Hitmap. Para ello, las estructuras de datos implicadas en las comunicaciones se substituyen por estructuras de tipo *HitTile*. La estructura *HitTile* debe especializarse al comienzo del programa según los diferentes tipos de datos de cada array que se vaya a declarar y manejar con Hitmap.

En fase de inicialización del programa se crean los arrays distribuidos y objetos de tipo *HitCom* con las especificaciones de las comunicaciones que serán invocadas en las iteraciones del bucle de simulación. Para las señales de control sólo es necesaria una variable de tipo entero en cada proceso, independientemente del número de agentes que se le asignen. Para cada flujo de datos entre el agente que gestiona el entorno y los agentes peaton, se declaran dos arrays distribuidos con un dominio de índices igual al número de agentes peatón. En uno se utiliza una política de distribución que reparte y asigna los elementos de forma equitativa entre los procesos. En el otro se utiliza una política que distribuye todos los elementos al proceso que ejecuta el entorno. Hitmap permite con una única llamada a una función construir un objeto *HitCom* que implementa un patrón que redistribuye los datos desde un array distribuido con una política cualquiera, a los correspondientes elementos locales o remotos de otro array con el mismo dominio, pero distribuido con otra política diferente. Esta técnica permite construir objetos de comunicación que moverán de forma transparente los datos entre las dos copias de cada array, la realmente distribuida y la que tiene todo el dominio en el proceso del entorno. El patrón de comunicación se adapta a los resultados de las políticas de partición independientemente del número de agentes y procesos. Este mecanismo soluciona de una forma única la construcción de los flujos de comunicación necesarios.

V. ESTUDIO EXPERIMENTAL

Esta sección describe el estudio experimental realizado para comprobar las ventajas de aplicar Hitmap en programas de simulación basados en agentes como MARL-Ped. El estudio se focaliza en mostrar la mayor capacidad de escalado en cuanto a número de agentes, y la predictibilidad de los parámetros de ejecución para obtener un mayor rendimiento.

A. Metodología de experimentación

En este estudio experimental se obtienen medidas de tiempos de ejecución de las dos versiones de código; MARL-Ped original y la versión utilizando Hitmap. Se presentan los mismos en términos de escalabilidad.

En este artículo nos focalizamos en la parte del proceso de aprendizaje, que es computacionalmente la más costosa, y que no implica operaciones de entrada/salida durante la fase de computación/comunicación. Los códigos han sido instrumentados para medir el tiempo de ejecución de cada proceso distribuido desde el momento en que comienza la inicialización de estructuras relacionadas con el paralelismo (MPI o Hitmap), hasta el momento en que termina la ejecución del aprendizaje, justo antes de comenzar a escribir en ficheros los resultados. De las mediciones obtenidas en cada proceso, se utiliza como resultado de la medida el tiempo mayor, es decir, el del proceso que ha tardado más en terminar. Cada experimento se repite varias veces para poder comprobar también la variabilidad en los resultados.

Dado que los tiempos de ejecución de un entrenamiento completo son extremadamente largos, para poder explorar un espacio de búsqueda amplio en cuanto a parámetros de ejecución, se ha limitado el programa a la ejecución de sólo 100 iteraciones de entrenamiento en todos los casos. Como se verá en los resultados, este número de iteraciones produce una carga y un número de fases de comunicación y sincronización suficientemente representativos. En todos los casos se ejecuta un escenario de ejemplo utilizado y validado en trabajos anteriores [3]. El escenario reproduce un dilema de navegación clásico en dinámica de peatones denominado “shortest path vs quickest path”. En este escenario un grupo de peatones debe pasar de una habitación a un lugar objetivo situado fuera de ésta. Existen dos salidas y una está mas cerca del objetivo que la otra. Los agentes deben aprender que si todos intentan salir por la puerta más cercana, se formará una aglomeración que ralentizará la evacuación. Una solución óptima es la división del grupo en peatones que eligen la salida con el camino más corto y agentes que eligen la otra salida, lo que proporcionará una evacuación más rápida.

En la configuración seleccionada se han situado 28 agentes en una habitación rectangular de longitud de 18 m. con dos salidas posibles de 1 m.de anchura que dificulta el paso simultáneo de más de un peatón. El objetivo de los agentes es alcanzar un punto de

encuentro al otro lado de las salidas.

Los códigos se han ejecutado en diferentes plataformas multicore donde las comunicaciones son menos costosas y destacan más los potenciales overheads asociados, entre otras cosas, a los cambios en la estructura de ejecución, al manejo de las estructuras propias de Hitmap, o a los cálculos y decisiones sobre las comunicaciones.

En este trabajo mostramos resultados obtenidos en una máquina denominada Chimera. Tiene dos CPUs Intel E5-2620 v2, a 210 GHz, con un total de 12 cores reales, con la opción de *hyperthreading* activada. Dispone de 8 Gb de memoria DDR4. El sistema operativo es CentOS 7.0 1406 x64. El compilador utilizado es GCC 4.8.2 con el flag de optimización -O3. La implementación de MPI utilizada es Mpiich 3.1.3.

B. Escalabilidad en número de agentes

Uno de los objetivos de este trabajo es conseguir una mayor escalabilidad en términos de número de agentes, gracias a la estrategia de asociar varios agentes a un mismo proceso con Hitmap. El primer estudio realizado comprueba experimentalmente la diferencia de escalabilidad entre la versión original de MARL-Ped y la versión que utiliza Hitmap. Los programas se han ejecutado en fase de entrenamiento fijando el número de procesos MPI al número de cores disponibles en la máquina, aumentando progresivamente el número de agentes por encima de ese número.

Los resultados se muestran en la figura 3. En el caso de MARL-Ped original, el número de procesos MPI necesarios crece con el número de agentes (siendo el número de agentes más uno). La estructura de estas aplicaciones de simulación, que utilizan comunicaciones colectivas con puntos claros de sincronización global alrededor de la ejecución del motor de simulación, no presentan la propiedad de *parallel slackness*, que aparece en ciertas aplicaciones cuando varios procesos asignados al mismo elemento de proceso solapan alternativamente fases de computación y comunicación. En los resultados se observa cómo el sobrecoste de realizar *oversubscribing* (lanzar más procesos que elementos de proceso disponibles) tiene un impacto negativo en el rendimiento y la escalabilidad de la aplicación. En la máquina Chimera, con 12 cores y que tiene activado *hyperthreading*, el efecto es mucho más notable a partir de 24 agentes, donde MARL-Ped ya utiliza 25 procesos MPI. La forma en los que los procesos se rotan en los planificadores adquiere relevancia. Por ello se observan mayores diferencias entre los tiempos de ejecución mínimos y máximos. La versión que utiliza Hitmap, mantiene una buena escalabilidad, ya que limita el número de procesos y ejecuta secuencialmente dentro de cada uno el código de varios agentes de una forma más eficiente.

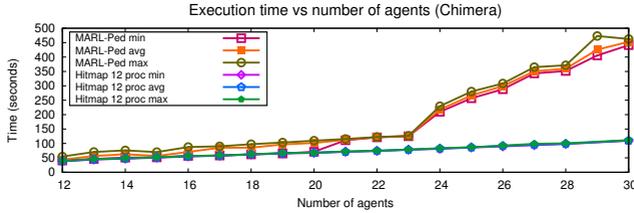


Fig. 3. Tiempos de ejecución en relación al número de agentes. La gráfica muestra el tiempo de ejecución de 100 iteraciones de entrenamiento para un número creciente de agentes. En el caso MARL-Ped original el número de procesos crece con el número de agentes. En el caso de MARL-Ped+Hitmap el número de procesos MPI es fijo, e igual al número de elementos de proceso (cores) reales disponibles.

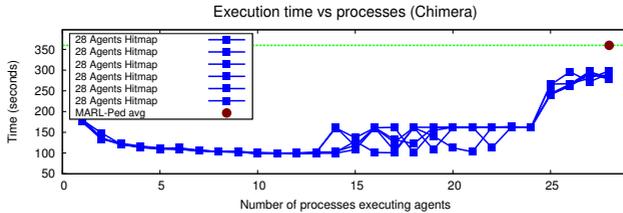


Fig. 4. Tiempos de ejecución en relación al número de procesos para un número de agentes fijo. La gráfica muestra el tiempo de ejecución de 100 iteraciones de entrenamiento. En el caso de MARL-Ped original el número de procesos MPI depende por diseño del número de agentes y es igual a 29. Para MARL-Ped+Hitmap se muestran varias líneas correspondientes a varios experimentos consecutivos, para mostrar la variabilidad de los resultados.

C. Impacto del número de procesos

En esta sección se estudia el impacto de modificar el número de procesos, y por tanto la distribución de agentes por proceso, en MARL-Ped+Hitmap. En la figura 4 se observan los resultados obtenidos para 28 agentes. En todos los casos los resultados en tiempos de ejecución de MARL-Ped+Hitmap son mejores que en MARL-Ped original debido al efecto de oversubscription comentado en la sección anterior.

Se observa que los resultados utilizando Hitmap mejoran ligeramente al ir repartiendo los agentes entre un número pequeño pero creciente de procesos, ya que aumenta el paralelismo. Al superar el número de procesos al número de cores reales (sin contar con hyperthreading), los resultados empeoran y se vuelven más inestables, ya que aparecen efectos negativos derivados del oversubscribing. Las políticas de planificación comienzan también a hacer los resultados más impredecibles. En algunos casos se obtiene tiempos tan buenos como antes de comenzar el oversubscribing, pero en muchos otros casos empeoran. Al superar el límite del número de threads disponibles contando con hyperthreading los resultados, como era de esperar empeoran notablemente.

Estos resultados indican que en MARL-Ped+Hitmap el número de procesos a escoger para el entrenamiento es predecible. Los tiempos

de ejecución son menores y más estables cuando se utiliza un número de procesos igual al número de elementos de proceso reales, sin tener en cuenta la opción de hyperthreading.

VI. CONCLUSIONES

En este artículo se presenta la aplicación de las técnicas y herramientas de la biblioteca de manejo de array distribuidos Hitmap a la simulación basada en agentes. Se utiliza la aplicación multi-agente para simulación de peatones MARL-Ped como caso de estudio. Se muestra cómo la utilización de arrays distribuidos y políticas de partición automáticas permite obtener una mayor productividad y escalabilidad, gracias a la capacidad de Hitmap de distribuir la carga entre procesos de forma transparente al programador. MARL-Ped+Hitmap permite hacer simulaciones con un número mucho mayor de agentes manteniendo tiempos de ejecución estables y predecibles.

El trabajo futuro incluye extender la aplicación de Hitmap a otros tipos de aplicaciones de simulación relacionadas; investigar la potencial eliminación de cuellos de botella en las fases de simulación; y utilizar la nueva versión de MARL-Ped+Hitmap para profundizar en el estudio de la calidad y resultados de las simulaciones de muchedumbres con un número mucho mayor de agentes.

AGRADECIMIENTOS

Esta investigación ha sido parcialmente financiada por el Ministerio de Economía y Competitividad (Spain) y el programa ERDF de la Unión Europea: Proyecto HomProg-HetSys TIN2014-58876-P, Proyecto TIN2015-66972-C5-5-R y COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

REFERENCIAS

- [1] M.J. Wooldridge and N.R. Jennings, "Intelligent agents: theory and practice," *The Knowledge Engineering Review*, vol. 10, pp. 115–152, 1995.
- [2] M. Wooldridge, *Multi-Agent Systems 2nd Edition*, chapter Intelligent Agents, pp. 3–50, MIT Press, 2013.
- [3] Francisco Martínez-Gil, Miguel Lozano, and Fernando Fernández, "MARL-ped: A multi-agent reinforcement learning based framework to simulate pedestrian groups," *Simulation Modelling Practice and Theory*, vol. 47, pp. 259–275, 2014.
- [4] A. Gonzalez-Escribano, Y. Torres, J. Fresno, and D.R. Llanos, "An extensible system for multilevel automatic data partition and mapping," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1145–1154, 2014.
- [5] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Phys. Rev. E*, vol. 51, pp. 4282–4286, 1995.
- [6] V. J. Blue and J. L. Adler, "Cellular automata microsimulation for modeling bi-directional pedestrian walkways," *Transportation Research Part B: Methodological*, vol. 35, no. 3, pp. 293–312, 2001.
- [7] R. L. Hughes, "The flow of human crowds," *Annu. Rev. Fluid Mech.*, vol. 35, pp. 169–182, 2003.
- [8] C. Reynolds, "Steering behaviors for autonomous characters," in *Game Developers Conference*, San Francisco, California., 1999, pp. 763–782, Miller Freeman Game Group.
- [9] Francisco Martínez-Gil, Miguel Lozano, and Fernando Fernández, "Multi-agent reinforcement learning for simulating pedestrian navigation," in *Adaptive and Learning Agents: International Workshop, ALA 2011, Held at AAMAS 2011, Taipei, Taiwan*, 2012, pp. 54–69, Springer Berlin Heidelberg.
- [10] M. Lozano, P. Morillo, J. M. Orduña, V. Caverio, and G. Viguera, "A new system architecture for crowd simulation," *J. Network and Computer Applications*, vol. 32, no. 2, pp. 474–482, 2009.
- [11] G. Viguera, M. Lozano, J. M. Orduña, and F. Grimaldo, "A comparative study of partitioning methods for crowd simulations," *Appl. Soft Comput.*, vol. 10, no. 1, pp. 225–235, Jan. 2010.
- [12] Guillermo Viguera, Juan M. Orduña, and Miguel Lozano, "A read-copy update based parallel server for distributed crowd simulations," *The Journal of Supercomputing*, vol. 64, no. 1, pp. 156–166, 2013.
- [13] Guillermo Viguera, Juan M. Orduña, Miguel Lozano, and Yvon Jégou, "A scalable multiagent system architecture for interactive applications," *Science of Computer Programming*, vol. 78, no. 6, pp. 715–724, 2013, Special section: The Programming Languages track at the 26th {ACM} Symposium on Applied Computing (SAC 2011); Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments.
- [14] E. Yilmaz, V. Isler, and Y. Y. Cetin, "The virtual marathon: Parallel computing supports crowd simulations," *IEEE Computer Graphics and Applications*, vol. 29, no. 4, pp. 26–33, 2009.
- [15] B.L. Chamberlain, D. Callahan, and H.P. Zima, "Parallel programmability and the chapel language," *Int. J. High Perform. Comput. Appl.*, vol. 21, no. 3, pp. 291–312, aug 2007.
- [16] D. A. Mallón, A. Gómez, J. C. Mourriño, G. L. Taboada, C. Teijeiro, J. Touriño, B. B. Fraguela, R. Doallo, and B. Wibecan, "Upc performance evaluation on a multi-core system," in *Proceedings of the Third Conference on Partitioned Global Address Space Programming Models*, New York, NY, USA, 2009, PGAS '09, pp. 9:1–9:7, ACM.
- [17] Arturo Gonzalez-Escribano and Diego R. Llanos, "Trasgo: a nested-parallel programming system," *The Journal of Supercomputing*, vol. 58, no. 2, pp. 226–234, 2011.
- [18] Basilio B. Fraguela, Ganesh Bikshandi, Jia Guo, María J. Garzarán, David Padua, and Christoph Von Praun, "Optimization techniques for efficient hta programs," *Parallel Comput.*, vol. 38, no. 9, pp. 465–484, sep 2012.
- [19] Yifeng Chen, Xiang Cui, and Hong Mei, "Parray: A unifying array representation for heterogeneous parallelism," *SIGPLAN Not.*, vol. 47, no. 8, pp. 171–180, feb 2012.
- [20] J. Fresno, A. Gonzalez-Escribano, and D.R. Llanos, "Blending extensibility and performance in dense and sparse parallel data management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2509–2519, 2014.

Migración portable y de altas prestaciones de aplicaciones MATLAB a C++: deconvolución esférica de datos de resonancia magnética por difusión

Javier García Blas¹, J. Daniel García¹, Manuel F. Dolz¹, Jesús Carretero¹, Yasser Alemán² y Erick Jorge Canales-Rodríguez³

Resumen— En muchos de los campos de la investigación científica, se ha establecido Matlab como herramienta *de facto* para el diseño de aplicaciones. Esta aproximación ofrece muchas ventajas como el rápido despliegue de prototipos, alto rendimiento en álgebra lineal, entre otros. Sin embargo, las aplicaciones desarrolladas son altamente dependientes del motor de ejecución de Matlab, limitando su despliegue en multitud de plataformas de altas prestaciones.

En este trabajo presentamos un caso práctico de migración de una aplicación inicialmente basada en Matlab a una aplicación nativa en lenguaje C++. Para ello se presentará la metodología empleada para la migración y las herramientas que facilitan esta tarea. La evaluación llevada a cabo demuestra que la solución implementada ofrece un buen rendimiento sobre distintas plataformas y sistemas altamente heterogéneos.

Palabras clave— Matlab, Imagen por Resonancia Magnética, Álgebra lineal.

I. INTRODUCCIÓN

DESPUÉS de décadas de evolución en el campo de la Imagen por Resonancia Magnética (IRM), la implementación de una gran variedad de métodos avanzados y distintas modalidades de procesamiento de imagen ha servido para mostrar los complejos patrones de organización del cerebro, presentes tanto a escala micro como macroscópica. La organización estructural de la sustancia blanca del cerebro se puede caracterizar con gran detalle mediante los métodos de reconstrucción intravoxel basados en datos de resonancia magnética por difusión. Este tipo de técnica permite estudiar los patrones de conectividad estructural del cerebro de forma no invasiva e *in vivo*. El estudio de la conectividad ofrece múltiples ventajas en el diagnóstico de enfermedades mentales como la esquizofrenia y trastornos bipolares, entre otras.

Del conjunto de métodos y algoritmos propuestos hasta la fecha para el procesamiento de este subconjunto de imagen médica, podemos destacar la colección de rutinas de MATLAB HARDI-Tools¹, dedicada al análisis de datos de difusión de alta resolución angular (del inglés, High Angular Resolution Diffusion

Imaging, HARDI). HARDI es un conjunto de código implementado y utilizado por investigadores de la Fundación de Investigación y Docencia FIDMAG (Barcelona, España), el Centro de Neurociencias de Cuba (La Habana, Cuba) y el Laboratorio de Imagen Médica del Hospital Gregorio Marañón (Madrid, España).

Por otro lado, desde décadas, el uso del lenguaje MATLAB se ha convertido en el estándar *de facto* para el diseño y el prototipado en una amplia variedad de aplicaciones de ciencia e ingeniería. Debido a su sencillez de programación, la comunidad científico-técnica ha optado por utilizar este lenguaje de alto nivel para el desarrollo de prototipos de aplicaciones que requieren el cómputo de problemas de álgebra lineal. Por contrapartida, muchos de estos prototipos han pasado a una etapa de producción sin haber sido convenientemente adaptados y optimizados para recibir grandes cargas de trabajo. Por ello, hoy en día se pueden encontrar múltiples ejemplos de aplicaciones MATLAB que se ejecutan en plataformas de producción de altas prestaciones pero que, debido a la naturaleza del propio lenguaje, no aprovechan correctamente los recursos proporcionados por éstas.

El objetivo de este trabajo es presentar las tecnologías y herramientas actuales para la migración de aplicaciones basadas en MATLAB. En particular, en este trabajo estudiaremos el método de deconvolución esférica RUMBA-SD (del inglés, Robust and Unbiased Model-BASED Spherical Deconvolution, RUMBA-SD [1]), debido a que esta técnica ha demostrado estar entre los métodos más avanzados hasta la fecha para inferir los cruces de fibras de la sustancia blanca, obteniendo el primer lugar en un concurso internacional de reconstrucción de imágenes².

Con el fin de mantener la compatibilidad y la solidez de esta aplicación, hemos implementado una versión equivalente C++ usando las bibliotecas Armadillo [2] y ArrayFire. Estas bibliotecas permiten representar las operaciones de álgebra lineal comunes como operadores aritméticos básicos de C++. Otra característica interesante es que este tipo de soluciones proporcionan compatibilidad con bibliotecas de álgebra lineal existentes, como OpenBlas, Intel MKL

¹Dpto. de Informática, Universidad Carlos III de Madrid, 28911-Leganés, e-mail: fjblas@inf.uc3m.es.

²BiiG, Hospital General Universitario Gregorio Marañón, 28007-Madrid

³Fundación para la Investigación y Docencia FIDMAG, CIBERSAM, 08035-Barcelona.

¹http://neuroimagen.es/webs/hardi_tools/

²Para más información ver: http://hardi.epfl.ch/static/events/2013_ISBI/

o ATLAS. Por lo tanto, nuestra solución puede beneficiarse de múltiples arquitecturas multi-núcleo, simplemente mediante el despliegue de múltiples hilos paralelos.

Las contribuciones de este trabajo son las siguientes. Primero, presentamos la implementación de un de los métodos de reconstrucción incluidos en HARDI llamado *RUMBA-SD*. Segundo, indicamos los pasos necesarios y herramientas empleadas para este propósito. Finalmente, a través de una evaluación exhaustiva, demostramos qué bibliotecas son más adecuadas para la migración de aplicaciones basadas en el motor de ejecución MATLAB.

El resto del trabajo está estructurado de la siguiente forma. En la Sección II se introducen las herramientas usadas en este trabajo. La Sección III contextualizamos el problema del procesamiento de *RUMBA-SD*. A continuación, en la Sección IV detallamos la metodología aplicada para la paralelización de una de las herramientas incluidas en HARDI. En la Sección V presentamos resultados de la evaluación realizada. Trabajos relacionados con nuestra solución son presentados en la Sección VI. Finalmente, en la Sección VII discutimos sobre los resultados arrojados en este trabajo.

II. ÁLGEBRA LINEAL PARA ENTORNOS BASADOS EN C++

En esta sección resumiremos las principales herramientas que existen en la literatura para el procesamiento de álgebra lineal para el lenguaje de programación C++. Destacamos entre las opciones dos aproximaciones: Armadillo y ArrayFire.

Armadillo [2] es una biblioteca de álgebra lineal para el lenguaje C++, diseñada con el objetivo de alcanzar un buen equilibrio entre la velocidad y facilidad de uso. Dentro de las principales ventajas, destacamos la sintaxis de alto nivel (API) similar a MATLAB. Armadillo es útil para el desarrollo de algoritmos directamente en C++, o la conversión rápida de código para entornos de producción.

En el siguiente fragmento de código se muestra un ejemplo de como se implementa una multiplicación de matrices con Armadillo:

```
#include <iostream>
#include <armadillo>

using namespace std;
using namespace arma;

int main(int argc, char** argv)
{
    mat A = randu<mat>(5000,5000);
    mat B = randu<mat>(5000,5000);
    mat C = A * B;
    return 0;
}
```

ArrayFire [3] es una biblioteca de funciones de código abierto con interfaces para C, C++, Java, R y Fortran. Se integra con cualquier aplicación CUDA, y contiene una API que facilita la programación. La biblioteca ArrayFire está diseñada para su uso en un amplio rango de sistemas, desde los sistemas compuestos por una sola GPU a grandes supercomputadores multi-GPU. En el siguiente ejemplo se mues-

tra un caso similar al implementado anteriormente en Armadillo:

```
#include <iostream>
#include <arrayfire.h>

using namespace std;
using namespace af;

int main(int argc, char** argv)
{
    array A, B;
    af_randn(&A,5000,5000,fp32);
    af_randn(&A,5000,5000,fp32);
    array C = matmul(A, B);
    return 0;
}
```

En este caso, tanto las matrices *A* y *B* son entidades abstractas cuya localización vendrá dada por el *back-end* seleccionado para su ejecución. En caso de escoger aceleradores GPU, ambas matrices estarán almacenadas íntegramente en la memoria del dispositivo GPU.

III. INTRAVOXEL FIBER RECONSTRUCTION (RUMBA-SD)

En esta sección introduciremos los conceptos básicos de *RUMBA-SD* para su contextualización y para mostrar el tamaño del problema.

En tejidos biológicos, la difusión del agua no es igual en todas las direcciones debido a la presencia de obstáculos o barreras que limitan el movimiento molecular. Para caracterizar cuantitativamente este fenómeno se usa el tensor de difusión (DTI, del inglés, Diffusion Tensor Imaging). Este tensor se define como una simétrica 3×3 y la matriz positiva definida (mostrada en la Ecuación 1).

$$D = \begin{pmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{yx} & D_{yy} & D_{yz} \\ D_{zx} & D_{zy} & D_{zz} \end{pmatrix} \quad (1)$$

El tensor de difusión puede ser visualizado como un elipsoide, donde las orientaciones de sus tres ejes principales se definen por sus vectores propios o auto-vectores y la longitud de los ejes por la magnitud de sus valores propios o difusividades (Figura 1).

El tensor de difusión resulta ser un modelo adecuado en el caso de que cada voxel del cerebro (pixel tridimensional) contenga un solo grupo de fibras paralelas (ver el panel (a) de la Figura 1). En este caso el eje principal del elipsoide asociado al tensor de difusión coincide con la orientación del grupo de fibras. De igual manera, el tensor de difusión es efectivo para caracterizar el proceso de difusión en regiones del cerebro donde la difusión ocurre sin interactuar con los tejidos y por tanto tiene una geometría esférica (ver el panel (c) de la Figura 1). Sin embargo, en regiones del cerebro donde varias fibras se cruzan e interceptan, el modelo del tensor de difusión no permite inferir las orientaciones de los haces de fibras (ver el panel b de la Figura 1). De hecho, diferentes cruces de fibras podrían dar lugar a tensores de difusión similares (ver Figura 2).

Tales limitaciones en el enfoque DTI han impulsado el reciente desarrollo de numerosos protocolos de

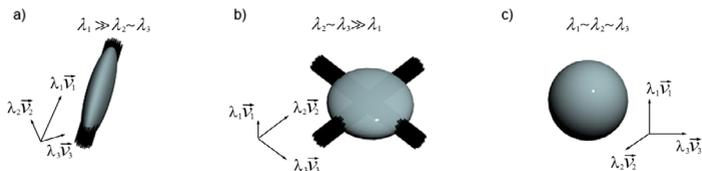


Fig. 1. La organización microestructural puede ser revelada por el elipsoide de tensor de difusión.

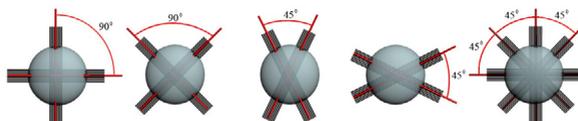


Fig. 2. Diferentes cruces de fibra pueden producir el mismo elipsoide de difusión.

muestreo, modelos de difusión y técnicas de reconstrucción. Entre estos métodos, la técnica RUMBA-SD se basa en asumir que cada voxel está compuesto por un número elevado de tensores de difusión y que cada tensor corresponde a un compartimento que contiene un solo grupo de fibras paralelas con una orientación predefinida. El objetivo del método es estimar la fracción de volumen de cada uno de estos compartimentos para obtener, en cada voxel, la función de distribución de orientación de las fibras (ODF, del inglés, *Oriental Distribution Function*). La Figura 3 muestra un ejemplo de ODF que corresponde al cruce de tres grupos de fibras perpendiculares. Las orientaciones donde la ODF alcanza sus valores máximos se corresponden con las orientaciones de las fibras nerviosas.

Como se muestra en la Figura 3, para facilitar la visualización, se asigna el código de color de la superficie de acuerdo con la dirección de difusión ($[x, y, z] - [r, b, g]$, donde r = rojo representa la probabilidad a lo largo de la orientación de izquierda a derecha, b = azul representa superior-inferior orientación, y g = verde representa la orientación antero posterior).

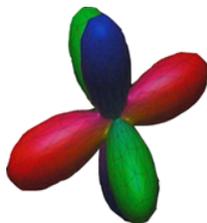


Fig. 3. Función de distribución de orientación.

IV. PARALLEL HARDI

Dentro del conjunto de utilidades de HARDI, en este trabajo nos hemos centrado en la paralelización de RUMBA-SD³. Sin embargo, la incorporación de nuevos métodos resulta sencillo debido al diseño modular de la solución.

Como se observa en la Figura 4, pHARDI posee un diseño basado en capas, el cual permite el uso de distintos aceleradores de álgebra lineal para una amplia gama de dispositivos. Estos dispositivos pueden ser multi-core, dispositivos GPU (tanto para CUDA como OpenCL) o incluso aceleradores basados en tecnología x86 como Intel Xeon Phi. En el caso de no contar con este tipo de dispositivos, nuestra solución soporta el acceso de múltiples bibliotecas optimizadas de cómputo de álgebra lineal. Para ello se han desarrollado dos versiones distintas: una puramente basada en Armadillo y otra con soporte a ArrayFire. Para esta aproximación, únicamente se ha utilizado ArrayFire para aquellos fragmentos de código computacionalmente más caros, también llamados núcleos de ejecución o *kernels*. Es importante destacar que ambas soluciones usan la misma disposición lógica de las matrices o volúmenes, tanto en CPU como en GPUs (*row-major*). Por lo tanto, no son necesarias transformaciones adicionales en el código fuente original.



Fig. 4. Arquitectura software de pHARDI.

pHARDI admite dos modelos de acceso a las imágenes para el procesamiento de datos de entra-

³<https://bitbucket.org/fjblas/phardi>

da. El primer patrón procesa por separado cada rodaja a partir de los volúmenes de datos de entrada ($x \times y \times z$), lo que resulta en un total de z rodajas, cada una compuesta por $x \times y \times t$ voxels, siendo t el número de orientaciones. El segundo diseño procesa todos los píxeles de los volúmenes en una sola matriz. Esta matriz tiene una dimensión de $n \times m$ elementos, donde n corresponde con la cantidad de orientaciones evaluadas (por ejemplo 100) y m con la cantidad de voxels en cada volumen (por ejemplo 125 megapíxeles).

Respecto a la gestión de datos, se ha optado por el uso de la biblioteca ITK [4]. Esta biblioteca facilita la gestión de lectura y escritura de datos en formatos de datos comunes en el ámbito de la imagen médica, como pueden ser DICOM y NIFTI. Otra de las ventajas del uso de ITK es el soporte de compresión automática de ficheros, reduciendo significativamente el espacio necesario en el almacenamiento, tanto para la entrada como para la salida de datos.

Listing 1

NÚCLEO DE EJECUCIÓN DE RUMBA-SD IMPLEMENTADO EN ARMADILLO.

```

for (size_t i = 0; i < Niter; ++i) {
    Ratio = mBesselRatio<T>(n_order,
        Reblurred.S);

    #pragma omp parallel for simd
    for (size_t k = 0; k < SR.n_cols; ++k)
        for (size_t j = 0; j < SR.n_rows; ++j)
            SR(j,k) = Signal(j,k) * Ratio(j,k);
    KTSR = KernelT * SR;
    KTRB = KernelT * Reblurred;

    #pragma omp parallel for simd
    for (size_t k = 0; k < fODF.n_cols; ++k)
        for (size_t j = 0; j < fODF.n_rows; ++j)
            fODF(j,k) = fODF(j,k) * KTSR(j,k) /
                (KTRB(j,k) + std::
                    numeric_limits<double>::epsilon
                ());

    Reblurred = Kernel * fODF;

    #pragma omp parallel for simd
    for (size_t k = 0; k < Signal.n_cols; ++k)
        for (size_t j = 0; j < Signal.n_rows;
            ++j)
            SUM(j,k) = (pow(Signal(j,k),2) +
                pow(Reblurred(j,k),2))/2 - (
                sigma2(j,k) * (Signal(j,k) *
                    Reblurred(j,k)) / sigma2(j,k)
                * Ratio(j,k));

    sigma2.i = (1.0/N) * sum( SUM , 0) /
        n_order;

    #pragma omp parallel for
    for (size_t k = 0; k < sigma2.i.n_elem; ++k)
        sigma2.i(k) = std::min<T>(std::pow<T>
            >(1.0/10.0,2), std::max<T>(sigma2.i(
                k), std::pow<T>(1.0/50.0,2)));

    sigma2 = repmat(sigma2.i, N, 1);
}

```

El Listado 1 muestra el núcleo de ejecución de RUMBA-SD. Este código se ejecuta un determinado número de iteraciones ($Niter$). Cada iteración se compone de tres multiplicaciones de matrices y varias

multiplicaciones elemento a elemento. Hasta donde sabemos, Armadillo carece de una implementación paralela de las llamadas de multiplicación elemento a elemento, por lo que estas llamadas se han paralelizado mediante OpenMP, preservando al mismo tiempo las primitivas de acceso a datos de Armadillo. Los bucles también están paralelizados aplicando la directiva OpenMP *pragmasimd*, con el objetivo de vectorizar de los contenidos de cada bucle.

V. EVALUACIÓN EXPERIMENTAL

La evaluación del rendimiento se ha llevado a cabo en un equipo con Ubuntu 14.04 x64. El hardware del sistema consiste en un Intel(R) Xeon(R) CPU E5-2630 v3 a 2.40GHz y 128GiB de RAM. Los compiladores utilizados son GCC 4.8 e Intel 15.1. La tarjeta gráfica empleada consiste en una NVidia Titan Black (perfil alto) y una GTX 680 (perfil bajo). La versión de CUDA usada en el proceso de compilador es 7.5. En todos los casos se ha compilado el código fuente con las opciones `-O3` y `-DNDEBUG`. Todos los resultados de la evaluación fueron adquiridos como el promedio de cinco ejecuciones consecutivas.

En este trabajo se han evaluado las siguientes configuraciones de pHARDI: BLAS, OpenBLAS, NVBLAS, Intel MKL y ArrayFire. Para ello se ha experimentado con tres casos de prueba distintos, en lo que varía el tamaño del conjunto de datos.

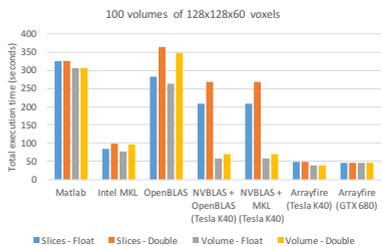


Fig. 5. Comparación de tiempo de ejecución de pHARDI sobre distintas soluciones de aceleración de álgebra lineal.

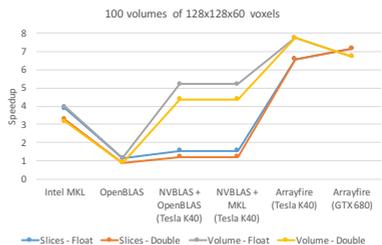


Fig. 6. Aceleración alcanzada sobre distintas soluciones de aceleración de álgebra lineal. Se toman los tiempos de ejecución de MATLAB como valores de referencia.

El caso de entrada evaluado en este trabajo consis-

ten en un estudio real compuesto por 101 volúmenes de $128 \times 128 \times 60$ píxeles cada uno. En la Figura 5 y 6 se muestran los tiempos de ejecución (en segundos) y la aceleración alcanzada de dos tipos de ejecución, respectivamente. Se ha procedido a ejecutar PHARDI sobre dos disposiciones de datos diferentes. En la primera, se procesa cada una de las rodajas de los volúmenes, resultado en 60 iteraciones (“slices”). En la segunda, todos los píxeles son procesados en una única matrix. Para ambos casos el proceso de reconstrucción se obtiene tras la ejecución de 300 iteraciones (“volume”). En la figura se incluye el tiempo total de ejecución, incluyendo los tiempos de entrada y salida.

A simple vista podemos observar que la solución que ofrece mayor rendimiento son las versiones implementadas con Arrayfire. Obviamente la ventaja viene dada por el aprovechamiento de la capacidad de cómputo en GPU. Sin embargo, apreciamos que no hay una gran diferencia entre los dos modelos de GPU analizados. En el futuro se pretende realizar un análisis más detallado sobre este aspecto. Adicionalmente observamos que la implementación compilada sobre la biblioteca Intel MKL es la que más beneficia el tiempo de ejecución de la aplicación. Hay que señalar que Matlab utiliza esta misma tecnología para la paralelización de álgebra lineal. Sin embargo, nuestra implementación hace un uso más eficiente en la carga y almacenamiento de datos.

Es importante destacar que NVBLAS únicamente es capaz de descargar operaciones relacionadas con BLAS3, por lo que limita sustancialmente el rango de mejora. Además, en cada una de las iteraciones, es necesario realizar copias de memoria entre el anfitrión y el dispositivo. Sin embargo, este proceso no es necesario con Arrayfire, ya que gracias al API de desarrollo es posible especificar que variables se mantienen en la memoria del dispositivo. Esta característica es especialmente beneficiosa en el caso de aplicaciones con un proceso iterativo.

VI. TRABAJOS RELACIONADOS

Para hacer frente a estos problemas de portabilidad de MATLAB, existen diferentes alternativas: *ii*) usar de transcompiladores (*source-to-source compilers*) para migrar códigos MATLAB a lenguajes compilados y orientados a computación de altas prestaciones, e.g., C/C++ o Fortran; *ii*) optimizar el código MATLAB mediante el uso de bibliotecas y modelos de programación paralela y *iii*) traducir, de forma manual, la aplicación MATLAB a un lenguaje HPC para finalmente optimizarla mediante paradigmas de programación paralelos.

Como bien es sabido, MATLAB es un lenguaje sencillo pero requiere de un intérprete para ejecutarse, convirtiéndolo en algunos casos en una solución más lenta frente a lenguajes compilados. Por ello, el compilador comercial MCC [5] permite traducir de forma automática un programa MATLAB a C/C++ o Fortran e incluso compilarlo en un ejecutable único MEX. No obstante, también pueden encontrarse soluciones

libres: el compilador Falcon [6] traduce aplicaciones MATLAB a códigos Fortran 90; otros compiladores como Conlab [7] y Otter [8] están orientados a generar código para plataformas de memoria distribuida y compartida. Sin embargo, transformar una aplicación MATLAB no sólo significa traducir instrucción por instrucción a un lenguaje orientado a HPC, sino también aplicar optimizaciones. Por ejemplo, el compilador Falcon reemplaza el código cuando encuentra coincidencias con patrones sintácticos por rutinas ya optimizadas. Otros, como el preprocesador Kapf [9] centra sus esfuerzos en detectar automáticamente porciones de código que encajan con operaciones BLAS y sustituirlas por su correspondiente llamada.

Cuando la traducción realizada por un transcompilador no genera las mejoras del rendimiento esperadas, se pueden aplicar mejoras directamente en el código MATLAB. Por ejemplo, se pueden utilizar versiones de bibliotecas propietarias de BLAS optimizadas para arquitecturas de propósito general (e.g. ACML [10] (AMD), MKL [11] (Intel) y ESSL [12] (IBM)) o específico (cuBLAS [13] para GPUs NVIDIA). También existen implementaciones de terceros de BLAS, como por ejemplo GotoBLAS [14], ATLAS [15] o GSL [16]. Otros *frameworks* permiten incluso el uso de modelos de programación paralela OpenMP o MPI y aceleradores [17], [18].

En caso de que sea posible traducir el código MATLAB de forma manual, se pueden utilizar algunas bibliotecas existentes en la literatura que, debido a su interfaz de alto nivel, permiten aliviar la tarea de traducción. Bibliotecas como Eigen [19] o VienaCL [20] en C++ proporcionan una interfaz a BLAS sencilla y utilizan, al mismo tiempo, modelos de programación OpenMP, OpenCL y/o CUDA. Otras bibliotecas en C++, como Armadillo [2], tienen el objetivo de alcanzar un buen equilibrio entre la velocidad y facilidad de uso. Dentro de las principales ventajas, destaca su sintaxis de alto nivel, muy similar a la de MATLAB. Armadillo es útil para el desarrollo de algoritmos directamente en C++, o la conversión rápida de código para entornos de producción. Otra alternativa es la biblioteca ArrayFire [3], que soporta interfaces para C, C++, Java, R y Fortran y es integrable con códigos basados en CUDA.

VII. CONCLUSIONES

En este trabajo se ha presentado un caso práctico de migración de una aplicación desarrollada inicialmente en el lenguaje MATLAB, a una nueva versión basada totalmente en el lenguaje C++. La principal ventaja de la solución propuesta es la portabilidad ofrecida, siendo compatible con un gran número de plataformas. Además, esta aproximación proporciona una solución flexible, permitiendo la integración de distintas bibliotecas o aceleradores de álgebra lineal.

Uno de los problemas que hemos observado en ArrayFire es la limitada cobertura para el desarrollo de aplicaciones basadas en múltiples dispositivos

GPU. Por ello, planeamos desarrollar bibliotecas que faciliten esta aproximación. Otra línea de trabajo que arroja este trabajo es la detección y aplicación de patrones de paralelismo al código implementado, con el objetivo de aprovechar todos los dispositivos disponibles en el computador, y no solo limitar su aplicación a GPUs.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Proyecto Europeo ICT 644235 “REPHRASE: REfactoring Parallel Heterogeneous Resource-Aware Applications” y el Ministerio de Economía y Competitividad, bajo el proyecto TIN2013-41350-P “Scalable Data Management Techniques for High-End Computing Systems”.

REFERENCIAS

- [1] Erick J Canales-Rodríguez, Alessandro Daducci, Stamatios N Sotiropoulos, Emmanuel Caruyer, Santiago Aja-Fernández, Joaquim Radua, Jesús M Yurramendi Mendizabal, Yasser Iturria-Medina, Lester Melie-García, Yasser Alemán-Gómez, et al., “Spherical deconvolution of multichannel diffusion MRI data with non-Gaussian noise models and spatial regularization.” *PLoS one*, vol. 10, no. 10, pp. e0138910, 2015.
- [2] Conrad Sanderson, “Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments.” 2010.
- [3] James Malcolm, Pavan Yalamanchili, Chris McClanahan, Vishwanath Venugopalakrishnan, Krunal Patel, and John Melonakos, “ArrayFire: a GPU acceleration platform,” in *SPIE Defense, Security, and Sensing*, International Society for Optics and Photonics, 2012, pp. 84030A–84030A.
- [4] Hans J Johnson, Matthew M McCormick, and Luis Ibanez, *The ITK Software Guide Book 1: Introduction and Development Guidelines-Volume 1*, Kitware, Inc., 2015.
- [5] The Mathworks, Inc., “C/C++ Compiler Suite,” <http://www.mathworks.com>, Online; accedido el 18 de Mayo de 2016.
- [6] Luiz De Rose and David Padua, “Techniques for the translation of matlab programs into fortran 90,” *ACM Trans. Program. Lang. Syst.*, vol. 21, no. 2, pp. 286–323, Mar. 1999.
- [7] Peter Drakenberg, Peter Jacobson, and Bo Kågström, “A conlab compiler for a distributed memory multicompiler,” in *PPSC*, 1993, pp. 814–821.
- [8] “Results from a parallel matlab compiler,” in *Proceedings of the 12th. International Parallel Processing Symposium on International Parallel Processing Symposium*, Washington, DC, USA, 1998, IPPS ’98, pp. 81–, IEEE Computer Society.
- [9] Kuck and Associates, Inc., “KAP for IBM Fortran and C,” <http://www.kai.com/product/ibminf.html>, Online; accedido el 18 de Mayo de 2016.
- [10] AMD Core Math Library (ACML), “<http://developer.amd.com/tools-and-sdks/cpu-development/and-core-math-library-acml/>.”
- [11] Intel Math Kernel Library (MKL), “<http://software.intel.com/en-us/intel-mkl/>.”
- [12] IBM, “Engineering Scientific Subroutine Library.”
- [13] nVidia, “cuBLAS Library User Guide,” <https://developer.nvidia.com/cublas>, 2012, Online; accedido el 18 de Mayo de 2016.
- [14] John Markoff, “Writing the fastest code, by hand, for fun: A human computer keeps speeding up chips,” *The New York Times*, 2005.
- [15] R. Clint Whaley and Jack J. Dongarra, “Automatically tuned linear algebra software,” in *International Conference on Supercomputing (ICS)*, 1998.
- [16] Brian Gough, *GNU Scientific Library Reference Manual - Third Edition*, Network Theory Ltd., 3rd edition, 2009.
- [17] The Mathworks, Inc., “Multicore-Capable Code Generation Using OpenMP,” <http://es.mathworks.com/products/matlab-coder/features.html#multicore-capable-code-generation-using-openmp>, Online; accedido el 18 de Mayo de 2016.
- [18] The Mathworks, Inc., “Parallel Computing Toolbox,” <http://es.mathworks.com/products/parallel-computing/>, Online; accedido el 18 de Mayo de 2016.
- [19] Gaël Guennebaud, Benoit Jacob, et al., “Eigen v3,” <http://eigen.tuxfamily.org>, 2010.
- [20] K. Rupp, F. Rudolf, and J. Weinbub, “ViennaCL - A High Level Linear Algebra Library for GPUs and Multi-Core CPUs,” in *Intl. Workshop on GPUs and Scientific Applications*, 2010, pp. 51–56.

Tecnologías Grid, clúster, plataformas distribuidas y Big Data

Towards Full GPU Virtualization in Cloud Infrastructures with rCUDA

Jose M. Claver, Carlos Pérez Conde, Juan Gutierrez-Aguado¹

Resumen— This paper presents a solution that enables the effective use of virtual GPUs, by using rCUDA, from virtual machines (VMs) in cloud infrastructures. The proposed solution has the following properties: ease integration in public clouds, dynamic assignment of virtual GPUs to VMs, and transparent physical location of the GPUs to the tenants, thus preventing overuse of GPU resources and denial of service to other users. For this purpose, the proposed architecture relies on different modules distributed in the infrastructure: network services (NS) at each compute node offering GPU access, daemons to configure rCUDA-servers in the nodes with GPUs, and, daemons to configure the network services.

We have analyzed different possible GPU utilization scenarios. Thus, depending on the location of the GPU with respect to the VM, we have considered a local GPU access (the VM is located in a node that has GPU devices) and a remote GPU access (the VM is located in a physical node without GPUs). Besides, the GPUs can be assigned to a VM as an exclusive resource (any other VM is not allowed to use the GPU) or shared (GPUs are shared among different VMs). We have performed experiments to validate the proposed architecture and the results show that the proposed solution does not introduce significant overhead to the performance obtained in previous environments.

Palabras clave— GPU, Cloud computing, HPC, rCUDA

I. INTRODUCTION

CLOUD computing is the technology paradigm that permits to perform a wide range of massive and complex computing in economy, science, and engineering. It allows at the same time energy and economy savings to the data centers [1]. Recently, a wide number of applications and services with high computational demands have been moved to the cloud. Thus, high performance cloud computing platforms have been proposed [2].

Today, high performance computing (HPC) makes an intensive use of computing accelerators, like graphic processing units (GPUs), field programmable gate arrays (FPGAs), or combinations of both [3], among others. But these resources have usually an intermittent and relatively low use and they have increased acquisition and maintenance costs. So, in order to overcome those problems, different remote GPU virtualization solutions have been proposed. The recent GRID vGPU [4] by NVIDIA and the KVMGT technology by Intel are both intended for desktop virtualization. The predominance of NVIDIA in the HPC programming field has made that most virtualization platforms use the CUDA architecture: GridCUDA [5], DS-CUDA [6], gVirtuS [7], vCUDA [8], rCUDA [9], [10], etc.

Each one of these solutions has different characteristics: vCUDA only supports the old CUDA 3.2 and its communications protocol has a considerable overhead; gVirtuS and GridCuda only support the old CUDA version 3.2. A comparison between the best previous proposals shows that the best performance solution using CUDA in virtualized environments is rCUDA [11].

Currently, a lot of applications demanding high performance computing on the cloud are using GPGPUs, but the available virtualization solutions are not consolidated on the cloud. Thus, a public cloud provider like Amazon Web Services¹ is offering, through Amazon Elastic Compute Cloud (EC2)², instances of virtual machines with access to GPU supporting CUDA and OpenCL, among other parallel computing frameworks for accelerators. Amazon EC2 G2 instances³ offer two different configurations, with 1GPU or with 4 GPUS. Each GPU has 1,536 CUDA cores and 4GB of video memory, which are the characteristics of one of the two GPUS in the NVIDIA GRID K2 [4] device. The CG1 is a previous generation instance of EC2, based on the NVIDIA Tesla M2050, and is still supported. However, these instances do not use full GPU virtualization, and users can only use the devices locally attached to the compute node using the PCI-Passthrough technique [12], in which a PCI device is assigned exclusively to one virtual machine. In this way, CUDA GPUs can be assigned to a virtual machine using PCI-Passthrough, but these GPUs do not have the Single Root I/O Virtualization (SR-IOV) specification. Hence, GPU are assigned to the virtual machines in exclusive mode. There are some proposals to change dynamically the assignment of a GPU to more than one virtual machine, but to change the assignment of the GPU supposes an overhead of at least 2 seconds [13].

Recently, some works have analyzed the use of virtualized GPUs from virtual machines (VMs), e.g., using rCUDA [14], [15]. But they are far from an actual integration in the cloud because they require to expose the physical location of the GPU to the VM. This can be allowed in a private cloud but should be avoided in public clouds to prevent tenant overuse of resources and denial of service to other tenants. Other solutions, like gVirtuS [7], allows virtualization of GPUs and makes them accessible to any virtual machine in the cluster, is strongly dependent on

¹Amazon web services: <http://aws.amazon.com>.

²Amazon EC2: <http://aws.amazon.com/ec2>.

³Amazon EC2, instance types: <http://aws.amazon.com/ec2/instance-types>.

¹Departament d'Informàtica, Universitat de València, e-mail: jclaver, cperez, jgutier@uv.es

the hypervisor. A similar solution is gCloud [16] but is not yet integrated in a Cloud Computing Manager, and application code has to be adapted to run in the vGPU environment. Another interesting solution is presented by Jun et al. [17], but shows some bottleneck for HPC applications and its full integration in OpenStack has not been proved.

The main contribution of our work is to provide an architecture to incorporate HPC in cloud infrastructures taking into account relevant aspects such as security, flexibility (in which different type of nodes and access modes are characterized) and transparency in the virtualization of GPGPUs. These characteristics are not critical in private clouds, but have to be addressed in public cloud environments. Furthermore, the proposed architecture has a negligible impact on performance.

The rest of the paper is organized as follows. In Subsection I-A, the technological background used in this work is described. Section II states the basis of the proposed architecture and the modes of use. Section III shows the experiments performed and the results obtained in the bandwidth and computational tests carried out to evaluate the overhead introduced by our approach. Conclusions and future work directions are present in Section IV.

A. Background: rCUDA and OpenStack

In this section, the main technologies used in this work are described. These are related with the GPU virtualization framework used, rCUDA, and the cloud platform for which we have proposed our architecture, OpenStack. We use rCUDA [18], [15] for the virtualization platform of GPUs due to the good results obtained in previous proposals. rCUDA follows a client-server distributed architecture and allows a transparent access to NVIDIA GPU devices in a cluster from all the compute nodes. The client middleware runs in the nodes, where applications require GPU-based acceleration capabilities. It intercepts, process and forwards the CUDA runtime calls from CUDA accelerated applications to the rCUDA server.

Each node offering a GPGPU service has a rCUDA-server daemon [18], which receives, interprets and executes CUDA call requests. This allows a flexible use of the GPUs either remotely where nodes without GPU can use remote GPU nodes through a high speed network, or locally where several clients share the same GPU.

Two execution modes have been proposed [18] for the remote GPU allocation, exclusive and shared. A configuration in exclusive mode prioritizes the performance, while the shared mode optimizes the GPU utilization, at the expense of a performance reduction. rCUDA supports runtime-loadable specific communication modules that currently target the InfiniBand network (using InfiniBand verbs) and general TCP/IP protocol stack.

When the rCUDA client intercepts the CUDA runtime calls, it forwards them to the address and

port of the rCUDA server to which it is associated. The NVIDIA proprietary drivers in the GPGPU servers manage the concurrent executions with its own scheduler in the same way as in a local GPU context. Currently, rCUDA v15.07 provides support for NVIDIA CUDA 6.0, 6.5 and 7.0, excluding calls related with Graphics Interoperability, Unified Memory Management and Module Management.

Among the current cloud solutions, OpenStack [19] and OpenNebula [20] are some of the most advanced cloud open source platforms. These projects can manage virtual infrastructures to build private, public and hybrid IaaS (Infrastructure as a Service). OpenStack [19] is one of the most complete and extended open-source cloud platforms. OpenStack offers a set of API services (Compute, Networking, Storage) that allows to manage the life-cycle of virtual machines through a Web interface (OpenStack Dashboard) or through a command line client API. OpenStack platform can be deployed over different underlying infrastructures [21], i.e. using physical, virtual machines or a hybrid approach [22].

OpenStack supports the most recent hypervisors and its architecture offers flexibility to create a adaptable cloud, integrating legacy systems, and non proprietary hardware and software requirements. For these reasons we have selected OpenStack to develop our proposal. Among the offer of current hypervisors (KVM, VMware, Xen, vBox, etc.), KVM [23] is one of the most extended, configurable, open-source, Linux compliant, etc., allowing different configuration possibilities to optimize memory, CPU and IO access. Furthermore, KVM is becoming the default hypervisor for OpenStack, it is easy to tune and supports the widest set of OpenStack features.

II. PROPOSED ARCHITECTURE

In this work, we establish four different use modes of the GPUs. These modes are a combination of two coordinates that determine the space-temporal use of the available GPUs by the virtual machines. Thus, depending on the location of the GPU with respect to the VM, we have considered a local GPU access (the VM is located in a node with GPU devices) and a remote GPU access (the VM and the GPUs are located in different nodes). Regarding the temporal dimension, the GPUs can be assigned to a VM as exclusive resource (any other VM is not allowed to use the GPU) or shared (GPUs are shared among different VMs). Therefore, we consider the following modes:

- **Local-Exclusive.** Where the VM uses the GPUs present in the compute node in which it is consolidated, and it monopolizes the use of the assigned GPUs.
- **Local-Shared.** Where the VM uses the GPUs present in the compute node in which it is consolidated, but the GPUs are partitioned and can be shared with other VMs.
- **Remote-Exclusive.** Where the VM uses the GPUs present in other nodes, different from the

compute node in which it is allocated, and it monopolizes its use.

- **Remote-Shared.** Where the VM uses the GPUs present in other nodes, different from the compute node in which it is consolidated, but the GPUs are partitioned and can be shared with other VM.

In order to enable these four modes of GPU utilization, we propose an architecture composed of four types of nodes (Figure 1):

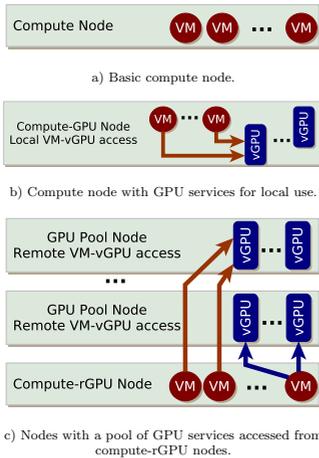


Fig. 1. Node types in the proposed architecture: a) Compute node without GPU/GPUs; b) Compute node with GPUS/GPUs for local vGPU access; c) Compute pool of GPUs for remote vGPU access and compute node with access to remote vGPUs.

- **Compute nodes:** these are standard compute nodes that can allocate tenant virtual machines.
- **Compute-GPU nodes:** nodes that have vGPUs and can allocate tenant virtual machines that will use the local vGPUS (Figure 1-b).
- **Compute-rGPU node:** these are compute nodes that allocate tenant virtual machines and have an additional network that allows remote GPU access (Figure 1-c).
- **GPU-pool nodes:** nodes that have vGPUs intended to be used from virtual machines allocated in compute nodes with remote GPU access(Figure 1-c).

The internal architecture of the Compute-GPU node is shown in Figure 2. In these nodes, running applications in VMs demanding GPGPU services can access the local GPUs using rCUDA. To do that, the VM executes the RCUDA-client while the physical node runs the rCUDA-server. We have launched a rCUDA-server daemon in the Compute-GPU node

for each VM allocated in that node and the network service is configured to properly route the packets from the VM to its assigned rCUDA-server daemon. This proposal allows to uncouple the real network (address,port) of the rCUDA-server from the network (address,port) of the rCUDA-client configured in every VM. This is possible thanks to the use of the network service (NS) that intercepts and analyzes all the network traffic between the VMs and the integration bridge (OVS br-int). Thus, the CUDA run-time calls forwarded by the rCUDA-client to the network (address, port) generic CUDA-server configured in the VM, is translated by the NS to the network (address, port) of the real rCUDA-server daemon assigned to the VM. Our NS is a process that makes use of the Linux iptables rules, allowing a transparent and secure use of the resources.

The consequences of this decision, for the applications and the VM in the cloud, are that:

- All the VM with the same need of GPU acceleration can be configured with the same network address and port. This reduces the effort devoted to the configuration of VM and avoids the need of internal intervention of the VM setup (which is not liked by users) .
- This clear separation of the VM configuration from the location of the real resource allows a real time assignment of the best GPU in function of its performance, economy or energy saving.
- Real network (address,port) of rCUDA-servers are hidden for applications and users from different VMs. This provides a security against the abuse of GPGPU resources.
- The network service blocks any other access so it is not possible to perform a denial of service to other users.

As a result, in the architecture of the Compute-GPU nodes the NS allows a “de facto” virtualization of the interface between rCUDA-client and RCUDA-server. Concretely, a network (address, port) used for a rCUDA-client in a VM could be translated on different real network (address, port) according on the rCUDA-server assigned.

On the other hand, GPU-pool nodes have vGPUs intended to be used from applications of VMs allocated in compute nodes that have access to a dedicated network where the GPU-pool nodes reside. The proposed architecture for the use of a GPU-pool node is shown in figure 3.

In the GPU-pool node will listen on the GPU network physical interface. In the compute nodes, the NS intercepts and analyzes the CUDA run-time calls forwarded by the rCUDA-client to the network (address, port) CUDA-server configured in the VM, and translates it to the network address of the GPU-node and port where the real rCUDA-server daemon is executed. The use of a dedicated network allows to have a clean separation of the traffic between VMs (this traffic goes through the VM Network) from the

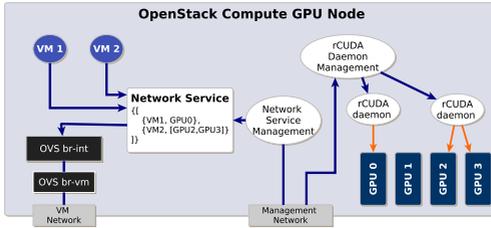


Fig. 2. Internal Architecture of a compute-GPU node.

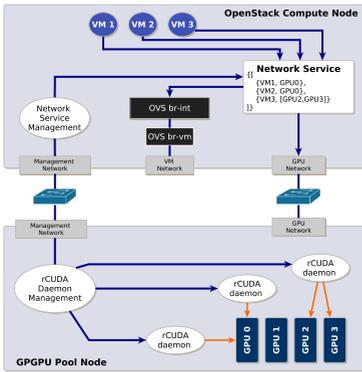


Fig. 3. Use of a GPGPU pool server from a compute node.

traffic destined to remote GPUs (this traffic goes through the GPU Network). In this way the rCUDA generated traffic do not interfere with other users of the infrastructure that are not using GPUs.

The dynamic assignation of a rCUDA-server to a particular VM should be made by a scheduler that has access to the physical resources available in the infrastructure and from real time monitoring data in order to make the best assignation. This scheduler should also orchestrate the rest of daemons in the proposed architecture (in order to configure the network services and configure the rCUDA-server daemons). This scheduler has not been implemented in the present prototype.

III. EXPERIMENTAL RESULTS

A. Physical setup

We have performed a proof of concept of our architecture with two nodes connected through a 1Gb Ethernet network switch. One node has 2 sockets with 4 cores per socket at 1.6GHz and hyperthreading (16 vCPUs), 24 GB RAM, two NVIDIA Tesla C2050, and KVM as hypervisor. This node will rep-

resent both the Compute-GPU node and the GPU-pool node. The other node has 2 cores at 3GHz, 4GB RAM, and KVM as hypervisor. This node will represent a compute node that allocates a virtual machine.

B. Performance test

We have carried out two performance tests: a bandwidth test and a computational test. In both cases, we have a local and a remote setup. The local setup corresponds to a local access to the GPU, whereas the remote setup implies to use a remote (located in another physical host) GPU. In the local case, we have the following scenarios: use of CUDA from the physical machine using the local GPU, use of rCUDA from the physical machine to use the local GPU, use of rCUDA inside the virtual machine to use the GPU of the same physical host, and the use of rCUDA inside a virtual machine using the NS service to use the GPU of the same physical node.

In the remote case, we have the following scenarios: use of rCUDA from the physical machine to access the remote GPU, use of rCUDA from the virtual machine to use a GPU installed in a remote host, and use of rCUDA from the virtual machine using the NS service to use a remote GPU. For each experiment involving virtual machines, the virtual machine was booted, all services were started, the experiment was performed, all the services were stopped, and the virtual machine was stopped before the next execution of the experiment.

The bandwidth results have been obtained averaging the results of five executions of the NVIDIA sample code `bandwidthtest` both for device to host and host to device transfer. Figure 4 shows the results for the local scenario. As can be seen, the results using rCUDA in the VM are very similar to those obtained using rCUDA in the physical machine. Note also that there is no noticeable overhead when the NS service is used. The results for the remote scenario are shown in Figure 5. The same behavior is obtained in this case. Note that the maximum bandwidth reached in the remote case is limited by the channel capacity (1Gb/s).

The computational results have been obtained running five times the NVIDIA sample code

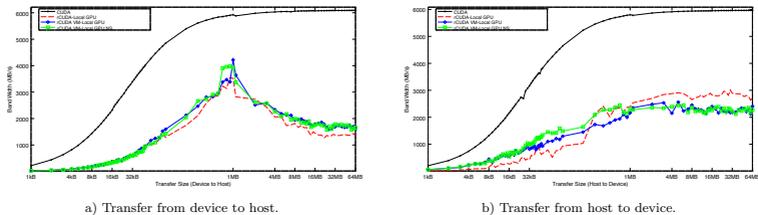


Fig. 4. Bandwidth test results for data transfer between the hot and the memory device, using pinned memory. Tests have been done in a local environment for CUDA native, rCUDA, and rCUDA using VM with and without NS.

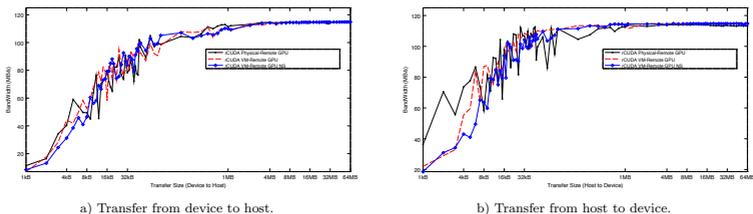


Fig. 5. Bandwidth test results for data transfer between the host and the memory device, using pinned memory. Tests have been done in a remote environment for direct rCUDA, and rCUDA using VM with and without NS.

`matrixMul` with matrices of size 8192×8192 and running 50 iterations. In this case, we have monitored the GPU to measure the *active time* and the *computing time*. The active time is the elapsed time between the allocation of the memory in the GPU until the disposal of that memory. The computing time is the time that the GPU is performing the computation. Figure 6-a) shows the results for the local scenario. As can be seen, the computation time is very similar in all cases. The main differences are observed in the active time as it is bigger when the rCUDA client runs in a virtual machine. Figure 6-b) shows the results for the remote scenario. The computation time is very similar in all cases and similar to the local scenario case. However, in this case there is more difference between the active time and the computation time due to the limitation of the network. It is assumed that if a faster network interconnection is used a reduction of the active time would be obtained, while the *computing time* would be the same.

C. Validation test

When the network service (NS) is not used, the IP address and port of the rCUDA daemon should be explicitly exposed in the tenant virtual machine by an environment variable. While this can be acceptable in a private cloud it is not a good practice in a public cloud, where the physical location of resources should not be exposed to the tenants. Using the proposed NS, any IP address and port can be configured in the virtual machine and the NS redirects the traffic to the real IP address and port. Any access to

other IP address or port is blocked by the NS, thus preventing the interference with the resources (GPGPUs) assigned to other tenants. Besides, when the virtual machine is stopped the NS destroys that mapping preventing the tenant to use the resources from other virtual machines.

IV. CONCLUSIONS AND DISCUSSION

In this paper we have proposed a new architecture that allows a full integration of GPU on the cloud. Our proposal is based on the virtualization of the interface between the client and the server of the GPU virtualization framework, in our case rCUDA. We have identified different types of nodes and have proposed different modes of access to vGPUs. Our architecture relies on different modules distributed across the infrastructure in order to achieve location transparency of the vGPUs, avoid overuse of vGPUs assigned to tenants and avoid denial-of-service to other tenants. This technique can be extended to other frameworks that use the client-server paradigm.

Results show that our proposed architecture maintains the performance obtained for VM in the use of GPU by rCUDA, while allows flexibility, security and better use of the GPGPUS resources in the Cloud. In the future we plan a full integration of rCUDA on the cloud introducing a more virtualized solution of the interface between VM and GPU resources, and a more flexible and automated scheduling of GPGPU resources in the NS management.

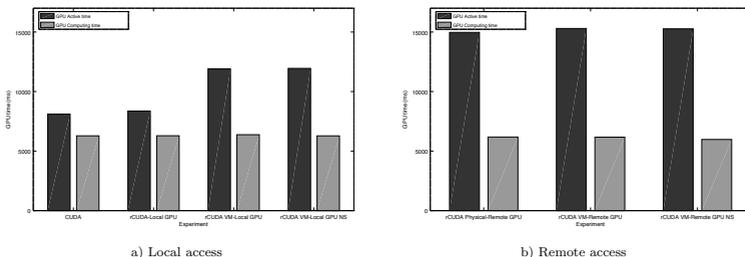


Fig. 6. Matrix multiplication average results for different configurations, using CUDA and rCUDA, and pinned memory. Tests have been done for a) local and b) remote access environments.

ACKNOWLEDGEMENTS

This work has been supported by the Universitat de València under the project number UV-INV-AE15-339582 and the Universitat de València Research Infrastructure Grant.

REFERENCES

- [1] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J-M Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 33:1–33:36, Dec. 2014.
- [2] V. Machi, M. Kunze, and M. Hillenbrand, "High performance cloud computing," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1408–1416, 2013. Including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for P2P Systems.
- [3] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs," in *Symposium on Application Specific Processors*, June 2008, pp. 101–107.
- [4] NVIDIA CRID Technology, "www.nvidia.com/object/grid-technology.html," 2015.
- [5] T.-Y. Liang and Y.-W. Chang, "Gridcuda: A grid-enabled CUDA programming toolkit," in *25th IEEE International Conference on Advanced Information Networking and Applications Workshops*, 2011, pp. 141–146.
- [6] M. Oikawa, A. Kawai, K. Nomura, K. Yasuoka, K. Yoshikawa, and T. Narumi, "DS-CUDA: A middleware to use many gpus in the cloud environment," in *High Performance Computing, Networking, Storage and Analysis (SCC)*, Nov 2012, pp. 1207–1214.
- [7] G. Giunta, R. Montella, G. Agrillo, and G. Coviello, *16th International Euro-Par Conference, Ischia, Italy, Proceedings, Part I*, chapter A GPGPU Transparent Virtualization Component for High Performance Computing Clouds, pp. 379–391, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [8] L. Shi, H. Chen, and J. Sun, "vCUDA: GPU accelerated high performance computing in virtual machines," in *IEEE International Symposium on Parallel Distributed Processing*, May 2009, pp. 1–11.
- [9] C. Reaño, R. Mayo, E. S. Quintana-Ortí, F. Silla, J. Duato, and A. J. Peña, "Influence of infiniband fdr on the performance of remote gpu virtualization," in *IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2013, pp. 1–8.
- [10] C. Reaño, F. Silla, A. J. Peña, G. Shainer, S. Schultz, A. Castelló, E. S. Quintana-Ortí, and J. Duato, "Pester: Boosting the performance of remote gpu virtualization using infiniband connect-ib and pcie 3.0," in *IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2014, pp. 266–267.
- [11] C. Reaño and F. Silla, "A performance comparison of CUDA remote GPU virtualization frameworks," in *IEEE International Conference on Cluster Computing*, Sept 2015, pp. 488–489.

- [12] J. P. Walters, A. J. Younge, D. I. Kang, K. T. Yao, M. Kang, S. P. Crago, and G. Fox, "GPU passthrough performance: A comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL applications," in *Cloud Computing (CLOUD), IEEE 7th International Conference on*, IEEE, 2014, pp. 636–643.
- [13] "Exploiting GPUs in virtual machine for biocloud," *BioMed Research International*, p. 11, 2013.
- [14] J. Duato, A. J. Peña, F. Silla, J. C. Fernández, R. Mayo, and E. S. Quintana-Ortí, "Enabling CUDA acceleration within virtual machines using rCUDA," in *18th International Conference on High Performance Computing*, Dec 2011, pp. 1–10.
- [15] J. Prades, C. Reaño, and F. Silla, "CUDA acceleration for Xen virtual machines in infiniband clusters with rCUDA," in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, New York, NY, USA, 2016, PPOPP '16, pp. 35:1–35:2, ACM.
- [16] K. M. Diab, M. M. Rafique, and M. Hefeeda, "Dynamic sharing of gpus in cloud systems," in *IEEE Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)*, May 2013, pp. 947–954.
- [17] T. J. Jun, V. Q. Dung, M.H. Yoo, D. Kim, H. Cho, and J. Hahn, "Gpgpu enabled hpc cloud platform based on openstack," in *The International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014.
- [18] A. J. Peña, C. Reaño, F. Silla, R. Mayo, E. S. Quintana-Ortí, and J. Duato, "A complete and efficient CUDA-sharing solution for HPC clusters," *Parallel Computing*, vol. 40, no. 10, pp. 574–588, 2014.
- [19] OpenStack, "Openstack: The open source cloud operating system," <http://www.openstack.org/software/>, 2015.
- [20] D. Milošević, I.M. Llorente, and Ruben S. Montero, "Opennebula: A cloud management tool," *Internet Computing, IEEE*, vol. 15, no. 2, pp. 11–14, March 2011.
- [21] OpenStack, "Install openstack," <http://www.openstack.org>, 2015.
- [22] E. Chirivella-Perez, J. Gutierrez-Aguado, J. M. Claver, and J. M. Alcaraz Calero, "Hybrid and extensible architecture for cloud infrastructure deployment," in *15th IEEE International Conference on Computer and Information Technology*, 2015, pp. 611–617.
- [23] I. Habib, "Virtualization with kvm," *Linux J.*, vol. 2008, no. 166, Feb. 2008.
- [24] J. Song, Z. Lv, and K. Tian, "KVMGT: a full gpu virtualization solution," 2014.
- [25] B. da Silva, A. Braeken, E. D'Hollander, A. Touhafi, J. G. Cornelis, and J. Lemeire, "Study of combining GPU/FPGA accelerators for high-performance computing," in *HLS4HPC workshop at the 9th International Conference on High-Performance and Embedded Architectures and Compilers*, Philippe Coussy and Cyrille Chavet, Eds. 2013, pp. 1–2, HIPEAC.
- [26] R. Moreno-Vozmediano, R.S. Montero, and I.M. Llorente, "Key challenges in cloud computing: Enabling the future internet of services," *Internet Computing, IEEE*, vol. 17, no. 4, pp. 18–25, July 2013.

Sentiment Analysis on Multilingual Tweets using Big Data Technologies

Rodrigo Martínez-Castaño, Juan C. Pichel and Pablo Gamallo¹

Abstract— In this paper a new parallel system for sentiment analysis on multilingual tweets based on Big Data technologies is presented. On the one hand, our sentiment classifier performs as well as other state-of-the-art classifiers considering tweets written in different languages. On the other hand, our system is capable of processing millions of tweets in short times taking advantage of Big Data processing and parallel architectures, showing a good scalability in all the considered scenarios.

Keywords— Sentiment analysis, Big Data, Performance, Twitter, Hadoop.

I. INTRODUCTION

Sentiment Analysis consists in finding the opinion (e.g. positive, negative, or neutral) from text documents such as movie reviews or product reviews. Opinions about movies, products, etc. can be found in web blogs, social networks, discussion forums, and so on. Companies can improve their products and services on the basis of the reviews and comments of their costumers. Recently, many works have stressed the microblogging service Twitter. As Twitter can be seen as a large source of short texts (tweets) containing user opinions, most of these works make sentiment analysis by identifying user attitudes and opinions toward a particular topic or product. The task of making sentiment analysis from tweets is a hard challenge. On the one hand, as in any sentiment analysis framework, we have to deal with human subjectivity. Even humans often disagree on the categorization on the positive or negative sentiment that is supposed to be expressed on a given text. On the other hand, tweets are too short text to be linguistically analyzed, and it makes the task of finding relevant information (e.g. opinions) much harder.

Useful conclusions can only be extracted when huge amounts of text or documents are analyzed. However, standard solutions cannot handle gigabytes or terabytes of text data in reasonable time. In this way, professionals demand scalable solutions to boost performance of the sentiment analysis process. To address this challenge we propose to take advantage of parallel architectures using Big Data technologies.

In this paper a new parallel system for sentiment analysis on multilingual tweets based on Big Data technologies is introduced. The goal of our system is twofold. First, the sentiment classifier should perform as well as other state-of-the-art classifiers considering tweets written in different languages. For this reason a thorough evaluation was carried out an-

alyzing Spanish and English tweets. We must highlight that our classifier took part in two different sentiment analysis contests showing a good behavior with respect to other approaches. Second, millions of tweets should be processed in short times. With this objective in mind tweets are analyzed in parallel taking advantage of a Hadoop [1] cluster which decreases the processing times noticeably. Performance results demonstrates the benefits in terms of scalability of our proposal.

II. BACKGROUND & RELATED WORK

A. Big Data processing

MapReduce [2] is a programming model introduced by Google for processing and generating large data sets on a huge number of computing nodes. A MapReduce program execution is divided into two phases: *map* and *reduce*. The input and output of a MapReduce computation is a list of key-value pairs. Users only need to focus on implementing map and reduce functions. In the map phase, map workers take as input a list of key-value pairs and generate a set of intermediate output key-value pairs, which are stored in the intermediate storage (i.e., files or in-memory buffers). The reduce function processes each intermediate key and its associated list of values to produce a final dataset of key-value pairs. In this way, map workers achieve data parallelism, while reduce workers perform parallel reduction. Note that parallelization, resource management, fault tolerance and other related issues are handled by the MapReduce runtime.

Apache Hadoop [1] is the most successful open-source implementation of the MapReduce programming model. Hadoop consists, basically, of three layers: a data storage layer (HDFS), a resource manager layer (YARN), and a data processing layer (Hadoop MapReduce Framework). HDFS is a block-oriented file system based on the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. For this reason, Hadoop shows good performance with embarrassingly parallel applications requiring a single MapReduce execution (assuming intermediate results between map and reduce phases are not huge), and even for applications requiring a small number of sequential MapReduce executions [3]. However, Hadoop performs poorly when applications do not fit into one of the previous workflows. For example, an iterative algorithm can be expressed as a sequence of multiple MapReduce jobs. Since different MapReduce jobs cannot shared data directly, intermediate results have to be written

¹Centro de Investigación en Tecnologías de Información (CITIUS), Universidad de Santiago de Compostela, Spain, e-mail: {rodrigo.martinez, juancarlos.pichel, pablo.gamallo}@usc.es

to disk and read again from HDFS at the beginning of the next iteration, with the consequent reduction in performance.

B. Sentiment analysis

There exist two types of approaches for sentiment analysis: Machine learning classification and lexicon-based strategy. Machine learning methods use several learning algorithms to determine the sentiment by training on a known dataset. Many of them rely on very basic classifiers, e.g. Naive Bayes [4] or Support Vector Machines [5]. They are trained on a particular dataset using features such as bag of words or bigrams, and with or without part-of-speech (PoS) tags. It is admitted that very basic language models based just on bag of words perform reasonably well [6]. The lexicon-based technique involves calculating sentiment polarity for a text using dictionaries or lexicons of words. The lexicon entries are annotated with their semantic orientation: polarity (positive, negative, or neutral) and strength [7].

To deal with short messages such as tweets and SMS, the state-of-the-art systems are based on machine learning techniques using as features polarity lexicons [8], [9]. Our strategy also makes use of polarity lexicons to enrich the set of features of the classifier. Most recent approaches to sentiment analysis on short messages and social media are endowed with rich linguistic information such as shallow syntactic structures [10] or syntactic dependency trees [11]. Following the tendency to use knowledge-rich linguistic features, our approach will be provided with shallow syntactic information to detect polarity shifters (e.g., negation markers).

We can find in the literature several interesting works which applied sentiment analysis to different document or text sources using Big Data technologies, for example, movie reviews [12] and hotel reviews [13]. Another authors analyzed tweets in real time [14]. Finally, some researchers have focused on the impact in the performance of different ways of distributing virtual machines (workers) on a cloud environment using sentiment analysis as case study [15].

III. ARCHITECTURE OF THE SYSTEM

The system is composed of three main modules illustrated in Figure 1. The first one is the Tweet Mining Module, which is the responsible of collecting the tweets for the future processing. This module has two components that will be explained in detail in the next section. The second module is, strictly speaking, the MapReduce application. This module is in charge of processing the stored tweets from a HBase¹ table. HBase is a non-relational and distributed column-oriented database built on top of HDFS. The mappers receive the tweets from HBase and process them through the sentiment analysis classifier if they match the query terms. Note that

¹<http://hbase.apache.org/>

there are two reducers, one is responsible of summarizing the number of successfully processed tweets and the other calculates the average positivity ratio and the number of matches for a particular query. This module is described in Section V.

The application can be executed through a web interface as shown in Figure 2, which corresponds to the third module of the system. The interface has two different views. A simple web form (main view, left image) allows to customize the MapReduce jobs that will be launched using two filters: start and end dates (an interval can be established) and the input terms. Several terms can be introduced separated by spaces. The second view shows the obtained result when the MapReduce job finishes (right image in the figure). The web back-end processes the data received from the web form, launches a Hadoop job and reads the outputs from HDFS, showing the result when it is finished.

IV. TWEET MINING MODULE

The Tweet Mining Module was developed in Java and consists of two components. The main module is a web scraper which acquires tweets directly from the official user interface for web browsers. The second component uses the Streaming API of Twitter through the help of the Twitter4J² library.

Twitter streams an arbitrary selection of tweets through the Streaming API. This represents a small fraction of the total volume of tweets published at each moment. The provided stream is the same for every consumer and it is very limited. It is said to provide about the 1% of the total tweets produced in every moment.

The idea to deal with the aforementioned problems is to retrieve tweets making queries to the Search API with common words of a particular language. Queries should be executed very often for each term when using this method, but this is not feasible due to the API limitations. However, these limitations can be bypassed considering web scraping techniques. This is the job of the second and main component of the Tweet Mining Module, which gathers tweets from the official web client making queries directly to twitter.com/search.

The module retrieves the HTML code that the web client returns for each query of every selected term and processes it using the HtmlUnit³ library. Through the web client, Twitter provides 20 tweets every 20 seconds for each query and for every consumer (at most, if available). Their system makes a selection of tweets as the Streaming API does, but, in this case, if the optimal terms are selected, a significant percentage of the total produced tweets for some language could be collected.

An important drawback of this method arises when lots of tweets contain only very common terms (usually there are more than those 20 tweets per term) forcing the loss of tons of tweets. This method

²<http://twitter4j.org/>

³<http://htmlunit.sourceforge.net/>

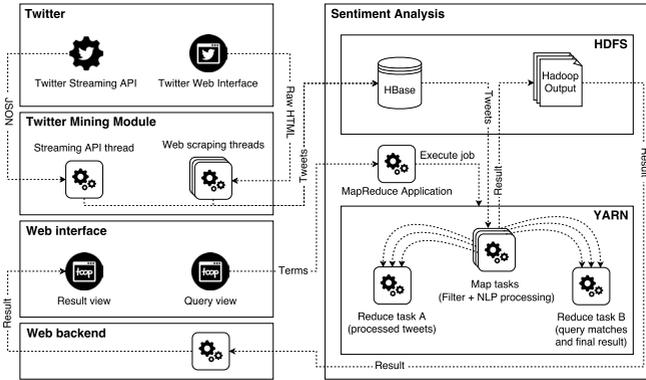


Fig. 1
 ARCHITECTURE DIAGRAM OF THE SENTIMENT ANALYSIS SYSTEM.



Fig. 2
 WEB GUI OF THE SENTIMENT ANALYSIS SYSTEM.

gets good results using frequency lists of certain languages and other type of terms like trending ones. This module and the idea behind it pretend to maximize the tweet retrieval at zero-cost.

When this module is executed, one thread is consulting the Streaming API while a configurable number of web scraper threads are retrieving information from the web client with the list of selected terms. In order to launch a mining job, the API tokens of a Twitter application must be set if using the official Streaming API. Other relevant parameters that can be set are the number of scraper threads to launch, the base size of the buffer and the step (in order to distribute the writes into HBase: the buffer of each thread is increased progressively with the step value), the number of laps (times that every web scraper thread consults its assigned terms) and the sleep time (number of seconds that every web scraper thread will pause its execution before starting a new lap).

V. SENTIMENT ANALYSIS MODULE

Our approach is based on a Naive Bayes (NB) classifier. NB combines efficiency (optimal time performance) with reasonable accuracy. The main theoretical drawback of NB methods is that it assumes conditional independence among the linguistic features. If the main features are the tokens extracted from texts, it is evident that they cannot be considered as independent, since words co-occurring in a text are somehow linked by different types of syntactic and semantic dependencies. However, even if NB produces an oversimplified model, its classification decisions are surprisingly accurate [16]. To improve the performance of the system, the classifier was enriched with lexicon-based features. Our sentiment analysis system is called *CitiusSentiment*⁴, it is multilingual and supports English and Spanish texts. It was implemented in Perl.

A. Strategy and features

Our classifier requires both a simplified annotated corpus of tweets (or short sentences) and a polarity lexicon with both Positive and Negative words. The annotated corpus only contains positive and negative examples of tweets. Neutral tweets are not required. As a result, a basic binary (or boolean) classifier which only identifies both Positive and Negative tweets is trained. In order to detect tweets without polarity (or Neutral), the following basic rule is used: if the tweet contains at least one word that is also found in the polarity lexicon, then the tweet has some degree of polarity. Otherwise, the tweet has no polarity at all and is classified as Neutral. The binary classifier is actually suited to specify the ba-

⁴freely available at <http://gramatica.usc.es/pln/tools/CitiusSentiment.html>

sic polarity between positive and negative, reaching a precision of more than 80% in a corpus with just these two categories.

The training corpus of tweets as well as the analyzed tweets are preprocessed as follows:

- Removing urls, references to usernames, and hashtags
- Reduction of replicated characters (e.g. *loooooveeee* → *love*)
- Identifying emoticons and interjections and replacing them with polarity or sentiment expressions (e.g. *:-)* → *good*)

The features considered by the classifier are lemmas, multiwords, polarity lexicons, and valence shifters. In the following, we describe the polarity lexicons and valence shifters.

Polarity Lexicons: For each language, we built a polarity lexicon with both *Positive* and *Negative* entries from different sources. The English lexicon is the result of unifying the following lexical resources:

- AFINN-111⁵ contains 2,477 word forms, which were lemmatized and converted into 1,520, positive and negative lemmas.
- Hedonometer [17] contains about 10,000 frequent words extracted from tweets which were classified as expressing some degree of happiness.
- Hu&Liu list [18] contains over 6,800 words out of which 5 positive and negative lemmas were selected 5,695.
- Sentiwordnet-3.0 [19] contains more than 100,000 entries. We selected a subset of 6,600 positive and negative lemmas with the highest polarity values.
- Finally, we have built a polarity lexicon with 10,850 entries by merging the previous ones.

The resources used to elaborate the Spanish lexicon are the following:

- Spanish Emotion Lexicon (SEL) [20] contains 2,036 words.
- A list of synonyms (ExpandSEL) was automatically extracted by expanding SEL using a dictionary of synonyms.
- A list of polarity words was semi-automatically extracted from the training corpus (CorpusLex).
- Finally, we have built a polarity lexicon with 4,564 entries by merging the previous ones.

Valence shifters: We take into account negative words that can shift the polarity of specific lemmas in a tweet. In the presented work, we will make use of only those valence shifters that reverse the sentiment of words, namely *negations*. The strategy to identify the scope of negations relies on the PoS (part-of-speech) tags of the negative word as well as of those words appearing to its right in the sequence.

⁵<http://arxiv.org/abs/1103.2903>

B. Integration into a Big Data infrastructure

In order to store the tweets collected by the Tweet Mining Module we have used Apache HBase. The Tweet Mining Module fills a buffer as the tweets are retrieved in such a way that when the buffer is full, all its content is dumped in the HBase corresponding table. Each row of the table has the following fields: a key (it matches the tweet original ID) and several attributes of the tweet stored in a single column family. These attributes are: user ID, nickname of the publisher, language, date of the post and the text of the tweet.

MapReduce (Hadoop) jobs can be launched when the Tweet Mining Module has finished or when it is still working. The MapReduce application performs reads from the HBase table which contains all the stored tweets. The initial and final date, the prefetch value, the table name, the internal task ID (it will determine the output file) and the target terms must be set when starting a job. Every tweet should match with each term to be processed (not necessarily in a contiguous manner).

HBase tables are split into regions and a mapper is launched for every region. While retrieving data from a HBase table, one region is equivalent to a split or map task. The MapReduce execution consist in a set of mappers, whose number is determined by the quantity of regions of the table, and two reducers. The mappers process a given set of terms.

Each accepted tweet is evaluated at the map function using the sentiment analysis classifier (*CitiusSentiment*) to get a result which takes the values -1, 0 or 1 (negative, neutral and positive, respectively). In order to integrate the classifier into the Hadoop infrastructure we should use Hadoop Streaming as it was originally implemented using Perl. Note that Hadoop Streaming is an utility provided by Hadoop to execute applications written in languages different from Java. However, important degradations in the performance were detected using Hadoop Streaming with respect to using Java codes. For this reason, we have used Perldoop [21] to translate Perl code into Java.

The reducer computes the total sum of the query. The final value contrasts the positive tweets with the negative ones. This value is eventually normalized from the total number of processed tweets, in a scale that goes from 0 to 1. This scale represents the positivity ratio of the query. Results lower than 0.45 are considered negative and upper 0.55, positive. The remaining intermediate values represent a neutral result. Both Hadoop output files and HBase data are stored in HDFS.

VI. PERFORMANCE RESULTS

In order to test our Sentiment Analysis system we set up a Hadoop cluster in Amazon EC2. The instances were created with Amazon Linux AMI running Apache Hadoop 2.7.2 and Apache HBase 1.1.4. Amazon gives their users the possibility of running a wide variety of virtual machines in their EC2 infras-

TABLE I
 AVERAGE NUMBER OF UNIQUE COLLECTED TWEETS PER SECOND, FILTERED BY LANGUAGE.

Spanish		English	
Streaming API	Full System	Streaming API	Full System
5.6	259.2 (46.3×)	15.1	275.4 (18.2×)

tructure. In our case, the cluster consists of 5 nodes of the `r3.2xlarge` instance type. This kind of EC2 instances has the following characteristics:

- CPU: Intel Xeon E5-2670 v2 (Ivy Bridge microarchitecture)
- Cores per node: 8
- RAM Memory per node: 61 GiB
- Disk: Each node has a 50 GB SSD General Purpose disk

According to the specifications provided by Amazon, the network performance of the `r3.4xlarge` instances is “high” so the theoretical bandwidth value is 1 Gbps.

The Tweet mining module has been evaluated using one `c4.8xlarge` instance. This system contains Intel Xeon E5-2666 v3 (Haswell microarchitecture) processors, there are in total 36 cores (72 threads).

A. Tweet mining module evaluation

In order to test the capabilities of this module, the performance was tested comparing the tweets per second collected considering the full system detailed previously in Section IV and also using the Streaming API thread working alone for Spanish and English in separated tests. The target terms were two lists of the 5,000 most frequent words for the Spanish and English languages. The *CREA* (Reference Corpus of the Current Spanish) list from RAE⁶ was used for Spanish. In the case of English the terms were extracted from a 6,000 words list from *insightin.com*⁷.

Tests were performed distributing the selected terms over 72 threads. Results are shown in Table I. The amount of tweets that the Streaming API can provide for a certain language is very low compared to the results obtained when executing our approach. For instance, our system collects 46.3× more tweets per second in Spanish than the Streaming API. Even if the language is not filtered when using the Streaming API the stream rate was at most 40 tweets per second.

B. Sentiment analysis evaluation

Our system, CitiusSentiment, took part as a participant in two different sentiment analysis competitions: TASS [22] and SemEval (task 9) [23]. TASS is an experimental evaluation workshop for sentiment analysis and online reputation analysis focused on Spanish language. SemEval (task 9 or 10) is focused on sentiment analysis in English microblogging. In

both competitions, the systems were trained with different training annotated corpora. In TASS we used as input dataset the training corpus of tweets provided by the organization. This set contains 7,216 Spanish tweets, which were tagged with several polarity values: Positive, Negative, and Neutral. In SemEval, we used the training dataset of English tweets provided by the organization, which contains 6,408 tweets. The systems were evaluated against different test datasets of tweets annotated with the polarity tags. In some subtasks, further tags were used in order to distinguish between strong and weak positive tweets, between strong and weak negative tweets, or between neutral (neither positive nor negative) and objective (no polarity at all) tweets.

Tables II and III show the performance results of our system in TASS and SemEval, respectively. We compare the F-score achieved by our system with respect to those reached by both the best and worst systems, as well as the average considering all systems in each competition. The F-score is a weighted average of the *precision* and *recall*. Precision is the number of all positive results returned by the system divided by the number of results which were actually returned, while recall is the number of correct positive results returned by the system divided by the number of positive results that should have been returned. F-score reaches its best value at 1 and worst score at 0. Each competition consists in several tasks. For example, in TASS, Task-1 makes use of 6 polarity tags, Task-2 only uses 4 tags, and Task-3 deals with determining the polarity at entity level. In SemEval, each task corresponds to a different test dataset: LiveJournal sentences (Task-1), SMS messages (Task-2), tweets of SemEval 2013 (Task-3), tweets of SemEval-2014 (Task-4), and tweets with sarcasm (Task-5).

For each subtask, the ranking of CitiusSentiment is enclosed in brackets. For instance, it was ranked as the 3th best system out of 13 participants in Task-2 of TASS, and it was the first system out of 5 in Task-3 of the same competition. The results obtained in TASS contest, focused on Spanish tweets, were significantly better than those obtained in SemEval, focused on English short messages. Yet, in all tasks, including those of SemEval, the scores of CitiusSentiment are clearly above the average.

C. Evaluation of the Big Data infrastructure

In order to test the processing times and scalability of our system 68 Spanish terms were selected as targets for the Tweet Mining Module. These terms were relevant in mid-February 2016, when the com-

⁶<http://corpus.rae.es/lfrrecuencias.html>

⁷<http://www.insightin.com/es/>

TABLE II
 F-SCORE AND RANKING OF OUR SYSTEM (CITIUSSENTIMENT) IN THE TASS COMPETITION: SENTIMENT ANALYSIS ON SPANISH TWEETS.

System	Task-1	Task-2	Task-3
Best system	61.6	68.6	39.4
CitiusSentiment	55.8 (4/13)	66.8 (3/13)	39.4 (1/5)
Worst system	12.6	23.0	30.7
Average	43.3	53.0	36.9

TABLE III
 F-SCORE AND RANKING OF OUR SYSTEM (CITIUSSENTIMENT) IN THE SEMEVAL COMPETITION, NAMELY TASK 9 FOCUSED ON SENTIMENT ANALYSIS IN TWITTER (ONLY ENGLISH MICROTTEXTS).

System	Task-1	Task-2	Task-3	Task-4	Task-5
Best system	74.8	70.2	72.1	70.9	58.1
CitiusSentiment	64.5 (29/50)	58.2 (24/50)	63.2 (24/50)	62.9 (27/50)	46.1 (25/50)
Worst system	29.3	24.6	34.2	33.0	28.9
Average	59.9	52.4	56.4	56.9	42.8

pilation of tweets started, e.g., political parties, famous people, new devices, etc. A total amount of 8,016,139 tweets written in Spanish (3.1 GiB) were stored in the HBase database. The selected terms for the tests were: *corrupción* (corruption), *gobierno* (government) and *elecciones* (elections). Table IV shows the number of matches (tweets containing the considered word) and the positivity ratio obtained by the system related to each term.

Due to the fact that Hadoop handles each region of a HBase table like a split, the original table was split into equal parts progressively, getting always a number of regions power of two from 1 to 32. A total amount of 50 GiB of RAM per node was configured for YARN containers, 7 GiB for every map or reduce container. 3GiB for HBase and the rest for other Hadoop processes and the OS. The number of cores per node (YARN) was set to 8.

HBase tables are split in regions according to the established policies. For this experiment, the split policy that was used was the `ConstantSizeRegionSplitPolicy`. Thereby, every region is split when one of their column families exceeds the value for the `hbase.hregion.max.filesize` attribute. This parameter was set to 20GiB, getting only one region in the table for all the tweets that were stored. For the purpose of obtaining the different number of regions, the tests were performed on the table in many iterations. In every iteration, each region was manually divided in two equal parts through the HBase Shell. It is important to set some properties to the Scan instance that will be used by the mappers. The cache blocks option must be disabled (the MapReduce job is a batch reading and this option is intended for frequently accessed rows). Furthermore, an important parameter when performing readings is the client prefetch (called caching). This refers about how many rows from the

table could be retrieved in a single RPC call by the client. Using the default value (1) will cause a huge degradation of the execution performance. For the tests, the chosen values were 50 and 5,000 (rows).

Results of the experiments carried out are displayed in Table V, which show that the system scales up quite good when the physical resources are enough to handle the launched tasks. Note that our system is capable of processing more than 8 million tweets in few minutes. Considering the term *elecciones*, the one with the lowest number of coincidences, using 32 mappers degrades the performance due to the overhead. In addition, low values for the caching parameter in the Scanner object causes worse results. This behavior is clearly detected with few regions. Additional tests were executed using other values for caching, but higher values for this parameter do not show relevant differences with respect to the results shown in Table V.

VII. CONCLUSIONS

Twitter can be considered as a large source of short texts (tweets) containing user opinions. Making sentiment analysis on tweets is challenging from the natural language processing perspective, but also in terms of performance when huge amounts of tweets should be processed. Both challenges are addressed in this paper.

First, our sentiment classifier (*CitiusSentiment*) has obtained good results considering both Spanish and English tweets. We must highlight that our classifier took part as a participant in two different sentiment analysis competitions, reaching scores clearly above the average with respect to other approaches.

Regarding the second challenge, we propose a new system that takes advantage of parallel architectures using Big Data technologies with the aim of speeding up the sentiment analysis process. The system consists of three modules. The first module is respon-

TABLE IV
 SUCCESSFULLY PROCESSED TWEETS, MATCHES AND POSITIVITY RATIO FOR THE SELECTED SPANISH TERMS.

Term	Processed tweets	Matches	Positivity ratio
<i>corrupción</i> <i>gobierno</i> <i>elecciones</i>	8,016,139	195,590	0.304 (negative)
		99,601	0.433 (negative)
		30,001	0.577 (positive)

TABLE V
 PROCESSING TIME (IN MINUTES) FOR THE SELECTED SPANISH TERMS.

No. of reg.	<i>corrupción</i>		<i>gobierno</i>		<i>elecciones</i>	
	Scanner caching (prefetch)					
	50	5,000	50	5,000	50	5,000
1	52.5	50.7	29.2	27.8	12.4	10.9
2	30.2	27.5	15.9	15.0	7.4	6.7
4	17.9	16.6	10.2	9.1	4.7	4.0
8	11.2	8.4	5.7	5.5	3.2	2.7
16	7.4	6.5	4.5	4.1	2.4	2.1
32	5.3	5.1	3.9	3.7	2.4	2.3

sible of collecting the tweets. It has been designed in such a way that captures 46x more tweets per second than the standard Twitter Streaming API. The second module uses Hadoop to process the tweets captured by the first module. Performance results show that our system is able to analyzed more than 8 million tweets in a few minutes on a small cluster. Finally, a GUI is also provided for the users.

ACKNOWLEDGMENT

This work was supported by MINECO (ref. TIN2014-54565-JIN) and RedPLIR-Xunta de Galicia (ref. CN2014/034). It was also supported by AWS in Education Grant award.

REFERENCES

[1] "Apache Hadoop home page," <http://hadoop.apache.org/>, [Online; accessed February, 2016].
 [2] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Symposium on Operating System Design and Implementation*, 2004, pp. 10–10.
 [3] Satish Narayana Srirama, Pelle Jakovits, and Eero Vainikko, "Adapting scientific computing problems to clouds using MapReduce," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 184–192, 2012.
 [4] Jared Kramer and Clara Gordon, "Improvement of a Naive Bayes Sentiment Classifier Using MRS-Based Features," in *Joint Conf. on Lexical and Computational Semantics*, 2014, pp. 22–29.
 [5] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan, "Thumbs Up?: Sentiment Classification Using Machine Learning Techniques," in *Conf. on Empirical Methods in Natural Language Processing*, 2002, vol. 10, pp. 79–86.
 [6] Franco Salveti, Christoph Reichenbach, and Stephen Lewis, "Opinion polarity identification of movie reviews," in *Computing Attitude and Affect in Text: Theory and Applications*, pp. 303–316, 2006.
 [7] Maite Taboada, Julian Brooke, Milan Tofloski, Kimberly Voll, and Manfred Stede, "Lexicon-based methods for sentiment analysis," *Comput. Linguist.*, vol. 37, no. 2, pp. 267–307, 2011.
 [8] Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu, "Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets," *CoRR*, vol. abs/1308.6242, 2013.

[9] X. Saralegi and I. San Vicente, "TASS: Detecting Sentiments in Spanish Tweets," in *Workshop on Sentiment Analysis at SEPLN*, 2012.
 [10] Aliaksei Severyn, Alessandro Moschitti, Olga Uryupina, Barbara Plank, and Katja Filippova, "Multi-lingual opinion mining on youtube," *Inform. Processing & Management*, vol. 52, no. 1, pp. 46–60, 2015.
 [11] David Vilares, Miguel A. Alonso, and Carlos Gómez-Rodríguez, "On the usefulness of lexical and syntactic processing in polarity classification of twitter messages," *Journal of the American Society for Information Science*, vol. 66, no. 9, pp. 1799–1816, 2015.
 [12] B. Liu, E. Blasch, Y. Chen, D. Shen, and G. Chen, "Scalable sentiment classification for big data analysis using naive bayes classifier," in *IEEE Int. Conference on Big Data*, 2013, pp. 99–104.
 [13] L. Banić, A. Mihanović, and M. Brakus, "Using Big Data and sentiment analysis in product evaluation," in *Int. Convention on Information Communication Technology Electronics Microelectronics*, 2013, pp. 1149–1154.
 [14] A. H. A. Rahnama, "Distributed real-time sentiment analysis for big data social streams," in *Int. Conf. on Control, Decision and Information Technologies*, 2014, pp. 789–794.
 [15] J. Conejero, P. Burnap, O. Rana, and J. Morgan, "Scaling archived social media data analysis using a hadoop cloud," in *Conf. on Cloud Computing*, 2013, pp. 685–692.
 [16] Chris Manning, Prabhakar Raghadvan, and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press, Cambridge, MA, USA, 2008.
 [17] Peter Sheridan Dodds, Cameron Decker Harris, Isabel M. Kloumann, Catherine A. Bliss, and Christopher M. Danforth, "Temporal patterns of happiness and information in a global social network: Hedonometrics and Twitter," *PLoS ONE*, vol. 6, no. 12, pp. e26752, 2011.
 [18] Bing Liu, Minqing Hu, and Junsheng Cheng, "Opinion Observer: Analyzing and Comparing Opinions on the Web," in *Int. World Wide Web conference*, 2005, pp. 342–351.
 [19] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani, "SentWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining," in *Int. Conf. on Language Resources and Evaluation*, 2010, pp. 2200–2204.
 [20] Grigori Sidorov, "Empirical study of opinion mining in spanish tweets," *Lecture Notes in Artificial Intelligence*, vol. 7629–7630, 2012.
 [21] J. M. Abujín, J. C. Pichél, T. F. Pena, P. Gamallo, and M. García, "PerLdoop: Efficient execution of Perl scripts on Hadoop clusters," in *Int. Conf. on Big Data*, 2014, pp. 766–771.
 [22] Pablo Gamallo, Marcos Garcia, and Santiago Fernández-

- Lanza, "A naive-bayes strategy for sentiment analysis on spanish tweets," in *Workshop on Sentiment Analysis at SEPLN (TAAS)*, 2013, pp. 126-132.
- [23] Sara Rosenthal, Preslav Nakov, Alan Ritter, and Veselin Stoyanov, "SemEval-2014 Task 9: Sentiment Analysis in Twitter," in *Int. Workshop on Semantic Evaluation*, 2014.

Evaluación del rendimiento de una implementación Cloud para un clasificador neuronal aplicado a imágenes hiperespectrales

Juan Mario Haut, Mercedes Paoletti, Javier Plaza y Antonio Plaza¹

Resumen—

La tecnología actual permite a los sensores hiperespectrales capturar cientos de imágenes, en diferentes longitudes de onda, sobre una misma zona sobre la superficie de la Tierra. Las denominadas *imágenes hiperespectrales* se caracterizan por su gran volumen y dimensionalidad, lo que complica considerablemente su almacenamiento y procesamiento. Si además tenemos en cuenta que gran parte de los algoritmos de procesamiento de este tipo de datos presentan una elevada complejidad computacional, resulta fácil justificar la aparición de implementaciones de este tipo de técnicas sobre arquitecturas de computación de altas prestaciones. En particular, las arquitecturas cloud permiten el procesamiento distribuido de los datos, que normalmente se encuentran ubicados en diferentes centros de procesamiento. En este artículo presentamos un estudio del rendimiento computacional de una implementación cloud (desarrollada utilizando Apache Spark) de una arquitectura de red neuronal orientada a la clasificación de datos hiperespectrales. Los resultados experimentales sugieren que las arquitecturas distribuidas tipo cloud permiten procesar de forma distribuida grandes conjuntos de datos hiperespectrales.

Palabras clave— imágenes hiperespectrales, computación neuronal, arquitecturas distribuidas, computación cloud.

I. INTRODUCCIÓN

Las imágenes hiperespectrales contienen información en cientos de bandas espectrales contiguas, lo que incrementa significativamente su tamaño en relación con las imágenes tradicionales, imponiendo nuevos requerimientos en términos del almacenamiento y el procesado de este tipo de datos. El crecimiento exponencial de estos requerimientos, debido al avance en la tecnología de desarrollo de los instrumentos de adquisición y a la disponibilidad de cada vez más nuevas misiones que generan un flujo continuo de datos multi/hiperespectrales, está provocando la aparición de repositorios de datos hiperespectrales de grandes dimensiones [1]. Por ejemplo, el sensor AVIRIS (Airbone Visible/Infrared Imaging Spectrometer) operado por el Jet Propulsion Laboratory de la NASA, presenta unas tasas de adquisición de datos de 2.5 MB/s (casi 9 GB/hora). Un caso similar es el sensor Hyperion [1], que adquiere casi 71.9 GB/hora (over 1.6 TB/día). La mayor parte de las misiones satélite que estarán operativas en breve, como el

programa de mapeo y análisis ambiental (EnMAP) ¹ presentan tasas de adquisición de datos similares.

En la actualidad, las plataformas de computación cloud están siendo utilizadas con el fin de procesar datos adquiridos de forma remota en arquitecturas distribuidas. En este sentido, la computación cloud ofrece avanzadas capacidades para computación orientada a servicios y computación de altas prestaciones. El uso de computación cloud para el análisis de grandes repositorios de datos hiperespectrales puede considerarse una solución natural, resultado de la evolución de técnicas previamente desarrolladas para otros tipos de plataformas de computación [2], [3]. Sin embargo, existen pocos ejemplos en la literatura reciente orientados al uso de infraestructuras de computación cloud para la implementación de técnicas de análisis hiperespectral en general, y para la clasificación supervisada de datos hiperespectrales en particular.

Desde los años 90, las redes neuronales han atraído la atención de gran cantidad de investigadores pertenecientes al área del análisis hiperespectral [4], [5] y, especialmente, los dedicados a la clasificación de datos hiperespectrales [6], [7], como una consecuencia directa de su éxito en el campo de reconocimiento de patrones [8]. La principal ventaja de este tipo de aproximaciones sobre los métodos probabilísticos radica en el hecho de que no necesitan conocimiento previo sobre la distribución estadística de las clases. Además, son una posibilidad atractiva debido a la disponibilidad de múltiples técnicas de entrenamiento para datos linealmente no separables [9], a pesar de que estas técnicas se hayan visto tradicionalmente afectadas por su complejidad algorítmica y computacional [10], así como por el número de parámetros que necesitan ser ajustados para su correcta aplicación. Han sido muchos los algoritmos neuronales propuestos en la literatura para la clasificación de datos hiperespectrales, incluyendo enfoques no parametrizados, tanto supervisados como no supervisados [11], [12], [13], [14], [15], siendo las redes de propagación hacia delante (*feedforward networks*, FN) las más comúnmente utilizadas para tareas de clasificación de datos hiperespectrales.

En este artículo, exploramos la posibilidad de utilizar arquitecturas distribuidas para el procesado de datos hiperespectrales. Utilizamos un clasificador neuronal como caso de estudio,

¹Dpto. Tecnología de los Computadores y de las Comunicaciones, Universidad de Extremadura, e-mail: juanmariohaut@unex.es, mpaolett@alumnos.unex.es, jplaza@unex.es, aplaza@unex.es

¹<http://www.enmap.org/>

centrándonos en el uso de arquitecturas neuronales de clasificación supervisada basadas en redes FN para demostrar la posibilidad de utilizar tecnología de computación cloud para clasificar de forma eficiente datos hiperespectrales, acelerando la computación asociada a este proceso.

El resto del artículo se organiza de la siguiente forma. La Sección II presenta el diseño del entorno de trabajo distribuido que será utilizado en la implementación evaluada. La Sección III describe el algoritmo de clasificación neuronal utilizado. En la Sección IV describimos la implementación distribuida. La Sección V evalúa el rendimiento de la implementación considerada bajo el punto de vista de su precisión y, especialmente, de su eficiencia computacional. Por último, la Sección VI expone una serie de conclusiones y consideraciones para posibles futuros trabajos.

II. DISEÑO DEL FRAMEWORK DISTRIBUIDO

Con el fin de desarrollar un marco distribuido para implementar el algoritmo del perceptrón multicapa en arquitecturas de computación *cloud*, hemos abordado dos cuestiones principales: 1) el modelo de programación distribuida y 2) el motor de la computación.

Para la programación distribuida, se recurre al modelo MapReduce [2], aprovechando al máximo las capacidades de alto rendimiento proporcionados por las arquitecturas de computación en *cloud*. En este modelo, una tarea es procesada por dos operaciones distribuidas: *map* y *reduce*. Los *datasets* están organizados como pares clave/valor, y la función *map* procesa los pares clave/valor con el fin de generar un conjunto de pares intermedios, dividiendo una tarea en varias sub tareas independientes para que se ejecuten en paralelo. La función *reduce* se encarga de procesar todos esos valores intermedios asociados con la misma clave intermedia, a continuación, se juntan todos los resultados intermedios y se genera el resultado final.

En cuanto al motor de computación distribuida, la primera solución considerada fue Apache Hadoop² debido a su fiabilidad y escalabilidad, así como su naturaleza *open source*. Sin embargo, Apache Hadoop sólo es compatible con cálculos simples, que se realizan de una sola pasada, por tanto, en general, no es adecuado para algoritmos iterativos tales como el perceptrón multicapa. Apache Spark³ es un motor de computación actual cuya finalidad es el procesamiento de grandes volúmenes de datos en las arquitecturas de computación *cloud*, es tolerante a fallos, y proporciona, de una manera rápida, un procesamiento de datos generales sobre grandes plataformas distribuidas. Como hemos dicho anteriormente, no sólo es compatible con cálculos simples, también con los algoritmos iterativos. Por todo lo indicado anteriormente, el diseño de nuestro framework distribuido en paralelo para la

clasificación de datos hiperespectrales utilizando Spark Apache es descrito gráficamente en la Fig. 1.

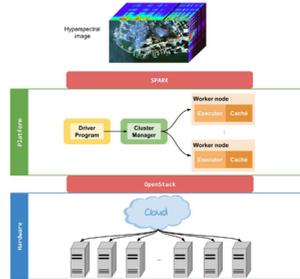


Fig. 1. Descripción de la arquitectura de Apache Spark utilizada en nuestros experimentos.

Como se muestra en la Fig. 1, la arquitectura tiene dos partes principales:

- La *zona de hardware*: contiene las máquinas físicas que soportan nuestras máquinas virtuales, que son creadas en la plataforma OpenStack⁴, el sistema operativo que controla las máquinas del *cloud*, almacenamiento y recursos de red a través de un centro de procesamiento de datos, todo ello gestionado a través de un panel de control que permite a los administradores controlar a la vez que provisionar recursos a las máquinas a través de una interfaz web.
- La *zona de la plataforma*: el framework Apache Spark en la Fig. 1) está instalado sobre un conjunto de máquinas virtuales Linux Ubuntu, creadas en OpenStack. Nuestro clúster tiene diferentes tipos de nodos. El algoritmo MLP se ha diseñado utilizando la librería de *Machine Learning* MLib⁵, y la aplicación se integra en Spark y el framework OpenStack, como se ilustra gráficamente en la Fig. 2. Cuando lanzamos una instancia del clasificador MLP, el nodo maestro gestiona los recursos del clúster y los esclavos realizan tareas individuales en los datos. El maestro divide el trabajo en tareas y coordina la asignación de las mismas, siguiendo el modelo MapReduce.

²<http://hadoop.apache.org>

³<http://spark.apache.org/>

⁴https://wiki.openstack.org/wiki/Main_Page

⁵<https://spark.apache.org/docs/latest/mllib-guide.html>

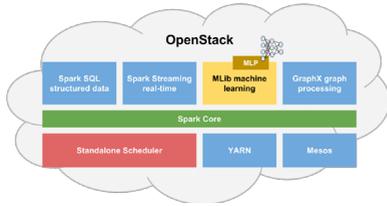


Fig. 2. Integración de Apache Spark y el framework OpenStack, ambos utilizados para la implementación del MLP.

III. ALGORITMO DE CLASIFICACIÓN BASADO EN COMPUTACIÓN NEURONAL

A. Redes de propagación hacia delante

Como hemos mencionado en la Sección I, las FNs han sido estudiadas en profundidad y ampliamente utilizadas desde la aparición del famoso algoritmo de retropropagación (backpropagation, BP) [16], un método de optimización basado en el método del gradiente de primer orden, que presenta dos inconvenientes principales: su convergencia es bastante lenta y existe la posibilidad de que el método se quede atrapado en un mínimo local, especialmente si los parámetros de la red no se han ajustado adecuadamente. Con el objetivo de contrarrestar los inconvenientes del algoritmo original, se han propuesto en la literatura diferentes estrategias de optimización de órdenes superiores, que se caracterizan por ser más rápidas y menos parametrizadas [17], [18].

En concreto, las redes de propagación hacia delante con una única capa oculta (*single layer feedforward network*, SLFN) son las más comúnmente utilizadas en el ámbito de la clasificación de datos hiperespectrales. Sean $\{(\mathbf{x}_i \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^d, \mathbf{t}_i \in \mathbf{R}^m, i = 1, 2, \dots, k\}$ K patrones de entrenamiento diferentes. Para una SLFN con L neuronas ocultas y función de activación $g_i(x)$, su salida o_j puede expresarse como:

$$o_j = \sum_{i=1}^L \beta_{ij} g_i(\mathbf{x}) = \sum_{i=1}^L \beta_{ij} g(\mathbf{w}_i \cdot \mathbf{x} + \mathbf{b}_i) \quad (1)$$

donde \mathbf{w}_i es el vector de pesos que conecta la i -ésima neurona oculta con las neuronas de la capa de entrada, β_{ij} es el vector de pesos que conecta la i -ésima neurona oculta con la j -ésima neurona de salida y \mathbf{b}_i es el bias de la i -ésima neurona oculta.

Podemos definir la función de error de una SLFN como el error cuadrático medio (*Mean Square Error*, MSE):

$$E = 1/2 \sum_{i=1}^K \|o_i - t_i\|^2$$

El objetivo del aprendizaje es minimizar la distancia entre la salida de la red y la salida

deseada ajustando los pesos de la red. Como hemos mencionado antes, tradicionalmente, dicho ajuste suele realizarse mediante algoritmos de aprendizaje basados en el descenso del gradiente, que suelen presentar dos problemas fundamentales: son costosos computacionalmente (especialmente ante datos de alta dimensionalidad) y pueden quedar atrapados en algún mínimo local de la superficie del error (crucial si el mínimo local está muy lejos del mínimo global). Sin embargo, la minimización de la función de error de una SLFN puede verse estrictamente como un problema de optimización. De esa forma, son muchas las alternativas basadas en métodos de optimización de segundo orden que pueden emplearse en combinación con el algoritmo BP [19], [20], [18], [21]. Diferentes implementaciones (Scikit, Spark, etc.) utilizan en su perceptrón el método de optimización L-BFGS, también llamado método de Broyden-Fletcher-Goldfarb-Shanno con límite de memoria [22].

B. Método de Broyden-Fletcher-Goldfarb-Shanno

Este método se clasifica dentro de los métodos de minimización multivariante quasi-newton, los cuales pretenden minimizar una función de error $E(x)$ con $x \in \mathbf{R}^n$ y E función real $E: \mathbf{R}^n \rightarrow \mathbf{R}$.

Los métodos de descenso de gradiente y de gradiente conjugado pueden minimizar esa función, pero son lentos, con un orden de convergencia lineal. Por otra parte, el método de Newton es ligeramente más rápido, con un orden de convergencia cuadrático, gracias al uso de la información de la matriz Hessiana⁶ de $E(x)$. Sin embargo, el cálculo de esta matriz es bastante costoso y a veces no es factible. Por ello, los métodos quasi-newton utilizan una aproximación al Hessiano mediante la información que obtienen del gradiente.

Tomemos $E(x) \in \mathbf{R}^n$ continua y con segundas derivadas parciales. Para los puntos x_k y x_{k+1} con gradiente $g_i = \nabla f(x_i)$ y matriz Hessiana constante F , se cumple:

$$q_k \equiv g_{k+1} - g_k, \quad p_k \equiv x_{k+1} - x_k$$

Por lo que $q_k = F \cdot p_k$. Entonces podemos aproximar sucesivamente F mediante matrices aproximadas H a partir de n direcciones linealmente independientes (p) y sus respectivos gradientes (q): $H = Q \cdot P^{-1}$, donde Q es la matriz de los sucesivos q_k y P la matriz de los sucesivos p_k . Por otra parte, se necesita calcular la inversa del Hessiano, F^{-1} . A partir de $q_k = F \cdot p_k$ podemos obtenerlo como $H \cdot q_k = p_k$, entonces tenemos $H = F^{-1}$.

⁶Matriz de segundas derivadas parciales de la función $E(x)$:

$$F = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Uno de los primeros métodos quasi-newton fue el Davidon-Fletcher-Powell (ver algoritmo 1).

$$Y = g(W^T \cdot X + B) \quad (4)$$

Donde:

Algorithm 1 Algoritmo Davidon-Fletcher-Powell

```

1: procedure DFP
2:    $d_k = -H_k \nabla E(x_k)$ 
3:    $p_k = \alpha_k \cdot d_k \leftarrow \alpha_k$  por búsqueda lineal
4:    $x_{k+1} = x_k + p_k$ 
5:    $q_k = \nabla E(x_{k+1}) - \nabla E(x_k)$ 
6:    $H_{k+1} = H_k + \frac{p_k \cdot p_k^t}{p_k^t \cdot q_k} - \frac{H_k \cdot q_k \cdot q_k^t \cdot H_k}{q_k^t \cdot H_k \cdot q_k}$ 
7: end procedure

```

- **Y** es la matriz que representa la salida con tamaño $L \times h$.
- W^T es la matriz de pesos de tamaño $m \times L$.
- **X** es el la matriz de entrada a la capa de tamaño $m \times h$.
- **b** es el vector bias con tamaño $L \times h$.
- **g** es la función de activación.

El método de Broyden-Fletcher-Goldfarb-Shanno aplica la fórmula $q_k = H \cdot p_k \rightarrow q_k = B_{k+1} \cdot p_k$, donde B_k es la k-ésima aproximación a la matriz hessiana. Esta expresión tiene la particularidad de que cualquier fórmula para para actualizar F^{-1} puede transformarse en una fórmula para actualizar la propia H , con tan solo sustituir F_k^{-1} por B_k e intercambiar q_k y p_k . Entonces, si intercambiamos esta forma en la fórmula de actualización de DFP obtendremos:

$$B_{k+1} = B_k + \frac{q_k \cdot q_k^t}{q_k^t \cdot p_k} - \frac{B_k \cdot p_k \cdot p_k^t \cdot B_k}{p_k^t \cdot B_k \cdot p_k}$$

Así pues, su inversa analítica será la actualización propiamente dicha de BFGS:

$$H_{k+1} = H_k + \left(1 + \frac{q_k^t H_k q_k}{q_k^t p_k}\right) \frac{p_k p_k^t}{q_k^t q_k} - \frac{p_k q_k^t H_k + H_k q_k p_k^t}{q_k^t p_k} \quad (2)$$

IV. IMPLEMENTACIÓN DISTRIBUIDA DEL CLASIFICADOR NEURONAL

A. Implementación Distribuida en Apache Spark

Apache Spark implementa una versión del MLP [23] que, básicamente, puede definirse como una FN que puede contener más de una capa oculta. En esta versión, se introduce el tamaño de bloque, *blocksize*, con el fin de acelerar el cálculo matricial. El algoritmo apila los datos en particiones de tamaño h , si el tamaño es mayor que los datos, entonces ajusta la partición a los datos. Esto se traduce en que la obtención tradicional de la salida de la capa oculta -ver ecuación (3) pasa a ser optimizado como se aprecia en la ecuación (4). Este proceso también es extrapolable a los cálculos realizados en la capa de salida.

$$y = g(W^T \cdot x + b) \quad (3)$$

Donde:

- **y** es un vector que representa la salida de la capa oculta, de tamaño L , siendo L el número de neuronas ocultas.
- W^T es la matriz de pesos de tamaño $m \times L$, siendo m la dimensionalidad de los datos de entrada.
- **b** es el vector bias con tamaño L .
- **g** es la función de activación.

Inicialmente, los datos son cargados en estructuras de tipo *dataframe*; este tipo de datos son una colección distribuida organizadas por columnas, las cuales tienen asociado un nombre. Al igual que los RDD, este tipo tiene evaluación perezosa. Es decir el cálculo que queremos realizar solo se lleva a cabo si representa una acción. Lo interesante de los *dataframes* es que todos los datos que contienen, se distribuyen de manera automática por los *workers*.

Función de los nodos:

1. El máster tiene una tarea primordial en esta implementación, sus principales funciones son la del envío de los parámetros a los esclavos, computar el gradiente final y decidir si volver a iterar o parar en función de la condición de salida.
2. Los nodos esclavos, por otra parte, se encargan de la realización de los cálculos.

Entrenar el clasificador: Esta etapa comprende dos acciones:

1. Propagación hacia delante: En esta fase, los pesos, se establecen de manera aleatoria. A partir de ahí, en la segunda y sucesivas, lo hace mediante la siguiente ecuación:

$$W' = W - \text{tamPasoIter} \cdot (\text{gradiente} + \text{regGradient}(w))$$

Por tanto, los pesos de la etapa k, W_k , serían los pesos de la etapa anterior, menos el tamaño del paso de la iteración (o paso del gradiente en k) multiplicado por la suma del gradiente, más el gradiente de la parte regularizada en la función de coste de los W_k .

2. Propagación hacia atrás: Una vez que los esclavos han calculado sus respectivos gradientes, el máster, obtiene el gradiente global mediante la siguiente ecuación:

$$\text{gradTotal} = \text{gradTotal} - \frac{\sum_{i=1}^K \text{gradientes}}{\text{numGradientes}}$$

De tal manera que el valor del gradiente se actualiza buscando la convergencia hacia el mínimo.

La Figura 3 muestra un esquema resumiendo el funcionamiento del algoritmo distribuido.

```

Algorithm 2 Algoritmo SLFN
1: procedure CLASIFICADOR_SLFN(maxIter,
   capas, tamBloque, maxTol semilla)
2:   for iteracion < maxIter & tolerancia <
   maxTol do
3:     Máster envía parámetros a esclavos
4:     ▷ + esclavos, - cómputo + comunicación
5:     ▷ - tamaño de bloque, - cómputo +
   comunicación
6:     Esclavos calculan el gradiente.
7:     Esclavos envían el gradiente al máster.
8:     Máster calcula el gradiente final.
9:   end for
10: end procedure
    
```

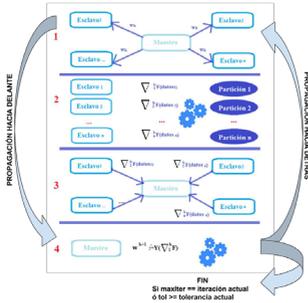


Fig. 3. Esquema de funcionamiento del MLP en Spark.

Número óptimo de workers: Para la elección del número de workers, se puede aplicar la siguiente expresión:

$$nWorkers = \max\left(\left\lceil \frac{m \cdot S \cdot \ln 2}{p \cdot \left(\frac{128 \cdot S}{2 \cdot b} + 2 \cdot c\right)} \right\rceil, 1\right)$$

Siendo:

- m = Tamaño de los datos
- S = Número de patrones * Número de clases
- p = FLOPS
- b = Velocidad de la red
- c = Congestión de la red

B. Implementación iterativa del Perceptrón Multicapa

Scikit-Learn [24] es una librería de *Machine Learning* para Python. Esta librería contiene su propia versión del perceptrón multicapa, *sklearn.neural_network*, las principales opciones configurables son el número de capas y neuronas en cada capa, *hidden_layer_sizes*, función de activación de las capas ocultas, *activation*, la tolerancia para el

proceso de optimización, *tol*, el número máximo de iteraciones, *max_iter* y el algoritmo de optimización, *algorithm*.⁷

V. VALIDACIÓN EXPERIMENTAL

A. Imágenes hiperspectrales

Las imágenes [25] utilizadas en los experimentos, fueron adquiridas por el sensor AVIRIS [26]. El conocido dataset Indian Pines fue obtenido en 1992 y comprende una zona agrícola con múltiples tipos de vegetación en múltiples zonas:

1. La primera escena (Fig. 4) tiene un tamaño de 145x145 píxeles, que mezcla zona forestal con agrícola. Tiene 220 bandas en el rango de 400 a 2500 nm, con una resolución espectral de 10 nm, resolución espacial moderada de 20nm y 16 bits de resolución radiométrica. Después de un análisis inicial, 8 bandas han sido eliminadas debido al ruido existente en los datos, quedando finalmente, 212 bandas útiles. En torno a la mitad de los píxeles de la imagen (10366 de 21025) tienen información del *ground-truth* comprendida en 16 clases diferentes.

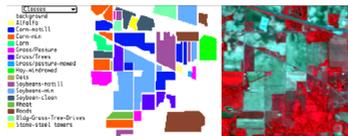


Fig. 4. Imagen en falso color y *ground-truth*

2. La segunda escena (Fig. 5) tiene un tamaño de 2678x614 píxeles, recogida sobre el mismo área pero es mucho más extensa. Contiene 220 bandas en el rango de 400 a 2500 nm, con una resolución espectral de 10 nm, resolución espacial moderada de 20nm y 16 bits de resolución radiométrica. Después de un análisis inicial, 8 bandas han sido eliminadas debido al ruido existente en los datos, quedando finalmente, 212 bandas útiles. Existen un total de 58 clases diferentes. El porcentaje de píxeles con *ground-truth* es de 20.33% (334245 de 1644292 píxeles).

⁷http://scikit-learn.org/dev/modules/generated/sklearn.neural_network.MLPClassifier.html

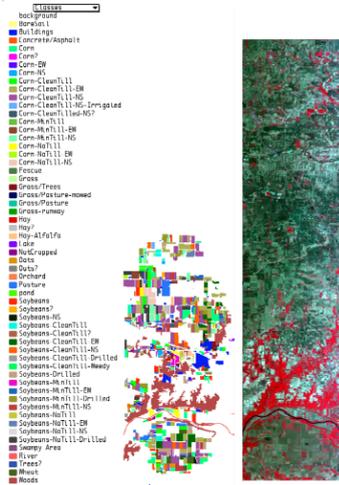


Fig. 5. Imagen en falso color y *grown-truth*

B. Descripción de los experimentos

El entorno distribuido con el que hemos probado nuestra implementación está descrito en la sección II. Como hemos mencionado en esa sección, la plataforma de *cloud computing* utilizada para nuestra evaluación experimental es OpenStack. Este software es un conjunto de tecnologías *Open Source* que proveen un desarrollo escalable en el entorno de *cloud computing*. Nuestro entorno está compuesto por Intel(R) Xeon(R) CPUs E5430 @ 2.66GHz (8 cores), 16 GB RAM, Shared storage, NetApp FAS3140. Los nodos virtuales que forman parte del clúster construido con Apache Spark en el hardware especificado tienen dos CPUs virtuales, 4GB de RAM y 40GB de disco duro cada uno. La máquina que ejecuta el algoritmo secuencial ha necesitado 8GB de RAM debido al desbordamiento de memoria causado por el procesamiento de la imagen grande de Indian Pines, el resto de parámetros mantiene la misma configuración que las del clúster. En nuestros experimentos, hemos usado Java 1.8.0_92-b14, Ubuntu 14.04 x64 LTS como sistema operativo, Python 2.7.10, las versiones actuales, en desarrollo, Scikit Learn 0.18.dev0 y Apache Spark 2.11.

Cabe destacar que, como el principal objetivo de este trabajo es analizar el rendimiento computacional de la versión distribuida del algoritmo, hemos fijado un error objetivo muy bajo y forzamos la detención del clasificador por número de iteraciones (a pesar de que el número de iteraciones pueda resultar excesivo en el contexto del problema). Así, logramos ejecutar siempre el mismo número de iteraciones, haciendo que el análisis de la aceleración tenga mayor

consistencia.

B.1 Algoritmo iterativo

En este primer experimento hemos lanzado 5 ejecuciones con la imagen pequeña de Indian Pines (en el caso de la imagen grande, solo hemos realizado una clasificación en serie debido a su elevado tiempo de ejecución), imágenes descritas en el apartado V-A. En estos tests, hemos establecido una configuración común de tolerancia de 1e-07, un número de iteraciones de 4000, sigmoide como función de activación de las neuronas de la capa oculta, el algoritmo de optimización 'l-bfgs'. Para la imagen pequeña, hemos establecido una capa con 135 neuronas ocultas mientras que para la imagen grande, una capa con 180 neuronas ocultas. La Tabla I muestra los tiempos medios de ejecución y la precisión media obtenidas al ejecutar el algoritmo de clasificación 5 veces sobre la imagen pequeña y 1 sobre la imagen grande.

Imagen	AVG Tiempo	Std Tiempo	AVG Precisión	Std Precisión
(145 x 145)	535.588	16.729	0.845	0.010
(2678 x 614)	38730.236	-	0.559	-

TABLA I

TIEMPOS DE EJECUCIÓN Y RESULTADOS DE PRECISIÓN OBTENIDOS SOBRE LOS DOS CONJUNTOS DE DATOS PROCESADOS UTILIZANDO LA IMPLEMENTACIÓN SERIE.

B.2 Algoritmo distribuido

En este experimento hemos lanzado 5 ejecuciones con las dos imágenes de Indian Pines descritas en el apartado V-A, variando el número de nodos con 1, 2, 4 y 8 nodos esclavos⁸. En estos tests, hemos establecido una configuración común de tolerancia de 1e-07, un número de iteraciones de 4000, sigmoide como función de activación de las neuronas de la capa oculta, el algoritmo de optimización 'l-bfgs'. Para la imagen pequeña, hemos establecido una capa con 135 neuronas ocultas mientras que para la imagen grande, una capa con 180 neuronas ocultas.

C. Discusión de Resultados

La Tabla II muestra los tiempos de ejecución y los resultados de precisión obtenidos sobre las dos imágenes consideradas utilizando diferentes números de nodos. Como puede observarse, los resultados de precisión son muy similares con respecto a los obtenidos por la implementación serie, mientras que el tiempo de procesamiento disminuye sensiblemente al utilizar la arquitectura distribuida considerando un número suficiente de nodos de computación.

⁸Excepto con Indian Pines pequeña que ha sido 1, 2, 4 y 7 nodos

Imagen	Nodos	Tiempo		Precisión	
		AVG	Std	AVG	Std
(145 × 145)	1	4306.1045	89.4959	0.8194	0.0
	2	2265.8825	228.8569	0.8216	0.0026
	4	1476.4164	80.4546	0.8220	0.0021
	7	1388.5527	14.6431	0.8322	0.0023
(2678 × 614)	1	141769.2273	9572.8992	0.5550	0.0
	2	76782.8940	178.7860	0.5569	0.0008
	4	49348.3759	313.6937	0.5562	0.0016
	8	18488.2113	221.1524	0.5501	0.0010

TABLA II

TIEMPOS DE EJECUCIÓN Y PRECISIÓN OBTENIDOS SOBRE LOS DOS CONJUNTOS DE DATOS PROCESADOS UTILIZANDO LA IMPLEMENTACIÓN DISTRIBUIDA Y DIFERENTES NÚMEROS DE NODOS DE COMPUTACIÓN.

Por razones ilustrativas, la Fig. 6 compara gráficamente el tiempo de ejecución entre la versión iterativa y la versión distribuida considerando la imagen pequeña (145 × 145, entrenando con un total de 1554 patrones de entrenamiento para las 16 clases presentes). De forma similar, la Fig. 7 compara el tiempo de ejecución entre la versión iterativa y la versión distribuida utilizando la imagen grande (2678 × 614, entrenando con un total de 50136 patrones de entrenamiento entre sus 58 clases). En el caso de la imagen pequeña, se observa como el tiempo de ejecución del entrenamiento se estabiliza en torno a los 1500 segundos. La carga de trabajo (patrones de entrenamiento) resulta insuficiente para ocultar los retardos de las comunicaciones en este caso, en el que podemos ver como la versión iterativa tiene mejor rendimiento aunque aumentemos el número de nodos de computación. Parece razonable pensar que debemos aumentar la carga de trabajo para hacer que el rendimiento de una versión distribuida pueda mejorar su rendimiento en relación a la versión iterativa.

En este sentido, si consideramos la imagen grande, podemos observar como la versión distribuida llega a mejorar a la versión iterativa al utilizar un número suficiente de nodos de computación.

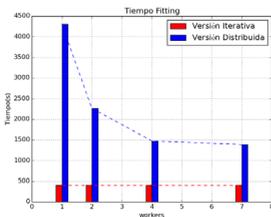


Fig. 6. Comparativa del tiempo de ejecución entre la versión iterativa y la versión distribuida aumentando el número de workers sobre la imagen pequeña

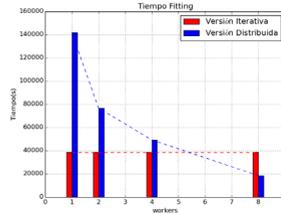


Fig. 7. Comparativa del tiempo de ejecución entre la versión iterativa y la versión distribuida aumentando el número de workers sobre la imagen grande

Finalmente, la Fig. 8 muestra los factores de aceleración (speedup) obtenidos por la versión distribuida frente a la versión iterativa, considerando diferentes números de nodos. Aunque los factores de aceleración obtenidos no son demasiado significativos, conviene destacar que la implementación cloud permite procesar grandes volúmenes de datos de forma distribuida y es escalable, por lo que en el futuro realizaremos experimentos adicionales con un mayor porcentaje de patrones de entrenamiento y un mayor número de nodos, analizando en mayor detalle los patrones de computación y comunicación.

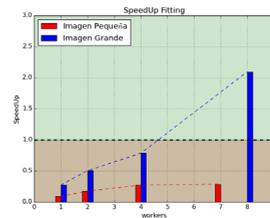


Fig. 8. Aceleración conseguida en la versión distribuida sobre la versión iterativa

VI. CONCLUSIONES Y LÍNEAS FUTURAS

En el presente trabajo se ha analizado la posibilidad de utilizar arquitecturas de computación cloud para el procesamiento de imágenes hiperspectrales. Como caso de estudio, hemos seleccionado una implementación cloud computing de un algoritmo de clasificación neuronal basado en SLFNs implementado en el framework Spark. La validación experimental evalúa la eficiencia de la implementación distribuida propuesta, no solo en términos de precisión, sino también en términos de complejidad computacional, demostrando que es posible trabajar con este tipo de implementaciones si procesamos grandes cantidades de datos. Como trabajo futuro, se planteará la evaluación de implementaciones cloud de otros algoritmos de

procesamiento de imágenes hiperespectrales.

AGRADECIMIENTOS

Este trabajo ha sido subvencionado por la Junta de Extremadura (a través del decreto 297/2014, ayudas para la realización de actividades de investigación y desarrollo tecnológico, de divulgación y de transferencia de conocimiento por los Grupos de Investigación de Extremadura, Ref. GR15005). Además, el presente trabajo ha sido llevado a cabo haciendo uso de la infraestructura de computación facilitada por el Centro Extremeño de Tecnologías Avanzadas (CETA-CIEMAT), financiado por el Fondo Europeo de Desarrollo Regional (FEDER). El CETA-CIEMAT pertenece al CIEMAT y al Gobierno de España.

REFERENCIAS

- [1] A. Plaza, J. Plaza, A. Paz, and S. Sanchez, "Parallel Hyperspectral Image and Signal Processing," *IEEE Signal Processing Magazine*, vol. 28, pp. 196–218, 2011.
- [2] Z. Wu, Y. Li, A. Plaza, J. Li, F. Xiao, and Z. Wei, "Parallel and Distributed Dimensionality Reduction of Hyperspectral Data on Cloud Computing Architectures," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. PP, no. 99, pp. 1–9, 2016.
- [3] J. Plaza, R. Pérez, A. Plaza, P. Martínez, and D. Valencia, "Parallel morphological/neural processing of hyperspectral images using heterogeneous and homogeneous platforms," *Cluster computing*, vol. 11, no. 1, pp. 17–32, 2008.
- [4] J. Plaza and A. Plaza, "Spectral mixture analysis of hyperspectral scenes using intelligently selected training samples," *IEEE Geoscience and Remote Sensing Letters*, vol. 7, no. 2, pp. 371, 2010.
- [5] J. Plaza, R. Pérez, A. Plaza, P. Martínez, and D. Valencia, "Mapping oil spills on sea water using spectral mixture analysis of hyperspectral image data," in *Optics East 2005*. International Society for Optics and Photonics, 2005.
- [6] J. A. Benediktsson, P. H. Swain, and O. K. Ersoy, "Conjugate gradient neural networks in classification of very high dimensional remote sensing data," *Int. Jour. Remote Sens.*, vol. 14, no. 15, pp. 2883–2903, 1993.
- [7] H. Yang, F. V. D. Meer, W. Bakker, and Z. J. Tan, "A backpropagation neural network for mineralogical mapping from AVIRIS data," *Int. Jour. Remote Sens.*, vol. 20, no. 1, pp. 97–110, 1999.
- [8] C. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag, New York, NY, USA, 2006.
- [9] J. A. Benediktsson, *Statistical Methods and Neural Network Approaches for Classification of Data from Multiple Sources*. Ph.D. thesis, PhD thesis, Purdue Univ., School of Elect. Eng. West Lafayette, IN, 1990.
- [10] J. A. Richards, "Analysis of remotely sensed data: The formative decades and the future," *IEEE Trans. Geos. Remote Sens.*, vol. 43, no. 3, pp. 422–432, 2005.
- [11] J. A. Benediktsson, P. H. Swain, and O. K. Ersoy, "Neural network approaches versus statistical methods in classification of multisource remote sensing data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 28, no. 4, pp. 540–552, 1990.
- [12] E. Merényi, W. H. Farrand, J. V. Taranik, and T. B. Minor, "Classification of hyperspectral imagery with neural networks: comparison to conventional tools," *Eurasip Journal on Advances in Signal Processing*, vol. 2014, no. 1, pp. 1–19, 2014.
- [13] F. Del Frate, F. Pacifici, G. Schiavon, and C. Solimini, "Use of neural networks for automatic classification from high-resolution images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 45, no. 4, pp. 800–809, 2007.
- [14] F. Ratle, G. Camps-Valls, and J. Wetson, "Semisupervised neural networks for efficient hyperspectral image classification," *IEEE Transactions on Geosciences and Remote Sens.*, vol. 48, no. 5, pp. 2271–2282, 2010.
- [15] Y. Zhong and L. Zhang, "An adaptive artificial immune network for supervised classification of multi-/hyperspectral remote sensing imagery," *IEEE Transactions on Geosciences and Remote Sens.*, vol. 50, no. 3, pp. 894–909, 2012.
- [16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [17] M. Moller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, no. 4, pp. 525–533, 1993.
- [18] M. T. Hagan and M. Menhaj, "Training feed-forward networks with the marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.
- [19] K. Levenberg, "A method for the solution of certain problems in least squares," *Quart. Appl. Math.*, vol. 54, pp. 164–168, 1944.
- [20] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *SIAM J. Appl. Math.*, vol. 11, pp. 431–441, 1963.
- [21] R. Battiti and F. Masulli, "Fgs optimization for faster automated supervised learning," in *Int. Neural-Network Conf.*, 1990, vol. 2, pp. 757–760.
- [22] J. Nocedal, "Updating quasi-newton matrices with limited storage," *Math. of Computation*, vol. 35, pp. 773–782, 1980.
- [23] A. Ulanov, "A Scalable Implementation of Deep Learning on Spark," Tech. Rep., Spark Summit, San Francisco, 2013.
- [24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [25] M. Baumgardner, L. Biehl, and D. Landgrebe, "220 Band AVIRIS Hyperspectral Image Data Set: June 12, 1992 Indian Pine Test Site 3," 2015.
- [26] R. O. Green, M. L. Eastwood, C. M. Sarture, T. G. Chrien, M. Aronsson, B. J. Chippendale, J. A. Faust, B. E. Pavri, C. J. Bovik, M. Solis, M. R. Olay, and O. Williams, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sensing of Environment*, vol. 65, pp. 227–248, 1998.

Aplicando un procedimiento de optimización paralelo *Teaching-Learning* para el enfoque automático de heliostatos

N.C. Cruz¹, J.L. Redondo², J.D. Álvarez³, M. Berenguel⁴, P.M. Ortigosa⁵

Resumen—La configuración operativa del campo solar de las centrales solares de recepción central es un aspecto fundamental de sus tareas de control. El subconjunto de heliostatos activos debe configurarse con precisión para alcanzar el estado de funcionamiento deseado y evitando distribuciones de flujo y picos de potencia peligrosos sobre la superficie del receptor. En este contexto, los autores de este trabajo están desarrollando una metodología de enfoque automático general. Sin embargo, la formulación matemática de este problema da lugar a un complejo problema de optimización de amplia escala en el que cada heliostato activo requiere un cierto punto de enfoque bidimensional sobre la superficie del receptor. En este trabajo se estudia la posibilidad de aplicar TLBO, un optimizador basado en poblaciones y diseñado para problemas de optimización de amplia escala. Teniendo en cuenta el potencial coste computacional de este proceso, se ha desarrollado una versión paralela preliminar de TLBO. Se describe la aplicación de este método para realizar una exploración amplia del espacio de búsqueda en un entorno de computación de altas prestaciones. La paralelización del optimizador resulta ser muy útil para acelerar el proceso de búsqueda en el presente problema. Por consiguiente, se facilita la posibilidad de incluir etapas adicionales dentro del método que permitan obtener mejores soluciones.

Palabras clave— Computación paralela, Optimización de amplia escala, TLBO, Enfoque de heliostatos.

I. INTRODUCCIÓN

LAS centrales solares de recepción central, CSRC en adelante, son instalaciones de producción de energía eléctrica basadas en el aprovechamiento de la energía solar mediante la concentración de la radiación incidente. En términos generales, y teniendo en cuenta el ámbito de este trabajo, se forman por un amplio conjunto de espejos orientables y altamente reflectantes y un receptor de radiación situado sobre una torre. Los espejos, llamados ‘heliostatos’, siguen el movimiento aparente del Sol a lo largo del día para concentrar la radiación incidente sobre el receptor. Por consiguiente, se genera una gran densidad de radiación sobre la superficie del receptor. Esta energía se transmite entonces a un fluido caloportador que circula por su interior. Después de aumentar considerablemente su temperatura, este fluido puede final-

mente emplearse en un ciclo termodinámico clásico para la generación de energía eléctrica. En la figura 1 se puede ver un esquema ilustrativo de este tipo de centrales. Algunos de sus aspectos clave son la estabilidad de producción y la eficiencia operativa dada la madurez de las tecnologías subyacentes en las que se basa. Se remite al lector interesado a [2], [10] para obtener más información de las centrales solares de recepción central.

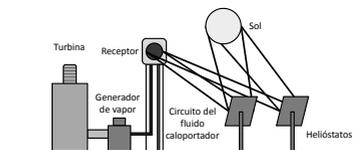


Fig. 1. Esquema de una central solar de recepción central.

El control de la distribución de flujo que forma el campo de heliostatos sobre la superficie del receptor es de vital importancia para evitar gradientes de temperatura peligrosos, estrés térmico y el envejecimiento prematuro de sus componentes [1], [3], [5], [8], [12]. Es un factor clave para aumentar el tiempo de vida útil del receptor, lo que tiene una repercusión directa en los costes de las CSRC como se destaca en [8]. Teniendo en cuenta que el campo de heliostatos está formado normalmente por cientos (incluso miles) de heliostatos, la definición del subconjunto de ellos a activar en un momento dado, así como sus correspondientes puntos de enfoque según la distribución de flujo a lograr, lleva a resolver un complejo problema de varios niveles de decisión. En [3], [8] este problema se aborda fijando un conjunto de heliostatos y otro de puntos de enfoque disponibles, para obtener una distribución homogénea, con buenos resultados. Sin embargo, en el contexto de este trabajo, se está desarrollando una generalización del concepto tratando de configurar automáticamente todo el campo, para un cierto instante de tiempo (una determinada posición solar), y una distribución de flujo cualquiera a obtener. El procedimiento diseñado habría de ser capaz incluso de desactivar aquellos heliostatos innecesarios para replicar una distribución de referencia deseada (dado que los campos solares suelen estar sobredimensionados para afrontar situaciones desfavorables como días nublados). En cualquier caso, esta etapa de selección quedaría fuera del alcance del

¹Dpto. de Informática, Univ. de Almería, CeiA3, e-mail: ncalvocr@ual.es.

²Dpto. de Informática, Univ. de Almería, CeiA3, e-mail: j.lredondo@ual.es.

³Dpto. de Informática, Univ. de Almería, CeiA3, e-mail: jbervas@ual.es.

⁴Dpto. de Informática, Univ. de Almería, CeiA3, e-mail: beren@ual.es.

⁵Dpto. de Informática, Univ. de Almería, CeiA3, e-mail: ortigosa@ual.es.

presente documento y se asume de entrada que el conjunto de helióstatos activos ya se conoce. En este punto, se obtienen en general buenos resultados aplicando optimizadores basados en la dirección del gradiente. Desafortunadamente, estas estrategias tienen un ámbito local y se sabe que la función objetivo tiene múltiples óptimos locales. Se pretende por tanto incluir en el procedimiento un optimizador global eficiente que permita obtener una perspectiva amplia del problema. Se ha decidido considerar un optimizador global basado en poblaciones existente y especialmente orientado a problemas de amplia escala, el algoritmo ‘Teaching-Learning-Based Optimization’ (TLBO) [6]. Por consiguiente, el despliegue y evaluación de este método en un entorno de computación de altas prestaciones es el principal objetivo de este trabajo.

En la sección II se describe formalmente el problema abordado. Seguidamente, en la sección III, se explica el algoritmo TLBO. En la sección IV se comenta la estrategia de paralelización del algoritmo seguida. Finalmente, la experimentación realizada y las conclusiones obtenidas se detallan en las secciones V y VI respectivamente.

II. DEFINICIÓN DEL PROBLEMA

Con objeto de modelar el presente problema es necesario, en primer lugar, definir la distribución de flujo que se pretende conseguir, F . Se trata de una matriz $Y \times X$ donde Y y X son el número de filas y columnas respectivamente. Todos los elementos de F son conocidos, tanto en magnitud (densidad de flujo) como posición, dado que forma parte de la información de entrada del problema. Esta matriz puede verse como una ‘imagen’ de la distribución de flujo a replicar sobre el receptor, que se modela como un rectángulo plano, con los helióstatos activos. Su dimensión Y_P está ligada a la vertical del plano, con dirección hacia el cénit desde la base de la torre en un sistema de coordenadas cartesianas tridimensional. En relación a su dimensión lateral X_P , se extiende en la dirección horizontal del plano desde el Oeste hacia el Este. En este contexto, los ejes de discretización son también conocidos como los vectores $Y' = y_0, \dots, y_Y$ y $X' = x_0, \dots, x_X$ cuya longitud es Y y X respectivamente. Por consiguiente, cada elemento de la matriz hace referencia a una zona particular del receptor, cuya superficie está implícitamente discretizada por el paso usado al definir F .

En relación a los helióstatos activos, se trata de un conjunto ordenado $H = \{h_1, h_s, \dots, h_T\}$ con cardinalidad T . Cada helióstato h_i proyecta una cierta distribución de flujo sobre el receptor cuando está operativo, que es una función bidimensional conocida de la densidad de radiación. Se define como una función de densidad gaussiana bidimensional como se muestra en la ecuación 1. En dicha ecuación, x e y son las coordenadas sobre el plano rectangular del receptor en sus dimensiones X_P e Y_P respectivamente, P es la contribución de potencia del helióstato h_i

sobre el receptor, ρ es la correlación entre x e y , y σ_x y σ_y son la desviación típica a lo largo de x e y respectivamente. En relación a μ_x y μ_y , las medias en la función de probabilidad gaussiana, definen el punto central de la distribución de flujo, es decir, el punto de enfoque del helióstato h_i . Este esquema de modelado es similar al seguido en [3], [8], donde se aplica una función de densidad gaussiana circular siguiendo el modelo de HFLCAL [9]. Todos los parámetros que definen la forma de la distribución de flujo de cada helióstato, es decir, P , ρ , σ_x y σ_y son conocidas (en nuestro caso a partir de simulaciones detalladas y procesos de ajuste de curvas). Sin embargo, su punto central de enfoque, (μ_x, μ_y) , debe determinarse para replicar la distribución de flujo de referencia. Una vez que todos los helióstatos tienen un cierto punto de enfoque asignado, el vector de configuración del campo, c , puede entonces definirse concatenando el par de coordenadas de enfoque de cada uno. Finalmente, la distribución de flujo obtenida según dicha configuración, F^*c , se forma a partir de la convolución de la correspondiente a cada helióstato sobre el receptor, a lo largo de los ejes de discretización y dando lugar a una matriz de las mismas dimensiones que F .

Teniendo en cuenta las definiciones previas, se puede plantear un problema de minimización $2T$ -dimensional sobre la acumulación de la diferencia al cuadrado, en cada punto de discretización, entre las matrices F y F^*c como se muestra en la ecuación 2

$$\min O = \min \sum_{x=x_0}^{x_X} \sum_{y=y_0}^{y_Y} (F(x, y) - F^*c(x, y))^2 \quad (2)$$

III. OPTIMIZACIÓN ‘Teaching-Learning’ (TLBO)

‘Teaching-Learning-Based Optimization’ (TLBO) es un algoritmo de optimización global estocástico, basado en poblaciones y orientado a problemas de amplia escala presentado en [6]. Se basa en modelar el comportamiento de una clase de estudiantes formada por un conjunto de soluciones candidatas conocidas. Estas soluciones se van mejorando progresivamente mediante la simulación tanto del proceso de enseñanza de un profesor como de la interacción entre los alumnos. Este algoritmo se caracteriza fundamentalmente por su rendimiento y por su falta de parámetros específicos de búsqueda, ya que sólo es necesario especificar el tamaño de la población y el número de ciclos según se requiera. A pesar de que en este trabajo se ha seleccionado para una primera exploración la versión básica del algoritmo para problemas de optimización continua sin restricciones, se recomienda al lector interesado la lectura de [7] para mayor información sobre TLBO, sus versiones y aplicaciones. Adicionalmente, la lectura de los trabajos de [4] y [11] es también muy recomendable.

Los ‘estudiantes’, es decir, las soluciones candidatas, se definen como vectores N -dimensionales, donde N es el número de dimensiones del problema

$$f_{h_i}(x, y) = \frac{P}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} e^{\left(-\frac{1}{2(1-\rho^2)}\left(\frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2} - 2\rho\frac{(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y}\right)\right)} \quad (1)$$

abordado. Cada dimensión se considera como una ‘asignatura’ en el contexto de TLBO, y la representación natural de la población o ‘clase’ es una matriz $P \times N$, siendo P el número de individuos disponibles. Se asume que la calidad o rendimiento de los estudiantes sigue una distribución normal cuyo valor medio debe mejorarse mediante la interacción académica. Por consiguiente, el valor de los estudiantes en cada asignatura se va alterando a lo largo de los ciclos con el objetivo de mejorar la media general. Para lograrlo, TLBO se apoya en dos pasos fundamentales por ciclo, las fases del profesor (*Teacher Phase (TS)*) y la de los alumnos o aprendices (*Learners Phase (LS)*). Ambas se resumen a continuación, para un cierto ciclo k , desde una perspectiva de problema minimización.

A. Fase del profesor (TS)

En esta etapa se comienza calculando el valor medio actual por asignatura o dimensión, es decir, por columnas en la matriz de población previamente comentada. De esta forma, se obtiene un vector N -dimensional M . Entonces, el mejor estudiante de la población, cuyo valor de la función objetivo es el menor de los conocidos, se selecciona como profesor del ciclo. Seguidamente, se calcula un entero aleatorio en el rango [1, 2] denominado factor de enseñanza (*Teaching Factor (T_F)*). Este valor supone un factor de ponderación general de las capacidades de enseñanza del profesor. Posteriormente, se genera un vector N -dimensional aleatorio de valores reales, r , en el rango [0, 1], para modelar la capacidad de docencia del profesor, en cada asignatura, en relación con los estudiantes. Con esta información se calcula entonces un vector de desplazamiento general DM , de dimensión N , según la ecuación 3 (donde los operadores tienen una componente dimensional o ‘elemento a elemento’). Finalmente, este vector se suma a cada individuo existente. Sin embargo, sólo aquellos que se ven mejorados por el cambio, tras evaluarlos, se mantienen. En caso contrario este cambio se deshace.

$$DM = r(T - T_F M) \quad (3)$$

B. Fase de los alumnos (LS)

En esta etapa se comienza por generar un vector N -dimensional de valores reales, r , en los mismos términos que el vector homónimo de la etapa anterior. En esta ocasión modela la capacidad puntual de avance de los alumnos, en cada asignatura, a través de su interacción. Seguidamente, cada individuo final i de la etapa anterior se empareja con otro, diferente de sí mismo, j . A continuación, se intenta desplazar al individuo i de su posición actual en una dirección que depende de su valor relativo con respecto a j , tal y como se comenta en la ecuación 4

(manteniendo el carácter ‘elemento a elemento’ de la operación). Finalmente, el individuo i sólo será actualizado si su valor se ve mejorado tras el cambio.

Llegados a este punto, la población del siguiente ciclo quedaría en principio definida. Sin embargo, aunque no se menciona en [6], es posible la inclusión de un paso adicional para la eliminación de soluciones duplicadas. Este detalle se destaca en [4] y se responde en [11]. Este proceso ha de buscar soluciones candidatas iguales y reinicializar aleatoriamente alguna dimensión de una de ellas dando lugar a una nueva solución candidata. Dicha fase también se ha incluido en la implementación realizada para este trabajo.

IV. ESTRATEGIA DE PARALELIZACIÓN

Teniendo en cuenta la descripción realizada de la estructura y funcionamiento de TLBO, y asumiendo además una función objetivo computacionalmente costosa de evaluar, los procesos iterativos subyacentes en las etapas TS y LS (el intento de actuación y evaluación de cada individuo) podrían asignarse a distintas unidades de ejecución. De esta forma, el procedimiento se mantendría perfectamente consistente con el secuencial pero, en cada ciclo, las tareas de gestión de los individuos se repartirían entre las unidades de ejecución disponibles. Por consiguiente, las evaluaciones requeridas de la función objetivo se repartirían entre las mismas. Esta es la estrategia que se ha seguido en el presente trabajo en un entorno de concurrencia real basada en hilos. Su eficiencia depende directamente del tamaño de la población y el coste de su evaluación. Queda, sin embargo, teóricamente desligada del número de ciclos de búsqueda.

Finalmente, es importante mencionar que para poblaciones extremadamente grandes (a fin de ampliar la búsqueda) y/o funciones objetivo muy costosas, la estrategia descrita podría ser fácilmente generalizada. En concreto, la población podría ser dividida en subconjuntos disjuntos y ser internamente gestionada en paralelo con puntos de intercambio y sincronización intermedios. De esta forma sería interesante para un entorno híbrido combinando procesos independientes e hilos cuando el problema abordado permitiera amortizar el coste de las comunicaciones.

V. EXPERIMENTACIÓN Y RESULTADOS

Una instancia de muestra del presente problema pretendería formar una distribución de flujo uniforme y plana sobre el receptor. Por consiguiente, la matriz F estaría formada por un único elemento replicado. En este caso de estudio se está ante una forma plana a $80kW/m^2$ sobre un receptor de 6×6 metros. El subconjunto de helióstatos a activar ya habría sido seleccionado por las capas superiores de

$$Estudiante_i = \begin{cases} Estudiante_i + r(Estudiante_j - Estudiante_i), & \text{Si } j \text{ mejor que } i \\ Estudiante_i + r(Estudiante_i - Estudiante_j), & \text{Si } i \text{ mejor que } j \end{cases} \quad (4)$$

nuestro procedimiento dando lugar a un conjunto de 110 elementos. En este contexto, se va a lanzar la implementación paralela de TLBO para evaluar su rendimiento computacional. Ésta ha sido desarrollada en el lenguaje de programación C con directivas OpenMP para temas de hebrado.

La plataforma de ejecución es un nodo de cluster con procesador Intel Xeon E5 2650v2 con 16 núcleos y 128 GB de memoria RAM. El número de ciclos se ha fijado a 150 tras una etapa de ajuste preliminar. En relación al número de individuos de la población, lo que marca el coste computacional directo de cada ciclo, se ha configurado para ser 50, 100, 200 y 400. Teniendo en cuenta el equipo disponible, el número de hilos desplegado en las pruebas ha sido de 2, 4, 8 y 16 para todos los tamaños de población.

En la figura 2 se muestra la aceleración obtenida con la versión paralela de TLBO para los distintos tamaños de población. Adicionalmente, se incluye como referencia una línea punteada negra que representa la aceleración teórica lineal. Los resultados mostrados se han promediado tras cinco ejecuciones. Como se puede apreciar, la aceleración conseguida es casi lineal en todos los casos. De hecho, con 2 y 4 hilos podría considerarse lineal independientemente del tamaño de la población. El pico de rendimiento se logra con la mayor población evaluada, 400 individuos, y 16 hilos, llegando a una aceleración de 14.10. Sin embargo, como es de esperar, se reduce conforme el tamaño de la población se reduce y el número de hilos es demasiado elevado. Dicho en otras palabras, la aceleración se va alejando progresivamente de la linealidad cuando el ratio entre individuos e hilos activos se reduce. Nótese, no obstante, la escalabilidad del procedimiento dada la tendencia monótonamente creciente de la aceleración en todas las configuraciones probadas.

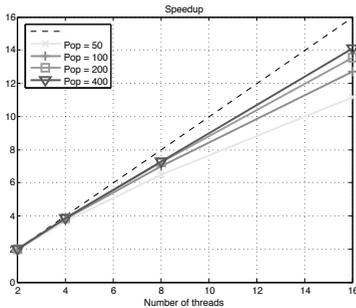


Fig. 2. Aceleración obtenida con la versión hebrada de TLBO.

Finalmente, en la figura 3 se muestra la distribución de flujo correspondiente a la mejor solución encontrada por TLBO. Se ha obtenido con una población de 400 individuos a lo largo de 150 ciclos de búsqueda. Como se puede apreciar, su zona superior es relativamente plana y próxima a $80kW/m^2$ como se pretendía. Por consiguiente, el algoritmo no sólo se muestra apto para ser paralelizado eficientemente con este problema, sino que sus resultados parecen prometedores como puntos de partida de optimizadores locales basados en gradiente.

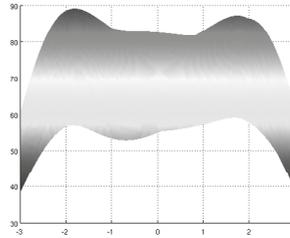


Fig. 3. Plano X-Z de la solución obtenida por TLBO al replicar una forma plana.

VI. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se ha presentado el problema que supone la definición del punto de enfoque de un conjunto de helióstatos en una central solar de recepción central. A continuación, se ha formalizado matemáticamente como un problema de optimización (minimización) de amplia escala sin restricciones. Se sabe de dicho problema que tiene numerosos óptimos locales y una complejidad que crece con el número de helióstatos. En este contexto, se ha decidido estudiar el comportamiento de un algoritmo de optimización global eficiente, basado en poblaciones y especialmente destinado a problemas de amplia escala, TLBO. Se pretende fundamentalmente que permita exploraciones amplias del espacio de búsqueda con grandes poblaciones. Por este motivo, y considerando además el coste computacional de la función objetivo, se ha implementado una versión de TLBO paralela, basada en hilos, bajo el soporte de Open-MP. Ha mostrado una aceleración casi lineal y un comportamiento muy escalable con resultados aceptables en el dominio del problema. Por consiguiente, TLBO parece adecuado para usarse como guía de ámbito global de un proceso con etapas de refinamiento o búsqueda local.

Como trabajo futuro, teniendo en cuenta los resultados positivos obtenidos, se va a considerar en

profundidad la inclusión de TLBO paralelo en el procedimiento de enfoque automático de heliostatos desarrollado por los autores de este trabajo. Adicionalmente, se abordará la posibilidad de desplegar una versión paralela híbrida de TLBO que pudiera ejecutarse en diferentes nodos de un cluster.

AGRADECIMIENTOS

Este trabajo ha sido financiado con fondos del Ministerio Español de Economía y Competitividad (TIN2015-66680-C2-1-R y ENERPRO DPI 2014-56364-C2-1-R), Junta de Andalucía (P11-TIC7176 and P12-TIC301). Nicolás Calvo Cruz es beneficiario de una beca FPU del Ministerio Español de Educación. Juana López Redondo y José Domingo Álvarez Hervás están dentro del programa de contratación 'Ramón y Cajal', co-financiado por el Fondo Social Europeo.

REFERENCIAS

- [1] M. Carasso y M. Becker. *Solar Thermal Central Receiver Systems: Performance evaluation standards for solar central receivers (Vol. 3)*. Springer Verlag, 1991.
- [2] O. Behar, A. Khellaf y K. Mohammedi. A review of studies on central receiver solar thermal power plants. *Renewable and Sustainable Energy Reviews*, 23:12-39, 2013.
- [3] S.M. Besarati, D.Y. Goswami y E.K. Stefanakos. Optimal heliostat aiming strategy for uniform distribution of heat flux on the receiver of a solar power tower plant. *Energy Conversion and Management*, 84:234-243, 2014.
- [4] M. Crepinsek, S.H. Liu y L. Mernik. A note on teaching-learning-based optimization algorithm. *Information Sciences*, 212:79-93, 2012.
- [5] A. Grobler y P. Gauché. A review of aiming strategies for central receivers. En 2nd SASEC Conference, 2014.
- [6] R.V. Rao, V.J. Sivasani y D.P. Vakharia. Teaching-learning-based optimization: an optimization method for continuous non-linear large scale problems. *Information Sciences*, 183(1):1-15, 2012.
- [7] R.V. Rao. *Teaching Learning Based Optimization Algorithm: And Its Engineering Applications*. Springer, 2015.
- [8] A. Salomé, F. Chhel, G. Flamant, A. Ferrère y F. Thierry. Control of the flux distribution on a solar tower receiver using an optimized aiming point strategy: Application to THEMIS. *Solar Energy*, 94:352-366, 2013.
- [9] P. Schwarzbözl, M. Schmitz y R. Pitz-Paal. Visual HFLCAL - A Software Tool for Layout and Optimisation of Heliostat Fields. En SolarPACES'09, 2009.
- [10] W. Stine y M. Geyer. *Power from the Sun*, 2001. Public website, available from <http://powerfromthesun.net/book.html> (Last access: May 10, 2016)
- [11] G. Waghmare. Comments on 'A note on teaching-learning-based optimization algorithm'. *Information Sciences*, 229:159-169, 2013.
- [12] C.J. Winter, R.L. Sizmann y L.L. Vant-Hull. *Solar power plants: fundamentals, technology, systems, economics*. Springer-Verlag New York, 1991.

Estudio del rendimiento de plataformas de computación voluntaria para aplicaciones intensivas en datos

Saúl Alonso Monsalve¹, Félix García Carballeira² Alejandro Calderón Mateos³

Resumen— Las aplicaciones intensivas en datos implican el procesamiento de grandes conjuntos de datos obtenidos a partir de simulaciones o de grandes experimentos que sean capaces de generar terabytes o petabytes de datos. Computacionalmente, existen diferentes soluciones que utilizan sistemas distribuidos para abordar la ejecución de aplicaciones intensivas en datos: basadas en cluster, computación grid, computación en la nube o cloud y computación voluntaria. La computación voluntaria es un paradigma en el que un gran número de ordenadores, pertenecientes a personas del público general, proporcionan recursos de almacenamiento y computación de manera voluntaria. En este artículo, hemos analizado el uso y rendimiento de proyectos de la plataforma BOINC, el principal sistema middleware de computación voluntaria. Con este objetivo, hemos desarrollado un simulador completo de las infraestructuras de BOINC, que tiene en cuenta todos los aspectos presentes en el sistema: servidores, servidores de datos, clientes, planificación, discos, y redes. El artículo describe el simulador y analiza los principales resultados obtenidos cuando proyectos de la infraestructura de BOINC se utilizan para ejecutar aplicaciones intensivas en datos.

Palabras clave— Computación voluntaria, BOINC, hosts, aplicaciones intensivas en datos, simulación, rendimiento.

I. INTRODUCCIÓN

Las aplicaciones intensivas en datos, tales como aplicaciones científicas en dominios tan diferentes como la física de altas energías, la modelación molecular, y las ciencias de la tierra, involucran el procesamiento de grandes conjuntos de datos [1], obtenidos a partir de simulaciones o de grandes experimentos que sean capaces de generar terabytes o petabytes de datos. Este tipo de aplicaciones de gran escala son conocidas como una parte fundamental de e-Science [2], una disciplina que pretende utilizar conjuntamente el uso de informática de alta gama, almacenamiento, redes, y tecnologías web para facilitar la colaboración en la investigación de aplicaciones intensivas en datos.

Computacionalmente, existen diferentes soluciones que utilizan sistemas distribuidos para abordar la ejecución de aplicaciones intensivas en datos: basadas en cluster, computación grid, computación en la nube o cloud y computación voluntaria.

La Computación Voluntaria (CV) [3] es un paradigma en el que un gran número de orde-

nadores, pertenecientes a personas del público general, proporcionan recursos de almacenamiento y computación de manera voluntaria [4]. Desde los comienzos de la CV, entre los primeros proyectos se encontraban GIMPS (Great Internet Mersenne Prime Search) [5], SETI@home [6], distributed.net [7] y Folding@home [8]. En la actualidad, la CV se utiliza en la física de altas energías, en biología molecular, medicina, astrofísica, estudio del clima, y muchas más áreas. Este tipo de plataformas se han utilizado principalmente para la ejecución de Bag-of-Tasks (Bolsa de tareas), ya que no requieren ninguna interacción entre los diferentes participantes [9].

Desde finales de los años 1990 [10], los proyectos de CV, como el proyecto SETI@home [6], se han convertido en los mayores y más potentes sistemas de cómputo distribuidos de de todo el mundo, ofreciendo una enorme capacidad de potencia de cálculo a un coste realmente bajo. El precio medio de un proyecto de CV es equivalente a una pequeña fracción del precio de un supercomputador construido a medida. Muchas aplicaciones de gran escala, pertenecientes a diferentes campos científicos, utilizan la potencia ofrecida por los sistemas de CV. Los sistemas de CV proporcionan a estas aplicaciones recursos informáticos que no habrían sido posibles de otra manera en la mayoría de casos.

Actualmente, BOINC (de sus siglas en inglés Berkley Open Infrastructure for Network Computing) [11] es el principal sistema middleware para CV, el cual facilita a los científicos la creación y manipulación de recursos compartidos y proyectos de cómputo.

En este artículo estamos interesados en analizar el rendimiento de aplicaciones de CV cuando se utilizan para procesar grandes cantidades de datos. Estamos especialmente interesados en analizar los límites y los posibles cuellos de botella que una arquitectura como BOINC presenta. Con este objetivo, hemos desarrollado un simulador completo de la infraestructura de BOINC, que tiene en cuenta todos los aspectos presentes en el sistema: servidores, servidores de datos, clientes, planificación, discos y redes. La idea es poder descubrir cuáles son los límites que presenta una plataforma de este tipo cuando se utiliza para procesar datos de diferentes tamaños por un gran número de nodos voluntarios.

El resto del artículo está organizado de la siguiente forma: la sección II describe las principales características de la plataforma BOINC. La sección III

¹Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: salonso@arcos.inf.uc3m.es.

²Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: felix.garcia@uc3m.es.

³Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: acaaldero@inf.uc3m.es.

presenta ComBoS¹, nuestro simulador, y analiza los diferentes simuladores de BOINC existentes hasta el momento. La sección IV muestra los principales detalles de nuestro simulador, el cual ha sido diseñado considerando todos los aspectos presentes en la infraestructura de BOINC. La sección V contiene un análisis de resultados obtenidos con el simulador. Esta sección trata de identificar los límites de la CV para aplicaciones intensivas en datos. Finalmente, la sección VI resume las conclusiones y describe los trabajos futuros.

II. BOINC

Los recursos informáticos que potencian la Computación Voluntaria (CV) son compartidos por los propietarios de las máquinas. Como estos recursos son donados voluntariamente, se realiza el máximo esfuerzo posible para garantizar que las tareas de CV no entorpecen las actividades del propietario de cada máquina. Una tarea de CV se suspende o se finaliza siempre que una máquina está siendo utilizada por una persona. Por lo tanto, los recursos de CV son volátiles en el sentido de que múltiples factores pueden producir que una tarea de CV no se complete en un tiempo máximo. Estos factores incluyen actividad del ratón o del teclado, la ejecución de otras aplicaciones, reinicios del sistema, o fallos hardware. Además, los recursos de CV son heterogéneos, ya que dependen del sistema operativo de cada máquina, de la velocidad del procesador, del ancho de banda de la red y de los tamaños de disco. Como consecuencia, el diseño de sistemas de CV eficaces se presenta como un gran desafío.

BOINC (Berkeley Open Infrastructure for Network Computing) [11], es un sistema middleware de CV, el cual facilita a los científicos la creación y manipulación de recursos compartidos y proyectos de cómputo. Permite que aplicaciones muy diversas, incluyendo aplicaciones con grandes requisitos de almacenamiento o con altos requisitos de comunicación, puedan especificar cómo asignar sus recursos entre estos proyectos. Actualmente, BOINC es utilizado por un gran número de proyectos, entre los que se incluyen SETI@home, Climaprediction.net, LHC@home, Predictor@home, y Einstein@Home. Los voluntarios participan ejecutando un programa cliente de BOINC en sus propios ordenadores. Los voluntarios pueden asignar a cada ordenador el conjunto de proyectos que deseen, y pueden controlar el porcentaje de recursos destinado a cada proyecto.

Algunos proyectos necesitan una replicación de datos eficiente. Por ejemplo, el proyecto: Einstein@home [12], [13] utiliza grandes ficheros de entrada (40 MB), y cada fichero de entrada debe ser enviado a un gran número de hosts (al contrario que otros proyectos como el SETI@home [14], [15], [6], donde cada fichero de entrada es diferente).

La arquitectura de BOINC [4] permite que los servidores de datos estén situados en cualquier lugar. Son servidores web simples que no tienen ac-

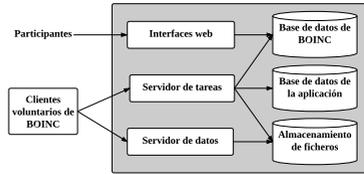


Fig. 1: Un servidor de BOINC está formado por múltiples componentes, compartiendo diferentes formas de almacenamiento.

ceso a la base de datos de BOINC. Los proyectos de BOINC que utilizan ficheros de gran tamaño (Einstein@Home [16] y Climateprediction.net) utilizan servidores de datos distribuidos y replicación, situados en instituciones asociadas. El tráfico de subida y bajada de ficheros se propaga a través de las conexiones a Internet de dichas instituciones.

Los proyectos de BOINC son autónomos. Cada proyecto utiliza al menos un servidor que consta de varios componentes:

- Interfaces web para gestión de cuentas y equipo, mensajes, y otras características.
- Un servidor de tareas que crea las tareas, las envía a los clientes, y procesa los resultados devueltos.
- Un servidor de datos desde el cual los clientes de BOINC descargan los ficheros de entrada y suben los ficheros de salida.
- Clientes voluntarios que piden trabajo, descargan ficheros de entrada, ejecutan, y suben ficheros de salida.

Estos componentes comparten datos almacenados en discos, incluyendo bases de datos relacionales y ficheros de entrada/salida (ver Figura 1).

Los servidores de datos manejan la subida de ficheros utilizando un mecanismo basado en certificados para garantizar que solo ficheros legítimos, con límites de tamaño establecidos, pueden ser cargados en el sistema. Las descargas de ficheros se realizan mediante el protocolo HTTP.

Los ficheros (asociados con versiones de aplicaciones, unidades de trabajo, o resultados) tienen nombres exclusivos para todo el proyecto y además son inmutables. Los ficheros pueden estar replicados: la descripción de un fichero incluye la lista de URLs desde las cuales puede ser cargado o descargado. Los ficheros pueden tener atributos asociados que indiquen, por ejemplo, que deben permanecer residentes en un host después de su uso inicial, que deben ser validados con una firma digital, o que deben comprimirse antes de ser transmitidos por la red.

El cliente descarga y sube ficheros y ejecuta aplicaciones maximizando la concurrencia, utilizando el número máximo de CPUs cuando sea posible y solapando la comunicación y la ejecución. El sistema

¹<http://www.arcos.inf.uc3m.es/~combos/>

TABLA I: Comparación de los principales simuladores de BOINC.

Característica	ComBoS	SimBOINC	SimBA	EmBOINC
Red	entrada	-	trazas	trazas
Número de hosts	500.000	1	40.000	100.000
Disponibilidad de los hosts	entrada (distribución estadística)	simulación	trazas	trazas
Potencia de los hosts	entrada (distribución estadística)	entrada	trazas	trazas
Planificación en los hosts	simulación	simulación	-	-
Fiabilidad de los hosts	entrada + simulación	-	trazas	trazas
Ejecución de los hosts	simulación	simulación	trazas	trazas
Organización de los hosts	hosts individuales + clusters	-	hosts individuales	hosts individuales
Acceso a disco	entrada	-	-	-
Número de tareas	100.000.000	-	200.000	350.000
Validación de tareas	entrada + simulación	-	simulación	emulación
Detalles de las tareas	entrada	trazas	entrada	trazas
Detalles de los proyectos	entrada	trazas	-	-
Número de servidores	N	N	1	1
Número de servidores de datos	N	-	-	-
Planificador de los servidores	simulación	-	simulación	emulación

computacional de BOINC también proporciona un servicio de almacenamiento distribuido (de ficheros de entrada o resultados, o de datos no relacionados con la computación distribuida) como un subproducto. Este servicio de almacenamiento es muy diferente al de sistemas peer-to-peer (P2P) como puede ser el caso de Gnutella, PAST [18] y Oceanstore [19]. En estos sistemas, los ficheros pueden ser creados por cualquier host (peer), y no existe una base de datos central donde se encuentre la localización de los ficheros. Esto produce una serie de problemas técnicos (por ejemplo, nombrado y localización de ficheros) que no están presentes en el servicio proporcionado por BOINC.

La arquitectura de BOINC está basada en un modelo estricto maestro/esclavo, con un servidor central responsable de dividir las aplicaciones en miles de pequeñas tareas independientes y luego distribuirlas a los nodos voluntarios que soliciten trabajo. Para simplificar las comunicaciones por red y evitar cualquier problema NAT (de sus siglas del inglés Network Address Translation) que pueda surgir de la comunicación bidireccional, el servidor centralizado nunca inicia la comunicación con los nodos de trabajo voluntarios. Toda la comunicación es iniciada por los nodos voluntarios cuando se necesita más trabajo o cuando los resultados están listos para ser enviados al servidor.

III. ComBoS

Para poder analizar el rendimiento y los límites de la Computación Voluntaria (CV) para aplicaciones intensivas en datos, hemos desarrollado ComBoS (del inglés Complete BOINC Simulator), un simulador completo de CV basado en la infraestructura de BOINC. ComBoS simula escenarios de CV reales. Estos escenarios están definidos por un gran número de parámetros que se especifican en un fichero XML, incluyendo el número de proyectos, las característi-

cas de cada proyecto y el entorno de red. Los resultados de las simulaciones contienen un gran número de estadísticas, como el número de FLOPS de cada proyecto, el número de tareas ejecutadas por los clientes, el número total de crédito obtenido por los clientes, y la carga media de los servidores. Hemos implementado ComBoS en el lenguaje de programación C, con la ayuda de las herramientas proporcionadas por SimGrid [20]. Por consiguiente, hemos conseguido realizar simulaciones masivas (más de 500.000 clientes) es solo unas horas.

No existen muchos simuladores de CV, aunque la mayoría se centran en BOINC. Como ComBoS, existen otros simuladores basados en las herramientas de SimGrid. Un ejemplo es SimBOINC, que simula el planificador del cliente de BOINC. SimBOINC [21] utiliza prácticamente el mismo código fuente que el planificador del cliente real de BOINC. Esta es la razón por la que las simulaciones de SimBOINC son casi perfectas. El código de SimBOINC es público. Sin embargo, aunque sus resultados son óptimos, el código del simulador es bastante extenso (más de 20.000 líneas de código fuente) y no excesivamente eficiente en cuanto a tiempo de ejecución. Además, al igual que otros simuladores, SimBOINC se centra en el lado del cliente, no simulando el resto de partes de la infraestructura de BOINC. En 2010, se creó un simulador con resultados similares a SimBOINC [22], pero más eficiente (alrededor de tres o cuatro veces más rápido) y con un código fuente de unas 800 líneas.

Al contrario de los simuladores mencionados, SimBA [23] (del inglés Simulator of BOINC Applications) es un simulador que reproduce la creación, descripción, y terminación de clientes voluntarios utilizando ficheros de trazas obtenidos de proyectos de BOINC reales. SimBA simula el planificador del servidor de BOINC y su interacción con un gran número de hosts clientes. Sus puntos débiles son que

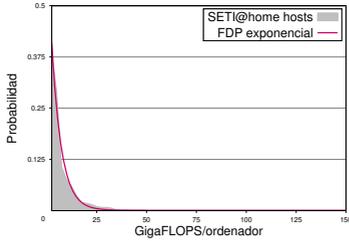


Fig. 2: Función de densidad de probabilidad de los NV de SETI@home.

no es altamente escalable (menos de 50.000 hosts, mientras que BOINC tiene proyectos con más de 100.000 hosts activos), cada proyecto tiene que simularse de manera individual y no cuenta con planificador en el cliente.

Finalmente, aunque es un emulador en vez de un simulador, creemos la necesidad de mencionar EmBOINC. EmBOINC [24] utiliza una población de clientes voluntarios y emula el lado del servidor. EmBOINC tampoco cuenta con planificador en el cliente. La Tabla I compara las principales características de los programas presentes en esta sección (SimBOINC, SimBA y EmBOINC) y ComBoS.

Nuestra intención fue crear un simulador que, al contrario que los existentes, pudiera simular escenarios reales teniendo en cuenta toda la infraestructura de BOINC. El resultado de nuestro trabajo es ComBoS, un simulador que ejecuta simulaciones complejas de entornos basados en BOINC, creando la plataforma de simulación a partir de un fichero XML y generando las salidas de la simulación en forma de resultados estadísticos, tales como los FLOPS ejecutados por proyecto, el número de tareas ejecutadas por los clientes o la ocupación media de los servidores de BOINC. Podemos tener múltiples proyectos y cientos de miles de hosts en la misma simulación. En la siguiente sección describiremos los componentes de simulación en detalles.

ComBoS ha sido desarrollado utilizando SimGrid. SimGrid [20] consiste en un conjunto de herramientas que proporcionan mecanismos para evaluar algoritmos y heurísticas cluster, grid y P2P. El núcleo de simulación de SimGrid [22] implementa y proporciona interfaces para simular un gran número de modelos diferentes, que pueden ser usados para simular diferentes tipos de recursos, tanto computacionales como de redes.

IV. COMPONENTES DE SIMULACIÓN

ComBoS es un simulador completo de las infraestructuras de BOINC, que simula el comportamiento de todos los componentes involucrados: proyectos, servidores, red, computación redundante, y nodos voluntarios. En esta sección se describen

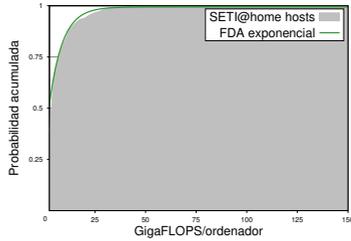


Fig. 3: Función de distribución acumulada de los NV de SETI@home.

las principales características teniendo en cuenta los elementos anteriores.

A. Servidores

Los servidores son responsables de gestionar proyectos. El lado del servidor de un proyecto se divide en dos partes [25]:

- Un *project back-end* (motor) que suministra trabajo a las aplicaciones y maneja los resultados computacionales. También incluye un *validador*, que examina grupos de resultados y selecciona resultados canónicos.
- Un *servidor complejo* que gestiona la distribución de datos y su colección. Incluye: uno o más servidores de tareas, que se comunican con los hosts voluntarios; y servidores de datos, que distribuyen ficheros de entrada y recogen los ficheros de salida.

ComBoS permite la definición de múltiples proyectos. Para cada proyecto, los usuarios deben definir:

- Número de servidores de tareas.
- Número de servidores de datos.
- Ancho de banda de los discos utilizados por los servidores.
- Potencia de CPU de los servidores, en GigaFLOPS.
- Duración de cada tarea: número medio de operaciones en coma flotante necesarias para completar una tarea.
- Fecha límite de cada tarea: tiempo máximo en el cual una tarea debe ser completada.
- Tamaño de ficheros de entrada: cantidad de datos que los voluntarios de datos descargan para procesar.
- Tamaño de ficheros de salida: cantidad de datos que un voluntario envía al servidor cuando ha completado la ejecución de una tarea.
- Quorum: mínimo número de resultados exitosos requeridos por el validador. Si una estricta mayoría coincide, son considerados correctos.
- Resultados objetivo: número de resultados creados inicialmente por unidad de trabajo.
- Máximo número de resultados erróneos: si el

TABLA II: Validación del simulador.

Proyecto	Hosts totales	Hosts activos	BOINCstats		ComBoS	
			GigaFLOPS	Crédito/día	GigaFLOPS	Crédito/día
SETI@home	3.970.427	175.220	864.711	171.785.234	865.001	168.057.478
Einstein@home	1.496.566	68.338	1.044.515	208.902.921	1.028.172	205.634.486
LHC@home	356.942	15.814	7.521	1.504.214	7.392	1.393.931

número de resultados erróneos de una unidad de trabajo supera este número, la unidad de trabajo se declara errónea.

- Máximo número de resultados totales: si el número de resultados totales de una unidad de trabajo supera este número, la unidad de trabajo se declara errónea.
- Máximo número de resultados exitosos: si el número de resultados exitosos de una unidad de trabajo supera este número y no se ha llegado a un consenso, la unidad de trabajo se declara errónea.
- Porcentaje de resultados erróneos.

B. Nodos voluntarios

Los Nodos Voluntarios (NV) son utilizados por los participantes que se unen a proyectos de BOINC. Cada NV en ComBoS se puede asociar a cualquier conjunto de proyectos, y el cliente realiza una planificación entre todos los trabajos en curso. Un NV es responsable de pedir a proyecto más trabajo, y planificar los trabajos de los diferentes proyectos. La planificación en el lado del cliente está basada en una planificación round-robin entre proyectos, con un peso según su compartición de recursos. Esta planificación se describe en detalle en [26].

La descripción de NV en ComBoS está basada en la definición de grupos. Para cada grupo de NV, la siguiente información debe detallarse:

- Número de NV del grupo.
- Número de proyectos asociados al grupo y prioridad de cada proyecto.
- Ancho de banda y latencia de la red que conecta el grupo con cada servidor.
- Modelo de potencia de CPU de los nodos del grupo (puede ser por trazas).
- Modelo de disponibilidad de los nodos del grupo (puede ser por trazas).

C. Validación del simulador

Para validar el simulador, hemos utilizado los datos publicados en el sitio web BOINCstats [27], el cual proporciona resultados oficiales de los proyectos de BOINC. En esta sección, analizamos el comportamiento de ComBoS considerando los resultados de simulaciones de los proyectos SETI@home, Einstein@home y LHC@home.

Para modelar la potencia de los NV del proyecto SETI@home, hemos analizado los 3.900.000 hosts que participan en este proyecto. Hemos descubierto

que la potencia de los hosts puede modelarse con una exponencial, como se puede ver en las Figuras 2 y 3. La distribución exponencial tiene una media de 5,871 GigaFLOPS por host. Para los NV de los proyectos Einstein@home y LHC@home, hemos utilizado las trazas de potencia de los hosts que forman cada uno de estos proyectos. No hemos utilizado ninguna otra traza en nuestras simulaciones.

Para modelar la disponibilidad y no disponibilidad de los hosts, hemos utilizado los resultados obtenidos en [28]. Este estudio analiza las trazas de disponibilidad de 230.000 hosts del proyecto SETI@home. Según este artículo, el 21% de los hosts presentan intervalos de disponibilidad aleatorios reales. También se mide la calidad del ajuste de las distribuciones resultantes. Para la disponibilidad de los hosts, los autores muestran que en la mayoría de los casos una distribución Weibull es un buen ajuste. Para la no disponibilidad, la distribución que ofrece un mejor ajuste es una log-normal. Los parámetros utilizados para la distribución Weibull son $shape = 0.393$ y $scale = 2.964$. Para la log-normal, los parámetros utilizados en ComBoS son una distribución con una media $\mu = -0.586$ y una desviación estándar $\sigma = 2.844$. Todos estos parámetros han sido obtenidos de [28] también.

La Tabla II compara los resultados reales de los proyectos SETI@home, Einstein@home y LHC@home con los obtenidos utilizando ComBoS. En concreto, se comparan los GigaFLOPS y los créditos resultantes. El error obtenido es 2,2% para el crédito/día y 0,03% para los GigaFLOPS del proyecto SETI@home; 1,6% para el crédito/día y para los GigaFLOPS del proyecto Einstein@home; y 7,9% para el crédito/día y 1,7% para los GigaFLOPS del proyecto LHC@home. Consideramos que estos resultados nos permiten validar el simulador.

V. EVALUACIÓN Y ANÁLISIS DE RESULTADOS

En esta sección vamos a presentar diferentes casos de prueba utilizando ComBoS. Nuestro objetivo es mostrar el rendimiento que puede resultar cuando plataformas de computación voluntaria son usadas para procesar grandes cantidades de datos. Estamos especialmente interesados en analizar los cuellos de botella y los límites que presenta una arquitectura como la utilizada por BOINC. Para ello, mostramos unos cuantos ejemplos prácticos del uso del simulador, con el respectivo análisis de los resultados de ejecución.

El escenario utilizado en la evaluación consiste en

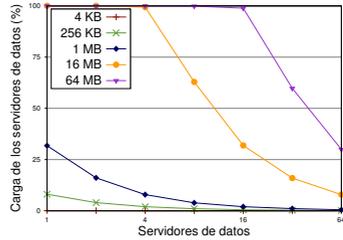
un único proyecto (un servidor de tareas y un número variable de servidores de datos) y tres grupos de Nodos Voluntarios (NV). Las redes que conectan cada grupo de NV con los servidores son fijas en todas las simulaciones. En cada caso de prueba, especificamos el número de NV por grupo, el número de servidores de datos, el tamaño de los ficheros de entrada, y la duración de las tareas. El resto de parámetros son fijos en todas las simulaciones, y el tamaño de los ficheros de salida no se ha tenido en cuenta. Cada ejecución en esta sección ha simulado 100 horas.

A. Carga de los servidores de datos

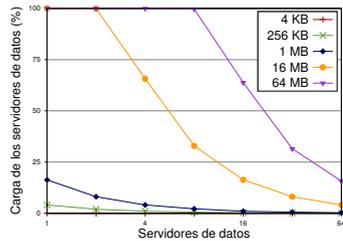
La mayoría de los participantes voluntarios utilizan ordenadores localizados en sus propios hogares, por lo que el tráfico de red se fundamenta en la conexión de internet que tiene contratada cada participante. Los mensajes de petición y respuesta que intercambian los voluntarios con el servidor de tareas tienen un tamaño de solamente 10 KB, por lo que el tráfico de datos por la red se basa exclusivamente en la subida y descarga de ficheros con los servidores de datos (ficheros de entrada y de salida), lo que puede ser una cuestión a optimizar [4]. Los NV descargan los ficheros de entrada de los servidores de datos. En este caso de prueba analizamos la carga de los servidores de datos de un único proyecto debido a la descarga de ficheros de entrada.

En este caso de estudio (Figura 4), utilizamos 30.000 NV (10.000 NV por grupo). En este experimento, nuestro objetivo era poder analizar la carga media de los servidores de datos variando el número de servidores de datos en cada simulación (de 1 a 64). Hemos recogido resultados de cinco tamaños de ficheros de entrada diferentes: 4 KB, 256 KB, 1 MB, 16 MB and 64 MB. El número medio de operaciones en coma flotante necesarias para completar cada tarea (flops, de sus siglas en inglés) en la Figura 4a es $3,6 \cdot 10^{12}$ (1 hora para una máquina de 1 GigaFLOP), $7,2 \cdot 10^{12}$ en la Figura 4b (2 horas para una máquina de 1 GigaFLOP), $1,4 \cdot 10^{13}$ en la Figura 4c (4 horas para una máquina de 1 GigaFLOP), y $2,9 \cdot 10^{13}$ en la Figura 4d (8 horas para una máquina de 1 GigaFLOP). El trabajo tiende a distribuirse uniformemente entre todos los servidores de datos, y la carga de los servidores de tareas no ha sido un factor a destacar en ninguna simulación.

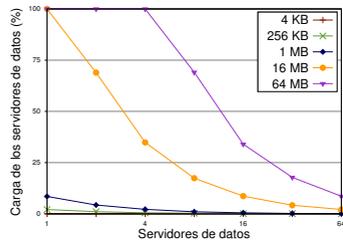
Utilizando estos resultados, podemos estimar la carga media de los servidores de datos en escenarios reales. Cuanto más grande es el tamaño de los ficheros de entrada y menor es la duración media de cada tarea, mayor es la carga en los servidores de datos. Por lo tanto, en algunas simulaciones, los servidores de datos pasan a ser un cuello de botella del sistema. Este tipo de simulaciones puede ayudar a diseñadores a comprobar la factibilidad de proyectos basados en BOINC. De esta manera, ComBoS puede servir como guía para diseñar proyectos de BOINC.



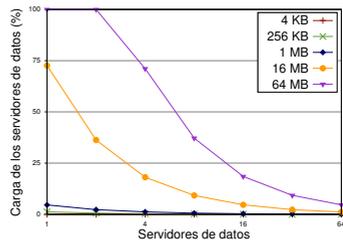
(a) Duración de tarea: $3,6 \cdot 10^{12}$ flops.



(b) Duración de tarea: $7,2 \cdot 10^{12}$ flops.



(c) Duración de tarea: $1,4 \cdot 10^{13}$ flops.



(d) Duración de tarea: $2,9 \cdot 10^{13}$ flops.

Fig. 4: Carga media de los servidores de datos con 30.000 NV (10.000 NV por grupo).

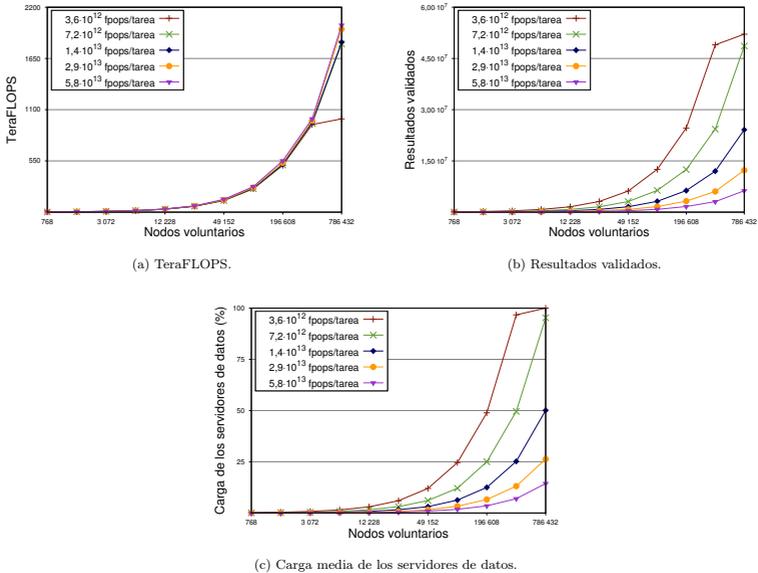


Fig. 5: Resultados combinados.

B. Resultados combinados

En el experimento anterior hemos estudiado la escalabilidad del entorno simulado, variando la duración de las tareas, el tamaño de los ficheros de entrada y el número de servidores de datos. Ahora queremos analizar el rendimiento del mismo entorno incrementando el número de NV y fijando el número de servidores de datos en 4 y el tamaño medio de los ficheros de entrada en 1 MB. Los resultados estas simulaciones se muestran en la Figura 5.

Los resultados obtenidos muestran que podemos estimar los TeraFLOPS del sistema (Figura 5a, constante para el mismo número de operaciones en coma flotante ejecutadas) y el número de resultados validados (Figura 5b, inversamente proporcional a la duración de las tareas). Además, como en el caso de prueba anterior, hemos incluido la carga media de los servidores de datos (Figura 5c). Como se muestra en la Figura 5c, para $3,6 \cdot 10^{12}$ operaciones en coma flotante por tarea, los servidores de datos se saturan cuando hay alrededor de 200.000 NV en el sistema, causando una severa deceleración en TeraFLOPS (Figura 5a) y resultados validados (Figura 5b).

VI. CONCLUSIONES Y TRABAJOS FUTUROS

Este artículo ha presentado un análisis del uso de computación voluntaria para aplicaciones intensivas

en datos. Para ello, hemos desarrollado, utilizando SimGrid, un simulador completo de las infraestructuras de BOINC que tiene en cuenta todos los aspectos fundamentales del sistema: servidores, servidores de datos, clientes, planificación, discos y redes. Hemos validado nuestro simulador con los resultados obtenidos para los famosos proyectos SETI@home, Einstein@home y LHC@home, en términos de créditos y GigaFLOPS. Por otro lado, hemos utilizado el simulador con diferentes cargas de trabajo y plataformas. Los resultados obtenidos muestran la factibilidad de este tipo de plataformas y el rendimiento de los proyectos simulados. Nuestro trabajo futuro se centrará en diseñar nuevas arquitecturas apropiadas para computación voluntaria y el uso de nodos voluntarios como servidores de datos.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por la subvención TIN2013-41350-P del Ministerio de Economía y Competitividad español.

REFERENCIAS

- [1] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A taxonomy of data grids for distributed data sharing, management and processing," Tech. Rep. GRIDS-TR-2005-3, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, Apr. 2005.
- [2] T. Hey and A. E. Trefethen, "The UK e-Science Core Programme and the Grid," *Future Gener. Comput. Syst.*, vol. 18, pp. 1017–1031, Oct. 2002.

- [3] S. Choi, R. Buyya, H. Kim, E. Byun, M. Baik, J. Gil, and C. Park, "A taxonomy of desktop grids and its mapping to state-of-the-art systems," Tech. Rep. GRIDS-TR-2008-3, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, Feb. 2008.
- [4] D. Anderson, E. Korpela, and R. Walton, "High-performance task distribution for volunteer computing," in *e-Science and Grid Computing, 2005. First International Conference on*, pp. 8 pp.–203, July 2005.
- [5] Mersenne Research, Inc., "Great Internet Mersenne Prime Search." Last visited, Apr. 2013.
- [6] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, 2002.
- [7] distributed.net, "distributed.net." Last visited, Apr. 2015.
- [8] S. M. Larson, C. D. Snow, M. R. Shirts, and V. S. Pande, "Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology," in *Computational Genomics: Theory and Application* (R. P. Grant, ed.), Horizon Bioscience, 2004.
- [9] F. Costa, J. Silva, L. Veiga, and P. Ferreria, "Large-scale volunteer computing over the internet," *Journal of Internet Services and Applications*, vol. 3, pp. 329–346, 2012.
- [10] INRIA/IN2P3, "XtremWeb: the open source platform for desktop grids," 2008.
- [11] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, (Washington, DC, USA), pp. 4–10, IEEE Computer Society, 2004.
- [12] LIGO Scientific Collaboration, "Einstein@Home search for periodic gravitational waves in LIGO S4 data," *Physical Review D*, vol. 79, p. 022001, 2009.
- [13] LIGO Scientific Collaboration and D. P. Anderson, "Einstein@Home search for periodic gravitational waves in early S5 LIGO data," *Physical Review D*, vol. 80, p. 042003, 2009.
- [14] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela, "SETI@HOME—massively distributed computing for SETI," *Computing in Science and Engineering*, vol. 3, no. 1, pp. 78–83, 2001.
- [15] P. Paul, "SETI@home project and its website," *Crossroads*, vol. 8, no. 3, pp. 3–5, 2002.
- [16] American Physical Society, "Einstein@home." Last visited, Apr. 2013.
- [17] Oxford University, "Climateprediction.net." Last visited, Apr. 2013.
- [18] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 188–201, 2001.
- [19] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weather- spoon, C. Wells, and B. Zhao, "Oceanstore: an architecture for global-scale persistent storage," *SIGARCH Comput. Archit. News*, vol. 28, no. 5, pp. 190–201, 2000.
- [20] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, pp. 2899–2917, June 2014.
- [21] D. Kondo, *Scheduling Task Parallel Applications for Rapid Turnaround on Enterprise Desktop Grids*. PhD thesis, University of California at San Diego La Jolla, 2005.
- [22] B. Donassolo, H. Casanova, A. Legrand, and P. Velho, "Fast and scalable simulation of volunteer computing systems using SimGrid," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10*, (New York, NY, USA), pp. 605–612, ACM, 2010.
- [23] "The Effectiveness of Threshold-Based Scheduling Policies in BOINC Projects," in *e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference*, p. 88.
- [24] K. R. D. P. A. Trilce Estrada, Michela Tauffer, "EmBOINC: An emulator for performance analysis of BOINC projects," in *Parallel and Distributed Processing Symposium, International*, pp. 1–8, 2009.
- [25] "Creating BOINC Projects." <https://boinc.berkeley.edu/boinc.pdf>, 2007.
- [26] D. P. Anderson, "Local scheduling for volunteer computing," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–8, IEEE, 2007.
- [27] "BOINCstats." <http://boincstats.com/en/stats>, 2016.
- [28] J.-M. V. D. P. A. Bahman Javadi, Derrick Kondo, "Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home," *Parallel and Distributed Systems, IEEE Transactions*, vol. 22, pp. 1896–1903, 2011.

Leveraging the Power of Big Data Tools for Large Scale Molecular Dynamics Analysis

Omar A. Mures¹, Emilio J. Padrón² and Bruno Raffin³

Abstract—Parallel Molecular Dynamics simulations are generating atom trajectories of growing sizes and complexity. Analyzing these trajectories is computationally expensive and time consuming. One of the main reasons is the lack of tools that enable the computational biologist to easily implement the analysis while ensuring reduced processing times by exploiting the benefits of parallel computer architectures. In this paper, we present a comparison between two parallel analytics frameworks based on the Map/Reduce paradigm: HiMach, a dedicated framework for trajectory analysis based on MPI, and Flink, a Big Data analytics framework. Both frameworks enable to hide the complexity of parallel code creation to the programmer, providing significant performance gains compared to a sequential execution. These two frameworks are the core components of the MDReduce system, which tries to simplify the creation of parallel Molecular Dynamics analysis code.

Keywords—Molecular Dynamics, Big Data, MapReduce, Flink, High Performance Data Analysis, Parallelization.

I. INTRODUCTION

RECENTLY in the HPC vendor and science community a new tendency is on the rise, since both fields face similar issues, these two worlds need to come together in order to solve bigger and more complex science problems. As a consequence, base technologies themselves are becoming more closely related. Nowadays scientists are taking advantage more than ever of the evolving computing resources available to them. From their local workstations to high performance clusters in large data centers, they rely on these tools to discover new phenomena in their respective fields.

To accomplish the aforementioned tasks, there is an inherent need to simplify the creation of analysis code that can exploit these heterogeneous resources. Big Data analytics tools based on the MapReduce paradigm [1], such as Hadoop [2], enable to take advantage of commodity compute clusters for analysing very large data sets generated from the web or business applications. But when trying to analyse complex scientific datasets they can fall short from a programmers perspective and performance wise.

Molecular Dynamics (MD) is a computer simulation technique for studying physical interactions of atoms and molecules [3]. It is a type of N-body simulation [4], which simulates a dynamic system of particles, typically under the influence of physical forces,

such as gravity. Atoms and molecules interact during a preset period of time, allowing the scientist to observe the dynamical evolution of the system. The most common approach determines the trajectories of atoms and molecules by solving Newton's equations of motion for a system of interacting particles. The forces and energies of said particles are computed using inter-atomic potentials or molecular mechanics force fields. This method is applied today in chemical physics, material science and the modeling of biomolecules.

Although plenty of progress has been made in the creation of efficient parallel code for Molecular Dynamics simulations, code for analysis of the results that they produce in parallel is scarce. As a consequence, scientist have to rely on serial tools to analyze the aforementioned data. In this paper we focus on the analysis of atom trajectories generated from Molecular Dynamics simulations. These datasets can be very large and time consuming to analyze, sometimes reaching thousands of gigabytes in size. The size of trajectories is expected to keep on growing further as exascale computers [5] become available, aggravating this need for parallel analysis code. Current serial approaches force the scientist to filter trajectories sometimes randomly in order to find the phenomena of interest, this is far from a perfect approach, and if not resolved, could hurt the scientists' opportunities to perform new discoveries and rendering the purpose of creating more precise simulations useless.

Existing analysis libraries such as MDAnalysis [6] or VMD [7] rely on traditional parallelization approaches based on MPI and/or OpenMP that are often complex to deploy at large scale and use efficiently for computational biologists. In this paper, we investigate how the Map/Reduce paradigm could be leveraged to enable the analysis of large trajectories. We have performed a comparison between two parallel analytics frameworks based on the Map/Reduce paradigm: HiMach, a framework for trajectory analysis based on MPI and VMD, and Flink, a new generation Big Data analytics engine. Both frameworks hide the complexity of parallel code creation to the user, providing significant performance gains compared to sequential executions. In addition, these two frameworks are the main components of the proposed solution, the MDReduce library, which aims to simplify the creation of parallel analysis code in Molecular Dynamics workflows.

The efficiency of our framework has been tested on a commodity Linux cluster using a trajectory analysis program. This scenario has achieved and speedup

¹Computer Architecture Group, Universidade da Coruña, e-mail: omar.alvarez@udc.es

²Computer Architecture Group, Universidade da Coruña, e-mail: emilioj@udc.es

³University Grenoble Alpes, INRIA, e-mail: bruno.raffin@inria.fr

of almost two orders of magnitude using MDReduce with minimal changes to the serial analysis code. Furthermore, we were able to perform said analysis on a 1 GB trajectory in less than 19 seconds with 208 cores.

A brief background overview of the subject is given in section II, the proposed solution is explained in section III and the results obtained are presented in section IV.

II. BACKGROUND

MapReduce is a programming model for processing and generating large data sets in parallel, on commodity clusters. It relies on a simple programming model that uses two main functions, Map and Reduce. They are inspired by the map and reduce functions that were commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms.

These two functions (see Figure 1) have to be defined by the programmer in order to obtain parallelism for data-intensive tasks. The Map function consumes key-value pairs and produces one or more intermediate pairs:

$$\text{Map}(k1, v1) \rightarrow \text{list}(k2, v2)$$

The Reduce function, processes a certain key and a list of values associated to it, usually outputting a list with less values than the input list:

$$\text{Reduce}(k2, \text{list}(v2)) \rightarrow \text{list}(v3)$$

For example, a typical MapReduce workload that counts words in a text, will Map (extract) the words in all lines of the text emitting $(k2, v2)$, where $k2$ is the word and $v2$ is 1. This will allow to Reduce (summarize) the results, counting the number of occurrences of each word. In order to do this, each reducer uses the input $(k2, \text{list}(v2))$; where $k2$ is the word and $\text{list}(v2)$ is the list of values associated to $k2$ (all ones), to sum up the values, obtaining the aforementioned count.

The MapReduce system performs the processing by marshaling the distributed servers, running the required tasks in parallel, managing all communications and data movement between the various parts of the system, while providing redundancy and fault tolerance. As such, a in a single-threaded environment, MapReduce will usually not be faster than a traditional implementation; any performance gains are usually only seen in multi-threaded scenarios.

From a performance standpoint, MapReduce has several limitations. First, as we have seen in Figure 1, after each Map phase, results are written to disk as Map Output Files (MOFs) even if they can fit in main memory, and will be later read by the Reduce phase in order to compute the final results. This is good from a fault tolerance point of view, but when trying to achieve high performance it presents a clear problem, as avoiding disk writes and reads could increase performance substantially. Furthermore, until every map task has finished, the reduce

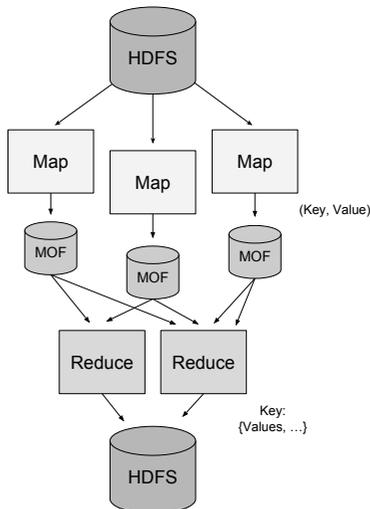


Fig. 1
 TYPICAL MAPREDUCE JOB WORKFLOW.

phase cannot start; even if the required data has already been obtained. Second, since the paradigm is quite simple, it presents almost no opportunities for the framework to perform automatic optimizations. While a experimented MapReduce programmer can use these two abstractions to model complicated algorithms, it is clear that this paradigm can fall short in some complex scenarios where more flexibility is required, apart from the fact that there is not enough semantic information in this two constructs in order to infer properties about the user defined functions.

Several efforts have been made to mitigate these problems, for instance the parallel trajectory analysis framework HiMach [8], is a MapReduce like system where the user writes sequential trajectory analysis code and the system carries out the parallel analysis of the trajectory automatically. This solution avoids the use of MOFs, as well as, using high performance MPI code that can take advantage of InfiniBand interconnects, improving performance substantially. Although this framework solves some of the issues that have been presented earlier, the parallelization paradigm offered still suffers from the same issues as MapReduce.

Since for complex analysis the MapReduce method is not optimal, a more advanced paradigm that relies on Directed Acyclic Graphs (DAGs) [9] has been chosen in order to allow the user to express more complex analytics tasks. This new approach uses Parallelization Contracts (PACTs) [10], which can be

thought of as a generalization of MapReduce in order to ease the expression of many operations, which in turn leads to better expressiveness and enables optimizations to be inferred automatically. The chosen framework is the result of the Stratosphere project [11], an open source project from the Apache Foundation called Flink [12]. To our Knowledge Stratosphere/Flink has never been used for analyzing trajectories.

The usage of such framework will allow computational biologists to more efficiently analyze massive trajectories, that until now had to be simplified, which is not ideal. With the presented framework, a scientist will be able to leverage in a simple way the power of new generation MapReduce tools, letting them worry about what analysis they want to perform instead of how to parallelize or deal with simplifying the data so it is usable.

III. MDREDUCE FRAMEWORK

Since our main objective is easing the creation of parallel code by the scientist, initially we have chosen the Python [13] language for its ease of use and plethora of external libraries. Python has been amply utilized by the scientific community, with libraries for efficient numerical analysis like Numpy [14]. This library is a key component of MDAnalysis, one of the tools that allows scientists to efficiently create parallel Molecular Dynamics analysis code using our system. This are the key technologies that were used in order to implement the MDReduce framework in conjunction with Flink and HiMach.

A. Architecture

Once the reader has been acquainted with the general concepts that have been used in the implementation of MDReduce, it is interesting to take a look at Figure 2, that presents a more accurate picture of the system. In the current MDReduce implementation, the **Core** module contains all the classes related to the system internals. First, the **Configuration** class takes care of parsing either a configuration file (it uses the YAML syntax to store several settings) or the arguments that the user passes to the class constructor. It will take care of initializing the requested back-end (Flink or HiMach), and other trivial setup tasks. Directly related to this class are the available **Backends**. These classes are in charge of dealing with Flink and HiMach internals, these are helper classes for initialing these frameworks and running the requested jobs on them.

The main supported operations that the user can employ to analyze trajectories in parallel are the following:

In addition, the **Analysis** module contains all the classes related to the Molecular Dynamics analysis methods included in the MDReduce framework. First, an **Algorithm** base class has been created, so that developing new analysis scripts is more intuitive for programmers. Second, several analysis classes have been included to perform the analyses that will

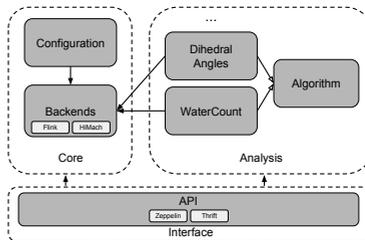


Fig. 2
 MDREDUCE SYSTEM ARCHITECTURE.

TABLE I
 SUPPORTED OPERATIONS. Note: The HiMach back-end only supports the Map and Reduce operations.

Operation	Description
Map	Takes one element and produces zero, one, or more elements.
Reduce	Combines a group of elements into a single element.
Filter	Evaluates a function for each element and retains those that fulfill it.
Aggregate	Aggregates a group of values into a single value. Minimum, maximum, and sums.
Join	Joins two data sets by creating all pairs of elements that are equal on their keys.
Union	Produces the union of two data sets.
Sort	Sorts dataset depending on a certain field.

be detailed in subsection III-B. These two classes, **WaterCount** and **DihedralAngles** offer an insight for programming analysis codes using the MDReduce platform.

Finally, through the MDReduce API, external programs that use the existing algorithms or perform new types of analyses that can be created by scientists and run on the platform. In order to ease the integration of the framework with external programs, output can be consumed by and external client, or stored in data files. These files can then be analyzed in local workstations, since the amount data should have been substantially reduced, and should now be usable on lesser machines. As can be seen in Figure 2, Zeppelin [15] a new framework that enables interactive data analytics has also been integrated with the system. This allows scientists to submit jobs interactively, and dynamically explore the resulting data.

This is a convenient tool to ease the creation and representation of different analyses, since it allows the user to interactively modify code and visualize analysis results.

B. Case Study

To test the feasibility of the proposed system for Molecular Dynamics analyses, we choose to implement two typical tasks for a computational biologist using MDReduce. First, obtaining a water count in a GLIC channel [16], [17]. Second, another type of analysis that is applied commonly to trajectories, which computes the dihedral angles [18] of the protein backbone.

B.1 Water Count in a GLIC Channel

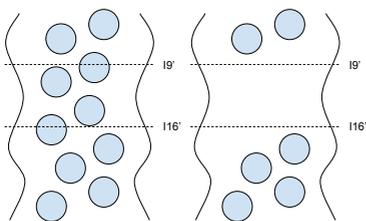


Fig. 3
 GRAPHICAL REPRESENTATION OF THE WATER COUNTING OPERATION IN AN ION CHANNEL.

As seen in Figure 3, the objective of this analysis is obtaining the amount of water ions that are comprised between certain residues in the ion channel. The final objective is obtaining a water ion count for every frame in the trajectory, since the user will vary the simulation parameters in order to isolate the causes of the closing and opening of the GLIC ion channel. This structure has both open and closed states, and the transition mechanism between these different states is unknown for now. Understanding this process is crucial, as empowering molecules such as anesthetics and alcohols is believed that depends on affecting transition barriers or the relative free energy of states.

To investigate these phenomena, extensive simulations of the GLIC channel with different starting conditions and parameters have been performed. This allows computational biologists to draw conclusions about the pore hydration, organization and the pore lining helix. To characterize the structure of this helix, the first part is performing this analysis. Counting the number of water molecules in the pore zone between the two hydrophobic residues I9 and I16. After this, the scientist can infer which region has been more affected by pore dewetting.

This type of analysis can be easily characterized in MDReduce using either the Flink or HiMach backends. Since this is an operation that needs to be

```

1 class MapCount(MapFunction):
2     def map(self, value):
3         u.trajectory[int(value[0])]
4         alignTo(u, uRef, select='protein and
5             backbone and resid 222:233')
6         selectWater = u.select.atoms(water)
7         if (selectWater.n.atoms > 0):
8             cylindre = selectWater[numpy.where(
9                 selectWater.coordinates()[:,0]**2 +
10                selectWater.coordinates()[:,1]**2 < 100)
11                [0]]
12         nbWater = cylindre.n.atoms
13     else:
14         nbWater = 0
15     return (value[0],nbWater)
    
```

Fig. 4
 WATER COUNT CODE EXAMPLE IN FLINK.

computed for every frame, this translates easily to the MapReduce paradigm. One just needs to write a mapper that will perform the water count operation for every frame. The Figure 4 represents how this analysis can be performed using the Flink backend.

The user has to override Flink's Map function, with the operations that will be applied to the trajectory. First, the frame number corresponding to the instance of the mapper is selected, then it is processed with MDAnalysis in order to align it to a reference frame and obtain the water count in the ion channel using atom selections. As can be seen, this code is completely straightforward requiring slight modifications to be run in parallel.

B.2 Dihedral Angles of the Protein Backbone

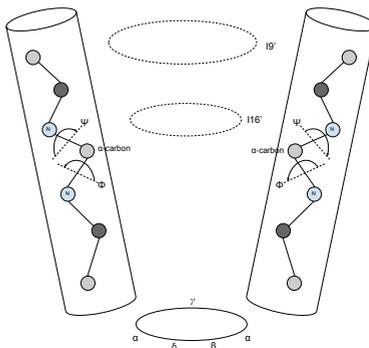


Fig. 5
 GRAPHICAL REPRESENTATION OF THE DIHEDRAL ANGLE CALCULATION OPERATION IN THE PROTEIN.

As seen in Figure 5, the objective of this analysis is obtaining a way to visualize energetically allowed regions for the backbone dihedral angles of amino acid residues in the protein structure. As mentioned earlier, understanding the reasons why the transition

```

1 class MapDihedral(Mapper):
2     def __init__(self, u, uRef):
3         self.u = u
4         self.uRef = uRef
5
6     def map(self, step):
7         cursor = step[0]
8         frameNo, frameBuf = cursor
9         assert frameBuf == None
10
11         self.u.trajectory[frameNo]
12
13         alignTo(self.u, self.uRef, select="
14             protein and backbone and resid 222:233")
15
16         phi_A, psi_A = angle_between_princ_axes(
17             TM2.A, alpha.carbones)
18         phi_B, psi_B = angle_between_princ_axes(
19             TM2.B, alpha.carbones)
20         phi_C, psi_C = angle_between_princ_axes(
21             TM2.C, alpha.carbones)
22         phi_D, psi_D = angle_between_princ_axes(
23             TM2.D, alpha.carbones)
24         phi_E, psi_E = angle_between_princ_axes(
25             TM2.E, alpha.carbones)
26
27         yield(frameNo, [phi_A, psi_A, phi_B,
28             psi_B, phi_C, psi_C, phi_D, psi_D, phi_E,
29             psi_E])
30
31     return

```

Fig. 6

DIHEDRAL ANGLE CODE EXAMPLE IN HiMACH.

between the open and closed states of the ion channel is crucial for several topics. This second analysis code is the missing link to study this phenomena that depends on the pore organization and lining of the M2 helix. To help in the description of the state of the pore, the tilt angles are computed and split in two angles: the radial and lateral tilt angles of the M2 helix around the center of mass in respect to the channel axis.

The second analysis, can be performed in MDReduce using either of the available back-ends. Since this is an operation that again, needs to be computed for every frame, it is an operation that can be easily adapted to the MapReduce paradigm. The user just needs to write a mapper that will perform the required operations for every frame. The Figure 6 represents how this analysis can be performed using the HiMach back-end.

IV. RESULTS

To test the proposed system we have used the Pluton cluster, a supercomputing facility situated in the Universidade da Coruña. In the Table II, the hardware that the cluster possesses can be seen. In our tests, we have used up to 15 nodes with the specifications shown in Table II.

The analysis algorithms explained in subsection III-B, have been used to run the performance tests carried out in this section. These tests have tested several different strategies, that access data with multiple I/O patterns, in order to discern the best one. The tested patterns are:

TABLE II
HARDWARE USED FOR TESTING.

Hardware	compute-0-16
CPU Model	2 x Intel Xeon E5-2660
Cores/Threads	16/32
CPU Speed/Turbo	2.20 GHz/3 GHz
Memory	64 GB DDR3 1600 Mhz
Disk	1 x HDD 1 TB SATA3 7.2K
Networks	InfiniBand FDR

- **Broadcast:** This strategy utilizes a single file that contains the trajectory, that is then distributed to every map task and accessed in parallel.
- **Split:** The trajectory in this strategy is split in one file per frame, which is then accessed independently by every map task in parallel.

All strategies were then executed using both back-ends available in MDReduce to see which of them fares better for each type of analysis.

Figure 7 shows the job execution times of the water count analysis. In it, the fastest serial execution time without using Flink or HiMach in order to avoid framework setup overheads, has been compared against parallel execution times using the parallelization frameworks.

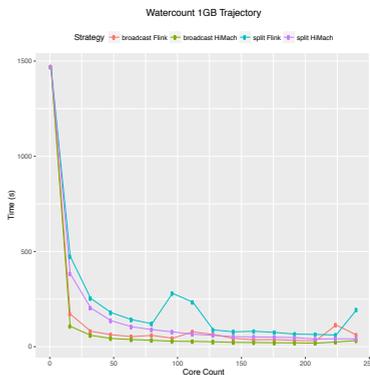


Fig. 7

RUN TIME WITH EACH BACK-END OF THE WATER COUNT JOB.

It is kind of difficult to see which approach fares better, since with every approach the execution time as we use more than one processor descends greatly. The fastest time for this workload, was achieved by the HiMach back-end with the broadcast strategy. This is due to the fact that the HiMach runtime uses MPI and the Infiniband capabilities of our cluster in an optimal way.

To further see what is happening in this analysis

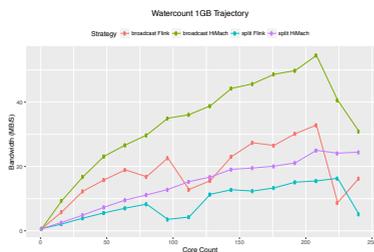


Fig. 8
 BANDWIDTH WITH EACH BACK-END OF THE WATER COUNT JOB.

code in Figure 8, we can see the bandwidth achieved by all the strategies. Here the reader can better discern which back-end is the fastest. This results mimic the ones obtained before, but paint a clearer picture about the performance that each strategy achieves. As was mentioned earlier, the best performance is achieved by HiMach followed closely by Flink. The first thing that one can deduce examining this graph, is that clearly the split access pattern is less efficient than the broadcast pattern. This was expected, since reading through small files generally causes lots of seeks and stresses the underlying file system, which results in an inefficient data access pattern.

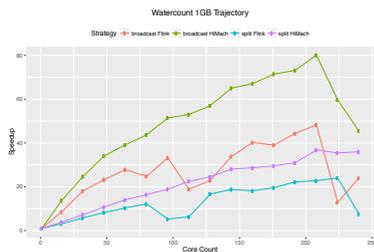


Fig. 9
 SPEEDUP WITH EACH BACK-END OF THE WATER COUNT JOB.

In addition, as to ascertain the efficiency of the tested frameworks, in Figure 9 we can see the speedups of all the different code versions. In this graph other interesting behavior can be observed. Both frameworks scale well but as the number of processors is increased a couple of discrepancies can be seen as the scaling weakens. Still, we have obtained speedups of up to $\sim 80\times$ using HiMach and up to $\sim 50\times$ using Flink, while keeping code changes to a minimum. In addition, we can see that with the

split strategy both back-ends have closer speedups.

The performance difference is more significant for the broadcast access pattern, probably due to the impact of the higher network performance that HiMach achieves, due to a more efficient use of the network layer than Flink. This is possible due to the use of low-level communication patterns available in MPI, in contrast with just using the InfiniBand IP layer in Flink.

In order to improve the performance of the split strategy in both frameworks and probably achieve better scaling for large amounts of nodes, frames should be grouped in batches. As a consequence, each mapper processes more than one frame. This would allow the system to better use the underlying I/O layer, while also keeping task overhead lower in the two back-ends.

V. FUTURE WORK

Since several inefficiencies in the current implementation are believed to be related to the usage of the Python interpreter from the Java Flink processes, it could be interesting to test the ZipPy [19] python implementation in our system. This Python 3 implementation uses Truffle [20] to generate optimized JVM bytecode from Python scripts. This could certainly improve the efficiency of the analysis performed using the MDReduce framework.

In addition, it could also be interesting to extend our study to include a Spark [21] back-end, today probably the most popular Big Dta analytics framework.

Moreover, to solve one of the main issues with the split data access pattern, other storage solutions like HBase [22] could be explored. This column oriented data store has the potential to avoid the bottleneck that the plethora of small files in a long trajectory could present to the MDReduce framework.

VI. CONCLUSIONS

Using MDReduce has proven to improve not only processing times, since it allows users to write almost sequential code that can then be exploited for parallel analysis, but also in terms of programming efficiency. It is clear that using the introduced system, is minimally more complex than writing sequential analysis code for straightforward tasks.

The usage of Flink, can yield performance advantages due to incorporated job optimizer, but also makes programming more complex analysis algorithms possible. It also allow the user to take advantage of the Hadoop ecosystem, for instance allowing the users to visualize data dynamically using Zeppelin.

Our initial performance tests have shown that HiMach, due to its simplicity, is the best option for running straightforward analysis tasks. However the analysis we have tested are simple map/reduce schemes. They did not allow us to probe the capabilities of Flink designed to support more complex parallelization schemes.

ACKNOWLEDGMENTS

This work was partly funded by the European Union's Seventh Framework Programme FP7, project VELA5SCo (N° 619439), Call FP7-ICT-2013-11; and partly by the Galician Government (Xunta de Galicia), under the Consolidation Program of Competitive Reference Groups (Ref. GRC2013/055), and the Ministry of Economy and Competitiveness of Spain and FEDER funds of the EU (Project TIN2013-42148-P).

REFERENCES

- [1] Jeffrey Dean and Sanjay Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [2] "Hadoop," <http://hadoop.apache.org/>, Accessed: 2016-05-01.
- [3] Berni J Alder and TE Wainwright, "Studies in molecular dynamics. i. general method," *The Journal of Chemical Physics*, vol. 31, no. 2, pp. 459–466, 1959.
- [4] John H Reif and Stephen R Tate, "The complexity of n-body simulation," in *Automata, Languages and Programming*, pp. 162–176. Springer, 1993.
- [5] Timothy Germann, "Exascale computing and what it means for shock physics," in *APS Shock Compression of Condensed Matter Meeting Abstracts*, 2015, vol. 1, p. 1002.
- [6] Naveen Michaud-Agrawal, Elizabeth J. Denning, Thomas B. Woolf, and Oliver Beckstein, "Mdanalysis: A toolkit for the analysis of molecular dynamics simulations," *Journal of Computational Chemistry*, vol. 32, no. 10, pp. 2319–2327, 2011.
- [7] William Humphrey, Andrew Dalke, and Klaus Schulten, "Vmd: visual molecular dynamics," *Journal of molecular graphics*, vol. 14, no. 1, pp. 33–38, 1996.
- [8] Tiankai Tu, Charles A Rendleman, David W Borhani, Ron O Dror, Justin Gullingsrud, Morten Ø Jensen, John L Klepeis, Paul Maragakis, Patrick Miller, Kate A Stafford, et al., "A scalable parallel framework for analyzing terascale molecular dynamics simulation trajectories," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for. IEEE*, 2008, pp. 1–12.
- [9] K. Thulasiraman and M.N.S. Swamy, *Graphs: Theory and Algorithms*, Wiley, 2011.
- [10] Alexander Alexandrov, Stephan Ewen, Max Heimel, Fabian Hueske, Odej Kao, Volker Markl, Erik Nijkamp, and Daniel Warneke, "Mapreduce and pact-comparing data parallel programming models," in *BTW*, 2011, pp. 25–44.
- [11] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, et al., "The stratosphere platform for big data analytics," *The VLDB Journal*, vol. 23, no. 6, pp. 939–964, 2014.
- [12] "Flink," <https://flink.apache.org/>, Accessed: 2016-05-01.
- [13] "Python," <https://www.python.org/>, Accessed: 2016-05-01.
- [14] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux, "The numpy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.
- [15] "Zeppelin," <https://zeppelin.incubator.apache.org/>, Accessed: 2016-05-01.
- [16] Prafulla Aryal, Mark SP Sansom, and Stephen J Tucker, "Hydrophobic gating in ion channels," *Journal of molecular biology*, vol. 427, no. 1, pp. 121–130, 2015.
- [17] Pierre-Jean Corringier, Marc Baaden, Nicolas Bocquet, Marc Delarue, Virginie Dufresne, Hugues Nury, Marie Prevost, and Catherine Van Renterghem, "Atomic structure and dynamics of pentameric ligand-gated ion channels: new insight from bacterial homologues," *The Journal of physiology*, vol. 588, no. 4, pp. 565–572, 2010.
- [18] Ricarda JC Hilf and Raimund Dutzler, "A prokaryotic perspective on pentameric ligand-gated ion channel structure," *Current opinion in structural biology*, vol. 19, no. 4, pp. 418–424, 2009.
- [19] Wei Zhang, Per Larsen, Stefan Brunthaler, and Michael Franz, "Accelerating iterators in optimizing ast interpreters," *SIGPLAN Not.*, vol. 49, no. 10, pp. 727–743, Oct. 2014.
- [20] Thomas Würthinger, Christian Wimmer, Andreas Wö8, Lukas Stadler, Gilles Duboscq, Christian Humer, Gregor Richards, Doug Simon, and Mario Wolczko, "One vm to rule them all," in *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, ACM, 2013, pp. 187–204.
- [21] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCanley, M Franklin, Scott Shenker, and Ion Stoica, "Fast and interactive analytics over hadoop data with spark," *USENIX, logn*, vol. 37, no. 4, pp. 45–51, 2012.
- [22] Mehul Nalin Vora, "Hadoop-hbase for large-scale data," in *Computer science and network technology (ICCSNT), 2011 international conference on. IEEE*, 2011, vol. 1, pp. 601–605.

HPSEngine: Motor de alto rendimiento y baja latencia para el procesamiento distribuido en tiempo real

Rafael Leira, Paula Roquero, Carlos Vega, Iván González, Javier Aracil
High Performance Computing and Networking
Escuela Politécnica Superior, Universidad Autónoma de Madrid
{rafael.leira, paula.roquero, carlos.vega, ivan.gonzalez, javier.aracil}@uam.es

Resumen— Entre los distintos sistemas de Big Data, el procesamiento en tiempo real (o streaming) destaca por su importancia y sus requisitos de alto rendimiento y mínima latencia. Sin embargo, disminuir la latencia de un sistema distribuido acostumbra a repercutir directamente en el ancho de banda (throughput) del sistema. El escalado horizontal enfrenta este problema requiriendo mayores costes de hardware. En ciertos casos de uso en los que el caudal de datos es muy elevado (¿20 Gbps) este coste se puede disparar debido al gran número de máquinas necesarias para alcanzar ese rendimiento con las herramientas actuales. Para estos casos el número de máquinas podría reducirse entre 4 y 100 veces si se hiciera un uso óptimo de los recursos. En unas pruebas preliminares, nuestro motor de Streaming es capaz de alcanzar más de 37 Gbps sostenidos entre dos o más nodos usando TCP+Ethernet sin perder por ello la capacidad de escalado horizontal. Este resultado supera con creces los obtenidos mediante las herramientas actuales contra las que lo hemos comparado.

Palabras clave— Big Data, Streaming, Storm, computación distribuida, PCAP

I. INTRODUCCIÓN

LA capacidad de cómputo actual se ha incrementado en los últimos años de forma exponencial, lo que ha permitido la popularización de los diferentes dispositivos “inteligentes”. Independientemente de si hablamos de un teléfono móvil, de una lavadora o de un ordenador convencional, comienza a ser difícil imaginarnos a dicho dispositivo desconectado de la red de redes: Internet. A diario, cada uno de estos dispositivos genera y consume enormes cantidades de datos con distintos tipos de información. Estos datos son muy diversos abarcando desde información estadística del uso y estado del dispositivo hasta datos de localización y de distintos tipos de aplicaciones del usuario, como por ejemplo las redes sociales, cuyos intercambios de datos pueden consistir desde simples mensajes de texto hasta vídeos de gran tamaño.

La cantidad de datos producidos a nivel global aumenta año a año junto al número de dispositivos “inteligentes” y la capacidad de cómputo de estos. En 2013 se llegó a superar los 28 TeraBytes por segundo, mientras que en 2002 apenas se generaba 0,1 [1]. Empresas, como Google, Yahoo o Facebook se lucran con la información que obtienen de sus usuarios, ya sea para optimizar las búsquedas web, mejorar la publicidad o simplemente vender la información a terceros. Ser capaz de procesar y analizar toda la

información que obtienen se ha convertido en algo prioritario y necesario para casi cualquier empresa tecnológica, y no solo las anteriormente mencionadas multinacionales.

El problema de analizar la información no solo radica en el tamaño en sí de los datos a analizar, sino en la complejidad del propio análisis. Un claro ejemplo es el análisis de voz o de imágenes, ya que ambos requieren de complejas y costosas redes neuronales [2]. Es bajo este contexto donde aparece el término Big Data para denominar a aquellos problemas que no pueden ser solventados por herramientas tradicionales en un tiempo razonable [3]. Por este motivo, los problemas de Big Data deben tratar varios TeraBytes de datos y ser resueltos de forma distribuida mediante un conjunto de máquinas.

Al hablar de computación de tipo Big Data, es común referirse en una arquitectura de tipo lambda [4]. Este tipo de arquitecturas se componen, principalmente, de 2 elementos de cómputo bien diferenciados: Un elemento de procesamiento en Flujo o Streaming, orientado al cómputo de tareas en tiempo real y con baja latencia, y un elemento de procesado por Lotes (Batch) offline, orientado a realizar tareas de cómputo con una elevada complejidad y cuyo resultado puede obtenerse varias horas, días, meses o incluso años más tarde de la obtención de los datos. Es posible afirmar que uno de los más importantes avances en el procesamiento de datos masivos ha sido la aparición del paradigma de computación MapReduce, el cual se apoya principalmente en la división del problema en tareas y nodos. Consta de dos tipos de tareas: Map y Reduce. Las tareas Map, realizan un determinado cómputo sobre un subconjunto de los datos en uno de los nodos de cálculo. Posteriormente, los resultados se mezclan y se comparten entre los nodos, de forma que uno o más nodos puedan realizar una tarea de Reducción (Reduce). Esta tarea realiza un cómputo sobre los resultados previos (Por ejemplo, sumar, ordenar, etc), generando un nuevo resultado. El proceso completo de MapReduce podría ejecutarse de forma iterativa si fuera necesario.

El paradigma MapReduce fue inicialmente desarrollado por Google junto con su sistema de almacenamiento distribuido GFS ¹[5]. Por otro lado Ya-

¹Google File System

hoo y la fundación Apache se encargaron de desarrollar de forma paralela el proyecto Apache Hadoop, la versión de código abierto del paradigma de programación MapReduce (Conocido actualmente como Yarn) [6], y el sistema de almacenamiento distribuido HDFS ²[7]. Ambas organizaciones a su vez han desarrollado la otra pieza fundamental que compone cualquier arquitectura Lambda, la pieza de procesamiento en tiempo real (o Streaming): Apache Storm [8].

No obstante, un sistema Big Data no se queda solo en la arquitectura de cómputo Lambda, sino que se extiende a un gran número de aplicaciones que utilizan por debajo estas herramientas básicas [9]. Un sistema Big Data completo está construido por diferentes capas, que abarcan desde el proceso de captura de la información, pasando por la anteriormente mencionada capa de proceso, hasta llegar al almacenamiento y representación de los datos (ver figura 1). A todas y cada una de estas capas que conforman un sistema Big Data se le suele denominar *ecosistema*. (muchas veces conocido como el *ecosistema Hadoop*).

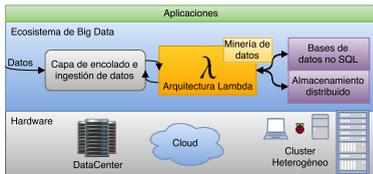


Fig. 1. Sistema típico Big Data

En este trabajo nos hemos centrado en desarrollar un nuevo motor de streaming con el objetivo de solventar la problemática del análisis distribuido de redes en tiempo real. Hemos intentado abordar este problema en trabajos anteriores [10] utilizando Hadoop y MapReduce. No obstante, procesar los datos con Hadoop puede conllevar una latencia de horas, lo que impide su uso en tiempo real. Por otro lado, tal y como se verá a lo largo de este trabajo, Storm no cumple los requerimientos de ancho de banda necesarios para llevar a cabo este tipo de análisis a alta velocidad (≥ 20 Gbps).

En la siguiente sección motivamos la creación de nuestro nuevo motor de streaming, así como un estudio del estado del arte de las diferentes herramientas actuales y su funcionamiento. En la sección III, entramos en el funcionamiento interno de la API, para entender su alcance, potencial y limitaciones. En la sección IV se muestran las diferentes pruebas realizadas y los resultados obtenidos, comparándolos con el estado del arte expuesto en las secciones previas. Finalmente, en la sección V se concluye este artículo y se presentan las diferentes líneas de mejora, desarrollo e investigación futuras.

²Hadoop Distributed File System

³Internet of Things

⁴Java Virtual Machine

II. MOTIVACIÓN

Este trabajo está motivado por la necesidad de procesar en tiempo real flujos de datos que alcanzan de manera sostenida velocidades de hasta 40+ Gbps. Existen diversos problemas que cumplen con estas características, por lo que en este trabajo nos hemos enfocado en plantear un sistema que (al menos) sea capaz de resolver dos de ellos: El análisis de datos de red y el análisis de Logs producidos por decenas de miles de dispositivos del IoT ³. Para afrontar problemas de tal magnitud y complejidad en tiempo real es necesario recurrir a los sistemas de Big Data, o más concretamente a los motores de procesamiento en Streaming. Los más conocidos son Apache Storm [8], Apache Spark Streaming [11], Apache Flink [12] y Apache S4 [13]. Tras evaluar dichos motores, decidimos implementar uno propio, ya que ninguno era capaz de superar los 5 Gbps entre dos nodos conectados con una interfaz de 40 Gbps. Para conseguir que alguno de los motores existentes alcance ese ancho de banda al procesar y transmitir datos, sería necesario utilizar entre **8 y 10 nodos como mínimo**. En los apartados siguientes analizamos el estado del arte de los sistemas Big Data y cómo han solventado los distintos problemas de rendimiento y conectividad. A continuación, analizamos la arquitectura e implementación de los diferentes sistemas de Streaming y enumeramos los motivos que finalmente nos han llevado a realizar nuestra propia implementación.

A. Estado del Arte

Los sistemas de Big Data han estado orientados desde sus inicios a la escalabilidad horizontal. No obstante, las implementaciones actuales descuidan frecuentemente el escalado vertical, o dicho de otra forma, la explotación de todos los recursos disponibles en la máquina anfitriona. Uno de los principales problemas relacionados con la eficiencia del uso de los recursos se encuentra relacionado con el lenguaje de programación utilizado: Java. Este lenguaje aporta uno de los pilares más fuertes de las herramientas de Big Data: una gran portabilidad de la cual no disponen otros lenguajes compilados y que permite ejecutar estos sistemas en distintas plataformas. No obstante, esta última cualidad del lenguaje hace que la aplicación deba ejecutarse en un entorno virtualizado llamado JVM ⁴. Con los parámetros adecuados, la JVM es capaz de optimizar en tiempo de ejecución lo suficiente como para igualarse en rendimiento a un programa compilado y, en raras ocasiones, incluso batirlo. Por otro lado, para que la JVM pueda hacer esto, el programador debe tener una elevada experiencia y tener muy presente como la JVM va a interpretar y ejecutar el código, cosa que rara vez ocurre.

Esta dificultad añadida al lenguaje, junto con otras limitaciones que examinaremos más adelante, causan que hayan surgido multitud de “parches” escritos en diferentes lenguajes de bajo nivel para optimizar los

diferentes componentes de los ecosistemas Big Data, así como aplicaciones completamente nuevas. Este tipo de parches aparecen en todas las capas que conforman un sistema Big Data, desde el almacenamiento hasta las bases de datos distribuidas que hacen uso de las capas subyacentes. Dado que el almacenamiento es una de las partes fundamentales, es la primera funcionalidad que los investigadores han analizado, comparado e intentado mejorar [14]. Los cambios propuestos a las diferentes herramientas abarcan desde implementaciones específicas de las funciones de comunicación [15], pasando por cómo se realiza la distribución de los datos almacenados [16], [17], hasta la realización de nuevas aplicaciones desde cero [18].

La velocidad de la comunicación entre los diferentes nodos es un factor crítico en cualquier aplicación distribuida, siendo de especial importancia en los motores de procesamiento en Streaming y en las bases de datos. HBASE, como ejemplo de base de datos distribuida y bien conocida, hace uso por defecto de la capa TCP/IP de red proporcionada por Java. No obstante, no es capaz de exprimir el ancho de banda y las latencias que ofrece una red infiniband. Por este motivo, se realizó un esfuerzo para hacer HBASE compatible con RDMA [19]. Otra base de datos distribuida muy conocida es Cassandra [20]. A pesar de ser capaz de procesar grandes cantidades de datos, el número de nodos para realizarlo puede resultar elevado dado que Cassandra no es muy eficiente ni en el uso de recursos ni en el tratamiento de las comunicaciones. Debido a ello nació el proyecto Scylla [18]. A pesar de que Scylla es completamente compatible con Cassandra, ha sido desarrollada partiendo desde cero en un lenguaje de bajo nivel (c/c++) y centrándose en obtener un gran ancho de banda y una latencia mínima en las conexiones entre nodos. Para lograrlo, los desarrolladores implementaron su propia pila TCP/IP utilizando Intel DPDK. Los resultados muestran que Scylla escribe y lee hasta 22 veces más entradas en la base de datos que la implementación original en Java [21].

B. Topologías y procesamiento de flujos

El problema del lenguaje lo podemos observar en las diferentes herramientas o aplicaciones de los ecosistemas de Big Data. No obstante, la implementación y la arquitectura de estos sistemas también puede afectar al rendimiento debido a algunas decisiones de diseño. Para detectar dichas ineficiencias, es necesario entender en profundidad el funcionamiento de un motor de Streaming.

Como todo componente de un sistema Big Data, un motor de Streaming parte de una problemática concreta a resolver. Este tipo de problemas comienzan con una fuente de datos constante y terminan con la necesidad de obtener algún tipo de resultado de los mismos en tiempo real. El significado de tiempo real puede variar en función del problema concreto a resolver, donde el tiempo que pasa desde la recepción de los datos hasta la salida de los resultados (latencia)

puede variar desde unos pocos milisegundos o microsegundos o hasta los 5 o 10 minutos.

Cada problema, a su vez, tiene su propio método de resolución así como sus propios métodos de particionado. La forma en la que se mueve el flujo de datos entre los distintos nodos y se resuelven cada uno de los subproblemas está definido en una *topología*. Una topología está compuesta fundamentalmente por: Los nodos de cómputo, qué aplicaciones se ejecutarán y cual es su interconexión. Existen dos tipos de aplicaciones fundamentales a desplegar en los nodos: Emisores y Trabajadores. Las aplicaciones emisoras obtienen datos de distintos orígenes, ya sea de un fichero, de la red, o de otra aplicación destinada para ello (Por ejemplo Kafka [22]) y la distribuyen a los trabajadores que hayan sido definidos por la topología. Los trabajadores, por otro lado, reciben datos, hacen algún tipo de proceso sobre ellos y, de forma opcional, pueden emitir datos a otros nodos (ya sean los mismos datos, resultados, o una combinación de estos). No existe, a priori, ninguna restricción de cuántos trabajadores pueden instanciarse o de la diversidad u homogeneidad de los mismos. En la figura 2 se muestra una topología de ejemplo.

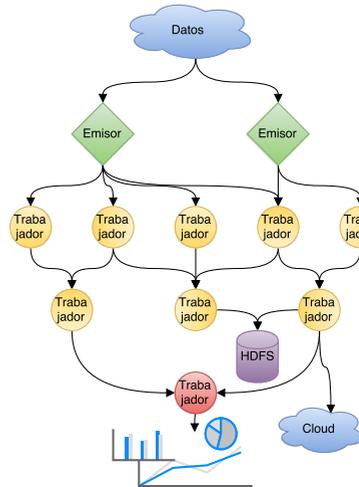


Fig. 2. Arquitectura para el procesamiento en Streaming

Cuando una topología define múltiples destinos para un nodo, este debe decidir a cuál enviar qué tipo de dato. En este punto, la definición de flujo (o Stream) en el sistema tiene cierta importancia. Un flujo de datos está compuesto por *tuplas* “clave, valor”, en las que ambos campos suelen ser (aunque no obligatoriamente) cadenas de texto (Strings). De este modo, un flujo de datos se puede definir como un

conjunto de datos que comparten la misma “clave”. Dicha clave puede ser utilizada para decidir a qué nodo y/o trabajador debe enviarse una determinada tupla. No obstante, existen otros métodos de encaminamiento de los datos que no dependen de la clave de la tupla (como el RoundRobin) y pueden ser utilizados para agilizar este proceso.

El punto más débil, y a su vez el más fuerte, es que este tipo de motores son capaces de enviar cualquier tipo de dato, ya sea un objeto, una cadena de texto o un array de bytes. Dado que fundamentalmente se trabaja con objetos o cadenas de texto, dichos elementos deben ser serializados y deserializados para poder ser transmitidos por la red, lo que supone un cómputo no despreciable para la JVM. Además, en ambos casos la JVM reserva memoria interna en el proceso, que hasta la siguiente llamada al recolector de basura no será liberada o reutilizada. Esta ineficiencia resulta especialmente perjudicial en el uso de cadenas de texto, dado que es la forma más habitual de transmitir información en estos motores. Las cadenas de texto (String) en Java tienen el inconveniente de ser inmutables, por lo que no es posible sobrescribir su memoria. Esta ineficiencia se ha solventado en Java mediante el uso de clases auxiliares como StringBuilder, las cuales, por otro lado, no son utilizadas por los motores actuales.

El último punto a tener en cuenta es la interoperabilidad entre lenguajes. Si una empresa o entidad tiene una determinada aplicación creada, evitará a toda costa utilizar una solución que requiera reescribirla. Por ello, los sistemas de Big Data tienden a ser compatibles entre lenguajes. No obstante, en el caso de algunos de estos sistemas, como Storm, esta implementación es muy ineficiente. Dado que estos sistemas están enmarcados en el uso de Java, los distintos lenguajes de la JVM pueden explotar “al máximo” las capacidades del motor de streaming en concreto. No obstante, la interoperabilidad con otros lenguajes como python o ruby suponen la serialización de los datos *en formato JSON*. Aunque este tipo de serialización es muy versátil y ofrece compatibilidad con distintas plataformas y lenguajes, degrada enormemente el rendimiento y desaprovecha en gran medida el ancho de banda de los enlaces.

C. Resumen

Tras analizar las diferentes implementaciones de los sistemas de procesamiento de flujos actuales, hemos detectado los siguientes puntos débiles, en los cuales nos apoyamos para tomar la decisión de crear un sistema nuevo:

- **Comunicación:** La comunicación, como piedra angular de cualquier sistema de procesamiento en streaming es imprescindible. Una buena implementación permitiría maximizar los anchos de banda de entrada o salida, permitiendo alcanzar más de 30 Gbps y reducir latencias.
- **Lenguaje de programación y portabilidad:** El lenguaje que un sistema de Big Data utilice debe ser tanto eficiente como interoperable.

Ambas cualidades pueden conseguirse partiendo de lenguajes como C/C++ y construyendo envoltorios eficientes en otros lenguajes. Uno de los objetivos a lograr es permitir que aplicaciones ya realizadas para otros motores puedan ejecutarse en el propuesto sin necesidad de realizar ningún cambio en el código.

- **Serialización y tipos de datos:** Creemos que el programador debe ser consciente del sobre coste de serializar diferentes objetos y estructuras. Por ello, un buen sistema de big data, además de proporcionar esta funcionalidad de forma eficiente, debe permitir al programador la capacidad de explotar la comunicación a bajo nivel, es decir, utilizar arrays de bytes.

III. ARQUITECTURA

Construir un motor completo de Streaming que implemente todas y cada una de las funcionalidades que los otros motores soportan actualmente supone, además de un esfuerzo titánico, un tiempo de desarrollo extremadamente elevado. Por este motivo, en este trabajo hemos desarrollado una implementación inicial, centrándonos en las características que consideramos que permitirán ejecutar la mayoría de aplicaciones (o casos de uso) sin echar de menos ninguna funcionalidad. Un ejemplo de funcionalidad que no hemos implementado en esta primera versión es la fiabilidad en la entrega de mensajes. Aunque a primera vista puede parecer una parte fundamental y esencial de estos sistemas, es comúnmente desactivada para mejorar el rendimiento de la aplicación final. La necesidad o no de esta funcionalidad varía en función del problema a resolver. Si nos fijamos en los casos de uso más típicos, como el análisis de twitter o procesamiento de logs, el gran volumen de datos a tratar disminuye la probabilidad de que perder una ráfaga de mensajes suponga un cambio relevante en el resultado o análisis final. No obstante, dado que es una funcionalidad importante para ciertos casos de uso, tenemos planeado añadirla en futuras versiones.

Para implementar esta versión inicial hemos decidido dividir *HPSEngine* en dos elementos independientes: El *Distribuidor* o *Coordinador*, encargado de repartir las tareas y gestionar las tareas entre los diferentes nodos, y las *Unidades de proceso (UP)*, encargadas de realizar una tarea concreta a partir de uno o más flujos de datos. (Ver Figura 3)

A. Distribuidor

Una parte esencial de cualquier sistema distribuido es la capacidad de gestión y coordinación de los diferentes elementos que componen al sistema, para lograrlo, hemos desarrollado un módulo de control llamado “Distribuidor”, encargado de repartir y ejecutar las tareas *UP* entre los distintos nodos.

Para lanzar una aplicación utilizando el Distribuidor, es necesario describir en un fichero en texto plano la topología del sistema (Ver figura 4). Cada *UP* está definida por un par de líneas, en las cuales figura un identificador único de la misma, el nom-

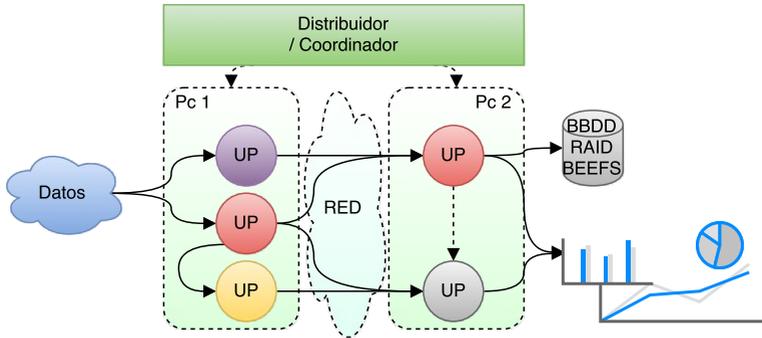


Fig. 3. Arquitectura del sistema propuesto

bre de la aplicación que se ejecutará y otros datos opcionales como la afinidad del proceso o si la comunicación utilizará SSL o no. La segunda línea de la definición, describe la comunicación de salida que tendrá el *UP* utilizando *expresiones-S* [23]. Estas expresiones están formadas por una serie de operadores que pueden enlazarse entre sí y terminan con los identificadores del destino al que se mandará el mensaje. A cada elemento después de un operador le llamamos “destino” independientemente de si es un nodo u otra regla recursiva. De este modo se pueden definir reglas complejas. Por ejemplo, siguiendo la regla de la figura 4, los mensajes de categoría 1 se “duplicarán” a los nodos 2 y 3. También se le aplicará de forma recursiva la regla RR, de modo que se envíen de forma alterna a los nodos 1 y 4. Los posibles operadores para utilizar en la definición de la comunicación son:

```
1 ProgramaEjemplo 192.168.1.103 0x3f SSL
  (Cat (1.(DUP 2 3 (RR 1 4))) (2.(RR 2
    3 4)) (3.(HASH 2 3 4)))
```

Fig. 4. Ejemplo de configuración de un nodo

- **DUP:** En caso de que una lista no tenga operador, se utiliza DUP como operador por defecto. Este operador es la abreviación de *duplicar*, por lo que cada mensaje se enviará por duplicado a todos y cada uno de los destinos a continuación del operador
- **RR:** El operador RR es acrónimo de *Round Robin*. Este operador enviará el mensaje a uno de los destinos especificados a continuación utilizando el algoritmo de *Round Robin*.
- **CAT:** El operador CAT es la abreviación de *Categoría*. Al especificar este operador, es necesario que el programa emisor asocie a cada mensaje una categoría numérica. Dicho operador va seguido de una serie de listas asociadas a cada

categoría que indican a qué destino se enviará el mensaje.

- **HASH:** El operador *Hash* está destinado a poder repartir los mensajes de forma uniforme entre los diferentes destinos. Dado que la naturaleza de los datos puede ser muy variable, es el programador usuario del motor el que debe aplicar una función hash sobre los datos e informar a la librería al enviar un mensaje. Al hash resultante se le aplica un módulo sobre el número de elementos de la lista a continuación del operador, de forma que se asegure que siempre el mismo destino reciba la misma información.

Tras leer la configuración, el Distribuidor copia las aplicaciones a los nodos mediante comandos *RSYNC* y las ejecuta, indicando mediante variables de entorno la dirección IP y el puerto del Distribuidor. Cuando una Unidad de Proceso inicia su ejecución, utiliza las variables de entorno para conectarse con el Distribuidor. Gracias a esta conexión permanente, es posible monitorizar el estado de las diferentes *UPs* de forma individual. Esto permite al distribuidor relanzar un proceso en caso de que haya finalizado de forma no esperada

B. Unidad de Proceso (UP)

Una *UP* es un programa externo a la librería que hace uso de la API de comunicación de esta. Estos programas se encuentran ya compilados y empaquetados en la carpeta desde la que se lanza el sistema completo. El usuario programador tiene la obligación de llamar a una función de inicio que será la que, utilizando las variables de entorno, se conectará al Distribuidor y descargará la descripción del enrutado. Dado que una de las características fundamentales del sistema es la alta velocidad, si las reglas de enrutado tuviesen que interpretarse con cada mensaje el ancho de banda se degradaría enorme-

mente. Para ello, estas reglas son interpretadas una única vez, generando como salida un fichero temporal en código C. Dicho código es compilado y enlazado en tiempo de ejecución, lo que no solo permite un rendimiento óptimo en el enrutado de cada mensaje, sino que a su vez permite cambiar el código de enrutado sin tener que relanzar la aplicación.

La API de comunicación con el usuario es simple y directa, proporcionando las funciones típicas de enviar y recibir. Dichas funciones también necesitan (para leer o escribir) un puntero a una estructura de configuración. En ella figuran datos del mensaje a enviar como la longitud en bytes, el tipo de mensaje a enviar (Un entero, un array, una cadena de texto, etc), la categoría o el hash asociado al mensaje. Casi todas las funcionalidades de esta API están encapsuladas en código Java para que un programa en este lenguaje pueda usarlas. Con el fin de realizar comparativas justas, hemos encapsulado la librería para que simule el comportamiento de Apache Storm, de forma que exactamente el mismo test pueda ejecutarse en ambas plataformas sin cambiar ni una línea de código. Dicha encapsulación presentó desafíos de diseño importantes, ya que Storm trabaja principalmente con Strings y, dado que las cadenas de texto son inmutables en la JVM, la memoria de cada mensaje es reservada y liberada constantemente. Para enfrentarnos a este desafío, realizamos una implementación básica con reservas de memoria y otra que accede directamente a la memoria de la JVM sobrescribiendo ilegalmente las cadenas de texto previamente reservadas. Aunque los resultados de estas pruebas motivan la realización del Hack (Como se comenta en la sección de resultados), el uso de este solamente puede hacerse en entornos muy controlados en los que siempre se transmiten mensajes de tamaños parecidos o iguales, ya que de lo contrario existe el riesgo de acceder a memoria no reservada y que el programa termine de forma inesperada.

B.1 Librería de red asíncrona

Para uso interno del sistema, hemos desarrollado nuestra propia librería de comunicación asíncrona, capaz de mandar información a velocidades de hasta 37 Gbps. Este ancho de banda se ha conseguido a cambio de aumentar ligeramente la latencia, ya que en nuestros casos de uso no supone un factor limitante.

Esta librería expone una API que permite mandar y recibir mensajes de forma bloqueante (síncrona) o no bloqueante (asíncrona) además de establecer conexiones TLS de forma transparente. En la explicación a continuación se usarán los términos HP-Sockets para referirse a los sockets expuestos por esta librería y BSDSockets para referirse a los sockets del sistema operativo.

La alta velocidad de esta librería se consigue abriendo un nuevo hilo de comunicación que usa BSDSockets TCP. Este hilo comparte datos con el programa principal usando un doble buffer almacenado en memoria compartida. Para evitar copias de

memoria innecesarias el hilo envía o recibe directamente en estos buffers mediante las funciones `send` y `recv` de la api BSD de sockets. Cada uno de estos buffers tiene un tamaño de 512 KB que hemos elegido tras realizar pruebas experimentales y comprobar que es el que ofrece mejor rendimiento. El acceso a la memoria compartida se controla mediante spinlocks, que ofrecen un mayor rendimiento que los semáforos convencionales.

El uso de esta librería es transparente al usuario, ya que en ningún momento accede a ella de forma directa, no obstante un programa que use esta librería deberá crear un HPSSocket de lectura o escritura. El HPSSocket se encarga de preparar el doble buffer compartido y lanzar un hilo de comunicación el cual utilizará un BSDSocket para comunicarse con el mundo exterior. A la hora de enviar datos el HPSSocket almacena en un buffer hasta que se llena, momento en el que se libera su spinlock permitiendo al hilo de comunicación que envíe los datos por el BSDSocket. Si el hilo se encuentra bloqueado enviando el buffer anterior, el programa tendrá que esperar hasta que este quede libre para poder continuar la copia de datos. La recepción funciona de forma similar. En este caso el hilo esperará a recibir datos de un BSDSocket para llenar los buffers circulares. Por otro lado, cuando el HPSSocket necesite hacer una lectura acudirá al buffer circular. En caso de que los buffers no contengan suficientes bytes para realizar la lectura, el HPSSocket tendrá que esperar a que uno de los buffers circulares se llene para continuar. La librería permite tanto enviar como recibir mensajes de mayor tamaño que el que tiene el doble buffer. En este caso el HPSSocket y el hilo se irán alternando leyendo y escribiendo del doble buffer hasta que se complete la operación.

IV. RESULTADOS

A la hora de evaluar nuestro sistema es necesario tener en cuenta las soluciones ya existentes en el estado del arte, ya que queremos comprobar que las mejoras obtenidas sean lo suficientemente sustanciales para compensar la reducción de funcionalidad respecto a los otros sistemas.

Estas pruebas han sido realizadas en un entorno de alto rendimiento compuesto por tres servidores. Dos de ellos, conectados mediante un switch Ethernet a 40 Gbps. El tercer nodo, se encontraba conectado por una interfaz de 1 Gbps y fue usado como Coordinador del cluster. Todos los servidores de proceso son Intel Xeon de doble socket. Los dos nodos de cómputo utilizan un Intel Xeon CPU E5-2620 v3 @ 2.40GHz con 64 GB de memoria RAM, mientras que el tercero y coordinador tiene un Intel Xeon E5-1650 v3 @ 3.50GHz con 32 GB de memoria. En ningún caso se utilizó ningún driver específico de alto rendimiento, siempre la versión original del fabricante distribuida por el Sistema Operativo: CentOS 7. Las aplicaciones realizadas, en todos y cada uno de los casos probados, han intentado tener el mínimo coste computacional posible, de forma que todas las aplicaciones se enfoquen en la transmisión y

recepción de mensajes preformados en memoria, utilizando una única instancia de emisión y una única instancia de recepción. El trabajo de cada nodo consiste únicamente en mandar o recibir el mensaje e imprimir estadísticas de ancho de banda cada cierto número de mensajes.

En primer lugar hemos evaluado el rendimiento de los sistemas existentes para compararlos entre ellos. Como se puede ver en la Figura 5, hemos comprobado que Storm (versión 0.10) ofrece mayor rendimiento para mensajes de tamaño medio pero Spark Streaming parece comportarse mejor cuando los mensajes son pequeños, lo cual parece razonable, ya que Spark Streaming utiliza micro-batches, es decir, agrupa mensajes. No pudimos hacer la prueba con mensajes muy grandes utilizando Storm (Mayores a unos pocos KiloBytes), ya que se quedaba sin memoria antes de terminar la ejecución, a pesar de permitirle reservar más de 12 GB por instancia. La figura también muestra el rendimiento de la librería de comunicación 0Mq. Esta librería fue utilizada como base para el diseño de las comunicaciones de Apache Storm. Se observa que la librería obtiene un rendimiento muy elevado cuando trabaja con redes infiniband a 10 Gbps⁵, rendimiento que no se alcanza al utilizar una tarjeta de Ethernet de 40 Gbps.

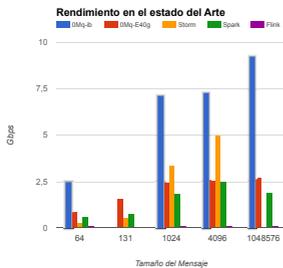


Fig. 5. Rendimiento de las diferentes soluciones del estado del arte

Tal y como se ha comentado anteriormente, nuestra API puede usarse como capa de comunicación entre otros lenguajes y sistemas de más alto nivel, por lo que resulta interesante comprobar la penalización de rendimiento que puede darse. En la Figura 6 podemos ver el ancho de banda conseguido por HPSE básico y los distintos usos de la API desde Java. Al usar mensajes pequeños en Java y en la micro-api de Storm el sobrecoste de la JVM es muy elevado y se reduce enormemente el rendimiento del sistema. Por otro lado, al utilizar mensajes más grandes, el número de veces que se debe realizar ll-

⁵Los datos de esa prueba fueron obtenidos desde <http://zeromq.org/area:results>

madas a funciones se reduce y con ello la diferencia entre la versión completamente nativa y la de Java es prácticamente indistinguible. Estas pruebas demuestran el impacto causado por la inmutabilidad de las cadenas de texto al comparar las dos versiones de la micro-api de Storm sobre HPSE. Aunque la versión con hack permite exprimir de forma efectiva el rendimiento, no es posible utilizar esa versión de forma estable con mensajes de tamaño variable.

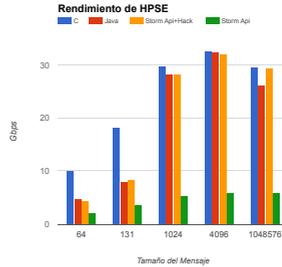


Fig. 6. Rendimiento de las diferentes implementaciones de HPSE

Por último, comparamos nuestro sistema con Storm ya que, según las pruebas que mostramos en la figura 7, es el sistema de procesamiento en Streaming con mejor uso del ancho de banda. En esta figura se muestran algunos de los resultados combinados de los dos gráficos anteriores. Podemos observar que el ancho de banda aprovechado en la implementación puramente nativa (C) es mucho mayor que el de Storm. Resulta más interesante comparar la versión estándar de Storm con la versión que funciona sobre nuestra API. Podemos comprobar que la versión modificada es más rápida en todos los casos, obteniéndose las mayores mejoras de rendimiento cuando los mensajes son más pequeños. No obstante, nuestra implementación utiliza 2 threads por instancia, mientras que la implementación de storm utiliza todos los cores de la máquina por instancia hasta casi el 100%. Por lo tanto, en una máquina con menos cores o habiendo instanciado más veces nuestra micro-api de recepción, la diferencia entre ambas soluciones habría sido mucho mayor.

V. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo hemos introducido un nuevo motor de procesamiento distribuido de alto rendimiento, capaz de procesar datos más rápido que otras soluciones existentes e incluso mejorar el rendimiento de algunas de ellas. Nuestras pruebas han demostrado que cumplimos los objetivos de rendimiento propuestos al inicio de este trabajo.

Consideramos que nuestra solución ayuda a afrontar el cada vez mayor volumen de datos que deben ser procesados por aplicaciones de big data,

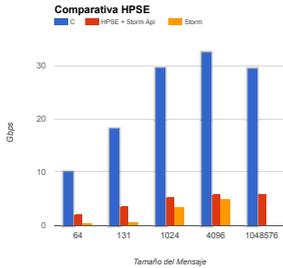


Fig. 7. Rendimiento de las diferentes soluciones del estado del arte

ofreciendo una API flexible que puede ser usada para solucionar los problemas presentes o futuros. Al mismo tiempo, esta API hace un uso eficiente de los recursos que permite reducir los costes del análisis de datos ya que aprovecha las máquinas escalando vertical y horizontalmente al mismo tiempo.

Este sistema nos servirá como base para desarrollar otras herramientas de procesado y análisis de datos, centrándonos en el análisis de datos de red y de logs de aplicación. También nos ayudará a acelerar sistemas existentes basados en tecnologías como Storm.

Como trabajo futuro planeamos centrarnos en tres mejoras distintas. La primera consiste en mejorar aún más la comunicación entre las unidades de proceso que se ejecuten en el mismo nodo. Para ello añadiremos la capacidad de elegir entre usar memoria compartida o sockets. La segunda mejora consiste en la implementación de un subsistema capaz de asegurar que los mensajes entre dos nodos sean procesados en algún momento, independientemente de si existe algún tipo de error o no. En la tercera mejora, queremos que nuestro sistema sea capaz de reconfigurar de forma dinámica las topologías para aprovechar al máximo el rendimiento de cada equipo que forme parte del cluster, de forma que el número de instancias de una determinada aplicación varíe en función de la carga que esté soportando. Esto tendrá el beneficio añadido de simplificar la configuración, ya que no será necesario indicar en qué nodo específico deberá ejecutarse cada instancia de una unidad de proceso.

REFERENCIAS

[1] Ben Walker, "Data generated per day," April 2015, [Disponible: <http://www.vcloudnews.com/wp-content/uploads/2015/04/big-data-infographic1.png>].

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., pp. 1097–1105. Curran Associates, Inc., 2012.

[3] Min Chen, Shiwen Mao, and Yunhao Liu, "Big data: A

survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.

[4] Nathan Marz and James Warren, *Big Data: Principles and best practices of scalable realtime data systems*, Manning, Publications Co., 2015.

[5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, January 2008.

[6] Tom White, *Hadoop: The Definitive Guide*, O'Reilly Media, Inc., 2012.

[7] K. Shvachko, Haihong Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, May 2010, pp. 1–10.

[8] "Apache Storm," [Disponible: <http://storm.apache.org/>].

[9] R. Ranjan, "Streaming big data processing in datacenter clouds," *IEEE Cloud Computing*, vol. 1, no. 1, pp. 78–83, May 2014.

[10] Ruben García-Valcárcel, Rafael Leira, Ivan Gonzalez, and Francisco J. Gomez Arribas, "Evaluando apache hadoop para análisis de tráfico de red," in *XXV Jornadas de Paralelismo 2015*, Córdoba, Spain, Sep 2015.

[11] "Apache Spark Streaming," [Disponible: <http://spark.apache.org/streaming/>].

[12] "Apache Flink," [Disponible: <https://flink.apache.org/>].

[13] L. Neumeier, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *2010 IEEE International Conference on Data Mining Workshops*, Dec 2010, pp. 170–177.

[14] J. Shafer, S. Rixner, and A. L. Cox, "The hadoop distributed filesystem: Balancing portability and performance," in *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, March 2010, pp. 122–133.

[15] N. S. Islam, M. W. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda, "High performance rdma-based design of hdfs over infiniband," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Los Alamitos, CA, USA, 2012, SC '12, pp. 35:1–35:35. IEEE Computer Society Press.

[16] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, J. Majors, A. Manzanares, and Xiao Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, April 2010, pp. 1–9.

[17] Yandong Wang, Xinyu Que, Weikuan Yu, Dror Goldenberg, and Dhiraj Sehgal, "Hadoop acceleration through network levitated merge," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2011, SC '11, pp. 57:1–57:10. ACM.

[18] "Scylla DB Architecture," [Disponible: <http://www.scylladb.com/technology/architecture/>].

[19] J. Huang, X. Ouyang, J. Jose, M. Wasi ur Rahman, H. Wang, M. Luo, H. Subramoni, C. Murthy, and D. K. Panda, "High-performance design of hbase with rdma over infiniband," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, May 2012, pp. 774–785.

[20] Christos Kalantzis, "Revisiting 1 million writes per second," <http://techblog.netflix.com/2014/07/revisiting-1-million-writes-per-second.html>, July 2014.

[21] Scylla, "Scylla vs. cassandra benchmark (cluster)," <http://www.scylladb.com/technology/cassandra-vs-scylla-benchmark-cluster-1/>, 2015, [Online; accessed 21-November-2015].

[22] "Apache Kafka," [Disponible: <http://kafka.apache.org/>].

[23] John McCarthy, "Recursive functions of symbolic expressions and their computation by machine, part i," *Communications of the ACM*, vol. 3, no. 4, pp. 184–195, 1960.

[24] V. Moreno, P.M. Santiago del Río, J. Ramos, J.L. García-Dorado, I. Gonzalez, F.J. Gomez Arribas, and J. Aracil, "Packet storage at multi-gigabit rates using off-the-shelf systems," in *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Em-*

bedded Software and Syst (HPCC,CSS,ICISS), 2014
IEEE Intl Conf on, Aug 2014, pp. 486-489.

Cloudification of a Legacy Hydrological Simulator using Apache Spark

Silvina Caño-Lores¹ Andrei Lapin² Peter Kropf³ and Jesús Carretero⁴

Abstract— Hydrologists usually rely on complex multiphysics systems and data collected from geographically distributed sensors in order to obtain good quality predictions and analysis of how water moves through the environment. Nowadays, the computational resources needed to run such complex simulations, and the increasing size of datasets related to the models, have arisen interest towards distributed infrastructures like clouds. This paper presents the results of applying a cloudification methodology to a legacy hydrological simulator (HydroGeoSphere), wrapped with an ensemble Kalman filter. This work describes how the methodology was applied, the particularities of its implementation and configuration for the Apache Spark iterative map-reduce platform, and the results of an evaluation in a commodity cluster against an MPI implementation of the simulator.

Keywords— Cloud Computing, Cloudification, Apache Spark, Hydrology, HydroGeoSphere, ensemble Kalman filter.

I. INTRODUCTION

HAVING good quality predictions of the behavior of hydrological systems is a key aspect for water resource management. Such systems are highly heterogeneous in terms of physical characteristics and parameters, relying on complex multiscale non-linear processes and matrix operations. These models can often benefit from their interoperation with models from other domains, yielding massively complex multiphysics simulations. Furthermore, their input data typically comes from several geographically distributed sensors, which may provide a huge amount of data that has to be efficiently integrated, processed, and visualized. Considering that it is sometimes necessary to meet severe time constraints due to the need for behaviour-related forecasts or reliable status reports for emergency management, being able to tackle such complex computations fast becomes critical.

Hydrologists have found in high-performance computing (HPC) and high-throughput computing (HTC) two key tools to meet such specific requirements. In particular, HPC parallelization techniques on multicore systems, and grid-like HTC technologies have increased the scale, size and complexity of the addressable problems in this domain. Nevertheless, the ever-increasing datasets related to the models have shifted the interest towards more data-intensive

infrastructures like compute clouds. Cloud Computing constitutes a paradigm in which flexibility and elasticity are key, since they allow to tailor the resource pool on-demand to reach a satisfactory cost-performance trade-off. Moreover, it holds further advantages for end-users, like ease of management and access to the platform.

In a previous work we introduced the architecture of a cloud-based system for environmental monitoring and simulation [1]. This paper is a follow-up work in which we aim to analyze the performance of the core simulator after applying a cloudification methodology for scientific simulators, specifically designed to maximize resource efficiency and performance in cloud infrastructures [2], [3]. The simulator workflow wraps two third-party hydrological kernel applications –GROK and HydroGeoSphere (HGS) [4]– as an ensemble Kalman filter (EnKF) [5], [6], which was implemented in Apache Spark¹, in addition to the resulting map-reduce jobs that resulted from the application of the methodology.

The rest of the paper is organized in the following way: Section II discusses relevant works related to the field; Section III depicts the target simulator and our motivation for the cloudification procedure; Section IV presents the current approach to cloudify the simulator; Section V provides a performance comparison and scalability assessment of the cloudified version of the simulator against the original MPI implementation of the EnKF, on a commodity cluster; finally, Section VI provides key ideas and insight on future works.

II. RELATED WORKS

Hydrologic modeling requires handling a wide range of large scale non-linear processes and matrix operations. The inherent computational complexity of these tasks have lead scientist to implement parallel versions of these models in the form of tailored simulators for multi-core environments. Keeping this platform in mind, Cheng et al. [7] propose a development toolkit for watershed simulations based on MPI. The abstractions proposed in that work allowed to hide the complexity of the MPI primitives from the end-user, while supporting domain partitioning, workload balancing and memory scaling across the cores. Another approach is found in the work of Hwang et al. [8], who proposed a version of the simulator we are tackling in this work, HGS, reworked to include parallelization techniques

¹Computer Science and Engineering Department, University Carlos III of Madrid, e-mail: scano@inf.uc3m.es

²Computer Science Department, University of Neuchâtel, e-mail: andrei.lapin@unine.ch

³Computer Science Department, University of Neuchâtel, e-mail: peter.kropf@unine.ch

⁴Computer Science and Engineering Department, University Carlos III of Madrid, e-mail: jesus.carretero@uc3m.es

¹The Apache Spark project is available at <http://spark.apache.org/>

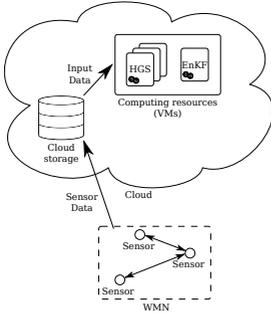


Fig. 1. Architecture of the real-time environmental monitoring and hydrological modelling system depicted in [1].

like OpenMP [9]. This allowed the execution of the simulator in a shared memory platform, thus reducing the execution time for matrix processing. We can also find applications of OpenMP in other simulators such as the Soil and Water Assessment Tool (SWAT) [10]. Nevertheless, the scientific community found the need for further improvements in computational power to handle larger and more complex scenarios. This led to the application of cluster and grid computing to the hydrology domain. For instance, we find cluster-oriented [11] and grid-oriented [12] MPI-based watershed models, complex multiphysics and multiscale problems solved in grids [13], and MPI legacy applications virtualized for grid environments [14].

But not only the computational complexity played a key role in this trend that aims to distribute hydrological applications. In geospatial fields like hydroclimatology, as well as hydrology, due to the amount of data that is constantly coming from numerous sensing devices, it is not feasible to store all the data in a single place. Lack of standardization and single data formatting specification make the situation even worse. Given the quantity and variety of data that feeds these models, faster processing of such volumes is key to conduct forecasts in a timely manner. In this context, the amount of data to exploit and the modeling requirements yield more and more computing and storage resources.

Considering that interacting efficiently with computing grids is costly in economic terms, and complex with respect to development and management, recent works have considered clouds as alternatives to traditional grids. The main advantages of clouds for this matter are the flexibility and elasticity in the resource selection, its ease of access, and its cost-performance ratio. Studies have shown the feasibility of running cloud-based frameworks for multiscale data analysis [15], [16], with complexities comparable to the hydrology domain. These works have shown that data and tool integration are easier for end-users in these environments, while performance and storage

capability remain comparable to grids.

Other works approached the benefits of Cloud Computing from a hybrid perspective, integrating data and computing infrastructures from grids and with external cloud providers like Amazon Elastic Compute Cloud (EC2) [17]. This work is particularly relevant, as it shows the suitability of clouds for a wide range of hydrological problems, covering both computationally intensive (traditional HPC), and multiple scenario applications (many-task computing (MTC)). In the hydrology domain, we can also find HydroCloud [18], a cloud-based data integration system to store and explore data. This constitutes an attempt to aggregate data from different sources and present it to the user in a single format for further analysis.

A similar research line is followed by our previous work [1]. In it we presented an architecture for a system that combines an environmental monitoring module, based on a wireless mesh network (WMN) as data source, and a cloud-based computing service to perform environmental simulations (Figure 1). Even though the system was tested in a real-world deployment in the Enmental in Switzerland, and proven to be operable, the performance of the core simulation procedures have not been assessed nor built for the target infrastructure. This work tackles this issue by developing a version of the simulator specifically tailored to the cloud, achieved by means of a previously proposed cloudification methodology [3].

III. PROBLEM STATEMENT AND MOTIVATION

In the domain of hydrology, recent technological and mathematical advances allow to significantly improve the precision of the simulations by integrating data acquisition techniques with the modelling process [19]. One of the state of the art simulators for this purpose is the EnKF-HGS [20] simulator. This is an MPI implementation of the ensemble Kalman filter (EnKF) technique for data assimilation, wrapping two proprietary simulation kernels: GROK and HydroGeoSphere (HGS) [21], [4]. HGS is an integrated hydrological modelling software, and GROK is a preprocessor that prepares the input files for HGS.

Data assimilation is the process of adjusting the simulated model to the environmental field measurements. Therefore, EnKF-HGS provides functionality for dynamic stochastic simulations of the groundwater and surface water profiles, and allows to constantly improve the simulated model by sequentially assimilating fresh field measurements. EnKF-HGS runs an ensemble of model instantiations – which we will call *realizations* – using HGS, and performs an update of the simulated realizations with the field measurements. As shown in Figure 2, HGS allows the numerical simulation of all the relevant surface water and groundwater processes in a pre-alpine type of valleys. Hence, each HGS simulation in the ensemble of realizations represents a long-running compute-intensive process, which comprises a sequential execution of two core binaries GROK and HGS, in a

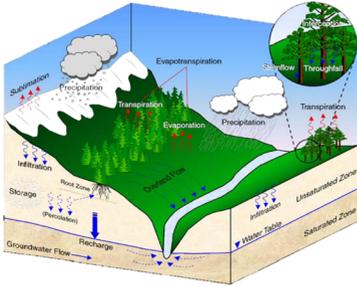


Fig. 2. Typical surface water and groundwater processes in a pre-alpine type of valleys.

pipelined manner.

As any other Monte Carlo based approach, the EnKF technique needs to perform hundreds of model simulations in order to achieve the required precision. This amount of computations implies having a dedicated HPC infrastructure, which is not always available to the end-user. Cloud Computing offers potentially unlimited computing resources and convenient and easy-to-use scaling and resource provisioning mechanisms. Hence, it is meaningful to explore the possibility to port EnKF-HGS to the cloud, as it would allow the implementation and execution of larger models, and the reduction of the overall execution time.

IV. CLOUDIFICATION PROCEDURE

The cloudification procedure described in [3] consists of three steps: an analysis of the original application to find a partitioning key, which will guide the domain distribution; an input adaptation stage that indexes all the necessary parameters for each value of the key; and a simulation kernel wrapping step in order to simulate each key-parameters tuple independently.

The original application consisted of an MPI implementation of an EnKF, which wraps two legacy binaries that execute the simulation (GROK and HGS). The EnKF operates with a set of realizations, which constitute instantiations of the underlying model. These realizations are simulated independently, and the output is gathered afterwards for further processing. Following the former methodology, we selected the realization identifier as key, and the collection of the model's data per realization as value. These key-value pairs can be distributed and executed independently across the nodes, and their results can be finally gathered in the driver process. This process is repeated several times in order to refine the final model.

A particularity of the kernel binaries is that they are pre-build black boxes. An effect of this is that they rely on hard-coded input paths for the inter-

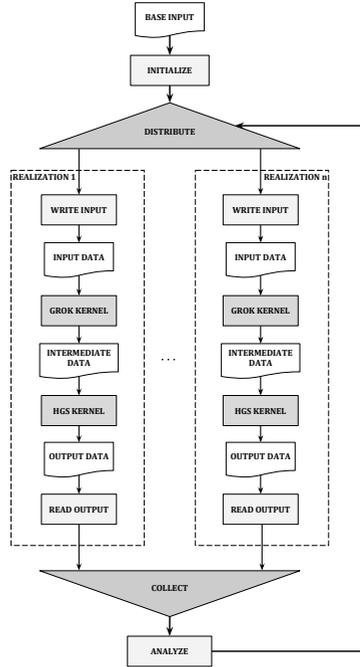


Fig. 3. Final Spark workflow for the cloudified EnKF-HGS.

mediate files they handle. This constitutes a limitation, as we are forced to execute both binaries in the same machine to ensure data locality per realization. To achieve this, we implemented the model in Apache Spark, and we exploited its partitioning mechanisms to ensure each full realization is computed in the same node in a pipelined manner. As seen in Figure 3, the current workflow forms a collection of independent pipelines that map to each realization, with the proper input data. There is the possibility to further optimize this, but currently it is a promising approach that can be generalized to many applications that make use of this filter.

V. EVALUATION

The resulting application was integrated with a Spark environment running on top of the Hadoop Distributed File System (HDFS) and Yet-Another Resource Negotiator (YARN)². To configure the environment, we profiled GROK and HGS binaries to estimate their resource usage. We found that GROK

²HDFS and YARN belong to the Apache Hadoop project, accessible at <http://hadoop.apache.org/>

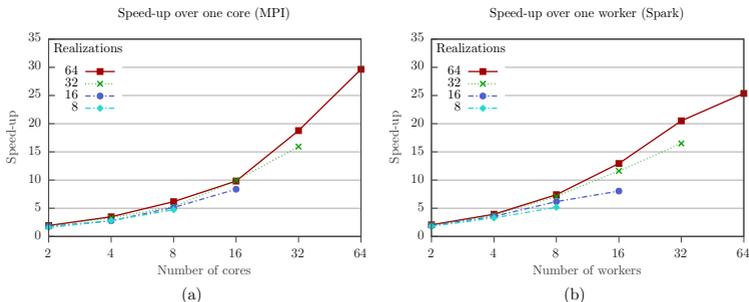


Fig. 4. Speed up results for the original MPI ensemble Kalman filter (a), and its cloudified version (b). Results for two and four realizations were not included for visibility reasons, as they are very similar to the results for eight realizations.

TABLE I
 CONFIGURATION PARAMETERS FOR THE SPARK PLATFORM AND
 THE YARN RESOURCE MANAGER.

YARN settings		
Virtual core allocation	min.	1
	max.	8
Memory allocation (MB)	min.	256
	max.	8192
Node CPUs		8
Node memory (MB)		8192
Spark settings		
Executer memory (MB)		471
Driver memory (MB)		7168
Driver cores		6
Executer memory overhead (MB)		384
Driver memory overhead (MB)		717

is I/O-intensive, while the HGS kernel is compute-intensive, and makes a full usage of the CPU resources of the machines. Neither of them consume a significant amount of memory in the worker nodes (less than 30MB for the current model).

The main source for memory usage is the EnKF workflow running on top of the kernels, due to the large matrices it uses for data distribution and processing. In Spark, these data structures are handled by a specific container acting as the driver of the workflow. For large experiments, this driver process requires up to 7GB of memory due to its computing needs and the overhead of the platform.

From the former profiling of the original kernels, for this preliminary evaluation we relied on eleven slave nodes with 8GB of RAM and two Intel Xeon E5405@2.00GHz processors, with four cores each; and a dedicated node for the driver container with an overall amount of 94GB of RAM and four In-

tel Xeon E7-4807@1.87GHz processors, with six cores each and hyper-threading enabled. Table I shows a summary of the configuration parameters considered in the Spark environment, and its underlying resource manager YARN, taking into account the former requirements with respect to resource consumption.

We have tested two versions of the simulator in the environment as described above in order to compare the speed-up of each implementation. Figure 4 shows the speed-up of the MPI and Spark implementations, respectively. For the Spark implementation we executed the simulator with 2 to 64 workers; similarly, the MPI version was executed with 2 to 64 MPI processes, where each process was provided with one CPU core. In both cases, we used a medium-size hydrological model as input, and we scaled the number of realizations from 2 up to 64 to test weak scaling.

The execution results show that our Spark implementation has a better speed-up for the majority of corresponding experiments. We are currently studying the rationale behind these results, which seem to be related to I/O contention, since the data-transferring mechanism between computations of the HGS model and the MPI EnKF is file-based.

However, the MPI implementation shows a noticeably better speed-up for the maximal size of the chosen model with the maximal number of computing workers/processes (64/64). This scalability issue appeared in the Spark implementation due to the specifics of the post-processing logic of the EnKF, which we did not tackle in the current work. The post-processing logic includes a set of operations with large-size matrices where complexity of the computation is directly proportional to the number of realizations in the ensemble, which resulted in the reduced performance of the implementation due to the amount of unnecessary data transfers. The MPI implementation performs the post-processing computation in a distributed manner, unlike our Spark implementation, which currently aggregates the output of the realizations in the driver node in order to as-

semble matrices, and performs the post-processing with no computation distribution. We are working towards implementing a fully distributed version of the cloudified application to compare performance fairly.

VI. CONCLUSIONS

This paper presents the results of applying a cloudification methodology to the legacy hydrological simulator EnKF-HGS. As a result, we obtained a functional application that is able to run in a distributed environment using the Apache Spark framework. The cloudified simulator was evaluated against the original MPI implementation, and showed comparable performance results on small and medium-sized realization sets. Nevertheless, scalability issues arose as the number of nodes and realizations increased beyond this point.

As future work, we will further assess the scalability and efficiency of the cloudified simulator, considering the implementation of relevant aspects of the post-processing stage that are included in the MPI version. As the speed-up results show that the cloudified application is competitive with MPI in an infrastructure favorable to the latter, we believe that solving this scalability problem would make our solution preferable in the cloud, due to the reduced network bandwidth that would hurt the performance of MPI. Additionally, we will also study the I/O throughput of both implementations, since EnKF uses files for input, intermediate, and output data, and this could easily induce I/O contention for very large-size models. It is left for future work to integrate the cloudified application with a memory-centric file system, as it may improve performance in these cases by improving intra-node data locality.

VII. ACKNOWLEDGEMENTS

This work has been partially funded under the grant TIN2013-41350-P of the Spanish Ministry of Economics and Competitiveness, and the COST Action IC1305 "Network for Sustainable Ultrascale Computing Platforms" (NESUS).

BIBLIOGRAPHY

- [1] A. Lapin, E. Schiller, P. Kropp, O. Schilling, P. Brunner, A. J. Kapic, T. Braun, and S. Maffioletti, "Real-time environmental monitoring for cloud-based hydrogeological modeling with hydroglobe," in *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on CyberSpace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPC, CSS, ICESSE), 2014 IEEE Intl Conf on, Aug 2014*, pp. 959-965.
- [2] Silvina Caño-Lores, Alberto García, Félix García Carballera, and Jesús Carretero, "A cloudification methodology for numerical simulations," in *Euro-Par 2014: Parallel Processing Workshops - Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part II*, 2014, pp. 375-386.
- [3] Silvina Caño-Lores, Alberto García Fernández, Félix García-Carballera, and Jesús Carretero Pérez, "A cloudification methodology for multidimensional analysis: Implementation and application to a railway power simulator," *Simulation Modelling Practice and Theory*, vol. 55, pp. 46-62, 2015.
- [4] Philip Brunner and Craig T. Simmons, "Hydroglobe: A fully integrated, physically based hydrological model," *Ground Water*, vol. 50, no. 2, pp. 170-176, 2012.
- [5] Geir Evensen, "Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics," *Journal of Geophysical Research: Oceans*, vol. 99, no. C5, pp. 10143-10162, 1994.
- [6] Gerrit Burgers, Peter-Jan van Leeuwen, and Geir Evensen, "Analysis scheme in the ensemble Kalman filter," *Monthly Weather Review*, vol. 126, no. 6, pp. 1719-1724, 1998.
- [7] Jing-Ru C Cheng, Robert M Hunter, Hwai-Ping Cheng, and David R Richards, "A parallel software development for watershed simulations," in *Computational Science-ICCS 2005*, pp. 460-468. Springer, 2005.
- [8] H-T Hwang, Y-J Park, EA Sudicky, and PA Forsyth, "A parallel computational framework to solve flow and transport in integrated surface-subsurface hydrologic systems," *Environmental Modelling & Software*, vol. 61, pp. 39-58, 2014.
- [9] Leonardo Dagum and Ramesh Enon, "OpenMP: an industry standard API for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46-55, 1998.
- [10] Seo Jin Ki, Tak Sugimura, and Albert S Kim, "OpenMP-accelerated SWAT simulation using intel C and FORTRAN compilers: Development and benchmark," *Computers & Geosciences*, vol. 75, pp. 66-72, 2015.
- [11] Tie-jian Li, Jia-hong Liu, Yang He, and Guang-qian Wang, "Application of cluster computing in the digital watershed model," *Advances in Water Science*, vol. 17, no. 6, pp. 841, 2006.
- [12] Isabel Campos, Ignacio Coterillo, Jesús Marco, Agustín Monteoliva, and Carlos Oldani, "Modelling of a watershed: a distributed parallel application in a grid framework," *Computing and Informatics*, vol. 27, no. 2, pp. 285-296, 2012.
- [13] G Lecca, Monique Petitdidier, L Huchy, M Ivanovic, N Kussul, Nicolas Ray, and V Thieron, "Grid computing technology for hydrological applications," *Journal of Hydrology*, vol. 403, no. 1, pp. 186-199, 2011.
- [14] Evangelos Floros and Yannis Cotronis, "Exposing MPI applications as grid services," in *Euro-Par 2004 Parallel Processing*. Springer, 2004, pp. 436-443.
- [15] Sifei Lu, Reuben Mingguang Li, William Chandra Tjhi, Kee Khooon Lee, Long Wang, Xiarong Li, and Di Ma, "A framework for cloud-based large-scale data analytics and visualization: Case study on multiscale climate data," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 618-622.
- [16] Chaowei Yang, Michael Goodchild, Quynh Huang, Doug Nebert, Robert Raskin, Yan Xu, Myra Bambacus, and Daniel Fay, "Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing?," *International Journal of Digital Earth*, vol. 4, no. 4, pp. 305-329, 2011.
- [17] Gen-Tao Chiang, Martin T Dove, C Isabella Bovolo, and John Ewen, "Implementing a grid/cloud science infrastructure for hydrological sciences," in *Guide to e-Science*, pp. 3-28. Springer, 2011.
- [18] Michael P McGuire, Martin C Roberge, and Jie Lian, "Hydrocloud: A cloud-based system for hydrologic data integration and analysis," in *Computing for Geospatial Research and Application (COM. Geo), 2014 Fifth International Conference on*. IEEE, 2014, pp. 9-16.
- [19] G. Bauser, Harrie-Jan Hendricks Franssen, Stauffer Fritz, Hans-Peter Kaiser, U. Kuhlmann, and W. Kinzelbach, "A comparison study of two different control criteria for the real-time management of urban groundwater works," *Journal of Environmental Management*, vol. 105, pp. 21-29, 2012.
- [20] Wolfgang Kurtz, Harrie-Jan Hendricks Franssen, Hans-Peter Kaiser, and Harry Vereecken, "Joint assimilation of piezometric heads and groundwater temperatures for improved modeling of river-aquifer interactions," *Water Resources Research*, vol. 50, no. 2, pp. 1665-1688, 2014.
- [21] R Therrien, R McLaren, E Sudicky, and S Panday, "A Three-dimensional Numerical Model Describing Fully-Integrated Subsurface and Surface Flow and Solute Transport," Tech. Rep., 2010.

Lenguajes, compiladores y herramientas de programación y ejecución paralela

Revisiting Conventional Task Schedulers to Exploit Asymmetry in ARM big.LITTLE Architectures for Dense Linear Algebra

Luis Costero¹, Francisco D. Igual², Katalin Olcoz³,
Enrique S. Quintana-Ort⁴, Francisco Tirado⁵

Resumen— Dealing with asymmetry in the architecture opens a plethora of questions from the perspective of scheduling task-parallel applications, and there exist early attempts to address this problem via *ad-hoc* strategies embedded into a runtime framework. In this paper we take a different path, which consists in addressing the complexity of the problem at the library level, via a few asymmetry-aware fundamental kernels, hiding the architecture heterogeneity from the task scheduler. For the specific domain of dense linear algebra, we show that this is not only possible but delivers much higher performance than a naive approach based on an asymmetry-oblivious scheduler. Furthermore, this solution also outperforms an *ad-hoc* asymmetry-aware scheduler furnished with sophisticated scheduling techniques.

Palabras clave— Dense linear algebra, task parallelism, runtime task schedulers, asymmetric architectures.

I. INTRODUCTION

The end of Dennard scaling has promoted heterogeneous systems into a mainstream approach to leverage the steady growth of transistors on chip dictated by Moore's law. ARM@ big.LITTLE™ processors are a particular class of heterogeneous architectures that combine two types of multicore clusters, consisting of a few high performance (big) cores and a collection of low power (LITTLE) cores. These *asymmetric multicore processors* (AMPs) can, in theory, deliver much higher performance for the same power budget. Furthermore, compared with multicore servers equipped with graphics processing units (GPUs), NVIDIA's Tegra chips and AMD's APUs, ARM big.LITTLE processors differ in that the cores in these systems-on-chip (SoC) benefit from sharing the same instruction set architecture and a strongly coupled memory subsystem.

Task parallelism has been reported as an efficient way to tackle the considerable number of cores in current processors. Several efforts aim to ease the devel-

opment and improve the performance of task-parallel programs by embedding task scheduling inside a *runtime* (framework). The benefits of this approach for complex dense linear algebra (DLA) operations have been demonstrated, among others, by OmpSs [1], StarPU [2], PLASMA [3], and libflame [4]. In general, the runtimes underlying these tools decompose DLA routines into a collection of numerical kernels (or tasks), and then take into account the dependencies between the tasks in order to correctly issue their execution to the system cores. The tasks are therefore the “indivisible” scheduling unit while the cores constitute the basic computational resource.

In this paper we introduce an efficient approach to execute task parallel DLA routines on AMPs via conventional asymmetry-oblivious schedulers. Our conceptual solution aggregates the cores of the AMP into a number of *symmetric virtual cores* (VCs) which then become the only basic computational resources that are visible to the runtime scheduler. In addition, an specialized implementation of each type of task, from an asymmetry-aware DLA library, partitions each numerical kernel into a series of finer-grain computations, which are efficiently executed by the asymmetric aggregation of cores of a single VC. Our work thus makes the following specific contributions:

- For the Cholesky factorization, we describe how to leverage the asymmetry-oblivious task-parallel runtime scheduler in OmpSs, in combination with a data-parallel instance of the BLAS-3 (*basic linear algebra subprograms*) from the BLIS library specifically designed for ARM big.LITTLE AMPs [5], [6].
- We provide practical evidence that, compared with an *ad-hoc* asymmetry-conscious scheduler integrated in OmpSs [7], our solution yields higher performance for the multi-threaded execution of the Cholesky factorization on an Exynos 5422 SoC comprising two quad-core clusters with different processing capabilities.

Compared with previous work [5], [7], this paper demonstrates that, for the particular domain of DLA, it is possible to hide the difficulties intrinsic to dealing with an asymmetric architecture (e.g., workload balancing, energy-aware mapping of tasks to cores, and criticality-aware scheduling) inside an asymmetry-aware implementation of the BLAS-3. As a consequence, our solution can refactor any con-

¹Departamento de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, Madrid, e-mail: lcostero@ucm.es.

²Departamento de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, Madrid, e-mail: figual@ucm.es.

³Departamento de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, Madrid, e-mail: katalin@ucm.es.

⁴Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaume I, Castellón, e-mail: quintana@uji.es.

⁵Departamento de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, Madrid, e-mail: ptirado@ucm.es.

ventional (asymmetry-agnostic) scheduler to exploit task parallelism in complex DLA operations.

II. SOFTWARE EXECUTION MODELS FOR ARM BIG.LITTLE SoCs

The target architecture for our design and evaluation is an ODRROID-XU3 board comprising a Samsung Exynos 5422 SoC with an ARM Cortex-A15 quad-core processing cluster (running at 1.3 GHz) and a Cortex-A7 quad-core processing cluster (also operating at 1.3 GHz). Both clusters access a shared DDR3 RAM (2 Gbytes) via 128-bit coherent bus interfaces. Each ARM core (either Cortex-A15 or Cortex-A7) has a 32+32-Kbyte L1 (instruction+data) cache. The four A15 cores share a 2-Mbyte L2 cache, while the four A7 cores share a smaller 512-Kbyte L2 cache.

Modern big.LITTLE SoCs, such as the Exynos 5422, offer a number of software execution models with support from the operating system (OS):

1. *Cluster Switching Mode (CSM)*: The processor is logically divided into two clusters, one containing the big cores and the other with the LITTLE cores, but only one cluster is usable at any given time. The OS transparently activates/deactivates the clusters depending on the workload in order to balance performance and energy efficiency.
2. *CPU migration (CPUM)*: The physical cores are grouped into pairs, each consisting of a fast core and a slow core, building VCs to which the OS maps threads. At any given moment, only one physical core is active per VC, depending on the requirements of the workload. In big.LITTLE specifications where the number of fast and slow cores do not match, the VC can be assembled from a different number of cores of each type. The *In-Kernel Switcher (IKS)* is Linaro's solution for this model.
3. *Global Task Scheduling (GTS)*. This is the most flexible model. All 8 cores are available for thread scheduling, and the OS maps the threads to any of them depending on the specific nature of the workload and core availability. ARM's implementation of GTS is referred to as big.LITTLE MP.

Figure 1 offers an schematic view of these three execution models for modern big.LITTLE architectures. GTS is the most flexible solution, allowing the OS scheduler to map threads to any available core or group of cores. GTS exposes the complete pool of 8 (fast and slow) cores in the Exynos 5422 SoC to the OS. This allows a straight-forward port of existing threaded application, including runtime task schedulers, to exploit all the computational resources in this AMP, provided the multi-threading technology underlying the software is based on conventional tools such as, e.g., POSIX threads or OpenMP. Attaining high performance in asymmetric architectures, even with a GTS configuration, is not as triv-

ial, and is one of the goals of this paper.

Alternatively, CPUM proposes a pseudo-symmetric view of the Exynos 5422, transforming this 8-core asymmetric SoC into 4 symmetric multi-core processors (SMPs), which are logically exposed to the OS scheduler. (In fact, as this model only allows one active core per VC, but the type of the specific core that is in operation can differ from one VC to another, the CPUM is still asymmetric.)

In practice, runtime task schedulers can mimic any of these OS operation modes. A straight-forward model is simply obtained by following the principles governing GTS to map ready tasks to any available core. With this solution, load unbalance can be tackled via *ad-hoc* (i.e., asymmetry-aware) scheduling policies embedded into the runtime that map tasks to the most "appropriate" resource.

III. PARALLEL EXECUTION OF DLA OPERATIONS ON MULTI-THREADED ARCHITECTURES

In this section we briefly review several software efforts, in the form of task-parallel runtimes and libraries, that were specifically designed for DLA, or have been successfully applied in this domain, *when the target is (an heterogeneous system or) an AMP*.

A. Runtime task scheduling of complex DLA operations

A.1 Task scheduling for the Cholesky factorization

We start by describing how to extract task parallelism during the execution of a DLA operation, using the Cholesky factorization as a workhorse example. This particular operation, which is representative of several other factorizations for the solution of linear systems, decomposes an $n \times n$ symmetric positive definite matrix A into the product $A = U^T U$, where the $n \times n$ Cholesky factor U is upper triangular [8].

Listing 2 displays a simplified C code for the factorization of an $n \times n$ matrix A , stored as $s \times s$ (data) sub-matrices of dimension $b \times b$ each, that overwrites the upper triangular part of A with the entries of U . This blocked routine decomposes the operation into a collection of building *kernels*: `po_cholesky` (Cholesky factorization), `tr_solve` (triangular solve), `ge_multiply` (matrix multiplication), and `sy_update` (symmetric rank-b update). The order in which these kernels are invoked during the execution of the routine, and the sub-matrices that each kernel read/writes, result in a direct acyclic graph (DAG) that reflects the dependencies between tasks (i.e., instances of the kernels) and, therefore, the task parallelism of the operation. For example, Figure 3 shows the DAG with the tasks (nodes) and data dependencies (arcs) intrinsic to the execution of Listing 2 for a specific matrix dimension.

The DAG associated with an algorithm/routine is a graphical representation of the task parallelism of the corresponding operation, and a runtime system can exploit this information to determine the task schedules that satisfy the DAG dependencies.


```

1 #pragma omp task inout ([b][b]A)
2 void po_cholesky (double *A, int b, int ld)
3 {
4     static int INFO = 0;
5     static const char UP = 'U';
6     dpotrf (&UP, &b, A, &ld, &INFO); // LAPACK
7     Cholesky factorization
8 }
9 #pragma omp task in ([b][b]A) inout ([b][b]B)
10 void tr_solve (double *A, double *B, int b, int
11 ld)
12 {
13     static double DONE = 1.0;
14     static const char LE = 'L', UP = 'U', TR =
15     'T', NU = 'N';
16     dtrsm (&LE, &UP, &TR, &NU, &b, &b,
17     &DONE, A, &ld, B, &ld); // BLAS-3
18     triangular solve
19 }
20 #pragma omp task in ([b][b]A, [b][b]B) inout ([b
21 ][b]C)
22 void ge_multiply (double *A, double *B, double
23 *C, int b, int ld)
24 {
25     static double DONE = 1.0, DMONE = -1.0;
26     static const char TR = 'T', NT = 'N';
27     dgemm (&TR, &NT, &b, &b, &b,
28     &DMONE, A, &ld, B, &ld,
29     &DONE, C, &ld); // BLAS-3
30     matrix multiplication
31 }
32 #pragma omp task in ([b][b]A) inout ([b][b]C)
33 void sy_update (double *A, double *C, int b,
34 int ld)
35 {
36     static double DONE = 1.0, DMONE = -1.0;
37     static const char UP = 'U', TR = 'T';
38     dsyrk (&UP, &TR, &b, &b,
39     &DMONE, A, &ld,
40     &DONE, C, &ld); // BLAS-3
41     symmetric rank-b update
42 }

```

Fig. 4. Labeled tasks involved in the blocked Cholesky factorization.

bottom-level longest-path priorities, keeps track of the criticality of the individual tasks, and leverages this information, at execution time, to assign a ready task to either a critical or a non-critical queue. In this solution, tasks enqueued in the critical queue can only be executed by the fast cores. In addition, the enhanced scheduler integrates uni- or bi-directional work stealing between fast and slow cores. According to the authors, this sophisticated *ad-hoc* scheduling strategy for heterogeneous/asymmetric processors attains remarkable performance improvements in a number of target applications; see [7] for further details.

When applied to a task-parallel DLA routine, the asymmetry-aware scheduler in Botlev-OmpS maps each task to a single (big or LITTLE) core, and simply invokes a sequential DLA library to conduct the actual work. On the other hand, we note that this approach required an important redesign of the underlying scheduling policy (and thus, a considerable programming effort for the runtime developer), in order to exploit the heterogeneous architecture. In particular, detecting the criticality of a task at execution time is a nontrivial question.

B. Data-parallel libraries of BLAS-3 kernels

B.1 Multi-threaded implementation of the BLAS-3

An alternative to the runtime-based (i.e., task-parallel) approach to execute DLA operations on multi-threaded architectures can rely on a library of specialized kernels that statically partitions the work among the computational resources, or leverages a simple schedule mechanism such as those available, e.g., in OpenMP. For DLA operations with few and/or simple data dependencies, as is the case of the BLAS-3, and/or when the number of cores in the target architecture is small, this option can avoid the costly overhead of a sophisticated task scheduler, providing a more efficient solution. Currently this is the preferred option for all high performance implementations of the BLAS for multicore processors, being adopted in both commercial and open source packages such as, e.g., AMD ACML, IBM ESSL, Intel MKL, GotoBLAS, OpenBLAS and BLIS.

BLIS in particular mimics GotoBLAS to implement all BLAS-3 kernels (including the matrix multiplication, GEMM) as three nested loops around two packing routines, which accommodate the data in the higher levels of the cache hierarchy, and a *macro-kernel* in charge of performing the actual computations. Internally, BLIS implements the macro-kernel as two additional loops around a *micro-kernel* that, in turn, boils down to a loop around a symmetric rank-1 update. For the purpose of the following discussion, we will only consider the three outermost loops in the BLIS implementation of GEMM for the multiplication $C := C + A \cdot B$, where A, B, C are respectively $m \times k, k \times n$ and $m \times n$ matrices, stored in arrays A, B and C ; see Listing 5. In the code, mc, nc, kc are cache configuration parameters that need to be adjusted for performance taking into account the latency of the floating-point units, number of vector registers, and size/associativity degree of the cache levels [12].

B.2 Data-parallel libraries for asymmetric architectures

The implementation of GEMM in BLIS has been demonstrated to deliver high performance on a wide range of multicore and many-core SMPs [13], [14]. These studies offered a few relevant insights that guided the parallelization of GEMM (and also other BLAS-3 routines) on the ARM big.LITTLE architecture under the GTS software execution model. Concretely, the architecture-aware multi-threaded parallelization of GEMM in [5] integrates the following three techniques:

- A dynamic 1-D partitioning of the iteration space to distribute the workload in either Loop 1 or Loop 3 of BLIS GEMM between the two clusters.
- A static 1-D partitioning of the iteration space that distributes the workload of one of the loops internal to the macro-kernel between the cores of the same cluster.

```

1 void gemm (double A[m][k], double B[k][n],
2           double C[m][n],
3           int m, int n, int k, int mc, int
4           nc, int kc)
5 {
6     double *Ac = malloc (mc * kc * sizeof (
7         double));
8     *Bc = malloc (kc * nc * sizeof (
9         double));
10    for (int jc = 0; jc < n; jc+=nc) {
11        // Loop 1
12        int jb = min(n-jc+1, nc);
13        for (int pc = 0; pc < k; jc+=kc) {
14            // Loop 2
15            int pb = min(k-pc+1, kc);
16            pack_buffB (B[pc][jc], Bc, kb, nb);
17            // Pack A->Ac
18            for (int ic = 0; ic < m; ic+=mc) {
19                // Loop 3
20                int ib = min(m-ic+1, mc);
21                pack_buffA (A[ic][pc], Ac, mb, kb);
22                // Pack A->Ac
23                gemm_kernel (Ac, Bc, C[ic][jc],
24                    mb, nb, kb, mc, nc, kc);
25            } // Macro-kernel
26        }
27    }
28 }
    
```

Fig. 5. High performance implementation of GEMM in BLIS.

- A modification of the control tree that governs the multi-threaded parallelization of BLIS GEMM in order to accommodate different loop strides for each type of core architecture.

The strategy is general and can be applied to a generic AMP, consisting of any combination of fast/slow cores sharing the main memory, as well as to all other Level-3 BLAS operations.

IV. RETARGETING EXISTING TASK SCHEDULERS TO ASYMMETRIC ARCHITECTURES

In this section, we initially perform an evaluation of the task-parallel Cholesky routine in Listings 2–4, executed on top of the conventional (i.e., default) scheduler in OmpSs linked to a sequential instance of BLIS, on the target Exynos 5422 SoC. The outcome from this study motivates the development effort and experiments presented in the remainder of the paper.

A. Evaluation of conventional runtimes on AMPs

Figure 6 reports the performance, in terms of GFLOPS (billions of flops per second), attained with the conventional OmpSs runtime, when the number of worker threads varies from 1 to 8, and the mapping of worker threads to cores is delegated to the OS. We evaluated a range of block sizes (b in Listing 2), but for simplicity we report only the results obtained with the value b that optimized the GFLOPS rate for each problem dimension. All the experiments hereafter employed IEEE double precision. Furthermore, we ensured that the cores operate at the highest possible frequency by setting the appropriate *cpufreq* governor. The conventional run-

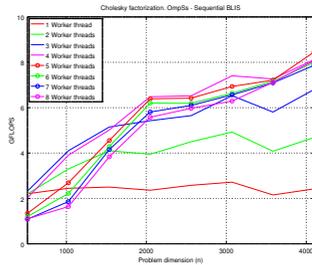


Fig. 6. Performance of the Cholesky factorization using the conventional OmpSs runtime and a sequential implementation of BLIS on the Exynos 5422 SoC.

time instance of OmpSs corresponds to release 15.06 of the Nanos++ runtime task scheduler. For this experiment, it is lined with the “sequential” implementation of BLIS in release 0.1.5. (For the experiments with the multi-threaded/asymmetric version of BLIS in the later sections, we will use specialized versions of the codes in [5] for slow+fast VCs.)

The results in the Figure reveal the increase in performance as the number of worker threads is raised from 1 to 4, which the OS maps to the (big) Cortex-A15 cores. However, when the number of threads exceeds the amount of fast cores, the OS starts binding the threads to the slower Cortex-A7 cores, and the improvement rate is drastically reduced, in some cases even showing a performance drop. This is due to load imbalance, as tasks of uniform granularity, possibly laying in the critical path, are assigned to slow cores.

An obvious solution to this problem consists in adapting the runtime task scheduler (more specifically, the scheduling policy) to exploit the SoC asymmetry [7]. Nevertheless, we part ways with this solution, exploring an architecture-aware alternative that leverages a(ny) conventional runtime task scheduler combined with an underlying asymmetric library.

B. Combining conventional runtimes with asymmetric libraries

B.1 General view

Our proposal operates under the GTS model but is inspired in CPUM. Concretely, our task scheduler regards the computational resources as four *truly* symmetric VCs, each composed of a fast and a slow core. For this purpose, unlike CPUM, *both* physical cores within each VC remain active and collaborate to execute a given task. Furthermore, our approach exploits concurrency at two levels: *task-level parallelism* is extracted by the runtime in order to schedule tasks to the four symmetric VCs; and each task is internally divided to expose *data-level parallelism*, distributing its workload between the two asymmetric physical cores within the VC in charge of its ex-

ecution.

Our solution thus only requires a conventional (and thus asymmetry-agnostic) runtime task scheduler, e.g. the conventional version of OmpSs, where instead of spawning one worker thread per core in the system, we adhere to the CPUM model, creating only one worker thread per VC. Internally, whenever a ready task is selected to be executed by a worker thread, the corresponding routine from BLIS internally spawns two threads, binds them to the appropriate pair of A15+A7 cores, and asymmetrically divides the work between the fast and the slow physical cores in the VC. Following this idea, the architecture exposed to the runtime is *symmetric*, and the kernels in the BLIS library configure a “black box” that abstracts the architecture asymmetry from the runtime scheduler.

In summary, in a conventional setup, the core is the basic computational resource for the task scheduler, and the “sequential” tasks are the minimum work unit to be assigned to these resources. Compared with this, in our approach the VC is the smallest (basic) computational resource from the point of view of the scheduler, while tasks are further divided into smaller units, and executed in parallel by the physical cores inside the VCs.

B.2 Comparison with other approaches

Our approach features a number of advantages for the developer:

- The runtime is not aware of the architecture asymmetry, and thus a conventional task scheduler will work transparently with no special modifications.
- Any existing scheduling policy (e.g. cache-aware mapping or work stealing) in an asymmetry-agnostic runtime, or any enhancement technique, will directly impact the performance attained on an AMP.
- Any improvement in the asymmetry-aware BLIS implementation will directly impact the performance on an AMP. This applies to different ratios of big/LITTLE cores within a VC, operating frequency, or even to the introduction further levels of asymmetry (e.g. cores with a capacity between fast and slow).

Obviously, there is also a drawback in our proposal as a tuned asymmetry-aware DLA library must exist in order to reuse conventional runtimes. In the scope of DLA, this drawback is easily tackled with BLIS. We recognize that, in more general domains, an ad-hoc implementation of the application’s fundamental kernels becomes mandatory in order to fully exploit the underlying architecture.

V. EXPERIMENTAL RESULTS

A. Performance evaluation of the asymmetric BLIS

As mentioned earlier, at execution time, OmpSs decomposes the routine for the Cholesky factorization into a collection of tasks that operate on sub-

TABLE I
 OPTIMAL BLOCK SIZES FOR THE CHOLESKY FACTORIZATION
 USING THE CONVENTIONAL OMPSS RUNTIME AND A
 SEQUENTIAL IMPLEMENTATION OF BLIS ON THE EXYNOS 5422
 SoC.

		Problem dimension (n)									
		512	1,024	1,536	2,048	2,560	3,072	3,584	4,096	5,120	6,144
1 WT	192	384	320	448	448	448	384	320	448	448	
2 WT	192	192	320	192	448	448	448	384	320	448	448
3 WT	128	192	320	192	384	448	320	320	448	448	
4 WT	128	128	192	192	192	320	320	320	448	448	

matrices (blocks) with a granularity dictated by the block size b (see Listing 2). These tasks typically perform invocations to a fundamental kernel of the BLAS-3, in our case provided by BLIS, or LAPACK (see Listing 4).

The first step in our evaluation provides a realistic estimation of the potential performance benefits of our approach (if any) on the target Exynos 5422 SoC. A critical factor from this perspective is the range of block sizes, say b^{opt} , that are optimal for the conventional OmpSs runtime. In particular, the efficiency of our hybrid task/data-parallel approach is strongly determined by the performance attained with the asymmetric BLIS implementation when compared against that of its sequential counterpart, for problem dimensions n that are in the order of b^{opt} .

Table I reports the optimal block sizes b^{opt} for the Cholesky factorization, with problems of increasing matrix dimension, using the conventional OmpSs runtime linked with the sequential BLIS, and 1 to 4 worker threads. Note that, except for smallest problems, the observed optimal block sizes are between 192 and 448. These dimensions offer a fair compromise, exposing enough task-level parallelism while delivering high “sequential” performance for the execution of each individual task via the sequential implementation of BLIS.

The key insight to take away from this experiments is that, in order to extract high performance from a combination of the conventional OmpSs runtime task scheduler with a multi-threaded asymmetric version of BLIS, the kernels in this instance of the asymmetric library must outperform their sequential counterparts, for matrix dimensions in the order of the block sizes in Table I. Figure 7 shows the performance attained with the three BLAS-3 tasks involved in the Cholesky factorization (GEMM, SYRK and TRSM) for our range of dimensions of interest. There, the multi-threaded asymmetry-aware kernels run concurrently on one Cortex-A15 plus one Cortex-A7 core, while the sequential kernels operate exclusively on a single Cortex-A15 core. In general, the three BLAS-3 routines exhibit a similar trend: the kernels from the sequential BLIS outperform their asymmetric counterparts for small problems (up to approximately $m, n, k = 128$), but from that dimension, the use of the slow core starts paying off. The interesting

aspect here is that the cross-over threshold between both performance curves is in the range, (usually at an early point,) of b^{OPT} ; see Table I. This implies that the asymmetric BLIS can potentially improve the performance of the overall computation. Moreover, the gap in performance grows with the problem size, stabilizing at problem sizes around $m, n, k \approx 400$. Given that this value is in the range of the optimal block size for the task-parallel Cholesky implementation, we can expect a performance increment in the order of 0.3–0.5 GFLOPS per added slow core, mimicking the behavior of the underlying BLIS.

B. Integration of the asymmetric BLIS in a conventional task scheduler

In order to analyze the actual benefits of our proposal, we next evaluate the conventional OmpSs task scheduler linked with either the sequential implementation of BLIS or its multi-threaded asymmetry-aware version. Hereafter, the BLIS kernels from first configuration always run using one Cortex-A15 core while, in the second case, they exploit one Cortex-A15 plus one Cortex-A7 core. Figure 8 reports the performance of both setups, using an increasing number of worker threads from 1 to 4. For simplicity, we only report the results obtained with the optimal block size. In all cases, the solution based on the multi-threaded asymmetric library outperforms the sequential implementation for relatively large matrices (usually for dimensions $n > 2,048$) while, for smaller problems, the GFLOPS rates are similar. The reason for this behavior can be derived from the optimal block sizes reported in Table I and the performance of BLIS reported in Figure 7: for that range of problem dimensions, the optimal block size is significantly smaller, and both BLIS implementations attain similar performance rates.

The quantitative difference in performance between both approaches is reported in Tables II and III. The first table illustrates the raw (i.e., absolute) gap, while the second one shows the difference per Cortex-A7 core introduced in the experiment. Let us consider, for example, the problem size $n = 6,144$. In that case, the performance roughly improves by 0.975 GFLOPS when the 4 slow cores are added to help the base 4 Cortex-A15 cores. This translates into a performance raise of 0.243 GFLOPS per slow core, which is slightly under the improvement that could be expected from the experiments in the previous section. Note, however, that the performance per Cortex-A7 core is reduced from 0.340 GFLOPS, when adding just one core, to 0.243 GFLOPS, when using all four slow cores.

C. Performance comparison versus asymmetry-aware task scheduler

Our last round of experiments exposes the performance advantages of different task-parallel executions of the Cholesky factorization via OmpSs. Concretely, we consider (1) the conventional task scheduler linked with the sequential BLIS (“OmpSs - Seq.

TABLE II
 ABSOLUTE PERFORMANCE IMPROVEMENT (IN GFLOPS) FOR THE CHOLESKY FACTORIZATION USING THE CONVENTIONAL OMPSS RUNTIME LINKED WITH THE MULTI-THREADED/ASYMMETRIC BLIS WITH RESPECT TO THE SAME RUNTIME LINKED WITH THE SEQUENTIAL BLIS IN THE EXYNOS 5422 SoC.

	Problem dimension (n)								
	512	1,024	2,048	2,560	3,072	4,096	4,608	5,120	6,144
1 wt	-0.143	0.061	0.218	0.289	0.326	0.267	0.259	0.313	0.340
2 wt	-0.116	-0.109	0.213	0.469	0.573	0.495	0.454	0.568	0.617
3 wt	-0.308	-0.233	-0.020	0.432	0.720	0.614	0.603	0.800	0.866
4 wt	-0.421	-0.440	-0.274	0.204	0.227	0.614	0.506	0.769	0.975

TABLE III
 PERFORMANCE IMPROVEMENT PER SLOW CORE (IN GFLOPS) FOR THE CHOLESKY FACTORIZATION USING THE CONVENTIONAL OMPSS RUNTIME LINKED WITH THE MULTI-THREADED/ASYMMETRIC BLIS WITH RESPECT TO THE SAME RUNTIME LINKED WITH THE SEQUENTIAL BLIS IN THE EXYNOS 5422 SoC.

	Problem dimension (n)								
	512	1,024	2,048	2,560	3,072	4,096	4,608	5,120	6,144
1 wt	-0.143	0.061	0.218	0.289	0.326	0.267	0.259	0.313	0.340
2 wt	-0.058	-0.054	0.106	0.234	0.286	0.247	0.227	0.284	0.308
3 wt	-0.102	-0.077	-0.006	0.144	0.240	0.204	0.201	0.266	0.288
4 wt	-0.105	-0.110	-0.068	0.051	0.056	0.153	0.126	0.192	0.243

BLIS”); (2) the conventional task scheduler linked with our multi-threaded asymmetric BLIS that views the SoC as four symmetric *virtual cores* (“OmpSs - Asym. BLIS”); and (3) the criticality-aware task scheduler in Botlev-OmpSs linked with the sequential BLIS (“Botlev-OmpS - Seq. BLIS”)—in this case, we use all four Cortex-A15 cores and evaluate the impact of adding an increasing number of Cortex-A7 cores, from 1 to 4, for Botlev-OmpSs.

Figure 9 shows the performance attained by the alternatives (1)–(3) on the Exynos 5422 SoC. The results can be divided into three groups along the problem dimension:

- For small matrices ($n = 512, 1,024$), the conventional runtime using exclusively the four big cores (that is, linked with a sequential BLIS library for task execution) attains the best results in terms of performance. This was expected and was already observed in Figure 8; the main reason is the small optimal block size, enforced by the reduced problem size, that is necessary in order to expose enough task-level parallelism. This invalidates the use of our asymmetric BLIS implementation in combination with a runtime due to the low performance for very small matrices; see Figure 7. We note that the ad-hoc Botlev-OmpSs does not attain remarkable performances either for this dimension range, regardless the amount of Cortex-A7 cores used.
- For medium-sized matrices ($n = 2,048, 4,096$), the gap in performance between the different

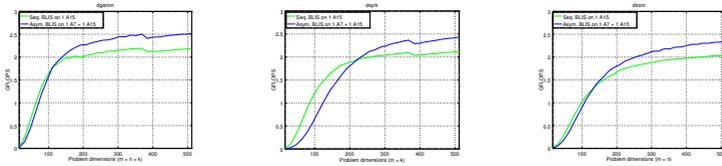


Fig. 7. Performance of the BLAS-3 kernels in the sequential and the multi-threaded/asymmetric implementations of BLIS, using respectively one Cortex-A15 core and one Cortex-A15 plus one Cortex-A7 core of the Exynos 5422 SoC.

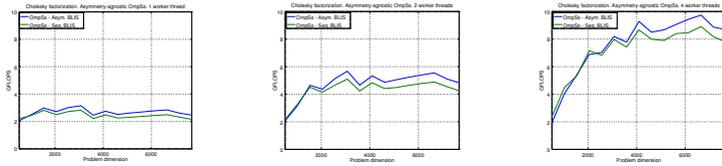


Fig. 8. Performance of the Cholesky factorization using the conventional OmpSs runtime linked with either the sequential or the multi-threaded/asymmetric implementations of BLIS in the Exynos 5422 SoC.

approaches is reduced. The variant that relies on OmpSs plus the asymmetric BLIS implementation commences to outperform the alternative implementations for $n=4,096$ by a short margin. For this problem range, Botlev-OmpSs is competitive, and also outperforms the conventional setup.

- For large matrices ($n = 6,144$) this trend is consolidated, and both runtime-based asymmetry-aware approaches deliver remarkable performance gains with respect to the conventional setup. Comparing both runtime-based asymmetry-aware solutions, our mechanism attains better performance rate, even when considering the usage of all available cores for the Botlev-OmpSs runtime version.

To summarize, our proposal to exploit asymmetry enhances the portability and programmability by avoiding a revamp of the runtime task scheduler for AMPs. In addition, our approach renders performance gains which are, for all problem cases, comparable with those of ad-hoc asymmetry-conscious schedulers; for medium to large matrices, it clearly outperforms the efficiency attained with a conventional asymmetry-oblivious scheduler.

D. Extended performance analysis

We next provide further details on the performance behavior of each one of the aforementioned runtime configurations. The following execution traces correspond to the Cholesky factorization of a single problem with matrix dimension $n = 6,144$ and block size $b = 448$.

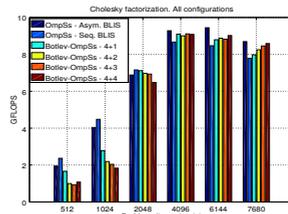


Fig. 9. Performance (in GFLOPS) for the Cholesky factorization using the conventional OmpSs runtime linked with either the sequential BLIS or the multi-threaded/asymmetric BLIS, and the *ad-hoc* asymmetry-aware version of the OmpSs runtime (Botlev-OmpSs) linked with the sequential BLIS in the Exynos 5422 SoC. The labels of the form “4-x” refer to an execution with 4 Cortex-A15 cores and x Cortex-A7 cores.

D.1 General task execution overview

Figure 10 reports complete execution traces for each runtime configuration. At a glance, a number of coarse remarks can be extracted from the trace:

- From the perspective of total execution time (i.e., *time-to-solution*), the conventional OmpSs runtime combined with the asymmetric BLIS implementation attains the best results, followed by the Botlev-OmpSs runtime configuration. It is worth pointing out that an asymmetry-oblivious runtime which spawns 8 worker threads, with no further considerations, yields the worst performance by far. In this case, the load imbalance and long idle periods, especially as the amount of concurrency

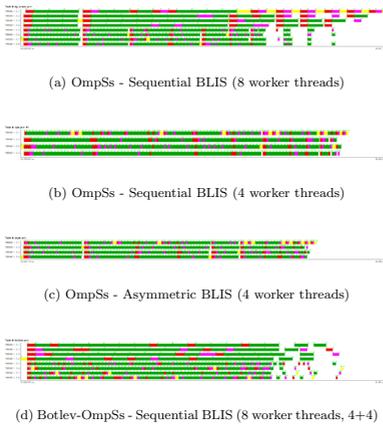


Fig. 10. Execution traces of the three runtime configurations for the Cholesky factorization ($n = 6,144$, $b = 448$). The timeline in each row collects the tasks executed by a single worker thread. Tasks are colored following the convention in Figure 3; phases colored in white between tasks represent idle times. The green flags mark task initialization/completion.

decreases in the final part of the trace, entail a huge performance penalty.

- The flag marks indicating task initialization/completion reveal that the asymmetric BLIS implementation (which employs the combined resources from a VC) requires less time per task than the two alternatives based on a sequential BLIS. An effect to note specifically in the Botlev-OmpSs configuration is the difference in performance between tasks of the same type, when executed by a big core (worker threads 5 to 8) or a LITTLE one (worker threads 1 to 4).
- The Botlev-OmpSs task scheduler embeds a (complex) scheduling strategy that includes priorities, advancing the execution of tasks in the critical path and assigning them to fast cores (see, for example, the tasks for the factorization of diagonal blocks, colored in yellow). This yields an execution timeline that is more compact during the first stages of the parallel execution, at the cost of longer idle times when the degree of concurrency decreases (last iterations of the factorization).

D.2 Task duration

Table IV reports the average execution time per type of task for each worker thread. These results show that the execution time per individual type of task is considerably shorter for our multithreaded/asymmetric BLIS implementation than for the alternatives based on a sequential version of BLIS. The

only exception is the factorization of the diagonal block (`dpotrf`) as this is an LAPACK-level routine, and therefore it is not available in BLIS. Inspecting the task execution time of the Botlev-OmpSs configuration, we observe a remarkable difference depending on the type of core tasks are mapped to. For example, the average execution times for `dgemm` range from more than 400 ms on a LITTLE core, to roughly 90 ms on a big core. This behavior is reproduced for all types of tasks.

From the execution traces, we can observe that the Botlev-OmpSs alternative suffers a remarkable performance penalty due to the existence of idle periods in the final part of the factorization, when the concurrency in the factorization is greatly diminished. This problem is not present in the conventional scheduling policies. In the first stages of the factorization, however, the use of a priority-aware policy for the Botlev-OmpSs scheduler effectively reduces idle times. Table V reports the percentage of time each worker thread is in **running** or **idle** state. In general, the relative amount of time spent in idle state is much higher for Botlev-OmpSs than for the conventional implementations (17% vs. 5%, respectively). Note also the remarkable difference in the percentage of idle time between the big and LITTLE cores (20% and 13%, respectively), which drives to the conclusion that the fast cores stall waiting for completion of tasks executed on the LITTLE cores. This fact can be confirmed in the final stages of the Botlev-OmpSs trace.

VI. CONCLUSIONS

In this paper, we have addressed the problem of refactoring existing runtime task schedulers to exploit task-level parallelism in novel AMPs, focusing on ARM big.LITTLE systems-on-chip. We have demonstrated that, for the specific domain of DLA, an approach that delegates the burden of dealing with asymmetry to the library (in our case, using an asymmetry-aware BLIS implementation), does not require any revamp of existing task schedulers, and can deliver high performance. This proposal paves the road towards reusing conventional runtime schedulers for SMPs (and all the associated improvement techniques developed through the past few years), as the runtime only has a symmetric view of the hardware. Our experiments reveal that this solution is competitive and even improves the results obtained with an asymmetry-aware scheduler for DLA operations.

ACKNOWLEDGMENTS

The researchers from Universidad Complutense de Madrid were supported by project TIN2015-65277-R (MINECO/FEDER). Enrique S. Quintana-Ortí was supported by projects CICYT TIN2011-23283 and TIN2014-53495-R as well as the EU project FP7 318793 “EXA2GREEN”.

TABLE IV

AVERAGE TIME (IN MS) PER TASK AND WORKER THREAD IN THE CHOLESKY FACT. ($n = 6,144$, $b = 448$), FOR THE RUNTIME CONFIGURATIONS.

	OmpSs - Seq. BLIS (4 worker threads)				OmpSs - Asym. BLIS (4 worker threads)				Botlev-OmpSs - Seq. BLIS (8 worker threads, 4+4)			
	dgemm	dtram	dayrk	dpotrf	dgemm	dtram	dayrk	dpotrf	dgemm	dtram	dayrk	dpotrf
WT 0	89.62	48.12	47.14	101.77	79.82	42.77	44.42	105.77	406.25	216.70	–	–
WT 1	88.96	48.10	47.14	–	78.65	42.97	44.56	76.35	408.90	207.41	212.55	–
WT 2	89.02	48.36	47.18	87.22	79.14	43.14	44.60	85.98	415.31	230.07	212.56	–
WT 3	90.11	48.51	47.42	–	79.28	43.10	44.59	67.73	410.84	216.95	216.82	137.65
WT 4	–	–	–	–	–	–	–	–	90.97	48.97	48.36	–
WT 5	–	–	–	–	–	–	–	–	90.61	48.86	48.16	90.78
WT 6	–	–	–	–	–	–	–	–	91.28	49.43	47.97	89.58
WT 7	–	–	–	–	–	–	–	–	91.60	49.49	48.62	95.43
Avg.	89.43	48.27	47.22	94.49	79.22	42.99	44.54	83.96	250.72	133.49	119.29	103.36

TABLE V

PERCENTAGE OF TIME PER WORKER THREAD IN IDLE/RUNNING STATE FOR THE RUNTIME CONFIGURATIONS AND THE CHOLESKY FACTORIZATION ($n = 6,144$, $b = 448$). WT 0 IS THE MASTER THREAD, AND THIS IS NEVER IDLE; FOR THIS THREAD, THE DIFFERENCE TO 100% IS DEVOTED TO **synchronization, scheduling and thread creation**. FOR OTHER THREADS, THIS AMOUNT OF TIME IS DUE TO **runtime overhead**.

	OmpSs-Seq. BLIS (4 worker threads)		OmpSs-Asym. BLIS (4 worker threads)		Botlev-OmpSs-Seq. BLIS (8 worker threads, 4+4)	
	idle	running	idle	running	idle	running
WT 0	–	98.41	–	97.85	–	86.53
WT 1	5.50	94.22	5.51	94.29	13.63	86.28
WT 2	3.14	96.67	5.27	94.53	13.94	85.98
WT 3	5.77	94.07	5.17	94.62	13.43	86.47
WT 4	–	–	–	–	19.26	80.51
WT 5	–	–	–	–	21.12	78.69
WT 6	–	–	–	–	20.84	78.97
WT 7	–	–	–	–	20.09	79.70
Avg.	4.84	95.89	5.32	94.90	17.47	82.89

REFERENCIAS

[1] OmpSs project home page, <http://pm.bsc.es/omps>, last visit: July 2015.

[2] StarPU project home page, <http://runtime.bordeaux.inria.fr/StarPU/>, last visit: July 2015.

[3] PLASMA project home page, <http://ic1.cs.utk.edu/plasma/>, last visit: July 2015.

[4] FLAME project home page, <http://www.cs.utexas.edu/users/flame/>, last visit: July 2015.

[5] S. Catalán, F. D. Igual, R. Mayo, R. Rodríguez-Sánchez, E. S. Quintana-Ortí, Architecture-aware configuration and scheduling of matrix multiplication on asymmetric multicore processors, ArXiv e-prints 1506.08988. URL <http://arxiv.org/abs/1506.08988>

[6] F. G. Van Zee, R. A. van de Geijn, BLIS: A framework for rapidly instantiating BLAS functionality, ACM Transactions on Mathematical Software 41 (3) (2015) 14:1–14:33. URL <http://doi.acm.org/10.1145/2764454>

[7] K. Chronaki, A. Rico, R. M. Badia, E. Ayguadé, J. Labarta, M. Valero, Criticality-aware dynamic task scheduling for heterogeneous architectures, in: Proceedings of ICS'15, 2015.

[8] G. H. Golub, C. F. V. Loan, Matrix Computations, 3rd Edition, The Johns Hopkins University Press, Baltimore, 1996.

[9] G. Quintana-Ortí, E. S. Quintana-Ortí, R. A. van de Geijn, F. G. Van Zee, E. Chan, Programming matrix algorithms-by-blocks for thread-level parallelism, ACM Transactions on Mathematical Software 36 (3) (2009) 14:1–14:26.

[10] R. M. Badia, J. R. Herrero, J. Labarta, J. M. Pérez, E. S. Quintana-Ortí, G. Quintana-Ortí, Parallelizing dense and banded linear algebra libraries using SMPSS, Concurrency and Computation: Practice and Experience 21 (18) (2009) 2438–2456.

[11] C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier, StarPU: A unified platform for task scheduling on heterogeneous multicore architectures, Concurrency and Computation: Practice and Experience 23 (2) (2011) 187–198.

[12] T. M. Low, F. D. Igual, T. M. Smith, E. S. Quintana-Ortí, Analytical modeling is enough for high performance BLIS, ACM Trans. Math. Soft. In review. Available at <http://www.cs.utexas.edu/users/flame>.

[13] F. G. Van Zee, T. M. Smith, B. Marker, T. M. Low, R. A. van de Geijn, F. D. Igual, M. Smelyanskiy, X. Zhang, M. Kistler, V. Austel, J. Gunnels, L. Kilgough, The BLIS framework: Experiments in portability, ACM Trans. Math. Soft. Accepted. Available at <http://www.cs.utexas.edu/users/flame>.

[14] T. M. Smith, R. van de Geijn, M. Smelyanskiy, J. R. Hammond, F. G. Van Zee, Anatomy of high-performance many-threaded matrix multiplication, in: Proc. IEEE 28th Int. Symp. on Parallel and Distributed Processing, IPDPS'14, 2014, pp. 1049–1059.

Evaluación del Consumo Energético de la Memoria Transaccional Software en Procesadores Heterogéneos

Emilio Villegas, Alejandro Villegas, Ángeles Navarro, Rafael Asenjo, Oscar Plata¹

Resumen—Actualmente existe una enorme cantidad de dispositivos y sistemas, como ordenadores portátiles y teléfonos móviles, que dependen de una batería para su funcionamiento. Como consecuencia, el hardware que incorporan debe ser energéticamente eficiente. La industria, para soportar este mercado, está desarrollando procesadores con el objetivo de reducir su consumo energético. Por ejemplo, ARM propone la arquitectura big.LITTLE como un procesador multi-núcleo heterogéneo: unos núcleos más rápidos para aplicaciones orientadas al rendimiento, y otros más lentos orientados a la eficiencia energética. Puesto que todos los núcleos acceden a la misma memoria física, las aplicaciones multi-hilo deben recurrir a algún tipo de sincronización para coordinar el acceso a los datos compartidos. La memoria transaccional (TM) es una solución optimista para ofrecer sincronización de hilos concurrentes en memoria compartida. En TM se permite el acceso en paralelo a los datos compartidos y, mediante un mecanismo de detección de conflictos, se puede garantizar la exclusión mutua.

Para beneficiarse de las ventajas que ofrece TM, así como de las características de los procesadores heterogéneos de bajo consumo, es necesario que las soluciones de TM tengan en cuenta los requisitos energéticos y de rendimiento de las aplicaciones en consonancia con lo que ofrece el procesador. Como paso inicial, hay que comprender el rendimiento y consumo energético de las soluciones TM actuales. Para ello, hemos realizado una evaluación de consumo y rendimiento de una librería de TM software, TinySTM, sobre un procesador del tipo big.LITTLE. Los resultados revelan una buena escalabilidad en los núcleos de bajo consumo para la mayoría de las aplicaciones evaluadas. Sin embargo, la aplicación con mayores requerimientos de cómputo resulta ser energéticamente más eficiente en los núcleos orientados al rendimiento, a pesar de su mayor consumo.

Palabras clave— Memoria transaccional software, Procesadores heterogéneos, Eficiencia energética.

I. INTRODUCCIÓN

Hoy día podemos encontrar dispositivos móviles y empotrados en una gran variedad de entornos. Ejemplos son los teléfonos móviles, ordenadores portátiles, controladores e infinidad de aplicaciones del *Internet of Things*. Todos estos dispositivos deben ser energéticamente eficientes. En primer lugar, suelen depender de una fuente de alimentación integrada (por ejemplo, una batería). En segundo lugar, el entorno en el que se sitúan puede tener restricciones en cuanto a su consumo y temperatura. Por ejemplo, la energía disipada (y, por tanto, la temperatura) de un teléfono móvil debe ser baja para que sea cómodo de utilizar por parte del usuario.

Conforme el mercado de este tipo de dispositivos crece, la industria se está centrando en diseñar procesadores de bajo consumo energéticamente eficientes. Concretamente, ARM desarrolla procesadores que se utilizan en dispositivos en los que debe mantenerse un equilibrio entre consumo de energía, temperatura y rendimiento. Para conseguir este equilibrio, han desarrollado la arquitectura multi-núcleo llamada big.LITTLE [1]. Esta arquitectura incorpora dos conjuntos de núcleos: unos núcleos más rápidos, orientados al rendimiento, y otros núcleos con menor capacidad de cálculo pero energéticamente más eficientes. Estos núcleos están organizados en 2 *clusters* que llamamos *cluster big* y *cluster little*, respectivamente. Los núcleos de ambos clusters implementan el mismo ISA y tienen acceso a la misma memoria física, por lo que una aplicación puede ejecutarse en cualquiera de los clusters indistintamente. Por tanto, las aplicaciones con altos requerimientos de cómputo serán (probablemente) planificadas en el *cluster big*, mientras que aquellas sin estos requerimientos pueden ejecutarse en el *cluster little*.

Las aplicaciones deben ser programadas con un diseño multi-hilo para que aprovechen las capacidades de los procesadores multi-núcleo. Programar este tipo de aplicaciones puede ser un reto para los programadores, especialmente cuando se necesita sincronización entre los hilos de ejecución para acceder a datos compartidos. La porción de código que accede a estos datos, conocida como sección crítica, debe garantizar la exclusión mutua en su acceso por parte de los hilos de ejecución. La exclusión mutua se implementa de forma tradicional utilizando cerrojos. Un cerrojo de grano grueso garantiza que ningún hilo puede acceder a la sección crítica si alguno de ellos ya se encuentra ejecutándola. Normalmente, esto tiene un gran impacto en el rendimiento de la aplicación ya que puede producir una serialización innecesaria si se necesita acceder a la sección crítica frecuentemente pero, dentro de ella, acceden a posiciones de memoria diferentes. Para obtener un mayor rendimiento en estos casos se puede recurrir a una implementación de cerrojos de grano fino. En ella se protegen individualmente las posiciones de memoria (o las variables u objetos), de forma que puede accederse de forma paralela a la sección crítica únicamente si van a modificarse datos disjuntos. Con esta implementación se pretende explotar un mayor paralelismo de la aplicación pero, a cambio, se requiere un esfuerzo de programación mucho mayor. Por ejemplo, com-

¹Universidad de Málaga, Andalucía Tech, Dept. Arquitectura de Computadores, 29071 Málaga. e-mail: emilio.villegas@uma.es, avillegas@uma.es, magonzalez@uma.es, asenjo@uma.es, oplata@uma.es

probar la corrección de esta implementación y que no existan *deadlocks* ni *livelocks* es una tarea difícil.

La Memoria Transaccional (TM) [2] se ha propuesto como una alternativa optimista a los cerrojos para implementar secciones críticas. Con TM, utilizamos transacciones para definir las secciones críticas. Las transacciones pueden ejecutarse en paralelo, pero los accesos a memoria que se realizan dentro de ellas deben ser registrados. Si dos o más transacciones han accedido a la misma posición de memoria y, al menos, un acceso es de escritura se registra un conflicto. En caso de conflicto, sólo una de las transacciones tiene permitido continuar o terminar su ejecución. El resto debe deshacer (o descartar) los cambios especulativos en memoria y reiniciar su ejecución. La idea es que las transacciones proporcionen al programador una interfaz similar al uso de cerrojos de grano grueso, pero un rendimiento similar a los cerrojos de grano fino. Además, la implementación de TM debe asegurar la corrección (esto es, que no existan *deadlocks* y el progreso esté garantizado). En las últimas décadas se han propuesto varias soluciones TM implementadas en hardware [3] y algunos procesadores multi-núcleo actuales incorporan este soporte [4]. Además de las soluciones hardware, se han propuesto numerosas soluciones software en forma de librerías [5], [6], [7], [8], [9]. Por último, también han aparecido soluciones híbridas que combinan las características de los TM software y hardware [10].

Adaptar las soluciones TM existente a los procesadores multi-núcleo heterogéneos de bajo consumo es un objetivo importante para los próximos años. Existen ya algunos análisis y propuestas de TM que tienen en cuenta el consumo de energía de los procesadores [11], [12], [13], [14], [15], [16], [17]. Sin embargo, estas propuestas no tienen en cuenta los procesadores multi-núcleo heterogéneos y están enfocadas a procesadores homogéneos. Además, ninguna de las propuestas existentes considera la evaluación sobre un sistema real: en todas ellas, tanto hardware como software, se han utilizado simuladores para estimar el consumo energético.

En este artículo evaluamos un sistema TM software (STM) ampliamente utilizado, TinySTM [5], [6], sobre la plataforma Odroid-XU3 [18], que incorpora un procesador tipo *big.LITTLE*. Para ello, hemos utilizado un conjunto de aplicaciones provenientes del STAMP [19] *benchmark suite* y los sensores de energía disponibles en el dispositivo. El objetivo es analizar el comportamiento de la librería sobre este tipo de procesadores y obtener datos e infraestructura sobre la que hacer propuestas futuras de TM con restricciones energéticas.

II. ANTECEDENTES

A. Arquitectura *big.LITTLE* de ARM

La arquitectura heterogénea *big.LITTLE* de ARM está pensada para adaptarse a todo tipo de escenarios, incorporando núcleos de alto rendimiento y otros de bajo consumo. Según sus propios datos [1], esto permite ahorrar hasta un 75 % de energía en

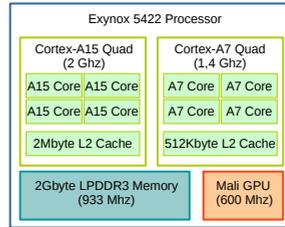


Fig. 1: Diagrama del procesador Exynos 5422 disponible en la plataforma ODRROID-XU3.

escenarios de bajos requerimientos de cómputo e incrementar hasta un 40 % el rendimiento en aplicaciones multi-hilo. En su primera generación, los núcleos de bajo consumo pertenecen a la familia Cortex-A7, mientras que los núcleos de alto rendimiento pertenecen a las familias Cortex-A15 o Cortex-A17. En su segunda generación, los núcleos de bajo consumo son de las familias Cortex-A35 o Cortex-A53, y los de alto rendimiento son de las familias Cortex-A57 o Cortex-A72. Los núcleos de cada tipo están organizados en 2 clusters. Por ejemplo, una configuración típica de la primera generación son 4 núcleos Cortex-A7 (para el cluster *little*) y 4 núcleos Cortex-A15 (para el cluster *big*). Ambos clusters incorporan una jerarquía cache coherente de dos niveles.

B. ODRROID-XU3

El dispositivo que hemos utilizado para la evaluación es ODRROID-XU3 [18]. Este dispositivo incorpora un procesador Samsung Exynos 5422, basado en la arquitectura *big.LITTLE* de ARM de primera generación. El procesador tiene 4 núcleos Cortex-A7 en el cluster *little* y 4 núcleos Cortex-A15 en el cluster *big*, con 512 Kbytes y 2 Mbytes de cache L2, respectivamente. El sistema incorpora una GPU MALI-T628 y 2 Gbytes de memoria DDR3. El sistema operativo instalado sobre el dispositivo es Linux odroid 3.10.59+. Además, se incluyen monitores INA231 de corriente/energía¹ capaces medir el consumo de energía del cluster *big*, cluster *little*, GPU y memoria. Estos sensores de energía son accesibles desde el directorio `/sys` del sistema de ficheros. Los datos proporcionados no son acumulativos, sino instantáneos.

Hemos desarrollado una librería que accede a dichos sensores y proporciona el consumo energético. Para ello, esta librería dedica un hilo de ejecución a leer estos sensores e integrar sus valores utilizando el reloj de tiempo real del sistema. Una vez que este hilo está ejecutando, podemos utilizar las funciones que proporciona para medir la energía consumida por las distintas secciones de nuestro código. La resolución de estas medidas es de 100 milisegundos.

¹<http://www.ti.com/product/ina231>

C. TinySTM

TinySTM [5], [6] es una librería de TM software ampliamente utilizada (disponible en <http://tware.org/tinystm>). Su implementación está basada en *timestamps* y trabaja a nivel de palabra de memoria. Para la gestión de transacciones, TinySTM incorpora varios modelos. En el modelo *write-back*, los cambios especulativos en memoria se almacenan en un *buffer* hasta el final de la transacción, momento en el que se hacen definitivos si no existen conflictos. En el modelo *write-through*, los cambios especulativos se guardan directamente en memoria y los valores antiguos (que deben restaurarse en caso de conflicto) se guardan en un *log*. En el modelo *commit-time locking*, se utilizan cerrojos durante la finalización (*commit*) de la transacción para proteger las posiciones que están siendo actualizadas. El caso contrario (esto es, los cerrojos se deben obtener en cada acceso a memoria en lugar de al final de la transacción) se llama *encounter-time locking*. Durante nuestros experimentos hemos utilizado la configuración por defecto de TinySTM, que emplea las opciones *write-back* y *encounter-time locking*.

Por último, hemos observado que TinySTM utiliza la librería *atomic_ops* para la implementación de algunas de sus funcionalidades. Una versión reducida de esta librería viene incluida en TinySTM pero, sin embargo, no puede ser utilizada en arquitecturas ARM. El motivo es que algunas de sus funciones están especialmente programadas para otro tipo de arquitecturas. Afortunadamente, el sistema operativo Linux odroid 3.10.59+ incorpora esta librería compilada para procesadores ARM.

D. STAMP benchmark suite

El conjunto de aplicaciones STAMP [19] es muy popular a la hora de evaluar distintos sistemas de TM, tanto software como hardware. Incluye, en total, ocho aplicaciones de diferentes dominios: ciencia, ingeniería, seguridad, aprendizaje computacional, etc... Para cada aplicación se pueden definir diversos parámetros de entrada con el objetivo de enfatizar las diferentes características del TM a evaluar. Además, se incluyen unos conjuntos de datos y parámetros de entrada por defecto que facilitan la comparación de distintos TM.

III. EVALUACIÓN DE ENERGÍA Y RENDIMIENTO.

A. Dispositivo

Como dispositivo para nuestras pruebas hemos elegido el ODRROID-XU3. La tabla I resume las características de este dispositivo explicadas anteriormente.

B. Benchmarks

Hemos seleccionado 5 aplicaciones de las disponibles en el STAMP benchmark suite: *intruder*, *kmeans*, *labyrinth*, *scaa2*, and *vacation*. Por defecto, hemos utilizado los parámetros de entrada ++, tal y como se definen en [19]. Las aplicaciones *kmeans*

Característica	Descripción
CPU	Samsung Exynos-5422 : Cortex-A15 y Cortex-A7 big.LITTLE
Memoria principal	2 Gbyte LPDDR3 RAM 933MHz
GPU	Mali-T628 MP6
Almacenamiento	32GB Sandisk iNAND Extreme
Medición de energía	Sensores separados para el cluster big, el cluster little, GPU y memoria.
S.O.	Linux odroid 3.10.59+

TABLA I: Sistema ODRROID-XU3 utilizado durante la evaluación.

y *vacation* disponen, a su vez, de entradas de baja y alta contención. Para nuestros experimentos hemos utilizado las entradas de alta contención. En la tabla II se resumen las principales características de las aplicaciones evaluadas.

De las aplicaciones disponibles, 3 han sido excluidas por diferentes motivos. En *bayes* se obtienen resultados irregulares tras varias ejecuciones del programa, por lo que decidimos excluirla de la evaluación. Este problema ha sido documentado con anterioridad en [20]. La aplicación *genome* presenta problemas de sincronización cuando se utiliza más de un hilo de ejecución para su evaluación. En *yada* se producen errores de falta de memoria para el conjunto de datos de entrada ++. Actualmente estamos investigando estos problemas.

C. Instrumentación del código

Por defecto, el código de STAMP está instrumentado con contadores de tiempo para medir el rendimiento de las distintas aplicaciones. Para nuestros experimentos, hemos desarrollado una librería de instrumentación propia y hemos reemplazado a la incorporada en STAMP. Esta librería proporciona acceso a los sensores de energía mencionados anteriormente: cluster big, cluster little, GPU y memoria. Además, se ha añadido un contador de tiempo de ejecución. Se ha comprobado que los resultados de este último y los proporcionados en la librería que STAMP

Aplicación	Longitud de transacción	Conjuntos de lectura y escritura	Tiempo en transacción
intruder	Corta	Medios	Medio
kmeans	Corta	Pequeños	Bajo
labyrinth	Larga	Grandes	Alto
scaa2	Corta	Pequeños	Bajo
vacation	Media	Medios	Alto

TABLA II: Aplicaciones utilizadas en la evaluación.

incorpora por defecto no tienen una variación significativa. Por último, como el objetivo es comprobar la energía consumida por todo el sistema, nuestros experimentos miden la energía de los cuatro sensores, integramos el consumo instantáneo a lo largo del tiempo de ejecución para cada uno de ellos y retornamos su suma.

D. Resultados experimentales

Durante los experimentos, examinamos tres métricas. En primer lugar, medimos el tiempo de ejecución normalizado al tiempo empleado por un hilo utilizando la librería TinySTM. En segundo lugar, medimos la energía consumida por la ejecución completa de cada aplicación. De nuevo, esta segunda medida está normalizada a la ejecución de un sólo hilo utilizando TinySTM. Por último, calculamos el EDP (*Energy-Delay Product*) con los datos anteriores. El EDP tiene en cuenta tanto el tiempo de ejecución como la energía consumida por la aplicación: valores pequeños muestran una mayor eficiencia. Para cada una de las métricas (tiempo de ejecución, energía consumida y EDP) se han representado las medias geométricas de las cinco aplicaciones evaluadas. En cada experimento se han llevado a cabo diez ejecuciones y se ha calculado el valor promedio para cada una de las métricas. No obstante, los resultados obtenidos han sido consistentes en cada una de las diez ejecuciones. La normalización de cada métrica se ha llevado a cabo utilizando una ejecución de TinySTM con un único hilo. No se ha utilizado para esta comparación una versión secuencial del código ya que la instrumentación es distinta y los resultados pueden no ser comparables con los obtenidos por TinySTM. La implementación de TinySTM está optimizada para otras arquitecturas y únicamente resulta competitiva (en términos de velocidad y consumo energético) para la aplicación labyrinth. Sin embargo, el objetivo es evaluar la escalabilidad, tanto energética como de rendimiento, de TinySTM en una arquitectura heterogénea, pero no su comparación con otras implementaciones de las aplicaciones.

D.1 Resultados de la ejecución

En la tabla III se muestran los tiempos de ejecución y consumo de energía de las aplicaciones utilizando cuatro hilos en los clusters big y little. En los siguientes apartados se representan gráficamente y se comentan estos resultados.

D.2 Análisis del cluster little

La figura 2 muestra los resultados de la evaluación del cluster little. En cuanto al tiempo de ejecución (fig. 2.a), TinySTM muestra una buena escalabilidad en los núcleos Cortex-A7 en todas las aplicaciones. La misma afirmación se aplica al consumo de energía (fig. 2.b) y EDP (fig. 2.c). Por ejemplo, utilizar cuatro hilos de ejecución proporciona una reducción del EDP en torno al 90%, según la media geométrica calculada. Recordar que el consumo de energía (y, por tanto, EDP) se miden para el procesador com-

Aplicac.	T.E. _{A7}	T.E. _{A15}	C.E. _{A7}	C.E. _{A15}
intruder	87.0	127.3	120.9	635.9
kmeans	22.3	37.5	31.9	187.7
labyrinth	91.4	34.5	136.9	295.9
ssca2	32.6	35.6	45.9	171.8
vacation	138.3	271.7	188.2	1317.0

TABLA III: Tiempo de ejecución en segundos (T.E.) y consumo de energía en Julios (C.E.) de las aplicaciones sobre los distintos clusters utilizando cuatro hilos.

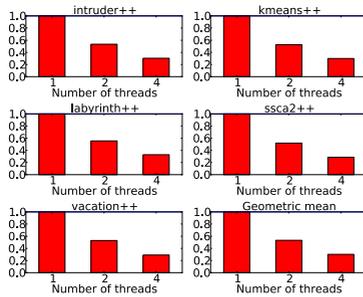
pleto: el hecho de que los procesadores Cortex-A15 no estén siendo utilizados resulta en un consumo de energía reducido.

D.3 Análisis del cluster big

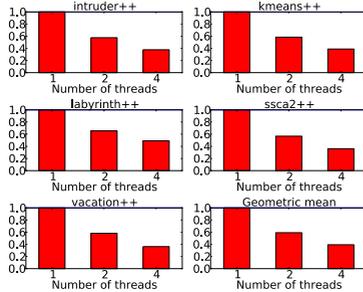
La figura 3 muestra los resultados de la evaluación del cluster big. En este caso, el escenario es diferente si lo comparamos con el cluster little. Los tiempos de ejecución (fig. 3.a) muestran escalabilidad, aunque de manera inferior al cluster little. Además, los núcleos Cortex-A15 no son tan eficientes como los Cortex-A7 en términos de energía (fig. 3.b). Por ejemplo, las aplicaciones intruder, kmeans y vacation muestran un incremento en el consumo energético cuando se activan los cuatro núcleos. Como resultado de este incremento en el consumo, el EDP (fig. 3.c) no resulta óptimo en todas las aplicaciones cuando ejecutan cuatro hilos. Las aplicaciones mencionadas anteriormente (intruder, kmeans y vacation) encuentran un EDP más eficiente utilizando únicamente dos hilos, lo cual aumenta levemente su tiempo de ejecución en comparación con la utilización de cuatro hilos. La media geométrica nos muestra que, en cuanto a tiempo de ejecución, es preferible utilizar cuatro hilos para la mayoría de aplicaciones. Sin embargo, utilizar cuatro hilos resulta en un consumo de energía un 20% superior en comparación con el uso de dos hilos. Estas diferencias se compensan mutuamente en el EDP: no existen diferencias apreciables en el EDP promedio comparando la ejecución utilizando dos hilos o cuatro hilos.

D.4 Evaluación de ambos clusters

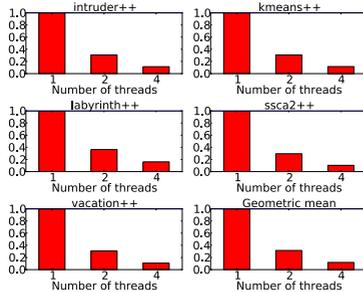
La figura 4 muestra la evaluación de las aplicaciones cuando incrementamos el número de hilos hasta ocho con el objetivo de utilizar ambos clusters. Antes de ejecutar estas aplicaciones, observamos el comportamiento del sistema operativo en cuanto a planificación de hilos. Durante nuestros primeros experimentos hemos comprobado que, utilizando cuatro hilos de ejecución o menos, el sistema operativo los planifica en el cluster big tan pronto como detecta algún tipo de carga computacional. En el momento que se utilizan los ocho hilos de ejecución, el sistema operativo decide hacer uso del cluster little junto al cluster big. Los resultados de tiempos de ejecución (fig. 4.a) muestran que, hasta cuatro hilos, la aceleración es similar al uso del cluster big. Cuando



a. Tiempo de ejecución normalizado a un hilo.



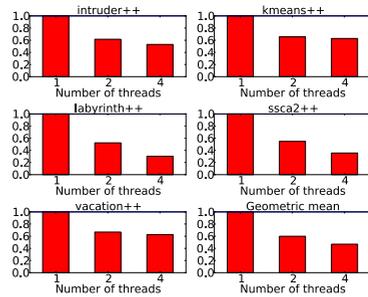
b. Consumo de energía normalizado a un hilo.



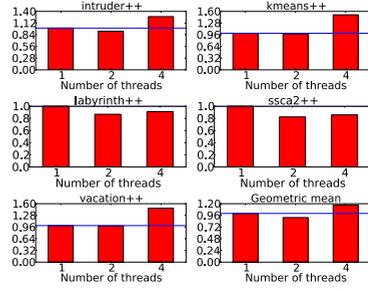
c. EDP normalizado a un hilo.

Fig. 2: Evaluación del cluster little.

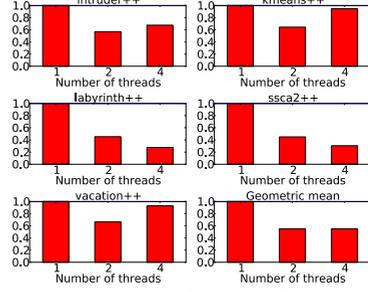
incrementamos el número de hilos hasta ocho podemos observar que los tiempos de ejecución mejoran. Además, el consumo de energía (fig. 4.b) mejora respecto al uso de únicamente el cluster big en todas las aplicaciones salvo kmeans, dónde empeora ligeramente. Sin embargo, este ligero aumento no tiene impacto en el EDP (fig. 4.c), ya que la mejora en los tiempos de ejecución compensa el incremento de consumo de energía. Por tanto, utilizar ambos clusters mejora el EDP en todos los casos.



a. Tiempo de ejecución normalizado a un hilo.



b. Consumo de energía normalizado a un hilo.

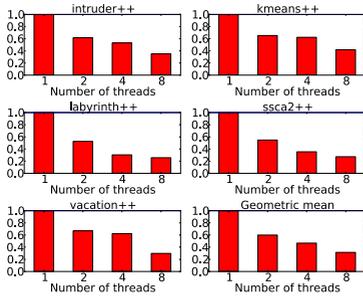


c. EDP normalizado a un hilo.

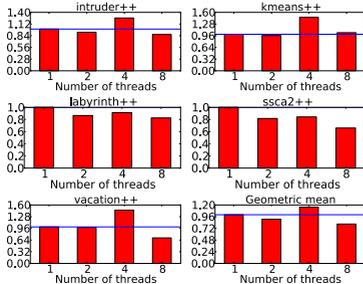
Fig. 3: Evaluación del cluster big.

D.5 Comparación entre los clusters little y big

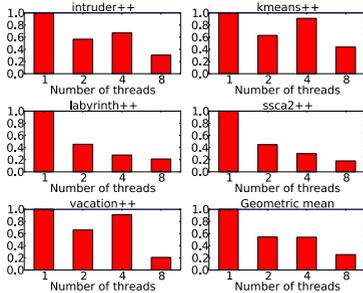
La figura 5 muestra una comparativa entre los clusters little y big. Para el tiempo de ejecución (fig. 5.a) representamos el cociente $TiempoEjec_{AT}/TiempoEjec_{A15}$. Valores mayores que 1 muestran un mejor rendimiento en el cluster big, mientras que los menores de 1 muestran un mejor rendimiento en el cluster little. La ejecución de la aplicación utilizando un único hilo muestra que el cluster big obtiene mejores tiempos gracias a su po-



a. Tiempo de ejecución normalizado a un hilo.



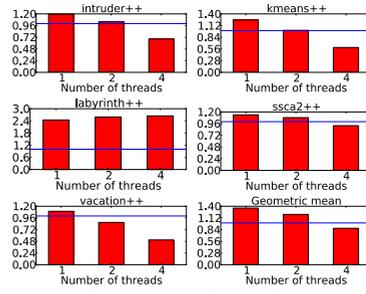
b. Consumo de energía normalizado a un hilo.



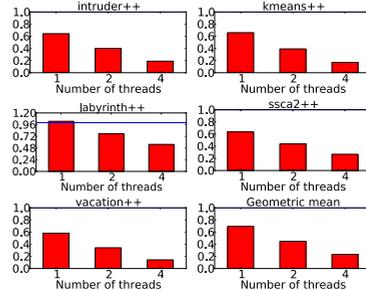
c. EDP normalizado a un hilo.

Fig. 4: Evaluación utilizando ambos clusters.

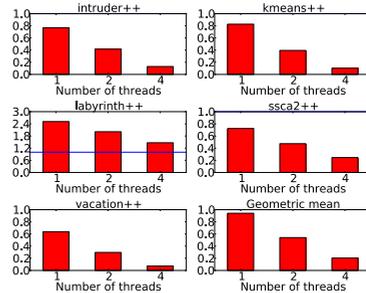
tencia de cálculo. Conforme el número de hilos aumenta, el rendimiento empieza a cambiar a favor del cluster little. Dado que el cluster big es capaz de reinventar las transacciones abortadas más rápidamente, también se produce un aumento significativo de aquellas que vuelven a tener algún conflicto durante dichos reintentos. Concretamente, en kmeans, intruder y scca2 se produce un aumento del 71%, 34% y 56%, respectivamente, en el número de transacciones abortadas cuando las ejecutamos sobre el cluster



a. $TiempoEjec.A7/TiempoEjec.A15$.



b. $ConsumoEnerg.A7/ConsumoEnerg.A15$.



c. $EDPA7/EDPA15$.

Fig. 5: Comparativa entre los clusters little and big.

big en comparación con el cluster little. Tal aumento en el número de reintentos no puede ser compensado por la mayor potencia de cálculo del cluster big, por lo que el cluster little resulta más eficiente. La aplicación labyrinth es una excepción a lo dicho anteriormente: dado que requiere gran potencia de cálculo, es capaz de aprovechar los recursos que el cluster big pone a su disposición. Además, la variación en el número de transacciones abortadas es de apenas un 4%, por lo que no se produce ningún efecto negativo sobre

el tiempo de ejecución. La misma comparativa se realiza para el consumo de energía (fig. 5.b) realizando la operación *ConsumoEnerg.A7/ConsumoEnerg.A15*. En todos los casos, salvo labyrinth con un sólo hilo, el cluster little produce mejores resultados que el cluster big. Para el EDP (fig. 5.c) comprobamos una mayor eficiencia del cluster little salvo en labyrinth, donde el cluster big es más apropiado. Dado que labyrinth presenta transacciones largas que modifican gran cantidad de datos, requiere una gran potencia de cálculo. Los núcleos Cortex-A15 son capaces de proporcionar esta potencia y amortizar el mayor consumo de energía.

IV. TRABAJO RELACIONADO

Recientemente se han realizado análisis del consumo de energía de TM sobre distintos tipos de procesadores y se han propuesto soluciones a partir de los resultados de dichos análisis. Gaona *et al.* [11] caracterizan el consumo de energía de dos TM hardware. Además, proponen la serialización dinámica de transacciones en hardware con el objetivo de reducir la energía consumida [12]. Esto lo consiguen tratando de minimizar el trabajo especulativo que se desaprovecha cuando las transacciones encuentran conflictos y deben abortar. Del mismo modo, Moreshet *et al.* [13] y Ferri *et al.* [21] realizan análisis energéticos de TM hardware utilizando simuladores. Sus resultados muestran una mejora en el consumo energético comparado con el uso de soluciones de exclusión mutua basadas en cerrojos. Baldassin *et al.* [14], [15] caracterizan la librería de TM software TL2 [9] sobre procesadores homogéneos de bajo consumo basados en la arquitectura ARMv7 utilizando simuladores. Además, proponen una solución basada en la variación dinámica del voltaje y el escalado de la frecuencia del procesador con el objetivo de reducir el consumo de energía. Sobre la misma plataforma, Klein *et al.* [16] proponen una estrategia basada en memoria *scratch-pad* para reducir el consumo energético. Para ello, también han utilizado simuladores para estimar la energía consumida. Sanjal *et al.* [17] proponen el uso de técnicas de *clock-gating* para reducir el consumo de energía en TM hardware y mejorar su rendimiento.

En la investigación previa no se han analizado procesadores con núcleos heterogéneos como los de la arquitectura big.LITTLE. En este trabajo nos centramos en analizar TM software sobre este tipo de procesadores. Además, dado que la plataforma hardware sobre la que realizamos nuestros experimentos nos proporciona sensores de consumo energético, podemos obtener mediciones reales de la eficiencia energética de la librería evaluada. Por contra, todos los análisis anteriores se han realizado utilizando simuladores.

V. CONCLUSIONES Y TRABAJO FUTURO

En este artículo presentamos el análisis de una librería TM software ejecutada sobre un procesador multi-núcleo heterogéneo basado en la arquitectura

big.LITTLE de ARM. Nuestra evaluación muestra que la escalabilidad de la librería, en términos de rendimiento y energía, es mejor sobre el cluster little compuesto de cuatro núcleos Cortex-A7. Sin embargo, una de las aplicaciones analizadas (labyrinth) ha mostrado mejor rendimiento y una utilización más eficiente de la energía (esto es, menor EDP) en los núcleos Cortex-A15. En todos los casos, utilizar simultáneamente el cluster big y el cluster little con ocho hilos de ejecución resulta en una mejoría en el rendimiento y la eficiencia energética.

Nuestro trabajo futuro se enfoca en utilizar los resultados e infraestructura expuestos en este artículo para construir un planificador para aplicaciones que utilizan TM sobre procesadores heterogéneos. El objetivo es permitir que las aplicaciones basadas en TM puedan situarse en los núcleos que les permitan obtener mayor eficiencia energética. Además, planeamos realizar un análisis más exhaustivo de la librería TinySTM (esto es, determinar el consumo de las funciones que TinySTM proporciona). Este segundo análisis nos permitirá diseñar mejoras en la librería enfocadas al aprovechamiento de los recursos de los procesadores heterogéneos.

AGRADECIMIENTOS

Este trabajo ha sido realizado gracias a la financiación de los proyectos TIN2013-42253-P del Ministerio de Economía y Competitividad, y P12-TIC-1470 y P11-TIC-08144 de la Junta de Andalucía.

REFERENCIAS

- [1] "ARM big.LITTLE technology," <http://www.arm.com/products/processors/technologies/biglittleprocessing.php>.
- [2] Maurice Herlihy and J. Eliot B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *20th Ann. Int'l. Symp. on Computer Architecture (ISCA '93)*, 1993, pp. 289-300.
- [3] Tim Harris, James Larus, and Ravi Rajwar, *Transactional Memory, 2nd Ed.*, Morgan & Claypool Publishers, USA, 2010.
- [4] Nasser A. Kurd, Muntasquim Chowdhury, Edward Burton, Thomas P. Thomas, Christopher Mozak, Brent Boswell, Praveen Mossilanti, Mark Neidengard, Anant Desval, Ashish Khanna, Nasirul Chowdhury, Ravi Rajwar, Timothy M. Wilson, and Rajesh Kumar, "Haswell: A family of IA 22 nm processors," *J. Solid-State Circuits*, vol. 50, no. 1, pp. 49-58, 2015.
- [5] Pascal Felber, Christof Fetzer, Patrick Marlier, and Torvald Riegel, "Time-based software transactional memory," *IEEE Trans. on Parallel and Distributed Systems*, vol. 21, no. 12, pp. 1793-1807, 2010.
- [6] Pascal Felber, Christof Fetzer, and Torvald Riegel, "Dynamic performance tuning of word-based software transactional memory," in *13th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP'08)*, 2008, pp. 237-246.
- [7] Aleksandar Dragojević, Rachid Guerraoui, and Michal Kapalka, "Stretching transactional memory," in *30th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI'09)*, 2009, pp. 155-165.
- [8] Luke Dalessandro, Michael F. Spear, and Michael L. Scott, "NOrec: Streamlining STM by abolishing ownership records," in *15th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP'10)*, 2010, pp. 67-78.
- [9] Dave Dice, Ori Shalev, and Nit Shavit, "Transactional Locking II," in *Distributed Computing*, vol. 4167, pp. 194-208, Springer, 2006.
- [10] Nuno Diegues, Paolo Romano, and Luís Rodrigues, "Virtues and limitations of commodity hardware transactional

- nal memory," in *23rd Int'l. Conf. on Parallel Architectures and Computation Techniques (PACT'14)*, 2014, pp. 3–14.
- [11] Epifanio Gaona-Ramírez, Rubén Titos-Gil, Juan Fernández, and Manuel E. Acacio, "Characterizing energy consumption in hardware transactional memory systems," in *22nd Int'l. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD'10)*, 2010, pp. 9–16.
 - [12] Epifanio Gaona, Rubén Titos-Gil, Manuel E. Acacio, and Juan Fernández, "Dynamic serialization: Improving energy consumption in eager-eager hardware transactional memory systems," in *20th Euromicro Int'l. Conf. on Parallel, Distributed and Network-based Processing (PDP'12)*, 2012, pp. 221–228.
 - [13] Tali Moreshet, R. Iris Bahar, and Maurice Herlihy, "Energy-aware microprocessor synchronization: Transactional memory vs. locks," *Fourth Annual Boston-Area Architecture Workshop*, p. 21, 2006.
 - [14] Alexandro Baldassin, Felipe Klein, Guido Araujo, Rodolfo Azevedo, and Paulo Centoducatte, "Characterizing the energy consumption of software transactional memory," *IEEE Computer Architecture Letters*, vol. 8, no. 2, pp. 56–59, Feb 2009.
 - [15] Alexandro Baldassin, Joao P.L. de Carvalho, Leonardo A.G. Garcia, and Rodolfo Azevedo, "Energy-performance tradeoffs in software transactional memory," in *24th Int'l. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD'12)*, 2012, pp. 147–154.
 - [16] Felipe Klein, Alexandro Baldassin, Guido Araujo, Paulo Centoducatte, and Rodolfo Azevedo, "On the energy-efficiency of software transactional memory," in *22nd Ann. Symp. on Integrated Circuits and System Design: Chip on the Dunes (SBCCI'09)*, 2009, pp. 33:1–33:6.
 - [17] Suthirtha Sanyal, Sourav Roy, Adrian Cristal, Osmand S. Unsal, and Mateo Valero, "Clock gate on abort: Towards energy-efficient hardware transactional memory," in *IEEE Int'l. Symp. on Parallel Distributed Processing (IPDPS'09)*, 2009, pp. 1–8.
 - [18] "ODROID — Hardkernel," http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127.
 - [19] Chi Cao Minh, Jaewoong Chung, Christoph Kozyrakis, and Kunle Olukotun, "STAMP: Stanford transactional applications for multi-processing," in *IEEE Int'l. Symp. on Workload Characterization (IISWC'08)*, 2008, pp. 35–46.
 - [20] Wenjia Ruan, Yujie Liu, and Michael Spear, "STAMP need not be considered harmful," in *9th ACM SIGPLAN Workshop on Transactional Computing (TRAN-SACT'14)*, 2014, pp. 1–7.
 - [21] Cesare Ferri, Amber Viescas, Tali Moreshet, Iris Bahar, and Maurice Herlihy, "Energy implications of transactional memory for embedded architectures," *Workshop on Exploiting Parallelism with Transactional Memory and other Hardware Assisted Methods (EPHAM'08)*, 2008.

Análisis comparativo del uso de STMs en códigos de reducción irregulares

Manuel Pedrero, Eladio Gutiérrez, Sergio Romero y Óscar Plata¹

Resumen— La memoria transaccional (TM) constituye un paradigma de concurrencia optimista en arquitecturas multinúcleo que puede ser de utilidad en la explotación de paralelismo en aplicaciones irregulares, en las que la información sobre las dependencias de datos no está disponible hasta la ejecución. Este trabajo presenta y discute cómo aprovechar las características de un sistema STM (software transaccional memory) en patrones de computación que involucren operaciones de reducción, ligadas frecuentemente a aplicaciones irregulares. Con el fin de comparar el uso de enfoques STM en esta clase de patrones con otras soluciones más clásicas, se ha implementado como prueba de concepto un sistema STM, que denominaremos ReduxSTM, que combina dos ideas: una consolidación (commit) ordenada de las transacciones, que asegura una equivalencia con la ejecución secuencial del código; y una extensión del mecanismo de privatización subyacente al sistema STM que contempla las operaciones de reducción.

Palabras clave— Software Transaccional Memory (STM), operaciones de reducción, aplicaciones irregulares.

I. INTRODUCCIÓN

La disponibilidad de múltiples núcleos compartiendo una memoria global en los computadores actuales está teniendo una gran influencia en cómo se diseñan las aplicaciones. Al descomponer un problema en tareas concurrentes, el rendimiento final está determinado en buena parte por cómo se gestionan las posibles dependencias de datos y de control. En general las dependencias se gestionan de una manera conservadora, especialmente cuando se resuelven en tiempo de compilación. En el caso de aplicaciones que presentan un patrón irregular de referencias de memoria ésta gestión puede resultar muy limitada, ya que muchas dependencias no se conocen completamente hasta que la aplicación no se ejecuta. En este contexto, la memoria transaccional (TM) [1] proporciona un modelo de concurrencia optimista en arquitecturas multinúcleo, facilitando al programador la explotación de paralelismo en códigos –como los de las aplicaciones irregulares– que no son fácilmente analizables de forma estática.

TM ha surgido como una alternativa que facilita la coordinación de threads concurrentes. TM proporciona el concepto de transacción: una estructura de programación que garantiza atomicidad, consistencia y aislamiento en bloque de las instrucciones que la componen. Las transacciones pueden ejecutarse concurrentemente, pero el sistema garantiza que los resultados de la ejecución sean los mismos que en una ejecución secuencial.

En un sistema TM, las transacciones se ejecutan de manera especulativa, de forma que las modificaciones en las posiciones de memoria quedan registradas por un gestor de versiones. Si dos transacciones concurrentes entran en conflicto (escritura/escritura, lectura/escritura en la misma posición de memoria), una de ellas debe abortar. Tras restaurar su estado inicial (rollback), la transacción que aborta reintentará la ejecución especulativa. Cuando una transacción termina su ejecución libre de conflictos, consolida sus datos modificados (commit) haciéndolos visibles al resto del sistema. Se dice que la gestión de versiones en un sistema TM es *eager* si los cambios se trasladan a memoria inmediatamente, manteniendo un buffer con los valores antiguos (*undo log*) que se usará para restaurar el estado inicial en caso de aborto. Por el contrario, en una gestión de versiones *lazy* las modificaciones se mantienen en un buffer de escritura (*redo log*), consolidándose al final de la transacción, durante la fase de commit. Con una terminología similar, la detección de conflictos se puede realizar inmediatamente (*eager*) o bien posponerse hasta el comienzo de la fase de commit (*lazy*). Encontramos en la literatura un buen número de propuestas TM, implementadas bien en software (STM), en hardware (HTM) o con enfoques híbridos (HyTM) [2].

Dadas las ventajas de TM, se han realizado esfuerzos encaminados a extraer paralelismo de aplicaciones secuenciales ya existentes mediante este paradigma. De hecho, muchas de las características básicas de TM, como la detección de conflictos, la privatización especulativa de posiciones de memoria y el aborto/reintento de la ejecución se pueden encontrar en otras técnicas como el multithreading especulativo (SpMT), o la especulación a nivel de thread (thread-level speculation ó TLS) [3], las cuales se han probado eficaces en la paralelización de programas con threads.

En general paralelizar un programa secuencial implica descomponer el programa en tareas y resolver correctamente las dependencias de datos entre las mismas. De este modo, la concurrencia optimista que ofrece TM puede ayudar en la paralelización de aplicaciones irregulares, donde un análisis estático no es suficiente para determinar la mayor parte de las dependencias. Las transacciones, definidas como secciones del programa secuencial, pueden ejecutarse concurrentemente, dejando a cargo del sistema TM la detección y resolución de los conflictos entre transacciones en tiempo de ejecución. Nótese sin embargo que, además de los conflictos de datos, podría requerirse una ordenación de las transacciones para

¹Dpto. de Arquitectura de Computadores, Universidad de Málaga, 29071 Málaga, e-mail: mpedrero@uma.es, eladio@uma.es, sromero@uma.es, oplata@uma.es

asegurar resultados correctos. Los sistemas TM convencionales no garantizan ningún orden de commit entre las transacciones.

En este trabajo se discute cómo utilizar un sistema TM para explotar patrones de reducción (operaciones asociativas y conmutativas) con accesos de memoria irregulares (no conocidos en compilación) y se compara esta alternativa con otras soluciones clásicas. Para ello se ha desarrollado ReduxSTM; un sistema STM que añade características específicas para este tipo de operaciones. ReduxSTM garantiza el commit de las transacciones en un orden equivalente al secuencial, y proporciona un tratamiento específico de las reducciones aprovechando la privatización subyacente al sistema STM. Con ello se pretende reducir el número de abortos derivados de las operaciones de reducción esperando trasladar este hecho a una mejora del rendimiento.

II. PATRONES DE REDUCCIÓN EN APLICACIONES IRREGULARES

Una sentencia de reducción es un patrón de la forma $O = O \oplus \xi$, donde \oplus es un operador asociativo y conmutativo aplicado a un objeto de memoria O (objeto o variable de reducción), y ξ es una expresión computada con objetos privados que no dependen de O . Se dice que un bucle es *completamente* de reducción (o reducción de histograma) si contiene sentencias de reducción y las únicas dependencias entre iteraciones son causadas por el objeto de reducción. Así mismo, el objeto de reducción no debe ser accedido por ninguna otra sentencia que no sea de reducción [4].

Las operaciones de reducción son parte habitual del núcleo de muchas aplicaciones computacionales como el álgebra de matrices dispersas, resolutores de ecuaciones diferenciales, etc. En estos casos el objeto de reducción (comúnmente un vector multidimensional) es accedido a través de índices indirectos, lo que confiere una naturaleza irregular a la aplicación.

Desde el punto de vista de las dependencias de datos, las sentencias de reducción causan dependencias entre iteraciones, ya que la variable de reducción es leída y a su vez escrita. No obstante, en los bucles completamente de reducción, las iteraciones se pueden reordenar sin problemas siempre que todas las reducciones sobre el objeto de reducción tengan el mismo operador, al ser éste conmutativo y asociativo.

No obstante puede haber situaciones donde la condición de bucle de reducción no se verifique completamente. Por ejemplo si se accede al objeto de reducción fuera de las sentencias de reducción, si se combinan varios operadores diferentes (aunque sean de reducción), o si ocurren otras dependencias entre las iteraciones debidas a otras variables no reductivas [5], [6]. A pesar de ello, puede que la condición de reducción se siga verificando para un subconjunto de iteraciones. Es lo que se conoce como reducciones parciales. Un ejemplo de esto se muestra en la figura 3.

A la hora de paralelizar bucles de reducción, podemos clasificar los métodos propuestos en la literatura en dos grandes grupos: (a) métodos que garantizan la exclusión mutua entre los accesos a los objetos de reducción [7], y (b) métodos que acumulan parcialmente el resultado en copias privadas de los objetos de reducción [8], [9], [10] y luego realizan una reducción global en el objeto de reducción original.

A. Exclusión mutua

Una forma obvia de resolver los conflictos causados por las sentencias de reducción es convertir el bucle secuencial completamente de reducción en un DOALL, encerrando las sentencias de reducción (o un grupo de ellas) en una sección crítica, de manera que sólo un thread accederá a las variables de reducción a la vez. El principal inconveniente de este enfoque es el alto grado de serialización, aunque puede mejorarse empleando locks de grado fino (*fine-grained locks*). Si el objeto de reducción es un vector, asociaríamos un lock por cada elemento del mismo, de modo que dos threads puedan acceder a elementos diferentes en paralelo, y sólo haya serialización en caso de producirse un conflicto real. Un enfoque similar se puede conseguir con operaciones atómicas, aunque su disponibilidad depende en gran medida de la arquitectura hardware.

En este grupo de técnicas podemos incluir modelos basados en tareas, como la cláusula *omp task depend* incluida recientemente en el estándar OpenMP u otras soluciones similares como las de [11] y [12]. En estos casos las variables de reducción se marcarían como una dependencia de entrada-salida de la tarea. La principal limitación de estos enfoques en códigos irregulares es su capacidad para expresar dependencias complejas, como las indirecciones, y también para expresar las dependencias cuando éstas se computan dentro de la tarea.

B. Privatización

Privatizar los objetos de reducción es una solución bastante eficaz y extendida a la hora de paralelizar bucles de reducción. En este caso se distribuye el espacio de reducción entre los threads, cada uno de los cuales opera sobre copias privadas de las variables de reducción. Estas copias deben inicializarse al elemento neutro del operador. Tras finalizar su trabajo, las reducciones parciales realizadas en las copias privadas han de reducirse de forma segura sobre los objetos de reducción originales. Dos técnicas representativas de este grupo son *Replicated Buffer* y *Array Expansion*, que se diferencian en cómo resuelven los conflictos en la reducción final. El principal inconveniente de la privatización es su gasto de memoria extra, ya que multiplicamos el tamaño de los objetos de reducción por el número de threads.

III. ENFOQUES TM EN PATRONES DE REDUCCIÓN

Al considerar la paralelización de un bucle de reducción completo, un enfoque directo mediante TM consiste en sustituir la sección crítica que abarca las

<pre> Read subscript arrays: edge(1,*), edge(2,*) do itime=1,nTimes do i=1,nEdges do i=1,nEdges n1 = edge(1,i) n2 = edge(2,i) Compute $\zeta_1, \zeta_2, \zeta_3$ vel(1,n1)=vel(1,n1)+ζ_1 vel(2,n1)=vel(2,n1)+ζ_2 vel(3,n1)=vel(3,n1)+ζ_3 vel(1,n2)=vel(1,n2)-ζ_1 vel(2,n2)=vel(2,n2)-ζ_2 vel(3,n2)=vel(3,n2)-ζ_3 enddo enddo </pre>	<pre> Compute subscript arrays: m(*) mbeg(*), mend(*) do irow=1,nRows do i=1,jdt+1 im=m(i) imb=mbeg(i) ime=mend(i) do is=imb,ime,2 Compute ζ_1, ζ_2, \dots do ilev=1,2*ilev f1(ilev,im)=f1(ilev,im)+ζ_1 f2(ilev,im)=f2(ilev,im)+ζ_2 enddo enddo </pre>	<pre> do ihop=1,nHops Update subscripts: B1(*),B2(*) do itime=1,nTimes do ih=1,nParticles j=B1(ih) j=B2(ih) Compute $r(i,j), \zeta(i,j), \eta(i,j), \theta(i,j)$ if (r.lt. Cutoff) then AX(i)=AX(i) + ζ AX(j)=AX(j) - ζ AY(i)=AY(i) + ζ AY(j)=AY(j) - ζ U = U + η P = P + θ endif enddo enddo </pre>
(a)	(b)	(c)

Fig. 1

ALGUNOS CÓDIGOS CON BUCLES DE REDUCCIÓN: (A) UNSTRUCTURED, (B) TRANSFORMADA DE LEGENDRE Y (C) DINÁMICA MOLECULAR 2D.

<pre> for (i=0; i<NInd; i++){ Compute ξ_1, ξ_2 #pragma omp critical{ ... A[idx1[i]] @= ξ_1 A[idx2[i]] @= ξ_2 ... } } </pre>	<pre> for (i=0; i<NInd; i++){ Compute ξ_1, ξ_2 BEGIN_XACT() ... TM_WRITE(A[idx1[i]]], TM_READ(A[idx1[i]]] @ ξ_1) TM_WRITE(A[idx2[i]]], TM_READ(A[idx2[i]]] @ ξ_1) ... END_XACT() } </pre>
(a)	(b)

Fig. 2

USO DE TRANSACCIONES (B) COMO REEMPLAZO DE UNA SECCIÓN CRÍTICA (A).

```

for (i=0; i<N; i++){
A[K[i]] = ...;
... = A[L[i]];
A[R[i]] = A[R[i]] @  $\xi$ ;
}
        
```

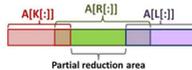


Fig. 3

EJEMPLO DE BUCLE CON UNA ZONA DE REDUCCIÓN PARCIAL. LOS SUBÍNDICES, K, L AND R, RESTRINGEN EL ACCESO A A TAL COMO SE MUESTRA. LOS ACCESOS A[K[i]] AND A[L[i]] NO SE SOLAPAN.

sentencias de reducción por una transacción, tal como se muestra en la figura 2. Esta solución es simple desde el punto de vista de la programabilidad, y los potenciales conflictos que surgen debido a las indirecciones son gestionados por el sistema TM. En caso de escenarios de baja contención, el sistema TM podrá mantener un buen nivel de paralelismo frente a la fuerte serialización que se produce en el caso de emplear secciones críticas (mutex, spinlocks, atomics, etc).

Si además imponemos cierto orden en el TM de forma que los commits de las transacciones tengan lugar de forma equivalente al orden de la ejecución secuencial, entonces podemos pensar en el sistema TM como una herramienta de apoyo a un enfoque

TLS. Esto es especialmente interesante para aquellos casos en los que la paralelización de las reducciones no se puede abordar con soluciones clásicas porque las condiciones de bucle de reducción no se verifican completamente.

Un ejemplo de esta situación se muestra en la figura 3, donde hay conflictos potenciales entre accesos a un vector en una sentencia de reducción y accesos fuera de ella. Sin embargo puede existir un subconjunto de iteraciones que cumplan la condición de reducción si existen restricciones en los subíndices como las mostradas en la figura [5]. En la figura 4 podemos observar otro ejemplo donde existen lecturas y escrituras a través de punteros que pueden ser alias de las variables de reducción. Este hecho impide saber en tiempo de compilación si se trata de un bucle completamente de reducción y si, consecuentemente, las iteraciones pueden o no reordenarse con seguridad [6]. Estos patrones, a los que hemos denominado *reducciones parciales*, han sido tratados en la literatura por medio de enfoques especulativos, como se discute en la siguiente sección.

IV. TRABAJOS RELACIONADOS

La idea de resolver la paralelización de bucles de reducción especulativamente no es nueva. En [13] se propone el test LRPD que, tras una fase de inspección, es capaz de seleccionar aquellas iteraciones que se pueden lanzar especulativamente en paralelo mediante una estructura DOALL. Esta idea ha sido reformulada recientemente con Privateer [4]. Otro enfoque especulativo reciente para bucles de reducción lo encontramos en [6], centrado en la detección y ejecución especulativa de variables de reducción parcial (PRV). En [14] y [15] se explora la posibilidad de utilizar un sistema TM en bucles de reducción, deshabilitando la detección de conflictos y extendiendo el buffer de escritura para tal fin. Este enfoque requiere el conocimiento previo de que el bucle sea completamente de reducción y no es aplicable en el caso de reducciones parciales. Otras propuestas más

```

for (termpr = ... ; termpr = termpr->nextterm) {
...
  for (netpr = ... ; netpr=netpr->nterm) {
...
    *costpr += ...; // Reduction sentence
  }
...
  rowsptr = tmp_rows[net] ;
  for (row = 0 ; row&lt;pr[row] == 0 ; row++) {
...
  }
...
  tmp_num_feeds[net] = f ;
...
  tmp_missing_rows[net] = -m ;
...
  delta_vert_cost += ( ... ) ; // Reduction sentence
}
}

```

Fig. 4

ESQUEMA DE UN BUCLE DE LA FUNCIÓN *new_dbox_a()* DEL CÓDIGO *300.twolf* (BENCHMARK SPEC CPU2000). LOS ALIAS ENTRE PUNTEROS HACEN IMPOSIBLE SABER SI SE TRATA DE UN BUCLE DE REDUCCIÓN AUNQUE CONTIENE SENTENCIAS DE REDUCCIÓN.

generales basadas en TM y que son aplicables a reducciones irregulares las encontramos en [16], [17], [18]. IPOT [16] es capaz de lanzar bloques de instrucciones en paralelo contenidas en estructuras similares a transacciones ordenadas, y permite relajar las restricciones de consistencia para mejorar el rendimiento. ALTER [17] propone un esquema TLS al estilo transaccional en el que las variables pueden ser anotadas permitiéndoles un chequeo de consistencia más permisivo. Una de estas anotaciones está pensada precisamente para las variables de reducción. En [18] se propone una técnica de paralelización automática basada en transacciones hardware ordenadas. Por último cabe destacar RMW (Read-Modify-Write without aborts [19]), un enfoque STM reciente que proporciona soporte específico para los patrones de lectura-modificación-escritura de los cuales las reducciones son un caso particular. Este enfoque general está limitado, no obstante, por su limitada escalabilidad, no contempla transacciones ordenadas y está pensado fundamentalmente para variables escalares, excluyendo vectores y estructuras multidimensionales que aparecen con frecuencia en códigos con reducciones.

V. REDUXSTM

Además de ser un sustituto directo de una sección crítica, las transacciones realizan una privatización selectiva de aquellas variables marcadas como transaccionales. La idea que se plantea es aplicar este mecanismo de privatización subyacente a las variables de reducción, permitiendo evitar aquellos abortos derivados de las sentencias de reducción siempre que sea posible. Las transacciones realizarán reducciones parciales sobre su versión local de la variable, realizándose la reducción global en la variable compartida durante la fase de commit. La asociatividad y conmutatividad del operador garantiza que esto pueda hacerse de forma segura. De esta manera el programador sólo tiene que marcar como transaccio-

TABLA I
DIFERENTES TÉCNICAS DE PARALELIZACIÓN DE BUCLES COMPLEMENTAMENTE DE REDUCCIÓN.

	Requerimiento de memoria extra	Paralelismo potencial	Sobrecarga de sincronismo	Reducción final
Privatización	muy alto	muy alto	muy bajo	si
Sección crítica	muy bajo	muy bajo	alto	no
Lock de grano fino	alto	alto	alto	no
Ops. atómicas	ninguno	muy alto	alto	no
TM	bajo	alto/medio	bajo	no
ReduxSTM	bajo	muy alto	bajo	en commit

nales las operaciones de reducción (sentencias) sin tener conocimiento de si se trata de un bucle completamente de reducción o de una reducción parcial. Aquellas sentencias de reducción que tengan conflictos con otros accesos a memoria (escritura o lectura) serán detectadas y corregidas por el STM de forma transparente al programador.

Nuestra propuesta, ReduxSTM, soporta por tanto patrones *potenciales* de reducción, donde resulta complicado determinar estáticamente si son reducciones parciales o totales. ReduxSTM ha sido planteado como una prueba de concepto para comprobar si un sistema TM se puede beneficiar de un tratamiento específico de los patrones de reducción. ReduxSTM ha sido construido como una librería desde cero. Una comparativa de los diferentes enfoques discutidos se incluye en la tabla I.

A. Características

Soporte de reducciones. Una de las características clave de ReduxSTM es la capacidad de explotar la privatización selectiva asociada al sistema transaccional. Al explotar esta privatización implícita se consiguen eliminar abortos innecesarios, mejorando el grado de concurrencia.

Con este propósito se introduce una nueva primitiva para las operaciones de reducción, que el programador podrá usar junto con las ya existentes de lectura y escritura. Esta primitiva es tratada por los gestores de conflictos y de versiones como una tercera operación básica: lectura (R), escritura (W) y reducción (Rdx). Su semántica es una lectura, seguida de una escritura en la misma posición de memoria tras haber realizado la operación de reducción. La nueva primitiva $Rdx(add, val, \oplus)$ es semánticamente equivalente a $W(add, R(add) \oplus val)$ pero permite una ejecución más eficiente.

Transacciones ordenadas. Una segunda característica importante de ReduxSTM es el mantenimiento de una restricción de orden entre los commits de las transacciones. De esta manera podemos garantizar que la ejecución especulativa lleva al mismo resultado que la versión secuencial. Obsérvese que, aunque los bucles completamente de reducción pueden ser reordenados sin problemas, esto no es así en el caso de las reducciones parciales, donde las reducciones coexisten con lecturas y escrituras sobre las variables de reducción fuera de las sentencias de reducción.

TABLA II
 CONFLICTOS TRANSACCIONALES POTENCIALES.

	STM Estándar	ReduxSTM (Orden + Rdx.)
R-W	aborto	no hay conflicto
W-R	aborto	aborto
W-W	aborto	no hay conflicto
Rdx-R	como R-W-R	aborto
R-Rdx	como R-R-W	no hay conflicto
Rdx-W	como R-W-W	no hay conflicto
W-Rdx	como W-R-W	no hay conflicto
Rdx-Rdx	como R-W-R-W	no hay conflicto

Esta restricción de orden en la fase de commit se convierte en el mecanismo básico de atomicidad de los commits, pero a su vez tiene un coste de rendimiento, ya que supone esperas no deseables en el turno de commit. No obstante, la oportunidad de mejora radica en el ahorro de abortos y rollbacks generados por las falsas dependencias asociadas a las reducciones [3], [15] como se muestra en la tabla II. La primera columna especifica dos operaciones realizadas por dos transacciones diferentes, donde la primera debe realizar la fase de commit antes de la segunda. Por ejemplo, R-W, estaría asociado a una anti-dependencia. La segunda columna corresponde al comportamiento de un TM convencional y la tercera a nuestra propuesta.

Gestión de versiones. Soportar reducciones (que pueden entrar en conflicto con lecturas y escrituras) junto con las transacciones ordenadas conlleva una gestión de versiones *lazy*, donde la consolidación de las posiciones de memoria se realiza al finalizar la transacción en la fase de commit. Para ello se introducen dos buffers privados que almacenan información disjunta: el *write buffer* y el *reduction buffer*. El primero ya existe en los sistemas transaccionales convencionales y el segundo es específico para las operaciones de reducción. Obsérvese que es necesario un *reduction buffer* por cada operación de reducción soportada (suma, producto, etc).

Cada vez que se ejecuta una reducción transaccional sobre una posición de memoria, se busca dicha posición en el *write buffer*. Si dicha posición está ahí, se reduce el valor especificado en la reducción con el valor almacenado en el buffer y se mantiene en el *write buffer* ya que se viola la condición de reducción. En caso contrario, el valor de reducción se opera con el valor correspondiente del *reduction buffer* (o con el elemento neutro del operador si es la primera operación sobre esta posición), actualizando dicha posición en el *reduction buffer* con el resultado de la operación (reducción parcial). Si una posición almacenada en el *reduction buffer* es escrita con posterioridad en la misma transacción, debe ser eliminada de este buffer e insertada en el *write buffer* ya que deja de cumplirse en este momento la condición de reducción. Obsérvese que una lectura de una posición que está marcada como reducción en una transacción implica combinar el valor de memoria con el acumulado parcialmente en el buffer de reducción.

Detección de conflictos. En ReduxSTM la va-

lidación/invalidación de las transacciones se realiza durante la fase de commit, por lo que la detección de conflictos se considera *lazy* [20].

Gestión de commits. Puesto que sólo una transacción puede estar en fase de commit para garantizar el orden, tal transacción es la responsable de comprobar y resolver posibles conflictos con otras transacciones activas. El orden de finalización garantiza la naturaleza atómica de la fase de commit, y por tanto actúa como el principal mecanismo de sincronización. La fase de commit está sujeta a las características particulares de la estrategia de implementación, tal como se discute en la sección V-B. Independientemente de la implementación, la fase de commit de una transacción debe (1) Esperar su turno de commit; (2) Comprobar y resolver posibles conflictos con otras transacciones; y (3) Consolidar (actualizar) la memoria principal con los valores almacenados en los buffers de reducción y escritura (los valores de reducción necesitarán ser acumulados según su operador asociado).

B. Implementación

Hemos seleccionado dos algoritmos STM bien conocidos, *Commit Time Invalidation* y *Time-Based Validation*, como base de nuestras implementaciones de ReduxSTM. La elección de estos algoritmos radica en que son adecuados para implementar de forma efectiva las características descritas anteriormente. Ambas implementaciones fueron codificadas desde cero.

Commit Time Invalidation (CTI)

En esta implementación la transacción que realiza el commit marcará como invalidadas (a abortar) aquellas transacciones activas que tengan algún conflicto con ella [21]. Los conflictos de datos se detectan a partir de las direcciones de memoria. Al comenzar su fase de commit, y tras esperar su turno, cada transacción comprueba si es o no válida, abortando en caso negativo. Si es válida, consolida (commit) los datos transaccionales en la memoria, tras lo cual comprueba posibles conflictos con las transacciones en ejecución, invalidándolas si sus conjuntos de datos de lectura no son disjuntos con los conjuntos de escritura y reducción de la transacción en el commit (ver tabla II). Nuestra implementación usa filtros de Bloom para representar cada uno de los conjuntos de datos (lectura, escritura y reducción).

Time-based validation (TS)

A diferencia de la estrategia anterior, ésta emplea marcas de tiempo (timestamps) para registrar *cuándo* tienen lugar las lecturas y actualizaciones [22]. Nuestra implementación usa como reloj global de estas marcas el orden global de la última transacción finalizada. Cada transacción mantiene una tabla privada para las marcas de tiempo de sus lecturas. Así mismo se mantiene una tabla global de marcas de tiempo para las escrituras y reducciones consolidadas en memoria. En la fase de commit se comprueba si hay lecturas cuya marca de tiempo sea posterior a la de la escritura consolidada corres-

TABLA III
 CÓDIGOS TESTEADOS

<i>Fluidanimate</i>	Forma parte de la suite PARSEC [23]. Esta aplicación simula flujidos usados en animaciones de tiempo real. Se comprobaron dos configuraciones: una de 100K partículas durante 5 fotogramas y otra de 500K partículas durante 500 fotogramas.
<i>MD2</i>	Esta aplicación [24] corresponde a una simulación de dinámica molecular 2D en sistemas con un número elevado de partículas con interacciones de corto alcance. Para limitar la complejidad del problema las interacciones se acotan a partículas cercanas mediante una lista de partículas vecinas.
<i>Unstructured</i>	Este código [25] resuelve las ecuaciones de Euler en simulaciones físicas. En cada paso de tiempo la aplicación computa fuerzas y velocidades en los nodos de una malla.
<i>Legendre</i>	Corresponde al núcleo de la transformada de Legendre [26] usada en predicción meteorológica. En cada paso de tiempo se invocan la transformada directa e inversa que llevan asociadas reducciones irregulares debido a los accesos a través de indirrecciones.
<i>300.twolf</i>	Es un simulador de <i>place and route</i> incluido en el benchmark SPEC 2000 [27]. Contiene patrones de reducción interesantes en la rutina <i>new_box.o</i> (<i>disbox.c</i>), donde pueden aparecer conflictos potenciales entre variables de reducción y de no reducción debido a los alias entre punteros.

pondiente, en cuyo caso la transacción debe abortar. Para reducir la memoria requerida por las marcas de tiempo, las tablas están limitadas en tamaño y son accedidas por un hash de la dirección de memoria, lo que implica cierta probabilidad de falsos positivos.

VI. EVALUACIÓN EXPERIMENTAL

En esta sección se evalúa experimentalmente cómo se comportan las técnicas STM en códigos dominados por operaciones de reducción, en especial cuando se incluye soporte para reducciones. Los experimentos se realizaron en un servidor con 256 GB de RAM y 16 cores (32 threads) Intel Xeon E5-2698 a 2.3GHz, con sistema operativo Linux kernel 3.13 (64 bits). Los programas se compilaron con GNU GCC 4.8.2 con opción de optimización -O2.

Como sistema STM de referencia a efectos de comparación se ha empleado TinySTM (v.1.0.5) [22], un STM que puede considerarse representativo del estado del arte actual. Se emplearon tanto la versión base como la versión ordenada de TinySTM.

En la evaluación se han seleccionado varios códigos representativos que se describen brevemente en la tabla III. *Fluidanimate*, *MD2*, *Unstructured* y *Legendre* contienen bucles completamente de reducción. Por su parte, el código *300.twolf* incluye patrones de reducción parcial.

Las técnicas de paralelización que se han comparado son las siguientes:

- *Locks de grano grueso* (CG Locks), que se corresponde con el uso de locks para proteger secciones críticas;
- *Locks de grano fino* (FG Locks), donde se usa un lock individual para proteger cada elemento del array de reducción compartido;
- Privatización completa de los arrays de reduc-

ción, implementada como *Array Expansion* [9];

- TinySTM (configuración por defecto) usado como referencia;
- TinySTM-ordered, la versión de TinySTM con commits ordenados;
- ReduxSTM en sus dos implementaciones: basada en marcas de tiempo (ReduxSTM-TS) y con invalidación en fase de commit (ReduxSTM-CTI).

En todos los sistemas STM la paralelización de los bucles de reducción se llevó a cabo descomponiendo los bucles en bloques de iteraciones consecutivas (chunks) y ejecutando cada bloque dentro de una transacción. Obsérvese que el número de iteraciones en cada bloque determina el tamaño de la transacción y es un parámetro relevante, que debe ser elegido cuidadosamente: transacciones muy grandes reducen la carga extra introducida por la instrumentación del sistema transaccional, pero implican mayores conjuntos de datos y por tanto mayor probabilidad de conflicto y de abortos. Las transacciones pequeñas tienen menor probabilidad de conflicto, pero implican un mayor coste de instrumentación debido al lanzamiento y cierre de las transacciones (por ejemplo de la inicialización de las estructuras de datos).

En la privatización y las técnicas basadas en locks, los bucles se han particionado equitativamente entre los threads, sin agrupar las iteraciones en chunks. Recuérdese que la privatización requiere una fase de inicialización y otra de reducción final.

Los locks se han implementado con *spinlocks* de POSIX. Adicionalmente, para los locks de grano fino, se ha optimizado el código mediante el uso de un mismo lock para aquellos bloques de sentencias de reducción cuyos accesos a los arrays de reducción están indexados por el mismo índice. Esto reduce el número de locks necesario y el número de pares lock/unlock ejecutados, lo que se traduce en una menor sobrecarga.

A. Comparación de rendimiento

A continuación se ofrece una comparativa de las diferentes técnicas en términos de rendimiento. El speedup observado ha sido calculado con respecto a la versión secuencial no instrumentada de los códigos, usando el menor tiempo de ejecución de entre al menos diez ejecuciones.

En los experimentos, las variables independientes consideradas son el número de threads y el tamaño de las transacciones (iteraciones por bloque) en los sistemas STM. En este caso, los speedups mostrados corresponden al mejor chunk, esto es, al tamaño de transacción que proporciona un speedup más alto. Las gráficas que muestran el speedup en función del tamaño de transacción muestran ejecuciones con 16 threads.

La figura 5 muestra los resultados para *Fluidanimate* usando dos conjuntos de datos de 100K y 500K partículas. Como se espera, los locks de grano grueso no aceleran en absoluto debido al uso de un sólo lock global. Este método podría tener sentido en aplicaciones en las que el tiempo de ejecución en sección

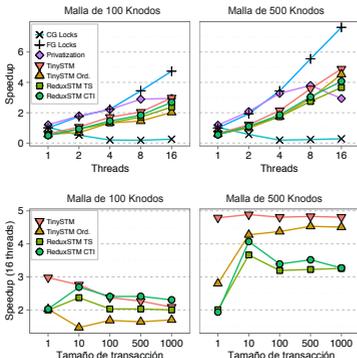


Fig. 5

FLUIDANIMATE: SPEEDUP PARA EL MEJOR TAMAÑO DE TRANSACCIÓN EN CADA CASO, E INFLUENCIA DEL TAMAÑO DE TRANSACCIÓN PARA 16 THREADS.

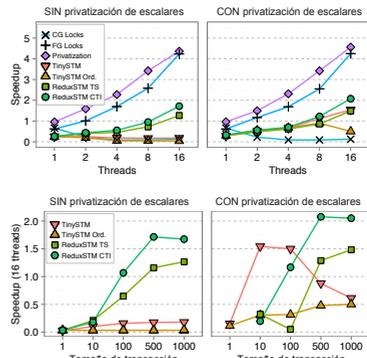


Fig. 6

MD2: SPEEDUP PARA EL MEJOR TAMAÑO DE TRANSACCIÓN EN CADA CASO, E INFLUENCIA DEL TAMAÑO DE TRANSACCIÓN PARA 16 THREADS.

crítica fuera despreciable con respecto al resto, pero no es el caso en este problema. Los mejores resultados se obtienen con locks de grano fino o privatización, debido principalmente a la baja contención del problema. Obsérvese que, para la malla más grande, el rendimiento de la privatización deja de escalar para un número de threads alto. Esto es consecuencia de las características NUMA de la arquitectura y la gran cantidad de memoria extra que necesita este método. Por su parte, todos los sistemas STM presentan un comportamiento similar, debido a que la tasa de abortos se mantiene baja. Esta situación no le da gran margen de ventaja a ReduxSTM. En este escenario los enfoques STM obtienen un speedup moderado sin grandes requerimientos de memoria. Con respecto a la influencia del tamaño de transacción, el comportamiento de los sistemas STM es muy dependiente de las características de la entrada. De esta manera, para la malla más pequeña la penalización de las transacciones grandes es más significativa que en el caso de la malla mayor, para la cual las transacciones más pequeñas implican una alta penalización.

MD2 incluye reducciones sobre variables escalares y sobre vectores (ver figura 1(c)). En este código se han probado dos estrategias, que se muestran en la figura 6. En la primera, todas las variables de reducción (escalares y vectoriales) han sido tratadas transaccionalmente. En la segunda, las variables escalares fueron privatizadas, de manera que el número de sentencias de reducción sobre objetos compartidos se reduce de 6 a 4 por iteración. Esta optimización es bastante común en este tipo de códigos [28], pero requiere un conocimiento mayor por parte del programador/compilador. Sin la privatización de escalares, la elevada contención causada por los escalares hacen

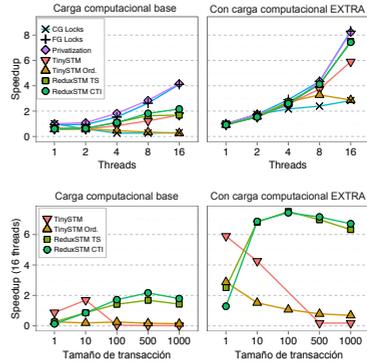


Fig. 7

UNSTRUCTURED: SPEEDUP PARA EL MEJOR TAMAÑO DE TRANSACCIÓN EN CADA CASO, E INFLUENCIA DEL TAMAÑO DE TRANSACCIÓN PARA 16 THREADS.

que TinySTM tenga una tasa de abortos muy alta, y por tanto un peor rendimiento. Como se observa, es precisamente en estos casos de alta contención en las variables de reducción donde ReduxSTM obtiene una ventaja en el rendimiento. En cualquier caso, el pequeño tamaño de los arrays de reducción hacen que la privatización obtenga los mejores resultados, sobrepasando a los locks de grado fino.

La figura 7 presenta los resultados de Unstruct-

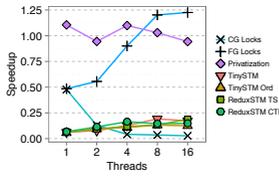


Fig. 8

LEGENDRE: SPEEDUP (CONSIDERANDO UNA ITERACIÓN POR TRANSACCIÓN).

red. En estos experimentos, el soporte para reducciones de ReduxSTM le permite tener ventaja con respecto a TinySTM. El rendimiento de TinySTM se degrada rápidamente por los conflictos. Por contra, ReduxSTM se beneficia de su filtrado de conflictos para poder conseguir mejores resultados con transacciones más grandes. Como en los casos anteriores, los locks de grano fino y la privatización siguen teniendo los mejores speedups. No obstante también se ha testado el código incorporando una carga computacional sintética adicional. En casos con mayor intensidad computacional, ReduxSTM es capaz de alcanzar el speedup de la privatización.

La figura 8 recoge los resultados para Legendre. En este caso no se han analizado diferentes tamaños de transacción, dado que el bucle exterior paralelizado no contiene un número elevado de iteraciones. En este código ninguna de las técnicas consigue un buen rendimiento. La baja intensidad computacional del mismo (es un problema *memory-bound*) hace que cualquier instrumentación adicional deteriore el rendimiento.

La figura 9 muestra los resultados obtenidos con 300.twolf. Los resultados están referidos a la rutina `new_dbox_a()`. Esta función presenta un patrón de acceso a memoria con alta contención, y la cantidad de paralelismo explotable está limitada por la baja intensidad computacional del código. Sólo se han considerado chunks de una y dos iteraciones por transacción, ya que transacciones mayores degradan el rendimiento. Es importante destacar que en este problema sólo son aplicables los métodos que garantizan el orden de la ejecución secuencial original, ya que las operaciones de reducción coexisten con otras operaciones de lectura y escritura potencialmente conflictivas. No obstante, aunque TinySTM no ordenado no cumple esta condición, se ha incluido a modo de referencia optimista. Nótese que ReduxSTM presenta un rendimiento significativamente mejor para todas las configuraciones. Cabe mencionar que, a pesar de que es difícil de explotar paralelismo en este benchmark, ReduxSTM es capaz de obtener aceleración hasta un número relativamente alto de threads aunque, para la carga computacional analizada, a partir de 8 threads los conflictos empeoran el rendimiento.

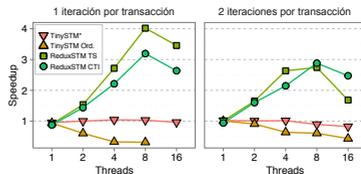


Fig. 9

300-TWOLF: SPEEDUP PARA TRANSACCIONES DE UNA Y DOS ITERACIONES. SÓLO LOS STM ORDENADOS GARANTIZAN EL RESULTADO CORRECTO. TINYSTM (NO ORDENADO) SE MUESTRA COMO UNA REFERENCIA OPTIMISTA.

VII. CONCLUSIONES

Es conocido que muchas aplicaciones con patrones irregulares de acceso de memoria son difíciles de paralelizar. En este contexto, la concurrencia optimista que proporcionan los sistemas de memoria transaccional (TM) puede resultar útil para extraer paralelismo. En este trabajo se ha presentado ReduxSTM, un sistema TM software con soporte específico para operaciones de reducción, un patrón común en muchas aplicaciones irregulares. Las características clave de ReduxSTM son que los commits de las transacciones se realizan en el orden equivalente al secuencial y que se utiliza la privatización subyacente al TM para filtrar los conflictos derivados de las operaciones de reducción cuando es posible, disminuyendo así el número de potenciales conflictos y, por tanto, de abortos. Comparado con técnicas clásicas de paralelización de bucles de reducción, se ha comprobado que los enfoques STM son un buen compromiso entre facilidad de programación, paralelismo explotado y sobrecarga de memoria extra necesaria; y que es posible mejorar el rendimiento de los STM si se añade soporte específico para reducciones, especialmente en patrones con una alta contención.

AGRADECIMIENTOS

Este trabajo ha recibido soporte del Gobierno de España (proyecto TIN2013-42253-P) y de la Junta de Andalucía (proyecto P12-TIC-1470).

REFERENCIAS

- [1] M. Herlihy and J.E.B. Moss, "Transactional Memory: Architectural support for lock-free data structures," in *20th Ann. Int'l. Symp. on Computer Architecture (ISCA'93)*, San Diego, CA, USA, 1993, pp. 289–300.
- [2] T. Harris, J. Larus, and R. Rajwar, *Transactional Memory, 2nd Ed.*, Morgan & Claypool Publishers, USA, 2010.
- [3] L. Porter, B. Choi, and D.M. Tullsen, "Mapping out a path from hardware transactional memory to speculative multithreading," in *18th Int'l. Conf. on Parallel Architectures and Compilation Techniques (PACT'09)*, Raleigh, NC, USA, 2009.
- [4] N.P. Johnson, H. Kim, P. Prabhau, A. Zaks, and D.I. August, "Speculative separation for privatization and reductions," in *33rd ACM Conf. on Programming Language Design and Implementation (PLDI'12)*, Beijing, China, 2012, pp. 359–370.

- [5] Lawrence Rauchwerger, "Speculative parallelization of loops," in *Encyclopedia of Parallel Computing*, pp. 1901–1912, Springer, 2011.
- [6] L. Han, W. Liu, and J.M. Tuck, "Speculative parallelization of partial reduction variables," in *8th Ann. IEEE/ACM Int'l. Symp. on Code Generation and Optimization (CGO'10)*, Toronto, Canada, 2010, pp. 141–150.
- [7] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, and T. Lawrence, "Parallel programming with Polaris," *IEEE Computer*, vol. 29, no. 12, pp. 78–82, 1996.
- [8] M.W. Hall, J.M. Anderson, S.P. Amarasinghe, B.R. Murphy, S.W. Liao, and E. Bu, "Maximizing multiprocessor performance with the SUIF compiler," *IEEE Computer*, vol. 29, no. 12, pp. 84–89, 1996.
- [9] P. Feautrier, "Array expansion," in *2nd Int'l Conf. on Supercomputing (ICS'88)*, Saint Malo, France, 1988, pp. 429–441.
- [10] H. Yu and L. Rauchwerger, "An adaptive algorithm selection framework for reduction parallelization," *IEEE Trans. on Parallel and Distributed Systems*, vol. 17, no. 10, pp. 1084–1096, 2006.
- [11] Josep M Perez, Rosa M Badia, and Jesus Labarta, "A dependency-aware task-based programming environment for multi-core architectures," in *10th IEEE Int'l Conf. on Cluster Computing*, 2008, pp. 142–151.
- [12] Carlos H González and Basilio B Fraguela, "A framework for argument-based task synchronization with automatic detection of dependencies," *Parallel Computing*, vol. 39, no. 9, pp. 475–489, 2013.
- [13] Lawrence Rauchwerger and David A Padua, "The LRPD test: Speculative run-time parallelization of loops with privatization and reduction parallelization," *IEEE Trans. on Parallel and Distributed Systems*, vol. 10, no. 2, pp. 160–180, 1999.
- [14] M. Gonzalez-Mesa, R. Quisiant, E. Gutierrez, and O. Plata, "Dealing with reduction operations using transactional memory," in *25th Int'l. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD'13)*, Porto de Galinhas, Brasil, 2013, pp. 128–135.
- [15] M.A. Gonzalez-Mesa, E. Gutierrez, E.L. Zapata, and O. Plata, "Effective transactional memory execution management for improved concurrency," *ACM Trans. on Architecture and Code Optimization*, vol. 11, no. 3, pp. 24:1–24:27, 2014.
- [16] C. von Praun, C. Ceze, and C. Cascaval, "Implicit parallelism with ordered transactions," in *12th ACM Symp. on Principles and Practice of Parallel Programming (PLDI'07)*, San Diego, CA, USA, 2007, pp. 79–89.
- [17] A. Udupa, K. Rajan, and W. Thies, "ALTER: Exploiting breakable dependencies for parallelization," *32nd ACM Conf. on Programming Language Design and Implementation (PLDI'11)*, pp. 480–491, 2011.
- [18] M. DeVuyst, D.M. Tullsen, and S.W. Kim, "Runtime parallelization of legacy code on a transactional memory system," in *6th Int'l. Conf. on High Performance and Embedded Architectures and Compilers (HiPEAC'11)*, Heraklion, Greece, 2011, pp. 127–136.
- [19] Wenjia Ruan, Yujie Liu, and Michael Spear, "Transactional Read-Modify-Write without aborts," *ACM Trans. on Architecture and Code Optimization*, vol. 11, no. 4, pp. 1–24, 2015.
- [20] Ricardo Quisiant, Eladio Gutierrez, Emilio L Zapata, and Oscar Plata, "Conflict detection in hardware transactional memory," in *Transactional Memory, Foundations, Algorithms, Tools, and Applications*, pp. 127–149, Springer, 2015.
- [21] J.E. Gottschlich, M. Vachharajani, and J.G. Siek, "An efficient software transactional memory using commit-time invalidation," in *8th Ann. IEEE/ACM Int'l. Symp. on Code Generation and Optimization (CGO'10)*, Toronto, Canada, 2010, pp. 101–110.
- [22] Pascal Felber, Christof Fetzer, Patrick Marlier, and Torvald Riegel, "Time-based software transactional memory," *IEEE Trans. on Parallel and Distributed Systems*, vol. 21, no. 12, pp. 1793–1807, 2010.
- [23] C. Bienia, S. Kumar, J.P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *17th Int'l. Conf. on Parallel Architectures and Compilation Techniques (PACT'08)*, Toronto, Canada, 2008, pp. 72–81.
- [24] J.J. Morales and S. Toxvaerd, "The Cell-Neighbour table method in molecular dynamics simulations," *Computer Physics Communications*, vol. 71, pp. 71–76, 1992.
- [25] Ian Foster, Rob Schreiber, and Paul Havlak, "HPF-2, scope of activities and motivating applications," Tech. Rep., Technical Report CRPC-TR94492, Rice University, 1994.
- [26] N. Mukherjee and J.R. Gurd, "A comparative analysis of four parallelisation schemes," in *13th Int'l. Conf. on Supercomputing (ICS'99)*, Portland, OR, USA, 1999, pp. 278–285.
- [27] Manohar K. Prabhu and Kunle Olukotun, "Exposing speculative thread parallelism in SPEC2000," in *10th ACM Symp. on Principles and Practice of Parallel Programming (PPoPP'05)*, Chicago, IL, USA, 2005, p. 142.
- [28] M. Dai Wang, M. Durcea, L. Li, et al., "Exploring the performance and programmability design space of hardware transactional memory," in *ACM Workshop on Transactional Computing (TRANSACT'14)*, Salt Lake City, UT, USA, 2014.

Extendiendo rCUDA con soporte para copias P2P entre GPUs remotas

Carlos Reaño¹ y Federico Silla¹

Resumen— Las unidades de procesamiento gráfico (GPUs, *Graphics Processing Units*) son actualmente utilizadas en muchos centros de supercomputación para acelerar aplicaciones paralelas. Sin embargo, el uso de GPUs para este propósito presenta algunas desventajas, tales como el aumento de los costes de adquisición así como mayor requerimiento de espacio. Además, suelen tener una baja utilización y, por otro lado, también consumen energía cuando están ociosas. Para hacer frente a estos inconvenientes, surge lo que se conoce como virtualización remota de GPUs, una técnica consistente en compartir GPUs entre los diferentes nodos de un supercomputador. La viabilidad de esta aproximación ya ha sido demostrada en anteriores trabajos. En el presente artículo extendemos una de las soluciones de virtualización remota de GPUs existentes, rCUDA, introduciendo el soporte para copias P2P (*peer-to-peer*, de igual a igual) entre zonas de memoria de GPUs remotas ubicadas en diferentes nodos del supercomputador.

Palabras clave— CUDA, GPU, evaluación, prestaciones

I. INTRODUCCIÓN

Las unidades de procesamiento gráfico (GPUs, *Graphics Processing Units*) son actualmente utilizadas en muchos centros de supercomputación para acelerar aplicaciones paralelas en ámbitos tan diversos como, por ejemplo, álgebra computacional [1], biología [2], finanzas [3], inteligencia artificial [4], cosmología [5], predicción meteorológica [6], etc.

Sin embargo, el uso de GPUs para este propósito presenta algunas desventajas. En primer lugar, este tipo de GPUs destinadas a centros de supercomputación presenta un coste muy superior en comparación con las GPUs convencionales. A esto hay que añadir el hecho de que ocupan un mayor espacio e incrementan la temperatura de los servidores. Además, suelen tener una baja utilización, ya que los fragmentos de las aplicaciones que pueden ejecutarse de forma paralela no suelen ser muy significativos. Por otro lado, también hay que tener en cuenta que las GPUs consumen una cantidad nada despreciable de energía incluso cuando no están siendo utilizadas. Con el objetivo de hacer frente a estos inconvenientes, surgió hace unos años lo que se ha denominado virtualización remota de GPUs. Esta técnica consistente en compartir GPUs entre los diferentes nodos de un supercomputador. De este modo, se reducirían los costes de adquisición y se aumentaría la utilización de las GPUs. Si bien la viabilidad de esta aproximación ya ha sido demostrada en anteriores trabajos [7], en el presente artículo extendemos una de las soluciones de virtualización remota de GPUs existentes,

rCUDA [8], introduciendo el soporte para copias P2P (*peer-to-peer*, de igual a igual) entre zonas de memoria de GPUs remotas ubicadas en diferentes nodos del supercomputador.

El resto del artículo está estructurado de la siguiente manera. Primeramente, en la Sección II, describimos con más detalle rCUDA. En segundo lugar, la Sección III analiza estudios previos relacionados con copias de memoria entre GPUs remotas. A continuación, en la Sección IV, mostramos cómo se ha extendido rCUDA para dar soporte a este tipo de copias. Seguidamente, la Sección V, presenta una evaluación del rendimiento obtenido por el mecanismo desarrollado. Por último, en la Sección VI, comentamos las principales conclusiones de este artículo.

II. rCUDA: REMOTE CUDA

CUDA [9] (*Compute Unified Device Architecture*, arquitectura de dispositivos de cómputo unificado) es una tecnología desarrollada por NVIDIA que proporciona una arquitectura de cálculo paralelo. Esta arquitectura aprovecha la gran potencia de las GPUs para acelerar determinadas partes de las aplicaciones, con la consecuente reducción en el tiempo de ejecución.

rCUDA (remote CUDA, CUDA remoto) [8], [10], [11] es una tecnología que permite el uso remoto de GPUs compatibles con CUDA. De esta forma, una GPU instalada en un nodo de un cluster (nodo servidor) puede ser utilizada por todos los nodos del cluster (nodos clientes) para acelerar aplicaciones que utilicen CUDA.

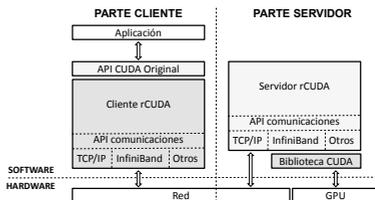


Fig. 1. Arquitectura de rCUDA.

En la Figura 1 podemos ver la arquitectura de rCUDA. Cuando una aplicación CUDA se ejecuta en un nodo cliente, dicha aplicación continúa utilizando la API original de CUDA. El cliente rCUDA intercepta las llamadas a dicha API y las redirige al servidor rCUDA a través de la red de comunicaciones.

¹DISCA, Universitat Politècnica de València (UPV), 46.022, Valencia (España). E-mail: carregon@gap.upv.es, fsilla@disca.upv.es.

La comunicación entre cliente y servidor puede realizarse mediante Ethernet (TCP/IP) o utilizando InfiniBand, en caso de que el cluster disponga de este tipo de red. No obstante, la capa de comunicaciones de rCUDA posee una API común que permite el uso de otros tipos de tecnologías de comunicación. Para ello únicamente habría que desarrollar un nuevo módulo que implemente la API de comunicaciones comentada. El hecho de que la capa de comunicaciones esté desacoplada del resto de capas de rCUDA permite optimizar las comunicaciones para cada tecnología concreta, obteniendo así los mejores resultados para cada una de ellas.

Una vez la llamada CUDA ha llegado desde el cliente al servidor rCUDA, se ejecuta la llamada utilizando la biblioteca de CUDA original y accediendo, por lo tanto, a la GPU física. Cuando la llamada CUDA finaliza, el resultado es devuelto al cliente rCUDA y, finalmente, a la aplicación que inició el proceso.

La última versión de rCUDA disponible, la versión 15.07, soporta la versión 7.0 de CUDA, a excepción de los módulos de interoperabilidad con gráficos. Además, también soporta bibliotecas CUDA específicas para diferentes áreas, tales como la existente para BLAS (Basic Linear Algebra Subprograms, subrutinas de álgebra lineal básica), denominada CUBLAS, o para FFT (Fast Fourier Transform, transformada rápida de Fourier), llamada CUFFT. Para utilizar rCUDA no es necesario realizar ninguna modificación en las aplicaciones. rCUDA puede obtenerse de forma gratuita en la web: www.rcuda.net.

Anteriormente a este trabajo, rCUDA no soportaba las copias de memoria entre GPUs remotas ubicadas en diferentes nodos de un cluster. En las próximas secciones exploraremos las diferentes alternativas existentes para soportar este tipo de copias y presentaremos el mecanismo finalmente implementado.

III. TRABAJOS RELACIONADOS

Con el fin de copiar eficientemente datos entre las memorias de GPUs ubicadas en nodos diferentes de un mismo cluster, NVIDIA introdujo *GPUDirect RDMA* [12] en 2012. Se trata de una tecnología que, utilizando características estándares del bus PCI Express (PCIe), proporciona una ruta directa para el intercambio de datos entre una GPU y un dispositivo de terceros, como, por ejemplo, una tarjeta de red InfiniBand (ver Figura 2).

Por lo que respecta a las tarjetas de red InfiniBand utilizadas en este artículo, fabricadas y distribuidas por Mellanox, el soporte para GPUDirect RDMA fue introducido [13] con el objetivo de proporcionar redes InfiniBand de altas prestaciones para la comunicación entre GPU.

Existe un extenso análisis de prestaciones [14] de NVIDIA GPUDirect RDMA sobre InfiniBand que muestra las capacidades reales de esta tecnología cuando se utiliza en plataformas modernas. La Tabla I presenta un resumen de los resultados relativos al ancho de banda extraídos de dicho análisis. Según

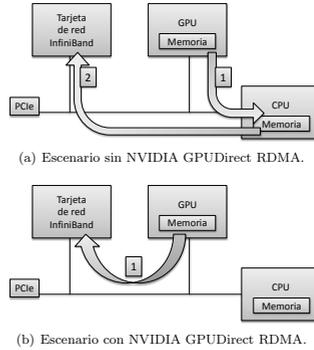


Fig. 2. Escenario con y sin NVIDIA GPUDirect RDMA utilizado con una tarjeta de red InfiniBand.

los autores de dicho estudio, el equipamiento utilizado estaba compuesto por dos servidores con procesadores Ivy Bridge Xeon (E5-2690v2 a 3.00GHz), tarjetas gráficas NVIDIA Tesla K40m y tarjetas de red de doble puerto Mellanox Connect-IB (donde cada puerto ofrece prestaciones FDR).

Los resultados de la Tabla I muestran el ancho de banda en GB/s obtenido al copiar mensajes de 64KB desde memoria principal a memoria principal (RDMA convencional, sin involucrar a las GPUs), y desde memoria de GPU a memoria de GPU (GPUDirect RDMA). Se barajan diferentes escenarios, dependiendo de la ruta existente entre la GPU y la tarjeta de red:

- Intra-socket: las GPUs y las tarjetas de red están ubicadas en el mismo socket en ambos servidores.
- Inter-socket: en uno de los servidores (servidor 1), la GPU y la tarjeta de red están instalados en diferentes sockets; en el otro servidor (servidor 2), la GPU y la tarjeta de red están instalados en el mismo socket. TX hace referencia al caso en el que los datos son copiados desde memoria de GPU del servidor 1 a memoria de GPU en el servidor 2. RX hace referencia al caso inverso.
- PCIe switch: las GPUs y la tarjetas de red están instaladas utilizando una *riser-card* y han sido conectadas a un switch PCIe.

Los resultados etiquetados como “FDR” fueron realizados utilizando sólo 1 puerto de la tarjeta de red de doble puerto Connect-IB, mientras que los resultados marcados como “FDRx2” fueron realizados usando ambos puertos.

Tal y como podemos ver, GPUDirect RDMA introduce, en general, una importante pérdida de prestaciones con respecto al RDMA utilizado habitualmente (usando memoria principal, referida como de CPU en la tabla, en lugar de memoria GPU). También po-

TABLA I
 RESUMEN DE LOS RESULTADOS RELATIVOS AL ANCHO DE BANDA EXTRAÍDOS DEL ANÁLISIS DE PRESTACIONES DE NVIDIA GPUDIRECT RDMA.

Ruta entre GPU y tarjeta red	Ancho de banda en GB/s por dirección de copia de memoria	
	CPU a CPU	GPU a GPU
Intra-socket (FDR)	6,1	3,7
Intra-socket (FDRx2)	12,3	3,7
Inter-socket (FDR)	6,1	1,1(TX)/0,25(RX)
PCIe switch (FDR)	6,1	5,8
PCIe switch (FDRx2)	12,3	7

demostremos observar que esta pérdida de prestaciones se ve claramente afectada por la ruta que atraviesan los datos. De esta manera, los mejores resultados se obtienen en el escenario en el que se usa el switch PCIe, donde el ancho de banda utilizando un puerto está próximo al ancho de banda obtenido en copias de memoria principal a memoria principal (5,8GB/s y 6,1GB/s, respectivamente). Sin embargo, cuando se utilizan los dos puertos de la tarjeta de red, se produce una clara bajada de prestaciones y el ancho de banda es muy inferior en comparación con el obtenido con memoria principal (7GB/s y 12,3GB/s, respectivamente). En el caso del escenario intra-socket, el ancho de banda disminuye a 3,7GB/s. Los autores del análisis señalan que la razón de dicha bajada puede deberse a que la interfaz PCIe integrada en la CPU está limitando el número de transacciones en vuelo. Por lo tanto, si no hay suficientes transacciones pendientes el ancho de banda de lectura pasa a estar limitado por la latencia. Los peores resultados son obtenidos en el escenario inter-socket, donde la pérdida de prestaciones es más evidente: 1,1GB/s y 0,25GB/s para TX y RX, respectivamente.

Los autores del referido análisis también proponían resultados en cuanto a la latencia se refiere. Para ello, han analizado la latencia al copiar mensajes de 4 bytes de tamaño. De esta manera, la latencia obtenida al copiar de memoria principal a memoria principal, utilizando el RDMA convencional, es de 1,3 μ s; mientras que la latencia obtenida al copiar datos de memoria GPU a memoria GPU es de 1,9 μ s en todos los escenarios. La única excepción la encontramos en el escenario inter-socket, donde las copias para el caso RX resultaron en una latencia de 2,2 μ s.

Finalmente, los autores de dicho estudio concluyen que:

- Ancho de banda: según los autores, GPUDirect RDMA sería más rápido que una solución basada en copias intermedias usando RDMA, y copias entre memoria principal local y memoria de GPU local, para mensajes de tamaño hasta 400KB-500KB. Para tamaños de copia mayores, la aproximación basada en copias intermedias usando RDMA probablemente obtenga mejores prestaciones.
- Latencia: GPUDirect RDMA proporciona una baja latencia que, en general, se encuentra

por debajo de 2 μ s, la cual es mejor que una solución basada en copias intermedias usando RDMA. Según los autores, dicha solución usaría copias sincronas (`cudaMemcpy`) o asincronas (`cudaMemcpyAsync`), las cuales tienen una latencia entorno a 8 μ s y 9 μ s, respectivamente. A dicha latencia, habría que sumar la latencia de copiar de memoria principal a memoria principal con RDMA convencional de InfiniBand: 1,3 μ s. De esta manera, la latencia total para copiar de memoria de una GPU 1 a memoria de otra GPU 2 sería: 8 μ s (`cudaMemcpy` de GPU 1 a CPU 1) + 1,3 μ s (copia de CPU 1 a CPU 2) + 8 μ s (`cudaMemcpy` de CPU 2 a GPU 2). Lo cual es claramente mayor que la latencia obtenida al utilizar GPUDirect RDMA.

IV. SOPORTE DE COPIAS P2P EN RCUA

En esta sección describimos cómo hemos incluido el soporte para copias de memoria entre GPUs remotas en rCUDA.

En la Figura 3 podemos ver una comparativa entre el escenario habitual con CUDA y los posibles escenarios con rCUDA cuando queremos realizar copias de memoria entre GPUs. Tal y como podemos observar, en el escenario con CUDA, Figura 3(a), las GPUs se encuentran conectadas por PCIe dentro de un mismo nodo. Sin embargo, en los escenarios con rCUDA tenemos dos posibilidades: (1) las GPUs remotas se encuentran en el mismo nodo, ver Figura 3(b); (2) las GPUs están en nodos diferentes conectadas por la red existente, Figura 3(c). En este trabajo asumiremos que se trata de una red InfiniBand.

Previamente a este trabajo, rCUDA ya soportaba el escenario 1 expuesto en la Figura 3(b). Así, era posible realizar copias de memoria entre GPUs remotas instaladas en el mismo nodo. Sin embargo, el escenario 2 expuesto en la Figura 3(c) no estaba soportado y es la contribución del presente trabajo. Para dar soporte a este tipo de copias en dicho escenario, hemos implementado un mecanismo que permite realizar copias de memoria entre la GPU1 y la GPU2, ver Figura 3(c), de forma transparente al usuario. De este modo, la aplicación original CUDA no tiene que ser modificada y el usuario tiene la misma experiencia que si se encontrara en el escenario de CUDA original, mostrado en la Figura 3(a).

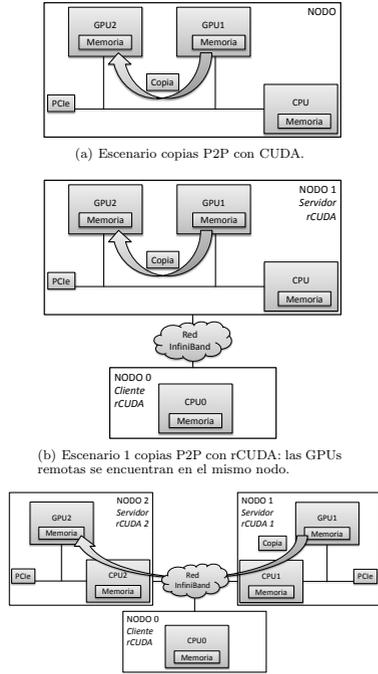


Fig. 3. Comparativa de los escenarios con CUDA y rCUDA para copias entre GPUs.

Con el fin de comparar prestaciones, se han desarrollado dos versiones diferentes:

- Versión “rCUDA P2P GDR”: las copias entre zonas de memoria de dos GPUs remotas se realizan utilizando la tecnología NVIDIA GPUDirect RDMA comentada en secciones anteriores
- Versión “rCUDA P2P RDMA”: las copias entre zonas de memoria de dos GPUs remotas se realizan utilizando una aproximación basada en copias intermedias usando RDMA.

En el caso de la versión “rCUDA P2P GDR”, la copia se realiza directamente desde la memoria de la GPU 1 en el Nodo 1, hasta la memoria de la GPU 2 del Nodo 2. En el caso de la versión “rCUDA P2P RDMA”, los pasos para realizar una copia serían los siguientes:

1. Copia de memoria de la GPU 1 a memoria principal del Nodo 1
2. Copia RDMA de memoria principal del Nodo 1 a memoria principal del Nodo 2
3. Copia de memoria principal del Nodo 2 a memoria de la GPU 2

V. EVALUACIÓN

En esta sección evaluamos las prestaciones del trabajo realizado. En primer lugar, en la Sección V-A presentamos los equipos utilizados en las pruebas y su configuración. A continuación, en la Sección V-B comparamos el rendimiento de CUDA y rCUDA.

A. Equipamiento utilizado

Para los experimentos con CUDA presentados a lo largo de esta sección hemos utilizado un servidor Supermicro SYS7047GR-TRF equipado con:

- 2 procesadores Intel Xeon E5-2620v2 de 6 núcleos a 2.10GHz
- 128 GB de memoria SDRAM DDR3 a 1600 MHz
- 4 GPUs NVIDIA Tesla K20m
- CentOS Linux Distribution 6.4
- CUDA 7.0 con NVIDIA driver 346.46

Para los experimentos con rCUDA presentados a lo largo de esta sección hemos utilizado tres servidores Supermicro 1027GR-TRF equipados con:

- 2 procesadores Intel Xeon E5-2620v2 de 6 núcleos a 2.10GHz
- 32 GB de memoria SDRAM DDR3 a 1600 MHz
- 1 GPU NVIDIA Tesla K20m
- CentOS Linux Distribution 6.4
- CUDA 7.0 con NVIDIA driver 346.46

Los servidores están interconectados mediante una red InfiniBand FDR a través de un switch Mellanox SX6025.

B. Ancho de banda y latencia en copias de memoria

En esta sección evaluamos el rendimiento de las copias de memoria entre GPUs con un test para medir el ancho de banda y la latencia. Para ello utilizamos el programa *bandwidthTest*, que se distribuye en el paquete NVIDIA CUDA Samples [15].

Las pruebas han sido realizadas utilizando diferentes escenarios:

- Pruebas con CUDA: la configuración utilizada ha sido similar a la mostrada en la Figura 3(a) de la Sección IV. De esta forma, se ha medido el ancho de banda y la latencia en copias de la memoria de la GPU 1 a la memoria de la GPU 2.
- Pruebas con rCUDA: la configuración utilizada ha sido similar a la mostrada en la Figura 3(c) de la Sección IV. De esta forma, se ha medido el ancho de banda y la latencia en copias de la memoria de la GPU 1 del Nodo 1, a la memoria de la GPU 2 del Nodo 2. El programa CUDA para medir dichos parámetros ha sido ejecutado en el Nodo 0, accediendo a las GPUs remotas gracias a rCUDA.

Los resultados de dichos experimentos los podemos ver en la Figura 4. En primer lugar, en la Figura 4(a), mostramos el ancho de banda en copias de memoria de gran tamaño (hasta 64MB). Como podemos ver, la versión de rCUDA utilizando GPUDirect RDMA, etiquetada como “rCUDA P2P GDR”, tiene un

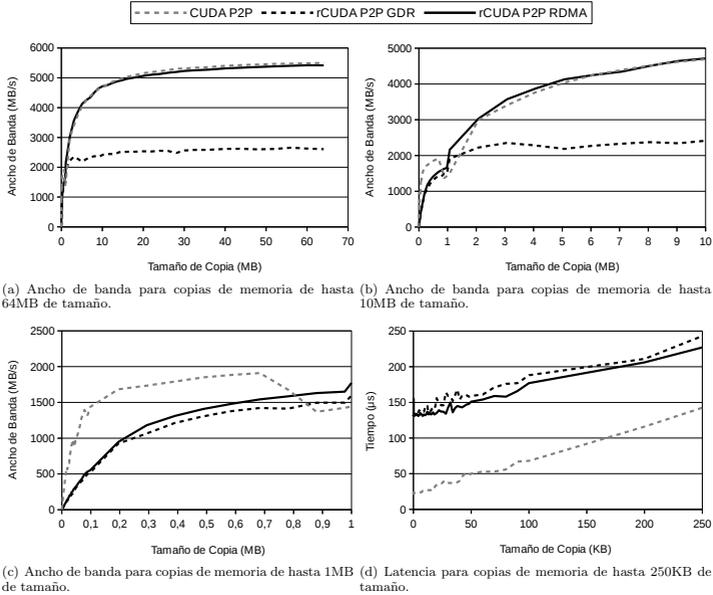


Fig. 4. Ancho de banda y latencia obtenidos en copias de memoria entre dos GPUs. En los experimentos con CUDA, etiquetados como “CUDA P2P” las copias se realizaron entre zonas de memoria de dos GPUs locales. En los experimentos con rCUDA, etiquetados como “rCUDA P2P GDR” y “rCUDA P2P RDMA”, las copias se realizaron entre zonas de memoria de dos GPUs remotas. Los resultados etiquetados con “rCUDA P2P GDR” hacen referencia a la versión de rCUDA implementada utilizando GPUDirect RDMA. Los resultados etiquetados con “rCUDA P2P RDMA” hacen referencia a la versión de rCUDA implementada utilizando RDMA convencional.

ancho de banda muy inferior a la versión basada en copias intermedias usando RDMA, etiquetada como “rCUDA P2P RDMA”. Estos resultados están alineados con las conclusiones extraídas de trabajos previos comentados en la Sección III, donde algunos autores ya señalaban que GPUDirect RDMA probablemente obtendría peores prestaciones que una solución basada en copias intermedias usando RDMA, para mensajes de tamaño superiores a 400KB-500KB. Respecto a la comparativa de rCUDA y CUDA, observamos que prácticamente se obtienen las mismas prestaciones para tamaños de copia grande cuando se utiliza la versión “rCUDA P2P RDMA”.

En segundo lugar, en la Figura 4(b), presentamos el ancho de banda en copias de memoria de un tamaño medio (hasta 10MB). Nuevamente, observamos que la versión de rCUDA utilizando GPUDirect RDMA (“rCUDA P2P GDR”) tiene un ancho de banda muy inferior a la versión que usa RDMA (“rCUDA P2P RDMA”). Al igual que ocurría con las copias de gran tamaño, estos resultados son los esperados por los mismos motivos expuestos anteriormente. En relación a los resultados de rCUDA en comparación con los de CUDA, el ancho de banda obtenido es muy si-

milar. Aunque si bien para copias grandes CUDA superaba ligeramente a rCUDA, en el caso de copias de tamaño medio, es rCUDA quien supera ligeramente a CUDA en algunos tramos.

A continuación, en la Figura 4(c) mostramos el ancho de banda en copias de memoria de un tamaño pequeño (hasta 1MB). En este caso, la diferencia entre ambas versiones de rCUDA es mucho menor, siendo las prestaciones muy similares hasta tamaños de copia de 200KB. A partir de ese tamaño, la versión que utiliza GPUDirect RDMA (“rCUDA P2P GDR”) se va distanciando cada vez más de la versión basada en RDMA (“rCUDA P2P RDMA”), la cual obtiene mejores prestaciones. Por lo que respecta a la comparativa de rCUDA y CUDA, podemos observar cómo para tamaños de copia de hasta 700KB, CUDA supera claramente a rCUDA. A partir de ese tamaño, se produce una clara bajada de prestaciones en CUDA. El test muestra la media de 10 repeticiones, por lo que asumimos que dicha bajada no se debe a una situación puntual.

Nótese que en el caso de rCUDA no se produce dicha bajada ya que las copias se realizan utilizando mecanismos diferentes:

- Copias con CUDA: se realizan usando llamadas a la API de CUDA. Los datos atraviesan el bus PCIe.
- Copias con rCUDA: se realizan usando llamadas a la API de InfiniBand Verbs. Los datos atraviesan la red InfiniBand, además de los buses PCIe de los dos nodos implicados en la copia.

Por último, en la Figura 4(d), podemos observar la latencia obtenida en copias de memoria de un tamaño muy pequeño (hasta 250KB). Como podemos observar, ambas versiones de rCUDA presentan prestaciones similares, aunque la versión que utiliza RDMA (“rCUDA P2P RDMA”) es algo mejor que la que usa GPUDirect RDMA (“rCUDA P2P GDR”). Estos resultados no eran los esperados y entran en contradicción con las conclusiones extraídas de trabajos previos comentados en la Sección III, donde algunos autores indicaban que GPUDirect RDMA, probablemente, obtendría mejores prestaciones que una solución basada en copias intermedias usando RDMA para mensajes de tamaño inferiores a 400KB-500KB. Con respecto a los resultados de rCUDA en comparación con los de CUDA, observamos que CUDA presenta una latencia muy inferior a la obtenida con rCUDA. Este sobrecoste en copias de tamaño muy pequeño es habitual en la tecnología rCUDA, la cual se ve penalizada cuando el tamaño de datos es muy pequeño. Ello se debe a que para compensar la inicialización del mecanismo de copia a la GPU remota, es necesario una cantidad mínima de datos. En el caso de los escenarios aquí tratados, ese tamaño de copia estaría en torno a los 700KB.

VI. CONCLUSIONES

En el presente artículo hemos extendido rCUDA, una de las soluciones de virtualización remota de GPUs existentes, introduciendo el soporte para copias P2P (*peer-to-peer*, de igual a igual) entre zonas de memoria de GPUs remotas ubicadas en diferentes nodos del supercomputador.

Se han presentado dos mecanismos diferentes. Uno de ellos utilizando la tecnología NVIDIA GPUDirect RDMA, que permite realizar copias entre zonas de memoria de GPUs remotas a través de dispositivos de terceros, como puede ser el caso de las tarjetas de red InfiniBand utilizadas en este trabajo. Alternativamente, se ha presentado otro mecanismo utilizando copias intermedias con RDMA convencional.

Los resultados demuestran que la solución basada en copias intermedias con RDMA convencional ofrece mejores prestaciones que la solución que utiliza NVIDIA GPUDirect RDMA. Incluso para tamaños de copias muy pequeños, donde la latencia juega un papel más importante que el ancho de banda, el mecanismo usando RDMA convencional presenta mejores resultados.

AGRADECIMIENTOS

El presente trabajo ha sido subvencionado por el Ministerio de Economía y Competitividad español

(MINECO) y por fondos europeos FEDER (Fondo Europeo de Desarrollo Regional) bajo el acuerdo TIN2015-66972-C5-1-R. Asimismo, los autores agradecen también a Mellanox Technologies el generoso apoyo recibido.

REFERENCIAS

- [1] Ichitaro Yamazaki, Tingxing Dong, Raffaele Solcà, Staninire Tomov, Jack Dongarra y Thomas Schultness, *Tri-diagonalization of a dense symmetric matrix on multiple GPUs and its application to symmetric eigenvalue problems*, *Concurrency and Computation: Practice and Experience*, vol. 26, no. 16, 2014.
- [2] Pratul K. Agarwal, Scott Hampton, Jeffrey Poznaniovic, Arvind Ramanathan, Sadaf R. Alam y Paul S. Crozier, *Performance modeling of microsecond scale biological molecular dynamics simulations on heterogeneous architectures*, *Concurrency and Computation: Practice and Experience*, vol. 25, no. 10, 2013.
- [3] Vladimir Surkov, *Parallel option pricing with Fourier space time-stepping method on graphics processing units*, *Parallel Computing*, vol. 36, no. 7, 2010.
- [4] Guo-Heng Luo, Sheng-Kai Huang, Yue-Shan Chang y Shyan-Ming Yuan, *A parallel Bees Algorithm implementation on GPU*, *Journal of Systems Architecture*, vol. 60, no. 3, 2014.
- [5] Yueqing Wang, Yong Dou, Song Guo, Yuanwu Lei y Dan Zou, *CPU-GPU hybrid parallel strategy for cosmological simulations*, *Concurrency and Computation: Practice and Experience*, vol. 26, no. 3, 2014.
- [6] V.T. Vu, G. Cats, y L. Wolters, *Graphics processing unit optimizations for the dynamics of the HIRLAM weather forecast model*, *Journal of Systems Architecture*, vol. 25, no. 10, 2013.
- [7] C. Reaño, F. Silla, G. Shainer, y S. Schultz, *Local and Remote GPUs Perform Similar with URG 100G InfiniBand*, en *International Middleware Conference (Middleware)*, 2015.
- [8] A. J. Peña, C. Reaño, F. Silla, R. Mayo, E. S. Quintana-Ortí y J. Duato, *A complete and efficient CUDA-sharing solution for HPC clusters*, *Parallel Computing*, vol. 40, no. 10, 2014.
- [9] NVIDIA, *The NVIDIA CUDA API Reference Manual Version 7.0*, NVIDIA, 2015.
- [10] C. Reaño, A. J. Peña, F. Silla, R. Mayo, E. S. Quintana-Ortí y J. Duato, *Influence of InfiniBand FDR on the Performance of Remote GPU Virtualization*, en *IEEE International Conference on Cluster Computing (Cluster)*, 2013.
- [11] C. Reaño, F. Silla, A. Castelló Gimeno, A. J. Peña, R. Mayo, E. S. Quintana-Ortí y J. Duato, *Improving the user experience of the rCUDA remote GPU virtualization framework*, *Concurrency and Computation: Practice and Experience*, vol. 27, no. 14, 2015.
- [12] NVIDIA, *Developing a linux kernel module using GPUDirect RDMA*, Disponible online: <http://docs.nvidia.com/cuda/gpudirect-rdma/index.html#ixzz42Pa6GGX>. Accedido por última vez: 15/05/2016.
- [13] Mellanox, *OFED GPUDirect RDMA Product Brief*, Disponible online: http://www.mellanox.com/related-docs/prod_software/PB_GPUDirect_RDMA.PDF. Accedido por última vez: 15/05/2016.
- [14] Davide Rossetti, *Benchmarking GPUDirect RDMA on Modern Server Platforms*, Disponible online: <http://devblogs.nvidia.com/parallelforall/benchmarking-gpudirect-rdma-on-modern-server-platforms/>. Accedido por última vez: 15/05/2016.
- [15] NVIDIA, *NVIDIA CUDA Samples Version 7.0*, NVIDIA, 2015.

Programación Paralela Estructurada del paradigma Divide y Vencerás. Una Propuesta Metodológica

Mario Rossainz López¹, Manuel I. Capel², Fernando Hernández Polo¹, Ivo H. Pineda Torres¹

Resumen—El trabajo propone la implementación del paradigma de diseño algorítmico Divide y Vencerás como un Patrón de Diseño Paralelo en JAVA usando objetos activos y con ello proporcionar al programador una forma simple de solucionar problemas diversos que pueden ser resueltos con la misma estructura de control común paralela, tales como problemas de ordenación, problemas de búsqueda o problemas de máximos y mínimos. Los algoritmos paralelos implementados como Patrones de Diseño Paralelos se basan en el desarrollo y uso de una metodología donde cada diseño algorítmico representa la estructura paralela de control común a una determinada técnica algorítmica, en este caso Divide y Vencerás (pero no limitativa pudiendo extenderse a técnicas de Ramificación y Poda, Autómatas Celulares y Multiplicación de Tuplas por citar algunos) generando un programa paralelo general del cual a su vez se deriven dos o más programas modelo que resuelvan problemas particulares. La implementación del paradigma propuesto se hace de forma concurrente mediante la creación de hilos, explicando los pasos a realizar en el desarrollo de éste tipo de Patrones de Diseño Paralelos hasta llegar a una propuesta de programación usando el paradigma de la Orientación a Objetos para facilitar la reusabilidad, genericidad y uniformidad.

Palabras Clave—Algoritmos Paralelos, Concurrencia, Objetos Paralelos, Patrones de Diseño Paralelos, Programación Paralela Estructurada, Divide y Vencerás, Programación Orientada a Objetos.

I. INTRODUCCIÓN

Actualmente la construcción de sistemas paralelos y concurrentes tiene cada vez menos limitantes y su existencia está sostenida en la obtención de la eficiencia en el procesamiento de datos. La utilización de tales sistemas se ha extendido a muchas áreas de la Ciencia Computacional e Ingeniería. Existen infinidad de aplicaciones que tratan de obtener el máximo rendimiento del sistema al resolver un problema utilizando máquinas con un solo procesador; sin

embargo, cuando el sistema no puede proporcionar el rendimiento esperado, una solución es optar por aplicaciones, lenguajes de programación, arquitecturas e infraestructuras de procesamiento paralelo y concurrente. Para incrementar el rendimiento de determinados sistemas, el procesamiento paralelo es una alternativa al procesamiento secuencial. Desde el punto de vista práctico, hoy en día existen arquitecturas de computadores implementadas en sistemas reales que proporcionan auténtico procesamiento paralelo que justifican el interés actual en llevar a cabo investigaciones dentro del procesamiento paralelo y áreas a fines (concurrencia, sistemas distribuidos, sistemas de tiempo real, etc.). Una parte importante de estas investigaciones se refiere a mejorar el diseño de algoritmos paralelos y distribuidos [14], [17], el desarrollo de metodologías y modelos de programación paralela que ayuden a la implementación de estos algoritmos sobre sistemas físicamente distribuidos, así como herramientas de razonamiento que permitan garantizar su corrección [2], [12]. El presente trabajo se centra en éste tipo de problemas particularizando su atención en el estudio de modelos que nos permitan la programación de aplicaciones paralelas de una manera estructurada; para conseguirlo, se propone una metodología de programación paralela utilizando el paradigma de la orientación a objetos para resolver problemas cuyos algoritmos son susceptibles de paralelización de acuerdo con esquemas que permiten alcanzar un rendimiento esperado. Se propone entonces bajo la programación paralela estructurada y el paradigma de la orientación a objetos una implementación de la técnica de diseño algorítmica denominada Divide y Vencerás como un “Patrón de Diseño Algorítmico Paralelo” mediante el cual, más de un problema distinto que utiliza dicho paradigma, pueda ser resuelto con el mismo patrón, proporcionando únicamente el algoritmo de solución secuencial. El paradigma de Divide y Vencerás es visto entonces como un Patrón de Diseño Algorítmico Paralelo independiente del problema a resolver, que debe estar bien definido y estructurado de forma lógica de tal manera que una vez identificados todos los componentes que lo conforman, pueda programarse y estar disponible como una abstracción de alto nivel en las aplicaciones del usuario, donde gracias al uso de la programación paralela orientada a objetos éste último podrá explotar los mecanismos de generalización por herencia y parametrización de forma que se puedan definir nuevos patrones paralelos de diseño algorítmico tomando como base uno existente o bien uno nuevo siguiendo la metodología

¹Universidad Autónoma de Puebla, Avenida. San Claudio y 14 Sur, San Manuel, Puebla, Puebla, 72000, México. E-mail: {rossainz, ipineda}@cs.buap.mx, 01conker@gmail.com.

²DPT. Ingeniería de Software, Colegio de Informática y Telecomunicaciones, Universidad de Granada, ETSIT Calle Periodista Daniel Saucedo Aranda s/n, 18071 Granada, España. E-mail: manuelcapel@ugr.es

propuesta en las secciones siguientes. Se muestra, el modelo de programación desarrollado, el análisis, el diseño e implementación de los algoritmos que conforman el paradigma de Divide y Vencerás como un Patrón de Diseño Algorítmico Paralelo y con él se demuestra su utilidad, funcionamiento, genericidad y abstracción, resolviendo problemas clásicos que utilizan dicho paradigma tales como los problemas de ordenación, búsqueda y problemas de encontrar máximos y mínimos de un conjunto de datos.

II. EL PARALELISMO ESTRUCTURADO

El enfoque estructurado para la programación paralela se basa en el uso de patrones de comunicación/interacción y de diseño algorítmico predefinidos entre los procesos de una aplicación de usuario [8], [9]. El enfoque del paralelismo estructurado, partiendo de la abstracción del patrón de diseño algorítmico permite diseñar aplicaciones en términos de paradigmas capaces de implementar patrones como el de Divide y Vencerás, Pares Totales, Multiplicación de Tuplas, Programación Dinámica, Automatas Celulares y Ramificación y Acotación por citar algunos.

La encapsulación de un Patrón en estos términos debe seguir el principio de modularidad y debe proporcionar una base para obtener la reusabilidad efectiva del comportamiento paralelo de la entidad software que éste implementa. Cuando se logra hacer esto, se crea un patrón paralelo de diseño algorítmico genérico que proporciona una posible representación de un paradigma independiente del problema concreto a resolver. La aportación de este esquema es que, en lugar de programar una aplicación paralela desde el principio y de programar con un cierto nivel de detalle, tanto en la creación de los procesos que intervienen en ella como en el paradigma que se utilice para su solución, el usuario simplemente deberá identificar el patrón de diseño algorítmico adecuado para las necesidades de solución de su aplicación y los utilice junto con el código secuencial que implementa los cálculos que individualmente realizan sus procesos. Sin embargo, hay que tener en cuenta que la identificación y definición no-ambigua de un conjunto completo de patrones que representen paradigmas de programación de una aplicación paralela dista todavía mucho de ser un problema resuelto, ya que no existe un acuerdo lo suficientemente general que permita definir formalmente sus semánticas [8].

III. EL ENFOQUE DE LA ORIENTACIÓN A OBJETOS (OO)

La orientación a objetos parece ser una posible solución al problema de la necesidad de encontrar nuevos conceptos, métodos, herramientas y algoritmos que hagan frente a las dificultades inherentes de la programación paralela y distribuida. En particular, el paradigma de la OO se utiliza en la presente investigación para encapsular y abstraer patrones comunes de diseño algorítmico paralelos hacia un estilo de paralelismo estructurado. La idea básica de programación es considerar a los dichos patrones como objetos; además, un patrón de éste tipo debe ser definido como un objeto encargado de controlar y coordinar la ejecución de sus

componentes internos que siguen una arquitectura o estructura de datos definida, por ejemplo un pipeline, un árbol, una malla, un cubo, un hiper-cubo, etc.. Bajo estas premisas se logra definir un ambiente de Patrones de Diseño Algorítmicos extensible que pretende superar las limitaciones de otros enfoques de programación paralela disponibles proporcionando características tan importantes citadas en [11] como las que se listan a continuación:

- Uniformidad: Todas las entidades dentro del ambiente son objetos, incluyendo a los paradigmas como patrones de diseño algorítmicos los cuales son descritos a un nivel de “clase”, como cualquier otro componente de la aplicación, y cualquier nuevo patrón paralelo puede ser definido al mismo nivel en el que se definieron los primeros mediante la definición de una nueva clase.
- Generalidad: La capacidad de generar referencias dinámicas en un ambiente orientado a objetos hace posible crear clases de patrones de diseño genéricas mediante la definición de sus componentes siempre como referencias a objetos. Además se tiene la reusabilidad de estos componentes por el mecanismo de actualización genérica de su patrón y por el mecanismo de herencia en su parte secuencial, ya que la especificación de la parte paralela puede ser separada de cualquier especificación funcional de los componentes internos al patrón de diseño. El paradigma Divide y Vencerás por ejemplo, puede ser definido sin especificar qué problema estará resolviendo.
- Reusabilidad: El mecanismo de herencia simplifica la definición de patrones paralelos especializados. En principio, para definir un nuevo patrón el usuario tiene que explicitar en qué se verá afectado debido al paralelismo; sin embargo, la herencia aplicada a la adaptación del comportamiento del objeto a los requerimientos de la aplicación puede ayudar a hacer más concurrente la implementación de un Patrón de Diseño Algorítmico. Considérese, por ejemplo, que el entorno inicialmente proporciona el paradigma Divide y Vencerás como un patrón el cual visto como un objeto, sólo está a la espera del problema a resolver, entonces es muy simple para el usuario especializar el comportamiento de dicho patrón mediante la herencia y reusar el código ya implementado.

IV. LOS PARADIGMAS DE PROGRAMACIÓN VISTOS COMO PATRONES DE DISEÑO ALGORÍTMICO PARALELO

El estudio de los algoritmos paralelos se centra en la obtención de patrones diseño algorítmicos que puedan derivar en dos o más programas concretos, llamados programas modelo [1] los cuales resuelven problemas particulares. No obstante, no es trivial encontrar estructuras de control común a problemas distintos que clarifiquen la generación de un Patrón de Diseño Algorítmico Paralelo común a tales problemas, pues generalmente el problema que se intenta resolver es en principio dependiente de un patrón conocido lo que da la idea de tener un programa particular. Existen dificultades prácticas que surgen al intentar clarificar las técnicas de programación paralela que se aplican a numerosos problemas prácticos, tal

como lo señala [1] y que motivan a la investigación sobre esta propuesta tales como las siguientes: Es difícil suponer cual es la estructura algorítmica común a todos los algoritmos de un mismo patrón de diseño generado. Los patrones de diseño propuestos en la literatura son demasiado dependientes del problema a resolver y no proporcionan intuición práctica para derivar nuevos algoritmos utilizando la orientación a objetos. Estos patrones de diseño se entienden mejor como programas particulares más que como auténticos patrones.

El patrón de diseño algorítmico paralelo que se trabaja en la presente investigación es el Paradigma de Divide y Vencerás que describe la estructura de control común compartida por todos los algoritmos que corresponden al mismo patrón. Ni los tipos de datos, ni el código de procedimientos específicos dependientes de los datos necesitan ser detallados en este nivel, ya que éstos dependen de problemas concretos como pueden ser problemas de ordenación, de búsqueda u optimización por citar algunos. La idea es implementar el patrón divide y vencerás como una composición de objetos activos [7] generando un árbol de procesos con el problema original como raíz y los sub-problemas como hojas de dicho árbol. La implementación debe ser abstracta, es decir, genérica para que sea independiente del problema concreto que se pretende resolver [18]. Para demostrar su utilidad y genericidad se resuelven problemas de ordenación como el uso de QuickSort y MergeSort, de búsqueda usando el algoritmo de Búsqueda Binaria y de Máximo-Mínimo como instancias del paradigma Divide y Vencerás. En cada uno de estos problemas secuenciales, el componente paralelo es el mismo, esto es, el paradigma como Patrón de diseño algorítmico y su implementación consiste de los siguientes componentes:

- Estructuras de datos para representar el problema y la solución (pipelines, árboles, mallas, cubos, hiper-cubos, etc.) [15].
- Un predicado que indique cuando un problema es indivisible o si éste puede ser dividido en el futuro. Cada problema es representado por un proceso que actúa de forma independiente pero que está conectado con otros en base a la estructura de datos utilizada, donde dicha conexión lleva inmersa restricciones de comunicación tales como la exclusión mutua y sincronización de procesos del tipo productor-consumidor.
- Una función que regrese problemas indivisibles en soluciones.
- Una función que divida problemas en sub-problemas, generando la estructura de datos o arquitectura de procesos requerida en la solución del problema de forma dinámica.
- Una función que combine soluciones en una sola solución. El uso de un lenguaje de programación orientado a objetos como lo puede ser JAVA para este tipo de patrones de diseño provee un buen camino para expresar estos parámetros y conservar el paralelismo de forma dinámica generando las tareas en tiempo de ejecución y generar a los procesos mediante una estructura en forma de árbol [20].

V. METODOLOGÍA DE DERIVACIÓN DE UN PATRÓN DE DISEÑO ALGORÍTMICO PARALELO

Algunas clasificaciones sobre algoritmos paralelos o paradigmas las podemos encontrar en [1], [4], [5], [6], [10] y [19]. Tomando como base éstas propuestas definimos un paradigma como: una clase de algoritmos que resuelve diferentes problemas que tienen la misma estructura de control. Ejemplos de ello son los paradigmas mostrados en la tabla 1.

Paradigma	Programa	Arquitectura
DIVIDE Y VENCERÁS	1. Ordenación 2. Búsqueda	Árbol de Procesos
PARES TOTALES	1. House-Holder 2. N-Body	Pipeline de Procesos
MULTIPLICACIÓN DE TUPLAS	1. Producto de Matrices 2. Rutas de Grafos	Pipeline de Procesos
AUTÓMATAS CELULARES	1. Laplace 2. Simulación	Matriz de Procesos

Tabla 1. Paradigmas Paralelos y sus programas modelo

Para cada paradigma se crea un Programa General que define la estructura de control común a aquellos problemas que puedan ser resueltos con una misma técnica de diseño. Al programa general se le da el nombre de Esqueleto Algorítmico o template. Posteriormente desde un programa paralelo general, se derivan dos o más Programas Modelo que ilustren el uso del paradigma para resolver problemas específicos. Un programa general incluye algunos tipos de datos que no se especifican y procedimientos que varían de una aplicación a otra. Un programa modelo se obtiene al reemplazar estos tipos de datos y procedimientos por los correspondientes tipos de datos y procedimientos de un programa secuencial que resuelve un problema específico [20]. En otras palabras, la esencia de esta metodología de programación es que un programa modelo tiene un componente paralelo que implementa un paradigma y un componente secuencial para una aplicación específica (Fig.1). Para cualquier paradigma entonces, se define el siguiente método de derivación:

1. Identificar dos o más problemas computacionales con la misma estructura de control.
2. Para cada problema escribir un tutorial que explique su teoría computacional e incluya un programa o algoritmo completo.
3. Escribir un programa paralelo para la programación del paradigma.
4. Probar el programa paralelo en una computadora secuencial.
5. Derivar un programa paralelo para cada problema mediante sustituciones triviales de algunos tipos de datos, variables, procedimientos, etc., y analizar la complejidad de esos programas.
6. Rescribir los programas paralelos en un lenguaje de implementación y medir su rendimiento en una multicomputadora.
7. Escribir descripciones claras de los programas paralelos.
8. Publicar los programas y sus descripciones en su totalidad.

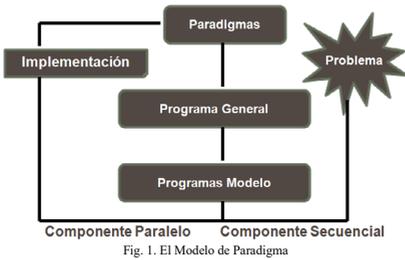


Fig. 1. El Modelo de Paradigma

VI. EL PARADIGMA DIVIDE Y VENCERÁS EN PARALELO

El paradigma de Divide y Vencerás se caracteriza por la división de un problema en sub-problemas que tienen la misma forma que el problema completo [3]. La división del problema en sub-problemas más pequeños se lleva a cabo utilizando la recursión. El método recursivo continúa dividiendo el problema hasta que las partes divididas ya no puedan dividirse más, entonces se combinan progresivamente y de forma ascendente los resultados parciales de cada sub-problema hasta obtener la solución al problema inicial [3]. El caso general de los algoritmos de divide y vencerás es:

```
función DyV(x)
    descomponer x en casos más pequeños x1,...,xn
    para i que va de 1 to 1 hacer yi= DyV(xi)
    recombinar los yi para obtener una solución y
    de x
    devolver y
```

En esta técnica, la división de cada problema suele hacerse a menudo en dos sub-problemas; por tanto, supondremos una formulación recursiva del método Divide y Vencerás con un esquema de división en forma de un árbol binario, cuyos nodos serán procesadores, procesos o threads (Fig.2) cuya comunicación entre ellos se lleva a cabo mediante el uso de memoria compartida aplicando técnicas de exclusión mutua para compartición de datos y estructuras, así como una comunicación basada en el modelo Productor-Consumidor [18].

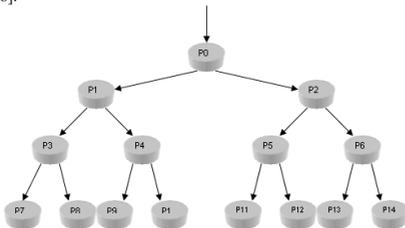


Fig 2. Representación gráfica de un árbol binario

El nodo raíz del árbol recibe como entrada un problema completo que se divide en dos partes, una se envía al nodo hijo izquierdo, la otra se envía al nodo hijo derecho (fig. 2). Se repite recursivamente el proceso de división hasta llegar a los niveles más bajos del árbol hasta que, transcurrido un cierto tiempo, todos los nodos hoja reciben como entrada un sub-problema de su nodo padre, entonces lo resuelven y le devuelven las soluciones. Cualquier nodo padre en el árbol obtendrá dos soluciones parciales de sus nodos hijos y las combinará para proporcionar una única solución que será la salida del nodo padre. Finalmente el nodo raíz proporcionará como salida la solución completa del problema inicial, [4]. En la fig. 2 se muestra un árbol binario completo, esto es, un árbol perfectamente balanceado con los nodos hojas en el mismo nivel, pero podría ocurrir que uno o más nodos hoja aparezcan en diferentes niveles del árbol si el número de sub-problemas no es potencia de 2. La transformación del algoritmo general en versiones paralelas de quicksort, búsqueda binaria, o encontrar el máximo y el mínimo de un vector de elementos es relativamente fácil. Mientras que, en una implementación secuencial un solo nodo del árbol puede ser visitado cada vez, en una implementación paralela de un algoritmo que siga este esquema se podría visitar más de un nodo al mismo tiempo, es decir, al dividir un problema en dos sub-problemas, ambos pueden ser procesados de manera simultánea y en cada nivel del árbol habrá una espera coordinada de los nodos que están en ese nivel aplicando el modelo Productor-Consumidor para esperar por los resultados en el backtracking de la recursividad. Es de nuestro interés utilizar este paradigma de programación como patrón de diseño algorítmico paralelo en una aplicación específica relacionada con problemas de ordenación de un conjunto de datos (quicksort y mergeSort), con problemas de búsqueda de información en un conjunto de datos (búsqueda binaria) y con problemas de máximos y mínimos de entre un conjunto de datos.

Crearemos entonces un programa general que representará al patrón de diseño Divide y Vencerás en sí mismo para que a través de él se defina la estructura de control común a aquellos problemas que puedan ser resueltos con este paradigma creando los programas modelos de los problemas que se resuelven aplicando Divide y Vencerás en paralelo (Fig 3 y Fig 4).

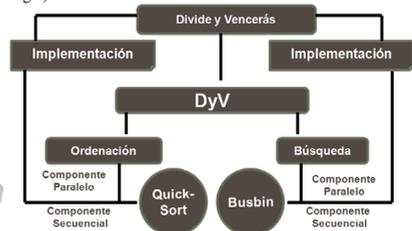


Fig. 3. El Paradigma Divide y Vencerás en Paralelo visto como un Patrón de Diseño aplicado a QuickSort y Búsqueda Binaria

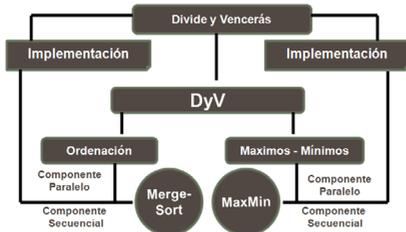


Fig. 4. El Paradigma Divide y Vencerás en Paralelo visto como un Patrón de Diseño aplicado a MergeSort yEl problema de MaxMin

En todos estos problemas secuenciales mostrados en las figuras 3 y 4, el componente paralelo es el mismo puesto que tal componente es el paradigma divide y vencerás como un patrón de diseño bajo el cual estos problemas son resueltos.

A. Ordenación Rápida o QuickSort

Supongamos que se tiene una lista de números enteros en desorden y queremos llevar a cabo la ordenación de dichos datos en orden ascendente, es decir, de menor a mayor, por medio de la técnica de divide y vencerás, utilizando para ello el algoritmo de ordenación rápida o QuickSort.

```

Procedimiento QuickSort (conjunto, inicio,fin)
    Si (inicio < fin)
        Entonces
            pivote=conjunto[inicio]
            nuevo_inicio= inicio
            Para I = (inicio+1) a fin hacer
                Si (conjunto[I] <= pivote)
                    Entonces
                        nuevo_inicio = nuevo_inicio+1
                        intercambia (conjunto[nuevo_inicio],
                                conjunto[I])
            intercambia (conjunto[inicio],
                        conjunto[nuevo_inicio] )
            QuickSort (conjunto, inicio, nuevo_inicio)
            QuickSort (conjunto, nuevo_inicio+1, fin)
        fin de QuickSort.
    
```

Como primer paso el algoritmo selecciona como pivote uno de los elementos del conjunto de datos que vaya a ordenar; a continuación, el conjunto se divide en dos partes, una a la izquierda y la otra a la derecha del pivote; se desplazan los elementos de tal manera que los que sean mayores que el pivote queden a su derecha mientras que los que sean menores queden a su izquierda, posteriormente las partes del conjunto que quedan a ambos lados del pivote se ordenan independientemente de forma paralela y recursiva; se llega al resultado final, que es el conjunto de datos completamente ordenado si cada nodo del árbol construye el array solución al sub-problema que le ha sido encomendado colocando el subarray que le entrega su hijo derecho detrás del que le entrega el izquierdo, antes de devolverla a su nodo padre [3].

B. Ordenación por Fusión o MergeSort

El método consiste en descomponer el arreglo o matriz T en dos partes cuyos tamaños sean tan parecidos como sea posible, ordenar estas partes mediante llamadas recursivas y después fusionar las soluciones de cada parte, teniendo cuidado en mantener el orden. Para hacer esto, se requiere de un algoritmo eficiente para fusionar dos arrays ordenados U y V en un único array T cuya longitud sea la suma de las longitudes de U y V [3]. Esto se logra de manera más eficiente si se dispone de espacio adicional al final de los arrays U y V para utilizarlo como centinela. A continuación el pseudocódigo que implementa el algoritmo:

```

Procedimiento merge(U[1..m+1],V[1..n+1],T[1..m+n])
{
    i=1; j=1;
    U[m+1]= ∞; V[n+1]= ∞;
    para k=1 hasta m+n hacer
        {
            si U[i] < V[j]
                entonces { T[k]=U[i]; i=i+1; }
                sino { T[k]=V[j]; j=j+1; }
        }
}
Procedimiento MergeSort(T[1..n])
{
    Array U[1..1+(n/2)], V[1..1+(n/2)];
    U[1..n/2]=T[1..n/2]
    V[1..n/2]=T[1+(n/2)..n]
    MergeSort (U[1..n/2])
    MergeSort (V[1..n/2]);
    merge (U,V,T)
}
    
```

C. Búsqueda Binaria o Busbin

Sea $T[1..n]$ un array ordenado por orden no decreciente, es decir, $T[i] < T[j]$ siempre que $1 \leq i < j \leq n$. Sea x un elemento. El problema consiste en buscar x en el array T, si es que está. Dicho problema se resuelve buscando x o bien en la primera mitad del array, o bien en la segunda mitad. Para averiguar cuál de estas búsquedas es la correcta, comparamos x con un elemento del array. Sea $k=n/2$; si $x \leq T[k]$ entonces se puede restringir la búsqueda de x a $T[1..k]$; en caso contrario basta con buscar en $T[k+1..n]$, [3], [16]. El algoritmo es el siguiente:

```

Procedimiento busquedabin(T[i..j],x)
{
    Si (i==j)
        entonces devolver i
    k=(i+j)/2
    si (x ≤ T[k])
        entonces devolver busquedabin(T[i..k],x)
        sino devolver busquedabin(T[k+1..j],x)
}
    
```

D. El problema de los máximos y mínimos o MaxMin

El problema de encontrar el valor máximo y el valor mínimo de un conjunto de datos no ordenados se clasifica en la serie de problemas que manejan como estructura de solución un árbol binario y devuelven un valor como resultado. El árbol binario representa las sucesivas divisiones del vector que contiene los datos a procesar en un orden descendente y la transmisión de los valores resultantes en orden ascendente [13], [20]. Su algoritmo es el siguiente:

```

MaxMinDV (i,j, var Max, Min)
{
    si (i<j-1)
    entonces {
        mit = (i+j) div 2;
        MaxMinDV (i, mit, Max1, Min1);
        MaxMinDV (mit+1, j, Max2, Min2);
        si (Max1>Max2)
        entonces {Max= Max1;}
        sino {Max = Max2;}
        si (Min1<Min2)
        entonces { Min = Min1; }
        sino {Min = Min2;}
    }
    sino { si (i=j-1)
    entonces {
        si (A[i]>A[j])
        entonces {
            Max = A[i];
            Min = A[j];
        }
        sino {Max = A[j] ; Min= A[i];}
    }
    sino { Max = A[i] ; Min = Max; }
    }
}
    
```

VII. EL PATRÓN DE DISEÑO ALGORÍTMICO PARALELO DIVIDE Y VENCERÁS

La implementación del paradigma Divide y Vencerás como Patrón de Diseño Paralelo a través de objetos activos en JAVA consiste de los siguientes componentes (ver Figura 5):

- Una interface (DyVable)
- Una clase que implemente el paradigma Divide y Vencerás, esto es, el programa general (SchDyVPar).
- Las clases que implementen los problemas secuenciales; ordenación por QuickSort, (ProQSort), ordenación por MergeSort (ProMergeSort), Búsqueda Binaria (ProBinSearch) y búsqueda del máximo y mínimo de un conjunto de elementos (ProMaxMin).
- Las clases solución de cada uno de los problemas secuenciales a paralelizar (SolqSort, SolMergeSort, SolBusBin, SolMaxMin).
- Una clase que contenga al programa principal (TestDyVPar) o bien, un programa principal para cada problema a resolver.

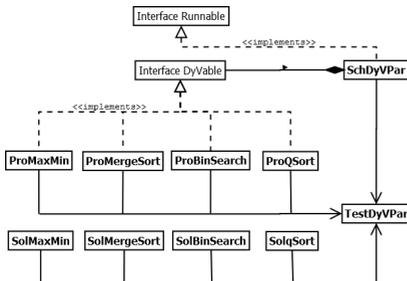


Fig. 5. Diagrama de clases del Patrón de Diseño Paralelo Divide y Vencerás

La Interface DyVable: Todos los problemas que se resuelvan mediante éste paradigma deberán de implementar esta interface. Con esto garantizamos que dichos problemas sean compatibles con ella y por tanto puedan implementar los métodos abstractos de la misma. De esta manera la clase que implementa el paradigma a la que le llamaremos SchDyVPar podrá resolver los problemas de forma genérica. La clase SchDyVPar recibe entonces referencias a objetos compatibles con la interface y por lo tanto podrá invocar los métodos abstractos de esta, los cuales son:

- `public boolean base():` retorna un valor TRUE si los datos del objeto representan un problema base o indivisible y FALSE en otro caso.
- `public Object resolver():` Retorna una solución a un sub-problema base cuando se invoca este método.
- `public Interfaz[] dividir():` Divide un problema no-base en un vector de sub-problemas paralelizables en su ejecución.
- `public Object combinar(Object[]):` Recibe un vector de objetos solución a sub-problemas, los combina y retorna una solución al problema combinado.

```

public interface DyVable
{
    public boolean base();
    public Object solve();
    public DyVable[] dividir();
    public Object combine(Object[] soluciones);
}
    
```

La clase SchDyVPar: Ésta producirá objetos activos es decir procesos que implementan la técnica divide y vencerás en paralelo, mediante un árbol de hilos con el problema original como raíz y los sub-problemas como hojas de dicho árbol sincronizados mediante el modelo cliente-servidor en su comunicación utilizando la exclusión mutua como restricción de sincronización. La implementación deberá ser independiente del problema concreto que se pretenda resolver. El constructor de esta clase iniciará entonces un hilo. Recibe un problema a resolver y devolverá su solución. Si el problema que recibe es base entonces se resuelve, y si no, se invoca al método divide(). Por cada sub-problema se creará una instancia de la clase DyVable. Se enviarán los sub-problemas a los hilos hijos y se obtendrán las soluciones de los sub-problemas, para después combinar dichas soluciones y devolver la solución del problema que se recibió al principio.

```

public class SchDyVPar implements Runnable
{
    DyVable problem;
    Thread t;
    static Object resul;
    boolean band;

    SchDyVPar(DyVable problem)
    {
        band=true;
        this.problem=problem;
        t=new Thread(this);
        t.start();
        try{
            t.join();
        }catch(InterruptedException e) {}
    }
}
    
```

```

    }
    public void run()
    {
        if (problem.base())
        {
            synchronized(this)
            {
                try{
                    while(!band)
                    {
                        wait();
                    }catch(InterruptedException e){}
                    band=false;
                }
                Object[]solucion=(Object[])
                    problem.solve();
                resul=problem.combine(solucion);
            }
        }
        else {
            DyVable[] division=problem.divide();
            for(int i=0; i<=division.length-1;i++)
            {
                new SchDyVPar(division[i]);
                synchronized(this)
                {
                    band=true;
                    notify();
                }
            }
        }
    }
}

```

La clase ProQSort: Clase utilizada para crear instancias del problema QuickSort resoluble con un algoritmo secuencial. Esta clase deberá implementar la interface DyVable.

```

public class ProQSort implements DyVable
{
    private int[] nums;
    private int inicio,fin;
    static int[] nums_sort;
    ProQSort(int[] nums,int inicio,int fin,
            int[] nums_sort)
    {
        super();
        this.nums=nums;
        this.inicio=inicio;
        this.fin=fin;
        this.nums_sort=nums_sort;
    }

    public synchronized DyVable[] divide()
    {
        DyVable[] p=new DyVable[2];
        int newinicio,piv,t;
        piv=nums[inicio];
        newinicio=inicio;
        for (int i=(inicio)+1;i<fin;i++)
        {
            if (nums[i]<=piv)
            {
                newinicio++;
                t=nums[newinicio];
                nums[newinicio]=nums[i];
                nums[i]=t;
            }
        }
        t=nums[inicio];
        nums[inicio]=nums[newinicio];
        nums[newinicio]=t;
        ProQSort a=new ProQSort
            (nums,newinicio+1,fin, nums_sort);
        ProQSort b=new ProQSort
            (nums,inicio,newinicio,nums_sort);
        p[0]=a;
        p[1]=b;
        return p;
    }
}

```

```

    }
    public synchronized boolean base()
    {
        if (inicio<fin)
            return false;
        else return true;
    }

    public synchronized Object solve()
    {
        Object[] objeto=new Object[2];
        if (fin<nums.length)
        {
            objeto[0]=new Integer(nums[fin]);
            objeto[1]=new Integer(fin);
        }
        return objeto;
    }

    public synchronized Object combine(Object[]
        resultado)
    {
        if (fin<nums.length)
        {
            int valor=
                ((Integer)resultado[1]).intValue();
            nums_sort[valor]=
                ((Integer)resultado[0]).intValue();
        }
        return resultado;
    }
}

```

La clase ProMaxMin: Clase utilizada para crear instancias del problema de Máximos y Mínimos resoluble con un algoritmo secuencial. Esta clase deberá implementar de igual forma la interface DyVable.

```

public class ProMaxMin implements DyVable
{
    private int[] nums;
    private int k;
    int max,min;

    ProMaxMin(int[] nums)
    {
        super();
        this.nums=nums;
        k=(nums.length/2)-1;
    }

    public DyVable[] divide()
    {
        DyVable[] p= new DyVable[2];
        int[] n= new int[k+1];
        int[] m= new int[nums.length-(k+1)];

        for (int i=0; i<=n.length-1;i++)
        {
            n[i]=nums[i];
            m[i]=nums[k+i+1];
        }
        ProMaxMin a=new ProMaxMin(n);
        ProMaxMin b=new ProMaxMin(m);
        new SchDyVPar(a);
        new SchDyVPar(b);
        if (a.max<b.max) this.max=b.max;
        else this.max=a.max;
        if (a.min>b.min) this.min=b.min;
        else this.min=a.min;
        p[0]=a;
        p[1]=b;
        return p;
    }

    public boolean base()
    {
        {

```

```

        if ((nums.length==1)|| (nums.length==2))
            return true;
        else return false;
    }
    public Object solve()
    {
        Object[] sol= new Object[2];
        if (nums.length==1)
        {
            sol[0]= new Integer(nums[0]);
            sol[1]= new Integer(nums[0]);
        }
        else {
            if (nums.length==2)
            {
                if (nums[0]<nums[1])
                {
                    sol[0]= new Integer(nums[1]);
                    sol[1]= new Integer(nums[0]);
                }
                else {
                    sol[0]= new Integer(nums[0]);
                    sol[1]= new Integer(nums[1]);
                }
            }
        }
        return sol;
    }
    public Object combine(Object[] resultado)
    {
        this.max=((Integer)resultado[0]).intValue();
        this.min=((Integer)resultado[1]).intValue();
        return resultado;
    }
}

```

La clase ProMergeSort: Clase utilizada para crear instancias del problema ordenamiento de datos por mezcla, resoluble con un algoritmo secuencial. Esta clase deberá implementar de igual forma la interface DyVable.

```

public class ProMergeSort implements DyVable
{
    private int[] nums;
    private int inicio,fin;
    static int[] nums_sort;

    ProMergeSort(int[] nums,int inicio,int fin,
                 int[] nums_sort)
    {
        super();
        this.nums=nums;
        this.inicio=inicio;
        this.fin=fin;
        this.nums_sort=nums_sort;
    }

    public DyVable[] divide()
    {
        DyVable[] p=new DyVable[2];
        int k=(inicio+fin)/2, h=inicio;
        int i=inicio, j=k+1;
        int[] aux=new int[nums.length];

        while ((h<=k) && (j<=fin))
        {
            if (nums[h]<=nums[j])
            {
                aux[i]=nums[h]; h++;
            }
            else { aux[i]=nums[j]; j++;}
            i++;
        }
        if (h>k)
        {
            for (int t=j;t<=fin;t++)

```

```

            { aux[i]=nums[t]; i++; }
        }
        else {
            for (int t=h;t<=k;t++)
            { aux[i]=nums[t]; i++; }
        }
        for (int t=inicio;t<=fin;t++)
        { nums[t]=aux[t]; i++; }

        ProMergeSort a=new
            ProMergeSort(nums, inicio, k, nums_sort);
        ProMergeSort b=new
            ProMergeSort(nums, k+1, fin, nums_sort);
        p[0]=a;
        p[1]=b;
        return p;
    }

    public boolean base()
    {
        if (inicio<fin) return false;
        else return true;
    }

    public Object solve()
    {
        Object[] objeto=new Object[2];
        if (fin<nums.length)
        {
            objeto[0]=new Integer(nums[fin]);
            objeto[1]=new Integer(fin);
        }
        return objeto;
    }

    public Object combine(Object[] resultado)
    {
        if (fin<nums.length)
        {
            int valor= ((Integer)
                resultado[1]).intValue();
            nums_sort[valor]=((Integer)
                resultado[0]).intValue();
        }
        return resultado;
    }
}

```

La clase ProBinSearch: Clase utilizada para crear instancias del problema de búsqueda binaria en un conjunto de datos previamente ordenado, resoluble con un algoritmo secuencial. Esta clase deberá implementar al igual que las anteriores, la interface DyVable.

```

public class ProBinSearch implements DyVable
{
    private int x,k;
    private int[] nums;
    static Object numfind;

    ProBinSearch(int[] nums,int x)
    {
        super();
        this.nums=nums;
        this.x=x;
        k=(nums.length/2)-1;
    }

    public DyVable[] divide()
    {
        int[] n=null;
        DyVable[] p=new DyVable[1];
        if (k!=-1)
        {
            if (x<nums[k])

```

```

n=new int[k+1];
for(int i=0; i<=n.length-1;i++)
    n[i]=nums[i];
for(int i=0; i<=n.length-1;i++)
    System.out.print(" "+n[i]);
}
else {
    if (x>nums[k])
    {
        n=new int[nums.length-(k+1)];
        for(int i=0; i<=n.length-1;i++)
            n[i]=nums[k+1+i];
        for(int i=0; i<=n.length-1;i++)
            System.out.print(" "+n[i]);
    }
}
}
p[0]=new ProBinSearch(n,x);
return p;

public boolean base()
{
    if(k==-1) return true;
    else if (x==nums[k]) return true;
    else return false;
}

public Object solve()
{
    Object[] objeto=new Object[1];
    if (k==-1){
        if (nums[nums.length-1]==x)
            objeto[0]=new Integer(x);
        else objeto[0]= new Character('a');
    }
    else objeto[0]= new Integer(x);
    return objeto;
}

public Object combine(Object[] resultado)
{
    numfind=resultado[0];
    return resultado;
}
}

```

Las clases Solución: Una instancia de esta clase contendrá una solución al problema que se pretende resolver, por ejemplo, en el caso de QuickSort, la clase solución deberá contener instancias de ella misma conteniendo un vector ordenado. De manera similar se codifican las clases solución para el problema de Máximos y Mínimos, Ordenación por MergeSort y Búsqueda Binaria.

```

//CLASE SOLUCION PARA EL PROBLEMA QuickSort
public class SolqSort
{
    SolqSort(int[] solution)
    {
        for (int i=0;i<=solution.length-1;i++ )
            System.out.print(solution[i]+" ");
    }
}

//CLASE SOLUCION PARA EL PROBLEMA MaxMin
public class SolMaxMin
{
    SolMaxMin(ProMaxMin p)
    {
        System.out.println("MAXIMO= "+p.max);
        System.out.println("MINIMO= "+p.min);
    }
}

```

```

//CLASE SOLUCION PARA EL PROBLEMA MergeSort
public class SolMergeSort
{
    SolMergeSort(int[] solution)
    {
        for (int i=0;i<=solution.length-1;i++ )
            System.out.print(solution[i]+" ");
    }
}

//CLASE SOLUCION PARA EL PROBLEMA BinSearch
public class SolBinSearch
{
    SolBinSearch(ProBinSearch solution, int n)
    {
        Object sol= solution.numfind;
        Object num= new Integer(n);
        boolean ban=true;

        if (!num.equals(sol)) ban=false;
        if (ban)
            System.out.println("El numero "+n+" si se encuentra en el vector...");
        else System.out.println("El numero "+n+" NO se encuentra en el vector...");
    }
}

```

Las clases Montaje: Son las clases que contendrán el programa principal o método main() de cada uno de los problemas propuestos. Para cada clase un vector de elementos es creado para después obtener una instancia de la clase que contendrá el problema original. Entonces un hilo es lanzado con el problema inicial para resolverlo al crear una instancia de la clase SchDyVPar, cuyo parámetro será el objeto que contenga al problema original, y posteriormente se espera recibir un objeto solución con el resultado de la clase SchDyVPar.

```

//CLASE MONTAJE QUE PRUEBA EL ALGORITMO DE QuickSort
public class TestDyVPar
{
    public static void main(String args[])
    {
        int[] arrayint={3,2,1,5,-8,4,3,40,
            10,-2,0,10,8,30,-15,6};
        int[] ordenado= new int[arrayint.length];
        ProQSort proqs=new ProQSort
            (arrayint,0,arrayint.length,ordenado);
        SchDyVPar dyvpar=new SchDyVPar(proqs);
        SolqSort solqs=new SolqSort(ordenado);
    }
}

//CLASE MONTAJE QUE PRUEBA EL ALGORITMO DE MaxMin
public class TestDyVPar
{
    public static void main(String args[])
    {
        int[] arrayint={3,2,1,5,-8,4,3,0,
            10,-2,0,10,8,30,1,6};
        ProMaxMin promm=new ProMaxMin(arrayint);
        SchDyVPar dyvpar=new SchDyVPar(promm);
        SolmaxMin solmm=new SolmaxMin(promm);
    }
}

//CLASE MONTAJE QUE PRUEBA EL ALGORITMO DE MergeSort
public class TestDyVPar
{
    public static void main(String args[])
    {
        int[] arrayint={3,2,1,5,-8,4,3,40,
            10,-2,0,10,8,30,-15,6};
    }
}

```

```
int[] ordenado= new int[arrayint.length];
ProMergeSort proms=new ProMergeSort
(arrayint,0,arrayint.length-1,ordenado);
SchDyVFar dyvpar=new SchDyVFar(proms);
SolmSort solqs=new SolmSort(ordenado);
}
}
//CLASE MONTAJE QUE PRUEBA EL ALGORITMO DE BinSearch
public class TestDyVFar
{
    public static void main(String args[])
    {
        int[] arrayint={-15,-8,-2,0,1,2,3,3,4,
            5,6,8,10,10,30,35,40};
        int num_buscar=30;
        ProBinSearch probin=new ProBinSearch
            (arrayint,num_buscar);
        SchDyVFar dyvpar=new SchDyVFar(probin);
        SolbinSearch solbin=new SolbinSearch
            (probin,num_buscar);
    }
}
```

VIII. CONCLUSIONES

Se ha desarrollado una metodología para la programación de algoritmos paralelos como Patrones de Diseño Algorítmicos. De forma particular hemos implementado el paradigma de divide y vencerás como un Patrón de Diseño Paralelo, comúnmente llamado Programa General el cual ha derivado en cuatro programas modelo que solucionan el problema de ordenación por QuickSort, el problema de ordenación por MergeSort, el problema de la búsqueda binaria y el problema de encontrar valores máximos y mínimos dentro de un conjunto de datos. Para ello se ha utilizado la Programación Paralela Estructurada bajo el lenguaje JAVA para demostrar que con conocimientos mínimos de Paralelismo y Concurrencia, el programador/usuario pueda explotarlos, mediante el uso de diferentes mecanismos de reusabilidad, genericidad y uniformidad y pueda definir sus propios patrones algorítmicos, adaptándolos a la estructura de comunicación que utilizan los procesos para comunicarse y resolver el problema mediante el concepto de Objeto Activo.

REFERENCIAS

- [1] G.R. Andrews, *Paradigms for Process interaction in Distributed Programs*. ACM Computing Surveys, Volume 15, Number 1, 49-90, 1991.
- [2] E. Blelloch, *Programming Parallel Algorithms*. Communications of the ACM. Volume 39, Number 3, 85-97, 1996.
- [3] G. Brassard, P. Bratley, *Fundamentos de Algoritmia*. Madrid, Prentice-Hall, 1997.
- [4] Brinch Hansen, *Model Programs for Computational Science: A programming methodology for multicomputer*. Concurency: Practice and Experience, Volume 5, Number 5, 407-423, 1993.
- [5] M. Cole, *Algorithmic Skeletons: Structured Management of Parallel Computation*. The MIT Press, 1989.
- [6] A. J. Collins, *Automatically Optimising Parallel Skeletons*. M.Sc. Thesis in Computer Science, School of Informatics University of Edinburgh, UK, 2011.
- [7] A. Corradi, L. Leonardi, *PO Constraints as tools to synchronize active objects*. Journal Object Oriented Programming 10, pp. 42-53. 1991.
- [8] A. Corradi, L. Zambonelli, *Experiences toward an Object-Oriented Approach to Structured Parallel Programming*. DEIS technical report no. DEIS-LIA-95-007. 1995
- [9] Darlington et al., *Parallel Programming Using Skeleton Functions*. Proceedings PARLE'93, Munich(D),1993
- [10] S. Ernsting, H. Kuchen, *Algorithmic skeletons for multi-core, multi-GPU systems and clusters*. Int. J. of High Performance Computing and Networking. Vol. 7, No.2, pp. 129 - 138. 2012.
- [11] Horstmann, *Computing concepts with JAVA. 3rd Edition*. John Wiley & Sons, Inc. USA. 2003.
- [12] Kumar, Vipin, et al. *Introduction to parallel computing. Design and Analysis of Algorithms*. USA, Benjamin-Cummings, 1994.
- [13] R.C.T Lee, S.S. Tseng, R.C. Chang, Y.T. Tsai, *Introducción al diseño y análisis de algoritmos, un enfoque estratégico*. Mc Graw Hill. 2007.
- [14] A. Levitin, *The Design of Analysis of Algorithms*. Wesley, 2003.
- [15] N. Marti, *Estructuras de datos y métodos algorítmicos ejercicios resueltos*. Pearson Prentice Hall. 2004.
- [16] J. F. Myoupo, V. K. Tchendji, *Parallel dynamic programming for solving the optimal search binary tree problem on CGM*. International Journal of High Performance Computing and Networking 2014. Vol. 7, No.4 pp. 269 – 280. 2014.
- [17] I. Parberry, *Problems on Algorithms*. Prentice-Hall. 2002.
- [18] M. Poldner, H. Kuchen, *Skeletons for divide and conquer algorithms*. Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN). ACTA Press. 2008.
- [19] Rabbi, Fethi, *A Parallel Programming Methodology based on Paradigms*. (18th WoTUG Technical Meeting), P. Nixon IOS Press, 239-249, 1995.
- [20] S. Sahni, *Data structures, Algorithms and Applications in Java*. Silicon Press. 2005.

Extendiendo Paraldroid para la Generación Automática de Código OpenCL

Sergio Afonso¹, Alejandro Acosta² y Francisco Almeida³

Resumen.—La popularidad de los dispositivos móviles (smartphones, tablets, ...) y sus crecientes capacidades computacionales abren una nueva era en términos de computación paralela. El uso eficiente de dichos dispositivos es todavía un reto. La heterogeneidad de los Sistemas en Chip (SoC) demanda un conocimiento muy específico de los dispositivos, lo cual representa una muy elevada curva de aprendizaje para programadores de propósito general. Para facilitar la tarea de desarrollo presentamos la extensión de Paraldroid para OpenCL. Paraldroid es un marco de desarrollo para programadores de propósito general de dispositivos móviles. Presenta un modelo de desarrollo que unifica los diversos modelos de Android y permite la generación automática de código paralelo. El desarrollador sólo implementa una aplicación Java orientada a objetos e introduce una serie de anotaciones específicas de Paraldroid en los elementos principales de las clases a optimizar. Las anotaciones están basadas en la especificación de OpenMP 4.0. Paraldroid soporta la generación automática de código nativo en C, Renderscript y OpenCL, donde los dos últimos permiten la ejecución en la GPU. La experiencia computacional prueba que los resultados son prometedores. El código generado por Paraldroid aprovecha la GPU y ofrece un buen rendimiento con un bajo coste de desarrollo, así que contribuye a incrementar la productividad a la hora de desarrollar código eficiente.

Palabras clave.— Android, anotaciones, OpenCL, paralelización automática, transformación fuente-a-fuente.

I. INTRODUCCIÓN

TECNOLOGÍAS previamente sólo disponibles en ordenadores de sobremesa están ya implementadas en sistemas integrados y dispositivos móviles. Podemos ahora encontrar nuevos procesadores que integran arquitecturas multi-núcleo, GPUs y DSPs desarrollados para este mercado. Nvidia Tegra [1], Qualcomm Snapdragon [2] y Samsung Exynos [3] son algunas plataformas que van en esta dirección. Conceptualmente, el modelo subyacente se puede ver como un sistema heterogéneo tradicional compuesto por CPU y GPU donde la memoria está compartida entre ambas unidades de proceso.

En las arquitecturas de memoria no unificada, es común disponer de sólo un subconjunto de la memoria direccionable por la GPU. Tecnologías como Algorithmic Memory [4], GPUdirect [5] y UVA (Unified Virtual Address) de Nvidia y HSA [6] de AMD van en la dirección de un sistema de memoria unificada entre CPUs y GPUs. Al mismo tiempo, el rendimiento de la memoria continúa siendo superado por la demanda cada vez mayor de procesadores más rápidos y arquitecturas paralelas.

Muchos entornos se han creado para apoyar el

desarrollo de software para estos dispositivos. Las principales compañías que compiten en este mercado tienen sus propias plataformas: Android de Google [7], iOS de Apple [8] y Windows Phone de Microsoft [9]. En cada una de estas plataformas un entorno de desarrollo de alto nivel se proporciona para facilitar el desarrollo de aplicaciones. Sin embargo, estos entornos están más orientados al desarrollo rápido de aplicaciones interactivas que para reducir la dificultad de aprovechamiento de la arquitectura paralela subyacente. Dada la alta heterogeneidad existente entre los dispositivos, se hace necesaria la creación de herramientas para incrementar la productividad de desarrollo mientras se explota la capacidad computacional de las distintas arquitecturas.

Android provee de tres modelos de programación con distintas características que se deben aplicar a distintas partes del código para obtener el mejor rendimiento. En [10] estos modelos se comparan al detalle y se resalta la necesidad de un modelo de programación unificado para Android.

- **Java:** Proporciona una API exhaustiva y es el modelo más sencillo de programar. La mayoría de aplicaciones Android están escritas en Java, por lo que los desarrolladores Android deberían estar familiarizados con este lenguaje.
- **Renderscript:** Diseñado para tareas de cómputo intensivo, principalmente SPMD. Requiere el aprendizaje de un nuevo lenguaje basado en C.
- **C nativo:** Proporciona acceso a bibliotecas nativas y tiene una menor sobrecarga que Java en tiempo de ejecución, lo que en ocasiones compensa el coste extra de desarrollo.

Paraldroid [11], [12] es un marco de desarrollo que permite el desarrollo automático de aplicaciones en C nativo y Renderscript — secuenciales y paralelas — en dispositivos móviles. En Paraldroid, el desarrollador anota un código secuencial escrito en un lenguaje de alto nivel que al final se ejecutará en su versión nativa. Renderscript y OpenCL. Paraldroid utiliza la información proporcionada por las anotaciones para generar un nuevo programa que incorpora las secciones de código a ejecutar en la CPU o GPU, usando el lenguaje de destino especificado. Paraldroid, por tanto, contribuye a unificar los distintos modelos de programación en Android.

En este artículo presentamos una extensión de Paraldroid para soportar la generación de código OpenCL. OpenCL es una biblioteca nativa y lenguaje de programación para escribir aplicaciones de alto rendimiento en sistemas heterogéneos, que soporta varios tipos de aceleradores. Proporciona un meca-

¹Universidad de La Laguna, email: safonsof@ull.es

²Universidad de La Laguna, email: aacostad@ull.es

³Universidad de La Laguna, email: falmeida@ull.es

nismo para la programación paralela y una API para la manipulación y la comunicación a bajo nivel entre los distintos dispositivos de cómputo. Actualmente, OpenCL no está soportado oficialmente en las implementaciones de Android proporcionadas por Google. Sin embargo, dada la heterogeneidad de los sistemas móviles, algunos fabricantes ofrecen OpenCL en sus dispositivos, así que es interesante soportarlo en Paralldroid.

Las principales contribuciones de este artículo son:

- Un nuevo modelo de programación se incluye en Paralldroid. La importancia de OpenCL en sistemas de sobremesa es bien conocida. Ahora, este modelo de programación se extiende a dispositivos móviles. Paralldroid permite la generación de código OpenCL para ejecutarse desde aplicaciones Android escritas en Java.
- El entorno de Paralldroid y la arquitectura objetivo elegida son contribuciones por sí mismas. La heterogeneidad de los modelos de programación de Android permite al programador obtener el mejor rendimiento, implementando cada sección de la aplicación utilizando el modelo de programación que mejor se ajusta a su código. Paralldroid permite generar código para cada uno de los lenguajes, facilitando el desarrollo de aplicaciones heterogéneas eficientes.
- Las metodologías de anotaciones y generación de código propuestas por Paralldroid simplifican la implementación de aplicaciones que utilizan OpenCL, lo cual ayudará a aumentar el uso de OpenCL en Android.
- El soporte de generación de OpenCL abre la posibilidad a extender Paralldroid a otras plataformas diferentes a Android. El único requisito es que la plataforma de destino soporte Java y OpenCL.
- Una nueva metodología de manejo de errores ha sido creada, la cual hace posible al desarrollador detectar y posiblemente reparar errores en tiempo de ejecución ocurridos al ejecutar código en el contexto de destino.
- Analizamos el rendimiento de los distintos modelos de programación para probar los beneficios de nuestra herramienta. Los resultados obtenidos muestran que Paralldroid es una herramienta útil para mejorar la programabilidad y explotar la potencia de cómputo disponible en dispositivos Android.

Algunas herramientas que generan código paralelo a partir de una extensión de Java han sido presentadas en [13], [14], [15]. En estos casos, la sintaxis de Java se modifica para introducir nuevos elementos sintácticos en el lenguaje. La principal desventaja de esta aproximación es que los nuevos elementos no son compatibles con la definición de Java, así que un compilador de Java estándar no puede compilar el código fuente con estas extensiones. La definición de Paralldroid no modifica las definiciones estándar de Java, sino que introduce un conjunto de anotaciones.

Un compilador estándar de Java puede simplemente ignorar las anotaciones de Paralldroid, aunque la semántica del programa no se conservaría en el caso de los métodos paralelos. En [16] los autores presentan un Lenguaje de Dominio Específico (DSL) para generar código Renderscript. Este DSL es específico para algoritmos de procesamiento de imágenes. Una desventaja de esta aproximación es que el usuario tiene que aprender un nuevo lenguaje. Nuestra propuesta, en contraposición, está basada en el lenguaje principal de Android y nuestros usuarios objetivo conocen este lenguaje.

Este artículo está estructurado como sigue: En la sección II se presenta el modelo de desarrollo en Android y las diferentes alternativas que ofrece para explotar los dispositivos. Algunas de las dificultades asociadas con cada modelo se muestran. La sección III da una visión general de las metodologías propuestas por Paralldroid. En la sección IV se presenta la extensión a Paralldroid para dar soporte a la generación de OpenCL. El rendimiento de los programas generados por Paralldroid se presenta en la sección V utilizando cuatro algoritmos de procesamiento de imágenes diferentes. Cada una de las aplicaciones es ejecutada con una implementación Java secuencial y con las implementaciones automáticamente generadas por Paralldroid para Renderscript y OpenCL y los resultados son comparados. El rendimiento medido prueba el incremento de rendimiento proporcionado por Paralldroid con un bajo coste de desarrollo. Finalizamos el artículo con algunas conclusiones y futuras líneas de investigación.

II. EL MODELO DE DESARROLLO EN ANDROID

Android es un sistema operativo basado en Linux principalmente diseñado para dispositivos móviles como smartphones y tablets. Las aplicaciones Android están escritas en Java, y el kit de desarrollo (SDK) de Android proporciona las bibliotecas y herramientas que se necesitan para compilar, probar y depurar aplicaciones. Desde la versión 5.0 de Android, la aplicación se ejecuta sobre el Android Run Time (ART) que gestiona los recursos del sistema asignados a esta aplicación, a través del kernel de Linux.

Además del desarrollo de aplicaciones Java, Android proporciona paquetes de herramientas y bibliotecas para desarrollar aplicaciones nativas: El kit de desarrollo nativo (NDK). El NDK permite implementar partes de la aplicación que se ejecuta en ART utilizando lenguajes nativos como C y C++. Este código nativo se comunica con la aplicación principal escrita en Java utilizando la Interfaz Nativa de Java (JNI). Los ficheros nativos se compilan utilizando el compilador de GNU (GCC). Nótese que utilizar código nativo no resulta en un aumento automático del rendimiento de la aplicación, pero siempre incrementa su complejidad, así que se recomienda sólo para operaciones de cómputo intensivo que no utilicen mucha memoria, como procesamiento de señales, simulaciones físicas, etc. El código nativo es útil tam-

bién para portar un código nativo existente a Android. Podemos acceder a OpenCL a partir del contexto nativo si las librerías en tiempo de ejecución de OpenCL están presentes en el dispositivo.

Para explotar las altas capacidades de cómputo en los dispositivos móviles actuales, Android proporciona Rendscript, que es una API de computación de altas prestaciones a nivel nativo y un lenguaje de programación basado en el lenguaje C. Rendscript permite la ejecución de aplicaciones paralelas bajo distintos tipos de procesadores como la CPU, GPU o DSPs, eligiendo automáticamente uno de ellos dependiendo de las características del hardware. El código Rendscript (en ficheros `.rs`) se compila utilizando un compilador LLVM basado en Clang. Además, genera un conjunto de clases Java que envuelven el código Rendscript, que actúan como interfaz entre el programa principal escrito en Java y el código Rendscript acelerado. De nuevo, el uso de Rendscript no resulta en un aumento automático del rendimiento, sino que es útil para aplicaciones que realizan procesamiento de imágenes, modelado matemático u otras operaciones que requieren mucha computación en paralelo.

III. PARALLDROID

Paralldroid está diseñado para facilitar el desarrollo de aplicaciones paralelas en la plataforma Android. Asumimos que los dispositivos móviles disponen de una CPU clásica y algún otro tipo de *co-procesador*, como una GPU, que se puede explotar a través de OpenCL o Rendscript. El modo en el que Paralldroid hace esto es a través de la transformación del código Java original en otro código que, conservando la misma semántica, se ejecuta de una manera más eficiente. La generación de código en otros lenguajes es también necesaria para poder aprovechar todos los modelos de desarrollo de Android, pero en cada algoritmo el mejor modelo de desarrollo puede ser diferente debido a sus características. Por este motivo el lenguaje de destino es algo que el usuario debe especificar explícitamente al utilizar Paralldroid.

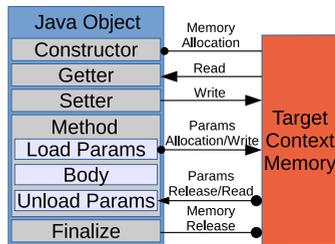
El paralelismo basado en directivas de compilación ha sido utilizado satisfactoriamente en aplicaciones para sistemas de Computación de Altas Prestaciones (HPC) durante años, y Paralldroid toma la misma aproximación en el mundo del desarrollo de aplicaciones móviles.

Las metodologías propuestas por Paralldroid pueden ser definidas en dos puntos:

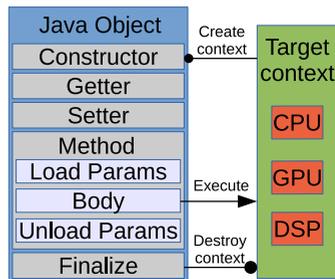
- Metodología de Anotaciones:** La directiva `target` de OpenMP 4.0 crea un entorno de datos que permite la gestión de la memoria y la ejecución de código en el dispositivo de destino. Consideramos que los elementos dentro de una clase (campos y métodos) pueden utilizarse para definir los modelos de datos y ejecución en el contexto del dispositivo. Paralldroid define una serie de anotaciones que se aplican a los campos de la clase y definiciones de métodos (Fig. 1). Es-

tas anotaciones permiten crear un entorno en el dispositivo, especificar cómo una variable se mapea en el entorno de datos del dispositivo (modelo de datos) y también especificar cómo una sección de código se ejecuta en el entorno del dispositivo (modelo de ejecución).

- Metodología de Generación:** El proceso de generación de código de Paralldroid está integrado en el proceso de compilación de Java de OpenJDK (Fig. 2(a)). En la Fig. 2(b) se muestra cómo integramos el proceso de generación de Paralldroid en el proceso de compilación de OpenJDK. Reutilizamos las fases y etapas de la implementación de OpenJDK relacionadas con la creación y análisis del Árbol Sintáctico Abstracto (AST) de Java. Incluimos tres nuevas etapas: *AnnotationsManager*, que analiza el AST de Java buscando anotaciones de Paralldroid; *Target Translator*, que genera el AST de destino y los ficheros en el lenguaje objetivo; y *Java Translator*, que modifica el AST de Java y genera los nuevos ficheros de Java. La metodología de generación propuesta por Paralldroid (Fig. 2(b)) permite extender OpenJDK para la generación de otros lenguajes objetivo como podrían ser OpenMP, CUDA, etc. Para soportar un nuevo lenguaje de destino, Paralldroid debe ser extendido con una nueva implementación de los traductores *Target Translator* y *Java Translator*.



(a) Modelo de datos



(b) Modelo de ejecución

Fig. 1. Modelos de ejecución y datos

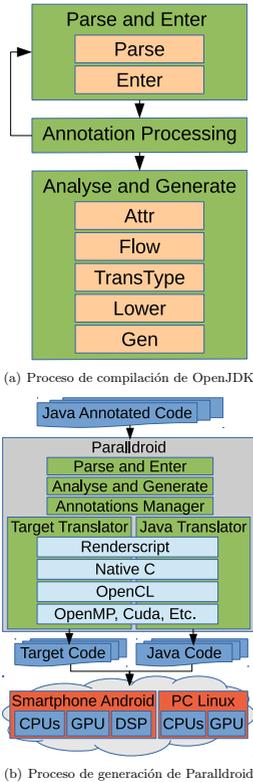


Fig. 2. Procesos de compilación y generación de OpenJDK y Paraldroid

IV. GENERACIÓN DE CÓDIGO OPENCL

El estándar OpenCL representa el esfuerzo más importante para crear una interfaz de programación de altas prestaciones común para dispositivos heterogéneos. El principal problema de OpenCL es la complejidad de su modelo de programación, que lo hace difícil de utilizar y de conservar la mantenibilidad de la aplicación.

La metodología basada en anotaciones propuesta por Paraldroid simplifica la complejidad asociada a OpenCL. Partiendo de la definición de una clase Java, el programador puede añadir anotaciones para generar código OpenCL que puede ser ejecutado transparentemente, debido a que se integra en flujo de trabajo de Java. Esto simplifica el desarrollo de aplicaciones en Android que utilizan OpenCL y ayuda a este estándar a tener una mayor adopción en la comunidad de desarrollo Android.

La Fig. 3 muestra cómo OpenCL se integra en el compilador de Paraldroid. Como para el resto de lenguajes de destino, la forma de generar nuevos ASTs desde el código fuente original escrito por el desarrollador Android es crear una clase de traducción por cada AST que se quiere crear. El traductor de AST de Java siempre debe crearse, ya que representa el código del usuario modificado que maneja los modelos de ejecución y datos de OpenCL, y delega la implementación de los métodos al contexto de destino, de acuerdo a la metodología explicada en la Fig. 1. Sin embargo, hay una diferencia evidente entre el modo en el que el conjunto de traductores de OpenCL y los demás traductores funcionan. Esta diferencia es el hecho de que hay tres traductores utilizados para la traducción a código OpenCL, en lugar de los dos traductores que se requieren para el resto de lenguajes objetivo. El problema es que el código Java no puede acceder directamente al contexto de OpenCL, así que, además de generar código Java y OpenCL, hay que generar código nativo que actúe como un puente entre los dos contextos. La Interfaz Nativa de Java (JNI) se utiliza para conectar el contexto de Java con el contexto nativo. El traductor de kernels OpenCL, también, es diferente a todos los otros traductores en que sólo puede generar código para métodos anotados, así que no es un traductor “autónomo”. Esto significa que este traductor tiene que ser llamado por otro traductor cuando se encuentre con un método anotado como paralelo. Su salida es el código OpenCL C que luego se inserta dentro del código nativo como una cadena de texto. Este complejo modelo es ocultado por Paraldroid; el programador sólo debe crear la clase Java y utilizar las anotaciones de Paraldroid.

```

1 @Target(OPENCCL)
2 public class GrayScale {
3     @Declare
4     private float gMonoMult[] = {0.299f, 0.587f, 0.114f};
5     @Map(TD)
6     private int width;
7     @Map(TD)
8     private int height;
9
10    public GrayScale(int width, int height) {
11        this.width = width;
12        this.height = height;
13    }
14    @Parallel
15    public void test(@Map(TD) int[] srcPxs,
16                  @NumThreads @Map(FROM) int[] outPxs,
17                  @Index int x) {
18        int acc;
19
20        acc = (int)((srcPxs[x] & 0xff) * gMonoMult[0]);
21        acc += (int)((srcPxs[x] >> 8) & 0xff) * gMonoMult[1];
22        acc += (int)((srcPxs[x] >> 16) & 0xff) * gMonoMult[2];
23
24        outPxs[x] = (acc) + (acc << 8) + (acc << 16) +
25                    (srcPxs[x] << 24);
26    }
27 }
    
```

Fig. 4. Escala de grises en Paraldroid

La Fig. 4 muestra la implementación en Java del algoritmo de conversión de una imagen a escala de grises utilizando las directivas de Paraldroid. La directiva **Target** (línea 1) especifica que la clase tie-

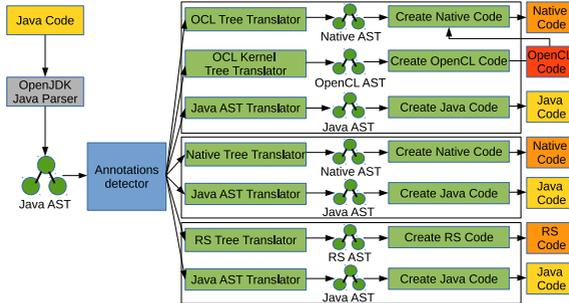


Fig. 3. Clases de traducción de Paralldroid

ne que crear la definición de un contexto OpenCL y que los elementos de la clase deben ser definidos en ese contexto. Los campos de la clase se definen entre las líneas 3 y 8. La directiva `Declare` especifica que el campo `gMonoMult` tiene que ser definido sólo en el contexto de OpenCL. En este caso el campo es inicializado, por lo que tiene que ser inicializado en el contexto OpenCL también. La directiva `Map(TD)` es similar a la directiva `Declare`, pero en este caso Paralldroid genera el método `setter` correspondiente. Estos métodos permiten a los programadores modificar los valores de los campos asociados en el contexto de OpenCL desde Java. El constructor se define en las líneas 10-13. El método `test` (líneas 14-26) define el algoritmo que transforma una imagen a escala de grises. La directiva `Parallel` indica que este método será ejecutado en paralelo. `srcPxs` y `outPxs` son los vectores que contienen las imágenes de entrada y salida, respectivamente. Nótese el uso de la directiva `Map` adecuada en cada una de ellas. La directiva `NumThreads` aplicada a un array significa que el método paralelo será ejecutado con tantos hilos como elementos hay en el array. Esta directiva se puede aplicar también a una variable escalar, lo que indicaría que se lanzará tantos hilos como se indique en la variable. La directiva `Index` especifica el índice utilizado en la ejecución paralela y se utiliza para acceder a los elementos de los vectores de entrada y de salida. El valor de esta variable se asigna automáticamente en tiempo de ejecución, y su valor variará en el rango desde cero hasta el número de hilos menos uno.

En la Fig. 5 se muestra el código de la clase Java generada por Paralldroid. Esta clase está basada en la clase definida en la Fig. 4. La clase generada tiene muchos de sus métodos marcados como nativos, así que lo primero que se hace cuando la clase se carga (líneas 2-4) es cargar la biblioteca nativa que contiene la implementación de dichos métodos. Esta biblioteca se obtiene de la compilación del código nativo también generado por Paralldroid. En las líneas 6-7 un conjunto de campos se añaden a la clase. `instanceCount` se utiliza para contar el número

```

1 public class GrayScale {
2     static {
3         System.loadLibrary("grayscale");
4     }
5
6     private static int instanceCount = 0;
7     private long instanceDataPtr;
8     private float[] gMonoMult = {0.299F, 0.587F, 0.114F};
9     private int width;
10    private int height;
11
12    public GrayScale(int width, int height) {
13        this.width = width;
14        this.height = height;
15        if (instanceCount == 0) initJNI();
16        ++instanceCount;
17        initGrayScale(gMonoMult, width, height);
18    }
19    public native void test(int[] srcPxs, int[] outPxs);
20    public native void setWidth(int width);
21    public native void setHeight(int height);
22    protected void finalize() {
23        destroyGrayScale();
24        --instanceCount;
25        if (instanceCount == 0) releaseJNI();
26    }
27    private native void initGrayScale(float[] gMonoMult,
28                                    int width, int height);
29    private native void destroyGrayScale();
30    private static native void initJNI();
31    private static native void releaseJNI();
32 }
    
```

Fig. 5. Código Java generado por Paralldroid

de instancias de la clase, para asegurar que la inicialización y liberación de las variables nativas globales se lleva a cabo, respectivamente, antes de crear la primera instancia y después de que la última es liberada por el recolector de basura. `instanceDataPtr` es un campo sólo accedido desde el código nativo, y se utiliza para mantener un puntero a una estructura nativa reservada dinámicamente que mantiene todos los datos nativos que representan una instancia de la clase. El constructor (líneas 12-18) se modifica para gestionar la inicialización de las variables globales nativas y para llamar al método que funciona como inicializador de la instancia nativa. El método anotado como `Parallel` ahora es un método nativo, con el parámetro `Index` retirado. En las líneas 20-21 se añaden los métodos `setter` para los campos `Map(TD)`. Si hubiera algún campo marca-

do como FROM o TOFROM, también se habría generado métodos *getter* para dichos campos. El método *finalize* revierte cualquier inicialización realizada en el constructor. Los métodos *initGrayScale* y *destroyGrayScale* gestionan datos de instancia, mientras que *initJNI* y *releaseJNI* gestionan variables globales y estáticas, y la configuración y liberación global de los recursos de OpenCL.

La Fig. 6 muestra el código nativo generado por Paraldroid. Los campos especificados en la clase anotada como *Target* están definidos en el contexto nativo (líneas 4-11). La implementación original del método *test* se encuentra ahora traducida como código de un kernel OpenCL y localizada en una cadena de texto (líneas 12-19), mientras que el método nativo se reemplaza por el código que interactúa con los contextos Java y OpenCL para conseguir la transmisión de datos en ambas direcciones y la ejecución en paralelo de los kernels OpenCL (líneas 23-43). Los *setters* (líneas 44-49) modifican el campo nativo asociado, y los *getters* recuperan el valor de la variable nativa asociada, aunque en este caso no hay ninguno. Los métodos *init* y *destroy* son responsables de reservar y liberar la memoria requerida en los contextos nativo y OpenCL y de inicializar y liberar el contexto OpenCL. Los métodos de inicialización nativa tienen como parámetros el conjunto de miembros de la clase que fueron inicializados en el código Java original, de forma que también se pueden inicializar en el contexto nativo.

A. Modelo de datos

La manipulación de memoria en la extensión OpenCL de Paraldroid sigue a grandes rasgos la metodología presentada en la Fig. 1(a), pero tiene que gestionar una capa intermedia adicional entre el contexto de Java y el contexto OpenCL.

- **Constructor:** Si se está creando la primera instancia de la clase, el constructor inicializa el contexto OpenCL y los campos nativos globales, correspondientes a los campos estáticos de la clase que han sido inicializados en el código Java original. Cuando se inicializa el contexto OpenCL, el programa OpenCL C obtenido a partir de los métodos anotados como *Declare* y *Parallel* se compila, obteniendo varios kernels que pueden ejecutar código en paralelo. Entonces la instancia se inicializa también, a través de la creación del objeto de instancia nativo y la inicialización de sus miembros con los valores proporcionados por los parámetros de la función.
- **Getters:** Cuando un *getter* es llamado, la variable nativa correspondiente es accedida y su valor devuelto. Cuando un campo es de tipo array, una operación de lectura se añade a la cola de comandos para obtener los valores más recientes del array porque, por motivos de rendimiento, los buffers de OpenCL no se transfieren de nuevo al host cada vez que se ejecuta un kernel. Después, el array nativo de datos obtenido se vuelve a transformar en un tipo de array JNI

que se devuelve y puede ser utilizado dentro del contexto de Java.

- **Setters:** Cuando se hace una llamada a un *setter*, la variable nativa asociada a éste es modificada. Cuando se asigna un nuevo valor a un array, un array nativo es creado para contener

```

1 #include <CL/cl.h>
2 #include <jni.h>
3 ...
4 typedef struct InstanceData {
5     float* gMonoMult;
6     jfloatArray gMonoMultGrayScale;
7     size_t sz_gMonoMult;
8     cl_mem gMonoMult_OCL;
9     int width;
10    int height;
11 } InstanceData;
12
13 const char* program_src =
14 "void __kernel test(__global float* m_gMonoMult, int m_width,
15  "int m_height, __global int* srcPxs, __global int* outPxs) {"
16  "int x = get_global_id(0);
17  "int acc;"
18  "acc = (int)((((srcPxs[x]) & 255) * m_gMonoMult[0]));
19  "...
20  "};
21
22 // Variables globales...
23 // Funciones de soporte...
24
25 JNIEXPORT void JNICALL Java...test(JNIEnv* env,
26  jobject obj, jintArray srcPxsTest, jintArray outPxsTest) {
27  InstanceData* self = getInstanceData(env, obj);
28  // Transformar arrays JNI en arrays nativos
29  int* srcPxs = (env)->GetIntArrayElements(env, srcPxsTest,
30  0);
31  ...
32  // Crear buffers OpenCL y copiar arrays @Map(TO)
33  size_t sz_srcPxs = (env)->GetArrayLength(...) * sizeof(int);
34  cl_mem srcPxs_OCL = clCreateBuffer(...);
35  err = clEnqueueWriteBuffer(queue, srcPxs_OCL, ...);
36
37  // Configurar la lista de parámetros del kernel y ejecutarlo
38  err = clSetKernelArg(kernel_test, 0,
39  sizeof(self->gMonoMult_OCL), &self->gMonoMult_OCL);
40  ...
41  err = clEnqueueNDRangeKernel(...);
42  // Leer objetos @Map(FROM) y actualizar su array JNI asociado
43  err = clEnqueueReadBuffer(queue, outPxs_OCL, ...);
44  ...
45  // Liberar objetos OpenCL y buffers reservados dinámicamente...
46 }
47
48 JNIEXPORT void JNICALL Java...setWidth(JNIEnv* env,
49  jobject obj, int width_PARAM) {
50  InstanceData* self = getInstanceData(env, obj);
51  self->width = width_PARAM;
52 }
53
54 JNIEXPORT void JNICALL Java...setHeight(...) { ... }
55
56 JNIEXPORT void JNICALL Java...initGrayScale(JNIEnv* env,
57  jobject obj, jfloatArray gMonoMultGrayScale_PARAM,
58  int width_PARAM, int height_PARAM) {
59  // Creación del objeto de instancia y actualización del
60  // miembro de la clase instanceData con el nuevo puntero
61  InstanceData* self = malloc(sizeof(InstanceData));
62  jfieldID bufferField = getInstanceDataField(env);
63  (env)->SetIntLongField(env, obj, bufferField, (long)self);
64  // Inicialización de las variables de instancia...
65 }
66
67 JNIEXPORT void JNICALL Java...destroyGrayScale(JNIEnv* env,
68  jobject obj) {
69  // Liberación de los datos de instancia...
70 }
71
72 JNIEXPORT void JNICALL Java...initJNI(JNIEnv* env) {
73  // Inicialización de referencias y variables OpenCL globales
74  jclass cla = (env)->FindClass(...);
75  openCLExceptionClass = (env)->NewGlobalRef(env, cla);
76  ...
77  err = clGetPlatformIds(1, &platform, NULL);
78  ...
79 }
80
81 JNIEXPORT void JNICALL Java...releaseJNI(JNIEnv* env) {
82  // Liberar referencias y variables OpenCL globales...
83 }

```

Fig. 6. Código nativo y OpenCL generado por Paraldroid

esos datos, porque los parámetros utilizan tipos de datos en formato JNI que para poder utilizarse deben ser convertidos a un tipo de datos nativo. Después de esto, un objeto de memoria OpenCL residente en el dispositivo acelerador es creado, y una operación de escritura a ese objeto se añade a la cola de comandos para enviar los datos nativos al mismo. Si había otro array previamente almacenado en este campo, se elimina de los contextos OpenCL y nativo antes de asignarle nuevos valores.

- **Parámetros de métodos:** Los parámetros de los métodos `Parallel` también necesitan una manipulación especial en algunos casos. El parámetro `Index` se elimina de la lista de parámetros y su valor se obtiene en tiempo de ejecución dentro del código OpenCL C. Los parámetros de tipo array se tratan de manera diferente dependiendo del tipo de `Map` que se haya especificado. En cualquier caso, sus valores son obtenidos y transformados en un tipo de datos nativo, y un objeto de memoria de OpenCL se crea para cada uno de ellos. Si el parámetro tiene una anotación `Map(TO)`, una operación de escritura se añade a la cola al principio de la función para enviar los datos nativos a OpenCL. Si el parámetro tiene una anotación `Map(FROM)`, después de que toda la computación haya terminado, una operación de lectura se añade a la cola de comandos para obtener los resultados, que son utilizados para actualizar el parámetro JNI asociado.
- **Destructor:** El destructor revierte las reservas de memoria que hayan sido realizadas por la instancia de la clase que esté siendo eliminada. Es responsable de liberar los objetos de memoria OpenCL, los arrays nativos y el objeto con los datos de instancia. Cuando se llama desde la última instancia existente de la clase, también libera la memoria reservada por los campos estáticos de la clase y las variables globales de OpenCL.

B. Manejo de errores

Una nueva metodología de manejo de errores también ha sido desarrollada como parte de esta extensión a Paraldroid. Fue diseñada para facilitar al desarrollador de aplicaciones la detección y manipulación de errores que pudieran ocurrir en el contexto de destino, y permitir a la aplicación notificar o resolver estos problemas en tiempo de ejecución.

Una clase `OpenCLException` fue creada, la cual es un tipo de `RuntimeException` que conserva datos específicos sobre el error sucedido en OpenCL. Esta excepción contiene el código de error, que puede ser utilizado para buscar la causa raíz del problema, y un mensaje, ya sea con el nombre de archivo y el número de línea donde se detectó el error, o con el informe de compilación si el error sucedió al compilar el código OpenCL C.

Después de cada llamada a una función de la

API OpenCL en el código nativo generado, el código de error devuelto por la función es comprobado y una excepción `OpenCLException` es lanzada si hubo algún error. Estas excepciones pueden ser manejadas desde el código Java del usuario, sin ninguna necesidad de conocer qué es exactamente lo que el código nativo está haciendo.

Esta metodología podría ser adaptada para los otros lenguajes objetivo de Paraldroid para proporcionar al usuario con un modo unificado de manejar errores que ocurren en los contextos de destino.

V. RESULTADOS COMPUTACIONALES

Dejando a un lado futuras investigaciones asociadas a otros requisitos existentes en las aplicaciones para smartphones y tablets, como el consumo energético, validamos el rendimiento del código generado por Paraldroid utilizando cuatro aplicaciones diferentes. Estas aplicaciones están basadas en el benchmark de procesamiento de imágenes de Renderscript [17]: transformar una imagen a escala de grises, ajustar los niveles de una imagen y convoluciones con ventanas de convolución de tamaños 3×3 y 5×5 . En todos los casos, implementamos dos versiones de código: la versión secuencial en Java y la versión Java con anotaciones de Paraldroid. A partir del mismo código anotado, las versiones Renderscript y OpenCL del código fueron automáticamente generadas por Paraldroid.

Las pruebas fueron realizadas en dos dispositivos de características diferentes. Éstos fueron el Odroid-XU3 y el Sony Xperia Z.

- **Odroid-XU3 (etiquetado XU3):** SoC Samsung Exynos 5422 Octa con dos CPUs Quad-core (ARM Cortex-A15 @ 2GHz y ARM Cortex-A7 @ 1,3GHz) con una arquitectura ARM big.LITTLE, 2GB de memoria compartida y una GPU ARM Mali-T628 MP6. Soporta OpenCL 1.1 Full Profile y Renderscript.
- **Sony Xperia Z (etiquetado SXZ):** Chipset Qualcomm APQ8064 Snapdragon S4 Pro con una CPU Krait Quad-core @ 1,5GHz, una GPU Adreno 320 y 2GB de memoria RAM compartida. Soporta OpenCL 1.1 Embedded Profile y Renderscript.

Utilizamos imágenes de tamaños variados para probar el rendimiento de cada una de las implementaciones y dispositivos descritos anteriormente. Para demostrar el rendimiento obtenido con el código generado, medimos los tiempos de ejecución y analizamos las aceleraciones obtenidas con respecto a la línea base establecida por la implementación secuencial en Java. En la Fig. 7 representamos las aceleraciones para el menor y mayor tamaño de imagen utilizado y para el algoritmo de grano más fino y grueso. Definimos la granularidad del algoritmo como la cantidad de cómputo que tiene que realizar cada uno de los elementos de proceso que se ejecutan en paralelo. Observamos que el algoritmo de menor granularidad es el de transformación a escala de grises y el de ma-

yor granularidad, el de convolución con tamaño de ventana 5×5 .

A. Análisis de rendimiento

Para analizar el rendimiento obtenido con los distintos modelos de programación de Android y con nuestro código OpenCL generado automáticamente, comparamos los tiempos de ejecución en cada uno de los problemas propuestos. Las distintas características de cada problema nos permiten analizar el comportamiento de cada modelo de programación en distintas situaciones.

Es importante a la hora de analizar los resultados tener en cuenta que las ejecuciones Rendscript y OpenCL en el dispositivo Odroid-XU3 han sido realizadas por distintos procesadores. Todas las ejecuciones OpenCL se realizan en la GPU del dispositivo que lo soporta, mientras que las ejecuciones Rendscript pueden ser llevadas a cabo tanto en la CPU del dispositivo como en la GPU, y es el sistema operativo el que decide esto en tiempo de ejecución. En nuestro caso todas las ejecuciones Rendscript en el Odroid-XU3 fueron realizadas por la CPU, lo que puede explicar las diferencias de rendimiento desfavorables para OpenCL con respecto a Rendscript en este dispositivo.

Lo primero que observamos es que los problemas de mayor granularidad experimentan mayores aceleraciones en todos los casos. Incluso vemos que, en algunos casos, el código generado es más lento que el código Java secuencial. Esto refuerza la idea que comentamos previamente de que utilizar Rendscript u OpenCL no nos proporciona automáticamente mejoras en el rendimiento, sino que deben aplicarse a problemas e instancias adecuadas. Cuando el problema es de granularidad fina y el tamaño de la entrada es pequeño es mejor mantener la versión original del código, pero un código paralelizable complejo que se deba aplicar a muchos datos es candidato de ejecución a través de Rendscript u OpenCL.

Cuando las ejecuciones se realizan en la GPU, OpenCL demuestra obtener menores tiempos de ejecución que Rendscript. Además, la flexibilidad que nos ofrece podría aprovecharse aún más para distribuir la carga entre todas las plataformas de cómputo disponibles. Sin embargo, el rendimiento del código OpenCL, aun siendo mejor, es más variable con respecto al tamaño de la entrada. La implementación Rendscript, por otro lado, parece mantener una aceleración casi constante para cualquier tamaño del problema. Esto es consistente con el objetivo de diseño de Rendscript, que es conseguir un mejor rendimiento medio a costa de un empeoramiento del rendimiento máximo. Nosotros pretendemos obtener un buen rendimiento de manera muy sencilla que pueda ser posteriormente optimizado a mano si se considera necesario.

VI. CONCLUSIONES Y TRABAJO FUTURO

Paralldroid es un entorno que simplifica el desarrollo de código nativo, Rendscript y OpenCL pa-

ra dispositivos Android. El usuario anota la definición de clases en Java con código secuencial y Paralldroid automáticamente transforma estas definiciones en una implementación realizada en uno de estos lenguajes. El código generado por Paralldroid puede ejecutarse en la CPU o en la GPU, y los detalles de implementación se abstraen al programador de aplicaciones.

En este artículo hemos presentado una extensión a Paralldroid que permite la generación automática de código para, de forma transparente, integrar la ejecución de código OpenCL en una aplicación Android escrita en Java. Esto mejora enormemente la productividad de los desarrolladores de aplicaciones sin tener que comprometer el rendimiento de éstas.

Los resultados demuestran que el código OpenCL consigue un rendimiento superior a Rendscript en la mayoría de instancias de nuestras pruebas donde la GPU es utilizada para llevar a cabo la computación. Las mejoras con respecto al código Java secuencial son claras, incluso aunque las diferencias en el código son muy reducidas. Como OpenCL es un estándar abierto disponible en multitud de plataformas de todo tipo, esta nueva extensión abre la puerta a la utilización de Paralldroid para acelerar cualquier aplicación Java, y no sólo Android.

Aún hay margen para mejoras futuras sobre el código OpenCL generado por Paralldroid, puesto que aquí presentamos una extensión directa y sencilla de Paralldroid para la ejecución de código OpenCL. Hay un gran número de optimizaciones que pueden hacer que el código generado sea más rápido y utilice más eficientemente los recursos disponibles.

- Utilización de todos los dispositivos visibles por OpenCL para acelerar el cómputo más allá de una única GPU, si el hardware lo permite. Se requiere una metodología de balanceo de cargas que permita aprovechar las posibilidades de cada dispositivo.
- En las arquitecturas de memoria unificada, la memoria principal está compartida entre los distintos dispositivos de proceso y aceleradores. Sería conveniente, por tanto, utilizar esta característica en nuestro favor para reducir la sobrecarga de las copias y transferencias de datos entre la CPU y los dispositivos aceleradores.
- Utilización de un contexto OpenCL global compartido por todas las clases generadas. Actualmente en Paralldroid se crea un contexto OpenCL para cada una de las clases generadas, aunque lo más adecuado sería compartir los datos globales como la cola de comandos o el identificador del dispositivo, ya que utilizan el mismo hardware. Esto puede redundar en un mejor aprovechamiento del hardware y en una disminución de la sobrecarga asociada a la gestión de los contextos OpenCL.
- Sería posible implementar paralelismo a nivel de tareas y no sólo a nivel de datos mediante la ejecución asíncrona de los kernels y la gestión de un grafo de dependencias en tiempo de ejecución.

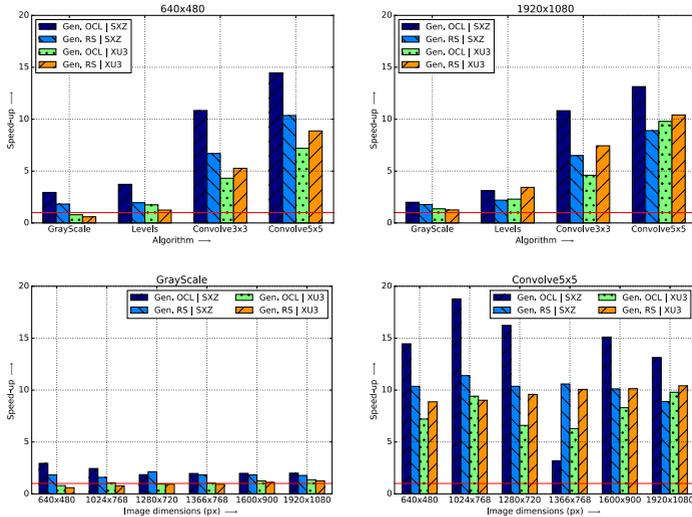


Fig. 7. Aceleración obtenida con respecto a la versión Java secuencial

- Permitir la vectorización del código puede hacer posible un mejor aprovechamiento de los dispositivos, ya sea mediante vectorización automática o explícita.

AGRADECIMIENTOS

Este trabajo ha sido apoyado por la UE (FEDER), el Ministerio de Educación, Cultura y Deporte a través del proyecto TIN2011-24598, por la red CAPAP-H4 y la NESUS IC1315 COST Action.

REFERENCIAS

- [1] NVIDIA, "Tegra mobile processors: Tegra2, Tegra 3 and Tegra 4," <http://www.nvidia.com/object/tegra-superchip.html>.
- [2] Qualcomm, "Snapdragon mobile processors," <http://www.qualcomm.com/snapdragon>.
- [3] Samsung, "Exynos mobile processors," <http://www.samsung.com/global/business/semiconductor/minisite/Exynos/>.
- [4] Memoir Systems, "Algorithmic Memory™ technology," <http://www.memoir-systems.com/>.
- [5] Nvidia, "GPUDirect Technology," <http://developer.nvidia.com/gpudirect>.
- [6] Anandtech, "AMD Outlines HSA Roadmap: Unified Memory for CPU/GPU in 2013, HSA GPUs in 2014," <http://www.anandtech.com/show/5493/>.
- [7] Google, "Android mobile platform," <http://www.android.com>.
- [8] Apple, "iOS: Apple mobile operating system," <http://www.apple.com/ios>.
- [9] Microsoft, "Windows Phone: Microsoft mobile operating system," <http://www.microsoft.com/windowsphone>.
- [10] Xiao Feng Li Xi Qian, Guangyu Zhu, "Comparison and analysis of the three programming models in google android," *First Asia-Pacific Programming Languages and Compilers Workshop (APPLC)*, June 2012.
- [11] Alejandro Acosta, Sergio Afonso, and Francisco Almeida, "Extending paralldroid with object oriented annotations," *Parallel Computing*, 2016.
- [12] A. Acosta and F. Almeida, "The particle filter algorithm: parallel implementations and performance analysis over android mobile devices," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 3, pp. 788–801, 2016.
- [13] Clément Valentin, Senn Christian, Knönen Pierre, Kilchoer Pr Francois, and Roche Jean-François, "Parallel object programming with java," <http://gridgroup.hefr.ch/popj/doku.php>.
- [14] Patrick Viry, "Ateji px for java-parallel programming made simple," *Ateji White Paper*, 2010.
- [15] Christophe Dubach, Perry Cheng, Rodric Rabbah, David F. Bacon, and Stephen J. Fink, "Compiling a high-level language for gpus: (via language support for architectures and compilers)," *SIGPLAN Not.*, vol. 47, no. 6, pp. 1–12, June 2012.
- [16] Richard Membarth, Oliver Reiche, Frank Hannig, and Jürgen Teich, "Code generation for embedded heterogeneous architectures on android," in *DATE*, 2014, pp. 1–6.
- [17] AOSP, "Android Open Source Project," <http://source.android.com/>.

Detección automática de los patrones de paralelismo Map y Farm en códigos secuenciales

David del Río Astorga¹, Manuel F. Dolz¹, Luis Miguel Sanchez¹ y J. Daniel García¹

Resumen.— A pesar de que los centros de cómputo incorporan arquitecturas paralelas multinúcleo, posiblemente con aceleradores y coprocesadores, una gran parte de estos recursos paralelos todavía siguen estando infrautilizados. Esto es debido a que las aplicaciones que se ejecutan hoy en día siguen siendo en gran parte secuenciales. Para hacer frente a este problema, la comunidad HPC desarrolla *frameworks* y modelos de programación que facilitan la paralelización de dichas aplicaciones. Por ejemplo, algunos *frameworks* permiten expresar de forma sencilla concurrencia a través de patrones paralelos en las aplicaciones secuenciales. No obstante, teniendo en cuenta que estas aplicaciones contienen miles, o incluso millones, de líneas de código, el esfuerzo necesario para identificar regiones paralelas es extremadamente alto. Para simplificar esta tarea, este artículo presenta una herramienta que analiza el código fuente de aplicaciones secuenciales, cuyo objetivo es detectar automáticamente regiones de código, que pueden ser ejecutadas en paralelo, y anotarlas usando pragmas del modelo de programación OpenMP. Concretamente, se detectan los patrones paralelos Map y Farm. Finalmente, a través de un análisis experimental, se evalúa la calidad de la detección y la mejora del rendimiento sobre un conjunto de *benchmarks* implementados en C++.

Palabras clave.— Análisis estático de código fuente, Paralelización automática, Patrones paralelos, OpenMP.

I. INTRODUCCIÓN

PESE a que la mayor parte de los componentes hardware actuales, tales como los procesadores multinúcleo, GPUs y/o co-procesadores, han sido diseñados para computar en paralelo, la mayor parte del software de producción sigue siendo secuencial [1]. En otras palabras, una gran parte de los recursos informáticos proporcionados por arquitecturas modernas están mayormente infrautilizados. Con el fin de aprovechar estos recursos, es necesario transformar software secuencial en código paralelo. Esta tarea requiere, sin embargo, un gran esfuerzo por parte de la comunidad científica e industrial. Para hacer frente a este problema, se han desarrollado diferentes soluciones para explotar de manera eficiente las arquitecturas paralelas [2]. Un ejemplo son los patrones paralelos junto con sus implementaciones, disponibles en *frameworks* de programación paralela. Algunos ejemplos de estos *frameworks* son OpenMP, Cilk o Intel TBB, para arquitecturas de memoria compartida; MPI o Hadoop, para plataformas distribuidas; y OpenCL o CUDA, orientados a diferentes aceleradores.

Múltiples ejemplos de aplicaciones, que debido a su naturaleza podrían aprovechar el paralelismo ofreci-

do por las arquitecturas actuales, son las de cómputo intensivo, comúnmente presentes en los ámbitos científicos e industriales. Un simple análisis sobre sus códigos revelaría que una gran mayoría de sus algoritmos y métodos podrían ser transformados aplicando patrones paralelos [3]. La forma tradicional de solventar este problema consiste en transformar manualmente el código en su versión paralela. Sin embargo, en la mayoría de los casos, esta tarea resulta compleja de realizar para aplicaciones de gran escala. Como alternativa se pueden utilizar herramientas de refactorización de código que transforman, de forma automática o semiautomática, el código secuencial en paralelo [4]. Aunque los códigos generados mediante estas herramientas a menudo no alcanzan el mayor rendimiento posible, ayudan en gran medida a reducir el tiempo necesario para llevar a cabo dicha transformación [5].

Por desgracia, las herramientas de refactorización se encuentran todavía en un estado muy prematuro. De hecho, muchas de éstas son supervisadas, siendo el desarrollador el único responsable de señalar las secciones específicas del código a refactorizar. Una de las claves para convertir este proceso semiautomático a completamente automático, son las herramientas de detección de patrones paralelos. Este hecho motiva el trabajo presentado en este artículo: una herramienta capaz de analizar de forma estática códigos secuenciales, con el objetivo de detectar y anotar patrones paralelos utilizando el modelo de programación paralelo OpenMP. Esta detección se lleva a cabo *offline*, reduciendo considerablemente los costes asociados a técnicas de *profiling* en tiempo de ejecución.

En general, este trabajo presenta las siguientes contribuciones:

- Se propone una herramienta capaz de analizar códigos secuenciales C++, detectar patrones paralelos y anotar, de forma automática, las aplicaciones analizadas para transformarlas en paralelas.
- Se proporciona soporte para la detección de los patrones paralelos Farm y Map.
- Se evalúa la calidad de la detección de patrones y el rendimiento de las aplicaciones paralelizadas con respecto a las versiones implementadas por un desarrollador.

Este documento se estructura de la siguiente manera: en la Sección II se revisa el estado de la cuestión sobre refactorización paralela y las herramientas de detección de patrones paralelos existentes en la literatura. En la Sección III se describen los patrones paralelos soportados. En la Sección IV se detalla el

¹Dpto. de Informática, Universidad Carlos III de Madrid, 28911-Leganés, e-mail: drio@pa.uc3m.es, {mdolz, lmsansche, jdgarcia}@inf.uc3m.es.

diseño de la arquitectura software de la herramienta, incluyendo los módulos de detección y anotación de los patrones paralelos *Map* y *Farm*. En la sección V se analizan y evalúan los resultados obtenidos experimentalmente. Por último, la Sección VI concluye este trabajo y enumera posibles trabajos futuros en esta línea.

II. ESTADO DE LA CUESTIÓN

En la literatura se pueden encontrar múltiples trabajos relacionados con la detección y refactorización de regiones paralelas. Sin embargo, este proceso está completamente ligado al lenguaje de programación necesitando, en la mayoría de ocasiones, técnicas de *profiling*. Por ejemplo, el trabajo realizado por Sean Rul et al. [6] utiliza LLVM para instrumentar bucles encontrados en el código secuencial. Mediante esta técnica es posible extraer información a nivel de LLVM-IR, permitiendo decidir si estos bucles pueden ser transformados usando algún patrón paralelo para realizar un proceso de refactorización. No obstante, esta aproximación presenta algunos problemas: *i*) es necesario instrumentar el código, lo que puede provocar variaciones en los resultados, *ii*) los resultados se obtienen después de múltiples ejecuciones de los programas, lo que requiere más tiempo para realizar el análisis y *iii*) el lenguaje usado por los autores es C, lo que limita el uso de estructuras y características de lenguajes más complejos como C++. Nuestra aproximación hace frente a estas limitaciones en dos aspectos: *i*) se realiza un análisis *offline* del código y se evita el proceso de *profiling* y *ii*) se soportan los lenguajes C y C++, lo que amplía el rango de aplicaciones donde se puede utilizar la herramienta de paralelización automática.

Otro trabajos, tales como el propuesto por Moli-toriz et al. [7] permiten detectar regiones de código potencialmente paralelizables mediante patrones paralelos. No obstante, esta técnica no comprueba que existan dependencias de datos, por lo que no se garantiza un funcionamiento correcto del código paralelo generado. Como solución a este problema los autores proponen el uso de técnicas para detectar condiciones de carrera en tiempo de ejecución. Nuestra propuesta lleva a cabo un análisis de los accesos a memoria en el código, lo que reduce el tiempo de detección de patrones. De forma similar, PoCC [8] es capaz de detectar y paralelizar bucles mediante el uso de técnicas de compilación poliédrica. Sin embargo, no es capaz de detectar patrones paralelos de alto nivel, tales como aquellos que paralelizan a nivel de tarea. Por otra parte, existen otros trabajos que realizan este tipo de búsqueda. Por ejemplo, Disco-PoP [9] utiliza técnicas de grafos de dependencias para detectar patrones. Una limitación importante de esta herramienta es que requiere el uso de técnicas de *profiling*, lo que implica un mayor tiempo de análisis. Otro ejemplo similar es el presentado por Tournavitis et al. [10], el cual permite detectar y transformar código secuencial en paralelo incluyendo la detección de patrones de alto nivel. Finalmente, la herramien-

ta *FreshBreeze* [11] aprovecha el uso de técnicas de análisis estáticas de código que permiten detectar dependencias de flujo y datos, y transforma el código mediante el uso de árboles estructurados de tareas.

Es importante destacar que las técnicas basadas en el análisis estático del código para detectar patrones paralelos no están todavía muy extendidas, debido a esta tarea resulta mucho más compleja que si se realiza de forma dinámica. En cambio, en el caso de lenguajes funcionales es posible simplificar esta tarea en una mayor medida que en el caso de los lenguajes estructurados. Por ejemplo, István Bozó et al. [12] presentan una herramienta capaz de detectar patrones paralelos estáticamente en un código secuencial Erlang y transformarlo a su correspondiente código paralelo. Sin embargo, la herramienta requiere técnicas de *profiling* para poder decidir qué patrón paralelo es el más adecuado para refactorizar en caso de conflicto.

III. PATRONES PARALELOS

En esta sección se ofrece una breve descripción acerca de los patrones paralelos *Farm* y *Map* soportados por la herramienta de detección de patrones [13]. Al mismo tiempo también se detalla cómo éstos se pueden paralelizar utilizando el modelo de programación paralela OpenMP.

A. Patrón paralelo *Farm*

El patrón paralelo de flujo *Farm* calcula de forma paralela la misma función $f: \alpha \rightarrow \beta$ sobre todos los elementos que aparecen en el flujo de entrada. Por lo tanto, para cada elemento x_i en el flujo de entrada, se genera un elemento $f(x_i)$ en el flujo de salida. En este patrón, los cómputos de f relativos a diferentes elementos en el flujo son completamente independientes y pueden ser procesados en paralelo. La implementación paralela del patrón *Farm* puede ser realizada por un conjunto de entidades $\{W_1, W_2, \dots, W_N\}$, también llamados hilos de trabajo, que calculan la función f en paralelo en diferentes tareas. Sin embargo, el cálculo de $f(x_i)$ sólo puede comenzar cuando x_i esté disponible en el flujo de entrada. Por lo tanto, si el tiempo de llegada de los elementos en el flujo de entrada es T_a y el cálculo de f se realiza en T_f , cómo máximo se computarán $\frac{T_a}{T_f}$ elementos en paralelo. Esta cifra representa la cota superior del número de hilos de trabajo que pueden ser utilizados en paralelo durante el cómputo de un determinado patrón *Farm*.

Código 1: Patrón *Farm* usando directivas OpenMP.

```

1 #pragma omp parallel
2 {
3     #pragma omp single nowait
4     {
5         while( !exit )
6         {
7             #pragma omp task
8             { ...
9             }
10        }
11    }
12    #pragma omp taskwait
13 }
14 }
    
```

De forma práctica, un patrón `Farm` puede ser paralelizado usando directivas del modelo de programación paralela `OpenMP`. Esta implementación se muestra en el Código 1. Para ello, se utiliza una región paralela `omp parallel` en la que un único hilo crea, de forma consecutiva, las tareas que se generan en cada iteración del bucle. Una vez el bucle termina, el hilo que creó las tareas espera con la directiva `omp taskwait` a que éstas finalicen para completar la ejecución del patrón.

B. Patrón paralelo Map

El patrón paralelo de datos `Map` calcula la función $f : \alpha \rightarrow \beta$ sobre los elementos de la colección de datos de entrada, donde los elementos de entrada y salida tienen el tipo α y β , respectivamente. El resultado de salida es una colección de elementos y_1, y_2, \dots, y_N , donde $y_i = f(x_i)$, para $i = 1, 2, \dots, N$ y x_i , es el elemento correspondiente en la colección de entrada. El único requisito del patrón `Map` es que la función f debe ser una función pura. Dado que cada elemento en la colección de datos de entrada es independiente de los otros elementos, todos ellos se pueden calcular en paralelo. Sin embargo, el número máximo de elementos que se pueden calcular en paralelo depende del grado de paralelismo del patrón, es decir, del número de hilos utilizados. A pesar de que los patrones `Farm` y `Map` parecen similares, la diferencia radica en el hecho de que el patrón `Farm` trabaja en sobre un flujo de datos de entrada, cuya longitud es desconocida, mientras que `Map` recibe una colección de datos con un número fijo de elementos. A pesar de ello, el enfoque de la paralelización del patrón `Map` es bastante similar al del `Farm`.

Código 2: Patrón `Map` usando directivas `OpenMP`.

```
1 #pragma omp parallel for
2 for(int i = 0; i < N; i++)
3 {
4     ...
5 }
```

Del mismo modo, el patrón `Map` se puede paralelizar de forma sencilla utilizando directivas de `OpenMP`. La implementación paralela de este patrón, usando este modelo de programación, se muestra en el Código 2. En este caso sólo es necesario emplear la directiva `omp parallel for`. Opcionalmente se pueden especificar estrategias de planificación estáticas o dinámicas, de modo que las iteraciones del bucle `for` se repartan de forma estática o dinámica entre los hilos de trabajo involucrados en la ejecución del patrón `Map`.

IV. DETECCIÓN DE PATRONES PARALELOS

Esta sección presenta la herramienta desarrollada en este trabajo, la cual es capaz de analizar de forma estática código C/C++ secuencial, detectar patrones paralelos y anotarlos, usando directivas `OpenMP`. De este modo las aplicaciones secuenciales procesadas con esta herramienta pueden ser transformadas automáticamente en paralelas.

A. Herramienta de análisis de patrones paralelos

En esta sección se describe en detalle la herramienta de análisis y detección de patrones paralelos desarrollada [14]. Específicamente, esta herramienta utiliza la API ofrecida por el compilador `Clang` para generar el árbol de sintaxis abstracta (AST), recorrerlo y analizarlo con el objetivo de identificar patrones paralelos en el código fuente. En esta versión de la herramienta se contemplan los patrones `Map` y `Farm`, no obstante se puede extender fácilmente para dar soporte a otro tipo de patrones paralelos, e.g., `Pipeline`.

El flujo de trabajo de esta herramienta se muestra en el esquema de la Figura 1. En primer lugar, la herramienta recibe los códigos fuente secuenciales que deben ser analizados y, a continuación, se ejecutan los siguientes pasos:

1. *Detección de bucles.* En este paso se detectan bucles que pueden ser potencialmente transformados en patrones paralelos. Básicamente, la herramienta recorre el AST y extrae información de los nodos asociados a estructuras de control iterativas, i.e., `for`, `while`, `do...while`, etc. Concretamente, se recoge información relacionada con las variables, llamadas a funciones y sentencias condicionales utilizadas en dichos bucles. Por otra parte, también se guarda la información sobre las funciones implementadas en el código de usuario.
2. *Extracción de características.* En este paso se utiliza la información almacenada en el paso previo para extraer características específicas sobre declaración de variables, referencias, llamadas a funciones, bucles internos, accesos a memoria y otro tipo de operaciones realizadas en el cuerpo de los bucles. A continuación, para cada sentencia analizada, también se almacena información sobre la localización en el código fuente original y tipos de acceso a variables locales y globales (escritura o lectura).
3. *Comprobación del tipo de acceso en argumentos de funciones.* En este último paso la herramienta comprueba, si es posible, el tipo de acceso sobre las variables pasadas como argumentos a funciones. En caso de que estas variables sean pasadas por valor o como referencias constantes, la herramienta las anota como de lectura. Sin embargo, para las variables pasadas por referencia es necesario llevar a cabo uno de los siguientes pasos:
 - a) Si la función está disponible en el código de usuario, se analiza el tipo de accesos sobre dicha variable en el cuerpo de la función. Si se detecta que la variable no se modifica, se anota como de sólo lectura. En caso contrario, si se detectan accesos de escritura, la herramienta anota que la variable se modifica durante la llamada a la función. Alternativamente, si se detectan dependencias de datos de tipo RAW

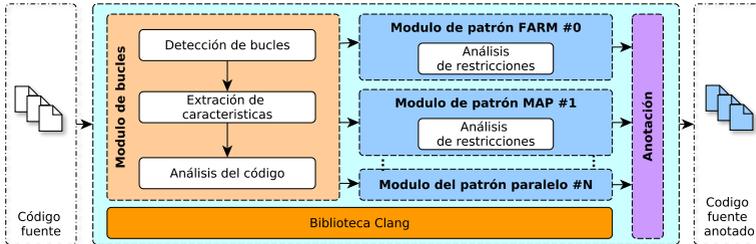


Fig. 1: Flujo de trabajo de la herramienta de detección y anotación de patrones paralelos.

(Read-After-Write), el tipo de acceso se anota como de lectura/escritura, puesto que dicho argumento podría generar una posible dependencia, o realimentación entre las iteraciones del bucle analizado.

- b) Por lo contrario, si no se dispone del código fuente de la función, no es posible analizar el tipo de accesos (lectura o escritura) sobre los argumentos de la misma. En este caso, la herramienta toma una decisión conservadora: se asigna el tipo lectura/escritura para todos los argumentos de la función. A pesar de ello, la herramienta inserta el nombre de la función y los tipos de acceso de sus argumentos en un diccionario con el objetivo de mejorar el proceso de análisis en futuros usos de la herramienta. De este modo, se da la opción al usuario de modificar posteriormente el diccionario para que pueda establecer correctamente los tipos de acceso de argumentos de estas funciones.

Más tarde, los bucles que han sido identificados en esta etapa se pasan a los módulos de patrones, encargados de determinar si un bucle cumple con las propiedades de un determinado patrón. Los módulos que soporta la herramienta (*Farm* y *Map*) se describen con mayor detalle en las siguientes secciones.

B. Módulo de detección del patrón *Farm*

El módulo del patrón *Farm* se encarga de comprobar si un determinado bucle cumple con las características de este esqueleto de programación paralela. En especial, se comprueba si el cuerpo de éste puede ser ejecutado en paralelo de forma independiente, es decir, sin que haya dependencias entre iteraciones. Nótese que, al tratarse de un patrón *Farm*, el número de elementos procesados puede que sólo se conozca en tiempo de ejecución. Para ello, se analiza si las instrucciones en el cuerpo de dicho bucle cumplen la condición de función pura. Concretamente se verifican las siguientes restricciones:

- *No deben existir dependencias RAW*. Esta restricción determina que no pueden existir dependencias de tipo RAW entre iteraciones del bucle,

es decir, las iteraciones deben ser independientes entre sí.

- *No se modifican variables globales*. Esta condición asegura que no hayan accesos de tipo escritura en variables globales durante la ejecución del bucle.
- *No deben existir sentencias de ruptura*. Finalmente, se comprueba que no existan sentencias de ruptura, e.g., *continue*, *break* o *return* en el cuerpo del bucle, en cuyo caso éste no puede ser paralelizado utilizando el patrón *Farm*.

En definitiva, si todas las condiciones anteriores se cumplen, la herramienta anota automáticamente el bucle usando las directivas de OpenMP que implementan el patrón *Farm*, tal y como se especifica en el Código 1.

C. Módulo de detección del patrón *Map*

El módulo para el patrón *Map* verifica si un determinado bucle satisface las condiciones de este esqueleto de programación paralela. Como ya se ha comentado, este patrón representa un código paralelo que ejecuta una función pura. Por otra parte, se añade la condición de que se generen elementos de salida sobre las condiciones impuestas para el patrón *Farm*. Nótese también que, a diferencia de *Farm*, para el patrón *Map* el número total de elementos de entrada se conoce de antemano. Para que un bucle cumpla con las condiciones de este patrón, el módulo verifica que se cumplan las siguientes condiciones:

- *Número conocido de elementos*. Los datos de entrada deben estar declarados y asignados antes del comienzo del bucle en cuestión.
- *Debe existir, al menos, un elemento de salida*. Según la definición del patrón *Map*, la función pura del patrón o cuerpo del bucle, un elemento de entrada se procesa para producir uno de salida. En otras palabras, el conjunto de elementos de salida debe tener la misma cardinalidad que el conjunto de elementos de entrada.

Finalmente, aquellos bucles que cumplen con las restricciones del patrón *Map* se anotan utilizando directivas de OpenMP que implementan dicho patrón en paralelo, tal y como se muestra en el Código 2. De

este modo, una vez la aplicación ha sido procesada con esta herramienta y compilada, la ejecución de los bucles identificados como **Map** o **Farm** y secuenciales en la versión original, se lleva a cabo en paralelo.

V. EVALUACIÓN

En esta sección se lleva a cabo una evaluación experimental de la herramienta de detección y anotación de patrones paralelos en diferentes *benchmarks* de aplicaciones científicas. Para ello, se utilizan los siguientes componentes hardware y software:

- **Plataforma.** La evaluación se ha realizado en una plataforma compuesta por 2 procesadores Intel Xeon Ivy Bridge E5-2695 v2 con un total de 24 núcleos a una frecuencia de 2.40 GHz, una caché L3 de 30 MB y 128 GB de memoria RAM tipo DDR3. El sistema operativo utilizado es Linux Ubuntu 14.04.2 LTS con el *kernel* v3.13.0-57.
- **Software.** La herramienta de detección y anotación de patrones paralelos se ha compilado usando la biblioteca de Clang parte de la infraestructura LLVM v3.7.0. Las anotaciones generadas por la herramienta utilizan el modelo de programación OpenMP v4.5.
- **Benchmarks.** Para evaluar la herramienta se han utilizado las versiones secuenciales de algunos de los *benchmarks* de las suites *Rodinia* [15] y *Parboil* [16]. Concretamente de la suite *Rodinia* se han utilizado los *benchmarks* *backprop*, *cfd*, *heartwall* y *nw*, mientras que de *Parboil* se usa *sgemm* y *spmv*. También se emplean las versiones paralelas proporcionadas por estas suites como punto de referencia para evaluar el rendimiento de los *benchmarks* paralelizados de forma automática.

En las siguientes secciones se analiza de forma cuantitativa tanto la calidad de la detección de patrones como el rendimiento de las aplicaciones automáticamente paralelizadas utilizando la herramienta presentada en este artículo. También se evalúan de forma cualitativa las ventajas e inconvenientes que aporta la misma.

A. Análisis de la detección de patrones

En primer lugar, se evalúa la calidad de la detección de patrones de la herramienta. Para ello, se lleva a cabo una comparación entre la inspección manual y la automática, usando la herramienta, de los bucles que aparecen en los códigos fuentes de los *benchmarks* analizados. Para llevar a cabo un estudio doble ciego, la inspección manual de bucles se ha realizado antes de la automática. Para cada *benchmark*, se contabiliza el número de bucles y patrones paralelos detectados. Después, se discuten los resultados con el objetivo de demostrar la calidad de la herramienta para la detección de patrones.

La Tabla I presenta el número de bucles totales, y el número de patrones detectados manualmente y por la herramienta de forma automática. Entre paréntesis

TABLE I: Resultados del proceso detección de patrones manual vs. automático. El valor principal indica el número de patrones detectados, mientras que la cifra entre paréntesis especifica cuántos de éstos han sido anotados con OpenMP.

Benchmark	Bucles	Manual		Automático	
		Farm	Map	Farm	Map
backprop	28	5 (0)	5 (1)	4 (0)	4 (1)
cfd	78	23 (0)	23 (22)	24 (3)	16 (13)
heartwall	54	4 (0)	3 (1)	18 (0)	18 (17)
nw	12	6 (0)	6 (2)	6 (0)	6 (4)
sgemm	5	2 (0)	2 (1)	2 (1)	2 (1)
spmv	3	2 (0)	2 (1)	2 (0)	2 (1)

se especifican aquellos que han sido finalmente anotados con directivas OpenMP. Como se puede observar, la herramienta desarrollada es capaz de detectar automáticamente la mayor parte de los bucles identificados manualmente como patrones **Map** y **Farm**. Las diferencias existentes entre ambos análisis se deben a que la herramienta detecta los patrones anidados, mientras que un desarrollador, de forma manual, los descarta directamente. Con respecto al número de patrones anotados, se puede observar que, en muchos casos, la herramienta anota más patrones que el desarrollador. Esto es debido a que la herramienta no tiene en cuenta la cantidad de cómputo de en el cuerpo de los bucles, por lo que anota aquellos que son de inicialización pero que el desarrollador desprecia. Una última observación a estos datos revela que el número de patrones detectados como **Map** y **Farm** son muy similares entre sí, siendo esta diferencia los patrones **Farm** que no pueden ser representados usando **Map**. Esto es debido a que los patrones detectados como **Map** pueden ser tratados a su vez como **Farm**.

B. Análisis del rendimiento de los benchmarks paralelizados de forma automática

A continuación, se evalúa el rendimiento de las aplicaciones paralelizadas de forma automática con respecto a las versiones paralelas desarrolladas de forma manual. Para llevar a cabo este estudio, se han ejecutado ambas versiones paralelas de los *benchmarks* analizados con diferente número de hilos. Los speedups obtenidos para dichas aplicaciones se muestran en la Figura 2. Como se puede observar, los rendimientos generados de las aplicaciones paralelizadas automáticamente alcanzan valores muy similares a los rendimientos de las versiones paralelas por defecto. Por otra parte, se aprecia que el rendimiento de alguno de los *benchmarks* es ligeramente inferior. Como por ejemplo *backprop* o *nw*. Esto es debido a que la herramienta presentada sólo paraleliza aquellos bucles que pueden ser transformados utilizando alguno de los patrones soportados. Además, en las versiones paralelas proporcionadas por las suites, se encuentran paralelizados bucles que encajan con otro tipo de patrones no soportados todavía por la herramienta, como por ejemplo, *Reduce* o *Stencil*. En general, gracias al uso de esta herramienta, se pueden obtener mejoras sustanciales en el rendimiento de los

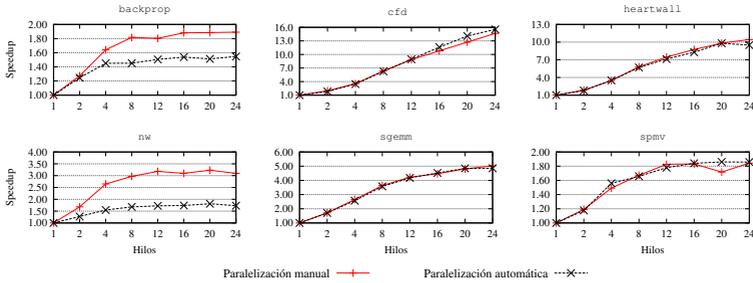


Fig. 2: Speedup con diferente número de hilos para las versiones OpenMP paralelizadas de forma manual y automática de los *benchmarks* analizados.

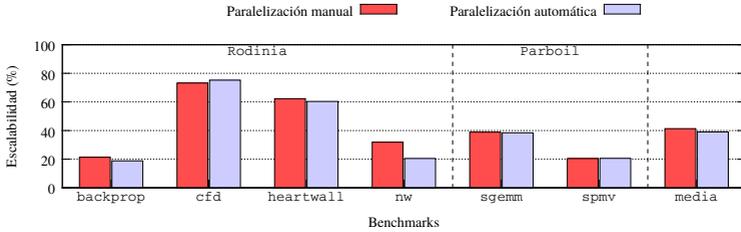


Fig. 3: Escalabilidad para las versiones OpenMP paralelizadas de forma manual y automática de los *benchmarks* analizados y su valor medio.

benchmarks con un esfuerzo relativamente menor al de realizar una paralelización de forma manual.

Por otra parte, la Figura 3 muestra valores de la escalabilidad obtenida durante la ejecución de los *benchmarks* paralelos de forma manual y automática. L muestra la escalabilidad de las aplicaciones analizadas. Nótese que estos porcentajes han sido obtenidos tras el cómputo de la media aritmética de los *speedups* normalizados para diferentes número de hilos de los datos de la Figura 2. Como se muestra, para prácticamente todos los casos estudiados, la escalabilidad de la versión automática está muy cerca de la manual. Como excepción, el *benchmark* *nw* escala en un menor grado que la versión manual, debido a que la herramienta no ha sido capaz de paralelizar un bucle con una cantidad de cómputo relativamente alta. Concretamente, dicho bucle accede a las posiciones un vector cuyo índice es desconocido en tiempo de compilación y, aunque no genere dependencias entre iteraciones, no es posible de determinar de forma estática este hecho. Finalmente, para tener una visión global de la escalabilidad generada, la última columna de la gráfica muestra la media aritmética de las escalabilidades de los *benchmarks* estudiados. La importancia de esta cifra no radica en la escalabilidad obtenida en sí, sino en la diferencia entre ambas versiones, manual y automática. Como se observa, la diferencia es menor del 5 % y puede tomarse co-

mo un indicador de los beneficios aportados por esta técnica de paralelización automática.

En general, tras esta evaluación, se desprenden las siguientes observaciones: *i)* la herramienta obtiene cifras de rendimiento cercanas a las versiones implementadas de forma manual, con un esfuerzo mucho menor que el que debería dedicar un desarrollador; y *ii)* en ciertos casos, la herramienta no genera la misma escalabilidad que la versión paralela por defecto, debido a que el desarrollador tiene un conocimiento global de la aplicación.

VI. CONCLUSIONS AND FUTURE WORKS

En este artículo se ha presentado una herramienta capaz de detectar bucles que pueden ser paralelizados utilizando los patrones paralelos *Farm* y *Map* y anotarlos convenientemente usando el modelo de programación paralela OpenMP. La evaluación experimental demuestra que la herramienta presentada es capaz de encontrar y paralelizar la mayor parte de los bucles que el desarrollador había paralelizado en la correspondiente versión de las aplicaciones analizadas. Esta técnica permite reducir el esfuerzo necesario para el desarrollo de aplicaciones paralelas.

Como se ha visto, la herramienta presentada en este artículo se diferencia de otras en tres aspectos principales: *i)* realiza un análisis estático y evita el uso de técnicas de *profiling*, y por lo tanto, es una

aproximación mucho más rápida que otras que llevan a cabo el análisis en tiempo de ejecución; y *ii*) garantiza que los patrones detectados cumplen una serie de requisitos que garantizan la corrección de la versión paralela generada.

Como trabajo futuro, se prevé la ampliación la herramienta con otros módulos de detección de patrones, por ejemplo, Pipeline, Reduce y Stencil. Además, se planea extender la herramienta con un sistema de decisión que seleccione el patrón paralelo más adecuado para un código dado, proporcionando el mayor rendimiento. Un último objetivo es extender el módulo de anotación, para poder paralelizar el código fuente usando otros modelos de programación paralela, como por ejemplo, Intel TBB [17] y el futuro STL paralelo de C++17.

AGRADECIMIENTOS

Este trabajo ha sido financiado por los Proyectos Europeos ICT 644235 “REPHRASE: REfactoring Parallel Heterogeneous Resource-Aware Applications” y FP7 609666 “REPARA: Reengineering and Enabling Performance And poweR of Applications”.

REFERENCIAS

- [1] H. Sutter, “Welcome to the Jungle,” <http://herbutsutter.com/welcome-to-the-jungle/>, 2012. [Last access 6th May 2015].
- [2] L. M. Sanchez, J. Fernandez, R. Sotomayor, S. Escolar, and J. D. Garcia, “A comparative study and evaluation of parallel programming models for shared-memory parallel architectures,” *New Generation Computing*, vol. 31, no. 3, pp. 139–161, 2013.
- [3] Michael McCool, James Reinders, and Arch Robison, *Structured Parallel Programming: Patterns for Efficient Computation*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.
- [4] Christopher Brown, Kevin Hammond, Marco Danelutto, Peter Kilpatrick, Holger Schöner, and Tino Breddin, “Paraphrasing: Generating parallel programs using refactoring,” in *Formal Methods for Components and Objects*, Bernhard Becker, Ferruccio Damiani, Frank S. de Boer, and Marcello M. Bonsangue, Eds., vol. 7542 of *Lecture Notes in Computer Science*, pp. 237–256. Springer Berlin Heidelberg, 2013.
- [5] Anne Meade, Jim Buckley, and J. J. Collins, “Challenges of evolving sequential to parallel code: An exploratory review,” in *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, New York, NY, USA, 2011, IWPSE-EVOL ’11, pp. 1–5, ACM.
- [6] Sean Rul, Hans Vandierendonck, and Koen De Bosschere, “A profile-based tool for finding pipeline parallelism in sequential programs,” *Parallel Computing*, vol. 36, no. 9, pp. 531 – 551, 2010.
- [7] Korbinian Mollitorisz, Tobias Müller, and Walter F. Tichy, “Patty: A pattern-based parallelization tool for the multicore age,” in *Proceedings of the Sixth International Workshop on Programming Models and Applications for Multicores and Manycores*, New York, NY, USA, 2015, PMAM ’15, pp. 153–163, ACM.
- [8] Louis-Noël Pouchet, Uday Bondhugula, Cédric Bastoul, Albert Cohen, J. Ramanujam, and P. Sadayappan, “Hybrid iterative and model-driven optimization in the polyhedral model,” Tech. Rep. 6962, INRIA Research Report, June 2009.
- [9] Zhen Li, Rohit Atre, Zia Ul-Huda, Ali Jannesari, and Felix Wolf, “DiscoPop: A profiling tool to identify parallelization opportunities,” in *Tools for High Performance Computing 2014*, chapter 3, pp. 37–54. Springer International Publishing, Aug. 2015.
- [10] Georgios Tournavitis and Björn Franke, “Semi-automatic extraction and exploitation of hierarchical pipeline parallelism using profiling information,” in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, New York, NY, USA, 2010, PACT ’10, pp. 377–388, ACM.
- [11] Xiaoming Li, Jack B. Dennis, Guang R. Gao, Willie Lim, Haitao Wei, Chao Yang, and Robert Pavel, “FreshBreeze: A Data Flow Approach for Meeting DDDAS Challenges,” *Procedia Computer Science*, vol. 51, no. Complete, pp. 2573–2582, 2015.
- [12] István Bozó, Viktoria Fordós, Zoltán Horvath, Melinda Tóth, Dániel Horpácsi, Tamás Kozsik, Judit Kőszegi, Adam Barwell, Christopher Brown, and Kevin Hammond, “Discovering parallel pattern candidates in erlang,” in *Proceedings of the Thirteenth ACM SIGPLAN Workshop on Erlang*, New York, NY, USA, 2014, Erlang ’14, pp. 13–23, ACM.
- [13] Timothy Mattson, Beverly Sanders, and Berna Massingill, *Patterns for Parallel Programming*, Addison-Wesley Professional, first edition, 2004.
- [14] David del Río Astorga, Manuel F. Dolz, Luis Miguel Sánchez, and J. Daniel García, “Discovering pipeline parallel patterns in sequential legacy C++ codes,” in *Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores*, PMAM@PPoPP, Barcelona, Spain, 2016, pp. 11–19.
- [15] C. Shuai, M. Boyer, M. Jiayuan, D. Tarjan, J. W. Sheaffer, L. Sang-Ha, and K. Skadron, “Rodinia: A benchmark suite for heterogeneous computing,” in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, Oct 2009, pp. 44–54.
- [16] J. A. Stratton, C. Rodrigues, I-J. Sung, N. Obeid, L. Chang, N. Anssari, G. D. Liu, and W. W. Hwu, “IMPACT Technical Report,” <http://impact.crc.illinois.edu/Shared/Docs/impact-12-01.parboil.pdf>, 2012.
- [17] J. Reinders, *Intel threading building blocks - outfitting C++ for multi-core processor parallelism*, O’Reilly, 2007.

Técnica de ocultación de malware

Javier Carrillo Mondéjar, José Luis Martínez Martínez

Resumen—Este artículo explica alguna de las técnicas utilizadas para la evasión de los antivirus y sistemas de detección de intrusos en red (NIDS), tanto de manera estática como dinámica mostrando los resultados obtenidos tras su aplicación sobre algunas muestras de malware.

Palabras clave—Crypter, RunPE, Antivirus, Seguridad, Malware

I. INTRODUCCIÓN

VIVIMOS en una sociedad donde la tecnología forma parte de nuestras vidas y se usa con más frecuencia, por lo que la seguridad es un aspecto importante a tener en cuenta. Las amenazas hacia nuestra privacidad o nuestros datos personales están en aumento y es interesante conocer los métodos y técnicas que los cibercriminales utilizan para poder defendernos.

Una de las herramientas o métodos más usados por los delincuentes para abarcar al mayor número de víctimas y obtener el mayor beneficio posible es el *malware*[1] software malicioso. El malware es uno de los grandes problemas a los que se enfrentan tanto las distintas organizaciones como los usuarios comunes de internet, ya que estos programas maliciosos pueden tener funcionalidades muy variadas que pueden ir desde la obtención de información de usuario, como por ejemplo contraseñas o nombres de usuario, a utilizar ese dispositivo infectado para atacar a organizaciones o gobiernos [2].

Aunque la existencia de este tipo de programas maliciosos puede remontarse al pasado varias décadas, la aparición de nuevos ejemplares aumenta cada día debido al crecimiento de las nuevas tecnologías. En el año 2008, “Symantec Enterprise Security” publicó un informe en el que sus resultados iniciales indicaban que, midiendo la prevalencia de aplicaciones del software en el mundo, el ratio de código malicioso y de programas no deseados excedía al software legítimo [3].

A consecuencia de las amenazas que existen a los sistemas e información personal, aparecen distintos softwares de seguridad que intentan mitigar o reducir el número de amenazas a nuestros sistemas. Entre estos softwares de seguridad se encuentran los Antivirus, cuyo objetivo principal es proteger los sistemas de los usuarios finales. Durante años los creadores de malware intentan descubrir y crear nuevas técnicas o métodos para evadir a los motores antivirus.

Existen un gran número de técnicas utilizadas para ocultar malware de los softwares antivirus y evitar así su detección. Una de estas técnicas se trata del uso de un “*crypter*” para cifrar el código malicioso y ser descifrado durante la ejecución del programa contenedor

y así pasar desapercibidos a los análisis estáticos de los antivirus [4]. El uso de estas técnicas de ofuscación tiene también como objetivo evitar la detección de malware por sistemas de detección de intrusos en red (NIDS) [5], pudiendo pasar inadvertido a estos sistemas de seguridad y conseguir que el archivo cifrado pueda ser transmitido por la red y no ser detectado.

El uso de esta técnica puede ser realmente eficaz evitando las detecciones basadas en firmas y la heurística de los motores antivirus, pero al ser ejecutado en un entorno real, donde los antivirus pueden analizar su comportamiento y ejecutar ese programa en sandboxes durante un corto periodo de tiempo, es posible que no sean tan eficaces como se pueda llegar a creer.

Por estas razones, se comprobará cuan efectivos son los softwares antivirus frente a técnicas de ocultación de malware como los *crypters* tanto a nivel de firmas y heurística como en un entorno de ejecución controlada en una máquina virtual con un sistema operativo “Microsoft Windows 7 Home”.

En los resultados se podrá observar que, aunque se puede conseguir evitar la mayoría de antivirus de manera estática, en la ejecución en un entorno real los *crypters* no son completamente indetectables.

A pesar de que muchos de los antivirus no analizan el espacio de memoria de usuario o no con la exhaustividad que deberían, si son capaces de detectar alguna de las muestras analizadas. En la mayoría de detecciones no detectan el malware cuando ha sido descifrado en memoria o su posterior ejecución desde la propia memoria, sino, por ejemplo, algún archivo que ha sido escrito en disco por el propio malware que se pretendía ocultar.

Este artículo se organiza de la siguiente manera: en la sección II se muestran las características y funciones principales de un producto antivirus y las distintas técnicas de detección comúnmente utilizadas y el funcionamiento y objetivos de un *crypter* [6]. En la sección III se describirán las distintas técnicas que se han empleado para evitar la detección por parte de los productos antivirus.

Por último, en la sección IV se describirá el entorno de laboratorio donde se han realizado las distintas pruebas y se mostrarán los resultados obtenidos en base a unos test, donde concluiremos en la sección V con una opinión personal a partir de dichos resultados.

II. CONCEPTOS TEÓRICOS

Los antivirus [7] son un software diseñado para prevenir la infección de un host detectando programas maliciosos que son conocidos por el nombre de *malware*. En última

¹ Instituto de Investigación en Informática. Universidad de Castilla-La Mancha, Campus Universitario, 02071, Albacete, SPAIN. e-mail: javier.carrillo@alu.uclm.es, jose.luis.martinez@uclm.es

instancia, cuando la infección ya se ha producido, intentan eliminarla y desinfectar el archivo malicioso del sistema operativo. Por tanto, se trata de una capa de seguridad que tiene como objetivo dar una protección extra a la ofrecida por los sistemas operativos.

Sus motores utilizan distintas técnicas [7] para detectar si un programa es considerado como malicioso o si por el contrario se trata de un programa “normal”. Para ello, sus motores se basan en:

- Firmas: comparar su firma digital única con la base de datos del programa antivirus.
- Heurística: buscar patrones que son utilizados habitualmente por programas no deseados.
- Comportamiento: detectar el comportamiento del malware una vez se ha ejecutado en el sistema.
- Sandbox: Se basa en la ejecución del programa en una máquina virtual para determinar si ejecuta instrucciones potencialmente peligrosas. La ejecución en máquina virtual se realiza durante un corto periodo de tiempo.

Si en alguna de estas fases el antivirus detecta algo inusual o que considera potencialmente peligroso, saltará una alarma y ese archivo pasará a estar en cuarentena hasta que el usuario decida qué hacer.

Por otro lado, el malware puede ser clasificado en distintas categorías [1] según su finalidad u objetivo. Estos pueden clasificarse como botnets [1], gusanos [1], virus [1], troyanos [1], ransomware [8], etc.

Cuando un malware ha sido identificado por las compañías de antivirus y ha sido añadido a su base de datos de firmas, ese archivo será detectado incluso antes de ser ejecutado. Aquí es donde los usuarios malintencionados utilizan técnicas para hacer a ese archivo indetectable a la mayoría de los motores antivirus más utilizados. Estas técnicas pueden dividirse en dos tipos:

- Estáticas: Intentan evitar las detecciones basadas en firmas [9] y la protección heurística de los antivirus [7].
- Dinámicas: Intentan evitar la detección en tiempo de ejecución, es decir, evitar la detección basada en su comportamiento o sandbox [7].

El uso de un *crypter* es una de las técnicas más utilizadas para evadir la detección de malware. Los *crypters* permiten cifrar un archivo para hacerlo invisible a los antivirus, puesto que pueden evadir las técnicas utilizadas por estos, como son las firmas o la heurística. Existen dos tipos de *crypters*:

- Scantime: Se llaman así a aquellos que una vez que el malware ha sido descifrado, lo escriben en disco y lanzan su ejecución. Estos *crypters* no se utilizan mucho puesto que el malware es detectado al ser escrito en disco por los antivirus. [10]
- Runtime: Se llaman así a aquellos que una vez descifran el malware, lo lanzan desde la memoria sin necesidad de escribirlo en disco. De esta forma

evita la protección heurística y la basada en firmas de los antivirus. [10]

Los *crypters* están compuestos de dos partes fundamentales:

- Builder: Es la parte del *crypter* que se encarga de construir el ejecutable final. Recibe como entrada el archivo que se quiere cifrar. La salida es un fichero ejecutable que contendrá el Stub y el archivo cifrado.
- Stub: Es la parte fundamental y está compuesta por el propio stub y el archivo cifrado. Básicamente, su funcionamiento consiste en la lectura del propio ejecutable y buscar donde comienza el archivo cifrado. Una vez que encuentra el comienzo del archivo que ha sido cifrado, se descifra y se escribe en disco para su posterior ejecución (*Scantime*) o se ejecuta desde la propia memoria (*Runtime*).

La Figura 1 muestra un diagrama de flujo del funcionamiento del crypter donde se ven identificadas cada una de las partes identificadas anteriormente.



Figura 1: Diagrama de flujo un crypter

Como ya hemos visto, la parte más importante del crypter es el stub, ya que es la que se enfrenta a los productos antivirus y en el momento que este sea detectado será inservible, puesto que cualquier archivo creado por el crypter será detectado como archivo malicioso.

III. ESTADO DEL ARTE

Bhaskar Mukherjee [11] describe la historia de internet y cómo las amenazas en este entorno digital han aumentado de manera exponencial con el crecimiento de este. Ha pasado mucho tiempo desde que el primer virus cifrado apareció en 1988 y aun así esta técnica de ofuscación sigue siendo muy utilizada.

Andreas Moser et al. [12] explora los límites del análisis estático para la detección de código malicioso. En su artículo presenta un esquema de ofuscación de archivos binarios mostrando como los detectores basados en patrones pueden ser evadidos, demostrando que las técnicas estáticas por sí solas no son suficientes para identificar el malware.

Wei Xu et al. [13] estudia la efectividad de los softwares antivirus frente a técnicas de ofuscación de código malicioso en JavaScript demostrando que los sistemas basados en firmas de estos pueden ser evadidos de manera efectiva con la ofuscación de código.

IV. TÉCNICA PROPUESTA

En esta sección se muestran las distintas técnicas que han sido empleadas para la implementación de un crypter. Estas técnicas tienen como objetivo final evitar la detección heurística y basada en firmas de los motores antivirus, así como la detección en la sandbox de los antivirus.

En la *crypter* se ha implementado un algoritmo de cifrado RC4 [14,15]. RC4 fue desarrollado por Ronald Rivest y es un algoritmo de cifrado en flujo que destaca por su sencillez y facilidad de implementación. Aunque RC4 es un algoritmo criptográfico muy inseguro y ha sido retirado de los estándares, es muy utilizado para la creación de *crypters*. El algoritmo está compuesto por dos partes:

- Extensión de la clave: A partir de la clave secreta se realiza una permutación de un arreglo de 256 bytes.
- Cifrado y descifrado: Para cada byte del mensaje se efectúa una XOR con una parte de la permutación.

La ejecución del archivo una vez ha sido descifrado desde memoria se ha realizado mediante una técnica conocida como RunPE [4,16,17] o *Process hollowing* [18]. Esta técnica consiste en ejecutar la imagen de un proceso dentro del espacio de memoria de otro proceso.

Por tanto, el stub actúa como un descifrador y un cargador de archivos con el formato *Portable Executable* (PE) [19] de Windows y se necesita conocer la estructura de un fichero PE [6,9,20,21]. En la Figura 2 se muestra un esquema de la estructura de los ficheros ejecutables de Windows.



Figura 2: Estructura de un fichero PE

Como vemos en la Figura 2, la estructura de un fichero PE se descompone en las siguientes estructuras de datos:

- DOS HEADER: Cabecera obsoleta incluida tanto en archivos de MSDOS como PE. Tiene dos campos importantes, *e_magic* que es la firma de MSDOS y *I_fanew* que es un puntero a NT HEADER.

- DOS STUB: Mensaje en sistemas MS-DOS.
- NT HEADER: Inicia en el valor apuntado por *I_fanew* y contiene la firma 'PE\0\0'.
- IMAGE FILE HEADER: Tiene la información general del archivo como la fecha de creación del archivo, el número de secciones, la arquitectura para la que ha sido creada el archivo, etc.
- IMAGE OPTIONAL HEADER: Tiene la información necesaria para que el cargador de Windows cargue correctamente el ejecutable en memoria. Por ejemplo, contiene la dirección del punto de entrada del ejecutable, la dirección base del ejecutable, el alineamiento de sección, tamaño de la imagen, etc.
- DATA DIRECTORY: Es una lista que tiene la dirección y tamaño de la tabla de exportación, de la tabla de importación, etc.
- SECTION HEADER: Existe una cabecera de sección para cada una de las secciones del ejecutable. La cabecera de sección contiene el nombre de la sección, la dirección de la sección, el tamaño, etc.
- SECTIONS: Contienen los datos y el código de un ejecutable.

Para la realización de la técnica RunPE se crea un nuevo proceso en estado suspendido usando la función *CreateProcess* [22]. Se desasigna la memoria de ese proceso mediante la función *NtUnmapViewOfSection* [23] y se obtiene el contexto. Se reserva en memoria la dirección de la base de la imagen del archivo ejecutable con el tamaño de la imagen y se escribe en memoria mediante la función *WriteProcessMemory* [24] la cabecera y las secciones con la alineación indicada por el campo alineación de sección. Se edita el registro EAX en el contexto con el nuevo punto de entrada y se edita EBX+8 por la nueva base de la imagen a ejecutar y se reanuda el proceso suspendido que ejecutara el malware en vez del ejecutable original. Se ha utilizado el propio stub para crear una instancia de sí mismo y correr el malware dentro de él.

Las llamadas a las funciones anteriormente mencionadas pueden ser detectadas por los antivirus, por lo que se ha usado una técnica sencilla para evitar su detección por la heurística de los antivirus. Esta técnica consiste en cargar dinámicamente las librerías necesarias y obtener la dirección de la función exportada por la librería mediante el uso de la función *GetProcAddress* [25] en tiempo de ejecución.

La última técnica utilizada para evitar que el ejecutable sea detectado en la sandbox, ha sido la ejecución de operaciones básicas en un bucle relativamente grande [16]. Esto consigue evitar la detección por la sandbox pero un ser humano no notará la diferencia en el inicio de la ejecución de un programa entre usar este sencillo código y no usarlo.

V. EVALUACIÓN DE PRESTACIONES

Para realizar las pruebas se han obtenido muestras reales de malware de "TheZoo" [26], que es un proyecto

educativo para hacer posible el análisis y estudio de malware y poder practicar los conocimientos adquiridos con malware real. Las muestras de malware pueden ser descargadas fácilmente del repositorio del proyecto creado en GitHub. El proyecto es mantenido por el usuario "ytisf".

A. Condiciones de simulación

Se han seleccionado cuatro muestras de malware con las que se probará el *crypter* y se evaluará el comportamiento de los productos antivirus tanto de manera estática como dinámica. Para la elección de las muestras de malware se ha buscado muestras que son conocidas y detectadas por los productos antivirus, de manera que se pueda comprobar si el uso de *crypters* es eficaz frente a estos productos de seguridad. Las muestras de malware que se han utilizado son las siguientes:

- Locky Ransomware: Se trata de un peligroso virus que cifra los archivos personales del disco duro y pide un rescate para recuperar esos archivos.
- Dexter: Se trata de un virus que puede robar información de las tarjetas de crédito de los sistemas.
- Lethic: Se trata de una botnet que en su momento fue el responsable del 10% del spam generado en todo el mundo.
- Ixeshe: Es una puerta trasera que permite el acceso y control de un ordenador.

Las pruebas en tiempo de ejecución han sido realizadas en VirtualBox [27], software de virtualización proporcionado por la compañía Oracle de manera gratuita. Se ha creado una máquina virtual para tal efecto con una arquitectura de 32 bits ejecutando el sistema operativo Windows 7 Home Service Pack 1.

Para el análisis estático de las muestras de malware se ha utilizado la aplicación web VirusTotal [28], que nos permite analizar archivos por la gran mayoría de motores antivirus existentes en el mercado de manera gratuita, aunque una de las condiciones que tiene es que cada archivo que se analiza será enviado a todas las compañías de antivirus que se utilizan en la aplicación web.

Para comprobar el comportamiento de los antivirus sobre un programa en ejecución se han utilizado las versiones de prueba gratuitas que ofrecen las compañías de antivirus con toda su funcionalidad. Para tal efecto se han seleccionado algunos de los más conocidos y utilizados motores de antivirus del mercado y se ha probado cada una de las muestras de malware cifradas con cada uno de los motores antivirus.

Para realizar esta tarea se creó una copia de seguridad de la máquina virtual antes de instalar ningún producto antivirus y de testear ningún malware. Una vez realizada la copia de seguridad se instaló un producto antivirus y se creó una imagen instantánea de la máquina virtual para poder volver atrás después de testear cada malware.

Una vez testeada la protección en tiempo real del antivirus con las distintas muestras de malware se recuperó la copia de seguridad para volver a testear otro software antivirus y así respectivamente.

Para la comparativa se ha utilizado como métrica el número de detecciones satisfactorias que han obtenido los distintos motores antivirus.

B. Resultados de la comparativa

Subimos las muestras de malware originales a VirusTotal para obtener cuantos antivirus detectan actualmente el malware de manera estática, es decir, basándose en las firmas de la base de datos de la del antivirus o en su protección heurística.

En la Tabla 1, podemos observar los resultados ofrecidos por VirusTotal, en donde podemos ver que, aunque el número de detecciones es bastante elevado obteniendo un porcentaje de acierto del 82,25 % de media, es relativamente bajo puesto que no se trata de malware creado recientemente, sino que llevan algún tiempo circulando por la red. Por ejemplo, las muestras de Lethic y Ixeshe fueron añadidas al repositorio de GitHub hace un año.

TABLA 1: DETECCIONES VIRUSTOTAL DE MUESTRAS ORIGINALES

Malware	Detecciones
Locky Ransomware	53/57
Dexter	51/57
Lethic	46/57
Ixeshe	39/57

Utilizamos el *crypter* sobre las muestras de malware originales y se suben los archivos cifrados a la aplicación web de VirusTotal.

En la Tabla 2 se muestra el número de detecciones que han obtenido las distintas muestras. Podemos observar que el número es muy bajo en comparación con las muestras sin cifrar analizadas en la Tabla 1, por lo que para evitar las detecciones basadas en firmas y la heurística de los antivirus, el uso de un *crypter* sigue siendo una técnica bastante eficaz. En los resultados proporcionados por VirusTotal, hay que destacar que "Microsoft Security Essentials" ha detectado las cuatro muestras cifradas.

TABLA 2: DETECCIONES EN VIRUSTOTAL MUESTRAS CIFRADAS

Malware	Detecciones
Locky Ransomware	2/56
Dexter	3/52
Lethic	2/56
Ixeshe	2/56

Pasamos a realizar las pruebas en ejecución. Para esta fase se ha obviado realizar las pruebas en ejecución para

las muestras de malware originales (sin cifrar), puesto que eran reconocidas de manera estática por los antivirus más utilizados del mercado y eran detectadas antes de ser ejecutadas. Se obviará también de las pruebas en tiempo real el producto de seguridad "Microsoft Security Essentials" ya que detecta los archivos como maliciosos de manera estática, por lo que no se puede evaluar su comportamiento frente a malware en tiempo de ejecución.

En la Tabla 3 se muestran los resultados obtenidos por los productos antivirus frente a las muestras de malware cifradas. Se observa que, aunque de manera estática eran indetectables a todos ellos, en tiempo de ejecución se han detectado algunas de las muestras de malware, aunque es importante destacar que sólo ha conseguido detectar todos los programas maliciosos el antivirus "Norton Security Deluxe".

TABLA 3: DETECCIONES ANTIVIRUS EN TIEMPO DE EJECUCIÓN

Antivirus/Malware	Locky Ransomware	Dexter	Lethic	Ixeshe
Avast Premier	No	Si	No	No
ESET Smart Security	No	Si	Si	No
Panda Antivirus Pro 2016	Si	No	No	No
Karperksy Internet Security	Si	Si	Si	No
McAfee Internet Security	No	Si	No	No
Malwarebytes Anti-Malware	Si	No	No	No
Norton Security Deluxe	Si	Si	Si	Si
Avira Free Antivirus	No	Si	Si	No

También se puede observar que el malware Ixeshe sólo es detectado por un producto antivirus y el malware Dexter es el más detectado de todos ellos. En este último caso, la mayoría de productos de seguridad testeados no detectan su presencia en memoria sino un archivo con extensión ".dll" cuando es escrita en disco.

VI. CONCLUSIONES

Viendo los resultados obtenidos podemos ver que el uso de un crypter es una técnica bastante eficaz para saltarse las detecciones basadas en firmas y la heurística de los antivirus.

En cuanto a evitar las detecciones en tiempo de ejecución, no se evitan las detecciones en muchos casos, pero en la mayoría de los productos antivirus no están detectando el malware en memoria o su posterior ejecución mediante la técnica conocida de RunPE, sino algunas de las tareas que realiza el propio malware como su escritura en disco para guardar persistencia o similares.

Por tanto, aunque en tiempo de ejecución no son completamente eficaces si que pueden evadir en algún caso a los productos antivirus, por lo que los usuarios no deben confiar completamente su seguridad en los

productos antivirus, sino que sólo son una capa más de seguridad.

Por tanto, aunque todavía tienen que mejorar la mayoría de ellos analizando la memoria o interceptando llamadas a funciones del sistema que son potencialmente peligrosas y usadas por el malware, tienen una capacidad de respuesta y una rápida actualización frente a nuevas muestras de malware que aparecen cada día.

AGRADECIMIENTOS

Este trabajo ha sido cofinanciado por el Ministerio de Economía y Competitividad y la Comisión Europea bajo el proyecto TIN2015-66972-C5-2-R (MINECO/FEDER).

REFERENCIAS

- [1] M. Sikorki and A. Honig. Practical Malware Analysis. No Starch Press, Inc. 2012.
- [2] ComputerHoy. ¿Qué es un ataque DDos? Así tumbaron los hackers PSN y Xbox. <http://computerhoy.com/noticias/software/que-es-ataque-ddos-asi-tumbaron-hackers-psn-xbox-17557>
- [3] Symantec Enterprise Security, 2008. Symantec Internet Security Threat Report.
- [4] A. Pasamar. Desmitificando el Antivirus. 2014
- [5] R. Carrasco, S. Gumiel y A. Vizcaino. Sistema de ofuscación de malware para la evasión de NIDS. Universidad Complutense de Madrid. 2013
- [6] C. Ammann. Hyperion: Implementation of a PE-Crypter. NULLSECURITY, 2012
- [7] J. Koret and E. Bachaalany. The Antivirus Hacker's Handbook. Wiley, 2015.
- [8] ¿Qué es un Ransomware? Panda Security. 2016. <http://www.pandasecurity.com/spain/mediacenter/consejos/que-es-un-ransomware>
- [9] InterNot Security Team. Bypassing Anti-Virus Scanners. 2012.
- [10] T. Vasilios. Bypassing Antivirus Detection with Encryption. University of Piraeus. 2014
- [11] B. Mukherjee. Threats to Digitization: Computer Virus. Published in: International CALIBER, Febrero 2008.
- [12] A. Moser, C. Kruegel and E. Kirda. Limits of Static Analysis for Malware Detection. Published in: Computer Security Applications Conference, 2007.ACSAC 2007. Twenty-Third Annual.
- [13] W. Xu, F. Zhang and S. Zhu. The power of Obfuscation Techniques in Malicious JavaScript Code: A Measurement Study. Published in: MALWARE '12 Proceedings of the 2012 7th International Conference on Malicious and Unwanted Software.
- [14] A. Jiménez. Criptografía de Clave Secreta. RC4. 2016
- [15] S. Bravo. Estudio comparativo de los algoritmos de cifrado de flujo RC4, A5 y SEAL. 2002
- [16] E. Nasi. Bypass Antivirus Dynamic Analysis. Limitations of the AV model and how to exploit them. 2014
- [17] E. Nasi. PE Injection Explained. Advanced memory code injection technique. 2014
- [18] M. Hale, S. Adair, B. Hartstein and M. Richard. Malware Analyst's Cookbook and DVD. Wiley Publishing, Inc. 2011.
- [19] Microsoft. Visual Studio, Microsoft Portable Executable and Common Object File Format Specification. Microsoft Corporation.
- [20] The Swash. Formato Portable Executable bajo Windows. 2011
- [21] G. Erdélyi. Reverse Engineering III: PE Format.
- [22] CreateProcess Function. 2016. <https://msdn.microsoft.com/es-es/>
- [23] NTAPI Undocumented Functions. Tomasz Nowak. 2016. <http://undocumented.ninternals.net/index.html>
- [24] WriteProcessMemory Function. Microsoft Corporation. 2016. <https://msdn.microsoft.com/es-es/>
- [25] GetProcAddress function. 2016. <https://msdn.microsoft.com/es-es/>
- [26] Ytsf. theZoo aKa Malware DB. 2016. <http://ytsf.github.io/theZoo/>
- [27] Oracle VM VirtualBox. 2016. <http://www.virtualbox.org>

- [28] VirusTotal – Free Online Virus, Malware and URL Scanner.
2016. <https://virustotal.com/>

Evaluación de prestaciones

Reduciendo el Tiempo de Ejecución de una Aplicación de Cálculo de Riesgos Financieros a través del uso de GPUs Virtuales

Javier Prades¹, Blesson Varghese², Carlos Reaño¹ y Federico Silla¹

Resumen— Las unidades de procesamiento gráfico (GPUs) se están convirtiendo en aceleradores de cómputo muy populares en los clusters de altas prestaciones (HPC). Sin embargo, la instalación de GPUs en cada uno de los nodos del clúster no está exenta de problemas, por un lado tenemos el alto coste de adquisición, mantenimiento y consumo energético de estos dispositivos y por otro lado, la baja utilización que generalmente se alcanza de estos dispositivos. Estos factores repercuten en una lenta amortización de la inversión inicial. En este trabajo evaluamos las mejoras, en prestaciones y consumo energético, experimentadas por una aplicación de cálculo de riesgos financieros al proveerle acceso remoto bajo demanda a tan solo unas pocas GPUs físicas virtualizadas. Nuestra hipótesis está basada en que al compartir una GPU física entre varios hilos de ejecución de la aplicación, a través de la virtualización remota de GPUs, se reduce tanto el tiempo de ejecución como la energía consumida por la aplicación, además de incrementar el uso de estas GPUs. Un aspecto clave en esta investigación son las transferencias de datos entre CPU y GPU, por lo que hemos analizado dos modos, transferencias concurrentes y transferencias secuenciales. Los resultados experimentales muestran que la compartición de unas pocas GPUs físicas usando transferencias de datos secuenciales disminuye el tiempo de ejecución y la energía consumida, mejorando así el rendimiento global de la aplicación.

Palabras clave— Virtualización de GPUs, CUDA, rCUDA, HPC

I. INTRODUCCIÓN

LOS aceleradores hardware están alcanzando un papel destacado en los modernos clusters de computación de altas prestaciones (HPC) ya que permiten ejecutar más rápidamente las aplicaciones. Una evidencia de esto es que cuatro de los diez mejores supercomputadores que aparecen en la lista Top500 (<http://top500.org>) y los diez primeros que aparecen en la lista Green500 (<http://www.green500.org>) (en noviembre de 2015) usan aceleradores hardware, tales como unidades de procesamiento gráfico (GPU).

La incorporación de GPUs en los clusters permite heterogeneidad, por lo que es posible que una aplicación realice el cómputo en un procesador regular, así como en una GPU [1], [2]. Estos clusters pueden configurarse de tal forma que cada uno de los nodos incorpore una o múltiples GPUs. Así, la aplicación que se está ejecutando en estos nodos simplemente puede hacer uso de estos aceleradores, reduciendo

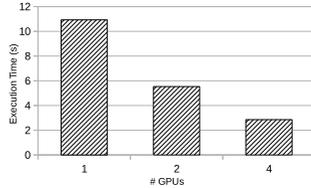


Fig. 1. Evolución del tiempo de ejecución de la aplicación de cálculo de riesgos financieros en función del número de GPUs utilizadas.

significativamente el tiempo de ejecución de las aplicaciones programadas para ejecutarse en GPUs (véase la Figura 1). Sin embargo, esta configuración no es eficiente, y tal vez no sea factible en un presupuesto limitado, debido a (i) costo relativamente alto de las GPUs, (ii) un mayor consumo energético debido a las GPUs [3] y (iii) infrautilización de las GPUs, ya que se encuentran en todos los nodos del clúster y es prácticamente imposible que las aplicaciones hagan uso de ellas todo el tiempo.

Una alternativa a esta configuración son los clusters en los que únicamente existe un pequeño número de GPUs y donde se proporciona a los nodos acceso bajo demanda a estas GPUs remotas [4], [5]. El nodo donde se ejecuta la aplicación que requiere aceleración GPU puede ser tratado como un cliente que solicita servicios de aceleración [6] de un servidor de GPUs o un conjunto de servidores de GPUs dentro del cluster. En consecuencia, no sólo se reduce el número de GPUs utilizadas para proporcionar aceleración, sino también la energía total consumida respecto a la configuración anterior, así como también se experimenta un aumento en la utilización de las GPUs del cluster [7].

Las contribuciones principales de esta investigación son: (i) análisis de la escalabilidad de una aplicación de cálculo de riesgos financieros, (ii) propuesta de dos enfoques para copiar datos entre la memoria principal y la memoria de la GPU, copias concurrentes y secuenciales, (iii) estudio y evaluación del uso de GPUs virtuales, (iv) desarrollo de técnicas que potencian el solapamiento de cómputo y transferencia de datos sobre una misma GPU física, (v) la evaluación del rendimiento de la aplicación usando GPUs virtuales, teniendo en cuenta el tiempo de ejecución, la utilización de GPU y la energía y potencia consumidas.

¹Departament d'Informàtica de Sistemes y Computadors, Universitat Politècnica de València, emails: {jprades, carregon}@gop.upv.es, fsilla@disca.upv.es

²School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, email: varghese@qub.ac.uk

El resto del trabajo se organiza del siguiente modo: en la Sección II son mostrados los trabajos más relevantes sobre virtualización de GPUs dentro del HPC relacionados con la industria de riesgos financieros. La Sección III resume las principales características del framework de virtualización remota de GPUs usado en este trabajo, rCUDA. Posteriormente, en la Sección IV las principales características de la aplicación de evaluación de riesgos financieros utilizada son introducidas. Finalmente, en las Secciones V y VI se realiza una exhaustiva evaluación de la aplicación y se muestran las principales conclusiones obtenidas a través de la realización de este trabajo.

II. TRABAJO RELACIONADO

Las soluciones de computación de altas prestaciones HPC son ampliamente usadas en la industria del riesgo financiero para acelerar los cálculos subyacentes de las aplicaciones usadas. Este tipo de soluciones van desde clusters de pequeña escala [8], [9] hasta grandes supercomputadores [10], [11]. Más recientemente, se están empezando a usar aceleradores hardware con procesadores multi-core y many-core. Por ejemplo, las aplicaciones de cálculo de riesgos financieros son aceleradas en procesadores Cell BE [12], [13], FPGAs [14], [15] y GPUs [16], [17].

Por tanto, muchos de los clusters HPC actuales ofrecen soluciones heterogéneas mediante el uso de aceleradores hardware, tales como las GPUs, junto con los procesadores de los nodos [1], [2]. El problema de estos clusters reside en el gran coste económico que suponen, tal como hemos visto en la Sección I. Sin embargo, actualmente existen soluciones más eficientes económicamente que nos permiten acceder a las GPUs solamente cuando es necesario. Este tipo de soluciones se basan en la virtualización de GPUs. Dado que actualmente no existen soluciones en la industria de cálculo de riesgos financieros que permitan aprovechar el potencial de la virtualización de las GPUs, en este trabajo, investigamos el uso de GPUs virtuales sobre una aplicación de estas características.

Los frameworks de virtualización de GPUs permiten que los nodos de un cluster que no poseen una GPU física puedan acelerar el cómputo de las aplicaciones accediendo a GPUs remotas. La aceleración se obtiene accediendo a las GPUs remotas de forma totalmente transparente, sin que los nodos o las aplicaciones ejecutadas en dichos nodos sean conscientes en ningún momento de que están accediendo a una GPU remota. Una aplicación (que se ejecuta en una máquina virtual (VM) o en un nodo de un cluster sin un acelerador hardware) se beneficia del uso de una GPU remota y así se reduce el tiempo de ejecución. La tasa de utilización de la GPU puede aumentar ya que múltiples aplicaciones o múltiples máquinas virtuales pueden acceder a una misma GPU remota.

En la actualidad existen diferentes frameworks de virtualización remota de GPUs. GridCuda [18] compatible con CUDA 3.2 (desgraciadamente no está disponible públicamente). vCUDA [19] soporta CUDA

3.2 e implementa un subconjunto del API de CUDA. GVim [20] está basado en CUDA 1.1 y no implementa toda la API del runtime de CUDA. gVirtus [21] es compatible con CUDA 2.3 pero tan solo soporta de nuevo una pequeña parte de la API. DS-CUDA [22] es compatible con CUDA 4.1 e incluye soporte de comunicación específico para InfiniBand Verbs (IBV). Sin embargo, DS-CUDA está limitado, ya que no permite transferencias de datos con memoria pinned, además únicamente permite transferencias de datos hasta un tamaño máximo de 32 MB.

En este trabajo hemos utilizado el framework de virtualización remota de GPUs rCUDA [23] ya que entre otras cosas es compatible binario con las versiones más recientes de CUDA.

III. rCUDA: CUDA REMOTO

Tal como hemos mencionado, en este trabajo usamos el framework rCUDA dado que soporta las versiones de CUDA más actuales. En esta sección se muestra rCUDA con más detalle.

rCUDA soporta la versión 7.0 de CUDA, siendo compatible a nivel binario con él, lo que significa que los programas CUDA no necesitan ser modificados para ser utilizados con rCUDA. Además, rCUDA implementa todo el API CUDA (a excepción de las funciones gráficas) y también proporciona soporte a las librerías incluidas en CUDA, como cuFFT, cuBLAS o cuSPARSE. Por otra parte, el middleware rCUDA permite que un solo servidor rCUDA pueda atender simultáneamente a varios clientes remotos que soliciten servicios de GPU. Esto se consigue mediante la creación de contextos de GPU independientes, cada uno de ellos asignado a un cliente diferente [23].

rCUDA proporciona adicionalmente apoyo específico a diferentes sistemas de interconexión [23]. Esto se consigue haciendo uso de una serie de módulos de comunicación específicos para cada tipo de red. Estos módulos son cargados en tiempo de ejecución, y han sido diseñados con el fin de obtener el mayor rendimiento posible de la red de interconexión subyacente. rCUDA dispone actualmente de dos módulos: uno destinado a redes compatibles TCP/IP y otro diseñado específicamente para InfiniBand.

En cuanto al módulo de comunicaciones de InfiniBand, está basado en el API InfiniBand Verbs. IBV ofrece dos mecanismos de comunicación: la semántica de canal y la semántica de memoria. El primero se refiere a las operaciones de envío/recepción estándar que típicamente están disponibles en cualquier librería de comunicaciones, mientras que el segundo ofrece operaciones RDMA donde son especificados los punteros de memoria origen y destino de una transferencia de datos, lo que resulta en una transferencia sin copias intermedias y una mínima participación de las CPUs. rCUDA utiliza ambos mecanismos de comunicación, seleccionando uno u otro en función de la tarea exacta que deba ser llevada a cabo [23].

Por otra parte, independientemente de la red utilizada, el intercambio de datos entre clientes y servidores rCUDA es realizado mediante segmentación

aprovechando al máximo el ancho de banda, tal como puede verse también en [23].

IV. APLICACIÓN DE ANÁLISIS DE RIESGOS FINANCIEROS

En esta sección se presenta una aplicación empleada en la industria de estudio de riesgos financieros conocida como *'Aggregate Risk Analysis'* [24]. Esta aplicación es usada para calcular el coste de seguros en función de una serie de parámetros. Generalmente es usada en clusters HPC y puede beneficiarse del uso de GPUs como aceleradoras de cómputo, en nuestro trabajo la utilizaremos para mostrar la viabilidad del uso de rCUDA en este tipo de aplicaciones.

El análisis de riesgos se realiza por medio de simulaciones muy costosas computacionalmente. El objetivo de estas simulaciones es proporcionar información y ayuda a la toma de decisiones, con millones de puntos de vista alternativos, sobre la posibilidad de que ocurran, por ejemplo, eventos catastróficos, como terremotos o huracanes, así como la magnitud y el grado de ocurrencia anual de los mismos en una determinada región geográfica. Por tanto, para obtener millones de puntos de vista alternativos, millones de pruebas son simuladas y con cada prueba se calcula un conjunto de posibles eventos catastróficos futuros y las pérdidas económicas generadas por los mismos.

Por lo general, este análisis se realiza periódicamente de forma rutinaria en los clusters de producción para derivar importantes medidas de riesgo. Tal puesta en marcha es suficiente cuando el análisis no tiene que ser realizado bajo demanda sino únicamente de forma periódica. Sin embargo, las métricas de riesgo necesitarán ser obtenidas en tiempo real, por ejemplo en un escenario de consulta de precios en línea. En tales configuraciones tendrán que ser recalculadas para satisfacer los parámetros de entrada dados por los distintos clientes. Esto genera un gran número de solicitudes que requerirán la ejecución de los análisis varias veces en función de la complejidad de las peticiones de cada cliente. Por lo tanto, estas peticiones serán imposibles de gestionar en el cluster anteriormente comentado ya que únicamente está preparado para la ejecución rutinaria del cálculo de riesgos. Aquí, las GPUs puede desempeñar un papel importante a la hora de resolver un gran número de solicitudes.

Sin embargo, aunque el uso de GPUs puede ser una solución factible, el empleo de un gran número de GPUs para responder a ráfagas de solicitudes será caro. Tal como se mostrará en la Sección V el uso de GPUs virtuales puede paliar este problema y ayudar a que la infrautilización de las mismas no sea tan elevada. En este contexto, usaremos GPUs virtuales dentro de un clúster HPC para conseguir una respuesta más rápida de las aplicaciones. Esto nos permitirá su uso en entornos de tiempo real. El framework rCUDA se adapta a dichas necesidades porque los cambios necesarios en el cluster son mínimos y la aceleración requerida para el análisis se obtiene como un servicio desde un host remoto. El

análisis de riesgos ha sido previamente testado en el contexto de arquitecturas many-core [25], pero creemos que las GPUs virtuales pueden ser una mejor opción.

V. EVALUACIÓN EXPERIMENTAL

En esta sección realizamos una exhaustiva evaluación experimental de la aplicación de riesgos financieros. Primero hemos realizado un análisis de la escalabilidad de la aplicación usando CUDA en un nodo del cluster equipado con 4 GPUs y usando rCUDA accediendo hasta un total de 16 GPUs sobre las redes de interconexión InfiniBand QDR y FDR. Finalmente, hemos resuelto, en parte, los problemas de escalabilidad de la aplicación, probando así la viabilidad del uso de GPUs virtuales en este tipo de aplicaciones y sobre un entorno de respuesta en tiempo real.

A. Banco de pruebas

El banco de pruebas utilizado en este trabajo está formado por nodos 1027GR-TRF Supermicro. Cada uno de estos nodos incluye dos procesadores Intel Xeon E5-2620 v2 con seis núcleos y arquitectura Ivy Bridge, que funcionan a 2,1 GHz y disponen de 32 GB de memoria DDR3 SDRAM a 1600 MHz. Cada nodo está equipado y configurado tanto para funcionar sobre una red de interconexión InfiniBand QDR como sobre una red InfiniBand FDR:

- Red de interconexión QDR, adaptador de red Mellanox ConnectX-2 VPI single-port conectado a switch MTS3600 (tasa máxima de transferencia de datos de 40 Gb/s).
- Red de interconexión FDR, adaptador de red Mellanox ConnectX-3 VPI single-port conectado a switch SX6025 (tasa máxima de transferencia de datos de 56 Gb/s).

Además, una GPU NVIDIA Tesla K20 está disponible en cada uno de los nodos.

Con el fin de evaluar las prestaciones de la aplicación usando GPUs locales con CUDA, uno de los servidores ha sido equipado con 4 GPUs NVIDIA Tesla K20 y 128 GB de memoria DDR3 SDRAM a 1600 MHz.

En cuanto al software, hemos utilizado Centos 6.4 como sistema operativo y la versión 2.4-1.0.4 del OFED de Mellanox (drivers de InfiniBand y herramientas de administración), junto con la versión 6.5 de CUDA y la versión 15.07 de rCUDA.

B. Escalabilidad de la aplicación usando CUDA

Tal como se vió en la Sección I el uso de múltiples GPUs reduce significativamente el tiempo de ejecución de la aplicación, ya que permite distribuir la carga computacional entre las diferentes GPUs disponibles. Sin embargo, un análisis más profundo muestra que los resultados vistos en la Figura 1 pone de manifiesto que la escalabilidad de la aplicación no es perfecta. Tal como podemos apreciar en la Figura 2 el tiempo dedicado a computación escala perfectamente a medida que aumentamos el número de GPUs.

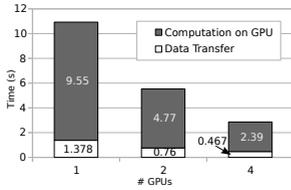


Fig. 2. Evolución del tiempo de ejecución de la aplicación de cálculo de riesgos financieros en función del número de GPUs utilizadas. El tiempo de ejecución mostrado está dividido en tiempo usado en copias de datos entre la memoria central y la memoria de la GPU y el tiempo de cómputo en la GPU.

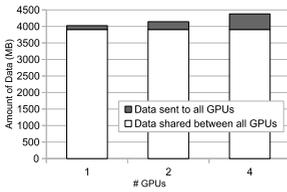


Fig. 3. Tamaño total de los datos copiados por la aplicación de riesgos financieros según varía el número de GPUs utilizadas.

Por el contrario, el tiempo invertido en las copias de datos entre la memoria del host y la memoria de la GPU no escala linealmente con el número de GPUs utilizadas. Esto se debe principalmente a dos factores: (i) la cantidad de datos copiados aumenta con el número de GPUs utilizadas, esto es debido a que en la aplicación estudiada, descrita en la Sección IV, existen estructuras de datos que no pueden ser divididas entre el número de GPUs, sino que tienen que ser copiadas íntegramente a la memoria de cada GPU utilizada. La Figura 3 muestra este aumento en el tamaño de los datos copiados según se incrementa el número de GPUs; (ii) la limitación impuesta por el canal de menor ancho de banda existente en el camino entre la memoria del host y la memoria de la GPU. Las copias de datos entre la memoria del host y la memoria de la GPU son realizadas de forma concurrente a todas las GPUs al inicio de la ejecución de la aplicación, por tanto, el ancho de banda en esta comunicación es clave para la escalabilidad de la aplicación. La Figura 4 muestra el ancho de banda medio alcanzado al realizar diferente número de transferencias simultáneas entre la memoria del host y la memoria de la GPU usando CUDA. Tal como puede observarse a medida que aumentamos el número de transferencias simultáneas (a medida que aumentamos el número de GPUs) el ancho de banda para cada una de las transferencias individuales va decreciendo, es decir el ancho de banda efectivo (suma del ancho de banda de todas las transferencias) no crece linealmente con el número de GPUs.

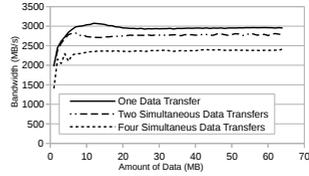


Fig. 4. Ancho de banda medio alcanzado en copias desde la memoria del host a la memoria de la GPU usando CUDA. Se muestran los anchos de banda medios para cada una de las transferencias individuales en los siguientes casos: una única transferencia, dos transferencias simultáneas y cuatro transferencias simultáneas.

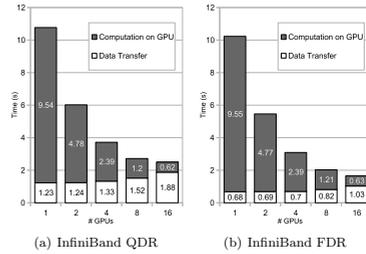


Fig. 5. Análisis de la escalabilidad de la aplicación de cálculo de riesgos financieros al usar rCUDA.

C. Escalabilidad de la aplicación usando rCUDA

Al usar rCUDA podemos aprovecharnos del uso de más GPUs, ya que a diferencia de CUDA, con rCUDA no tenemos la limitación de usar exclusivamente las GPUs locales de un nodo. Sin embargo, al igual que al usar CUDA la escalabilidad de la aplicación de análisis de riesgos financieros tampoco es perfecta. Tal como podemos apreciar en la Figura 5 tanto al usar rCUDA sobre InfiniBand QDR como sobre InfiniBand FDR, el tiempo dedicado a cómputo escala perfectamente con el número de GPUs pero el tiempo que la aplicación dedica a copias de datos no escala de forma lineal con el número de GPUs usadas¹. Es decir, la aplicación se comporta exactamente igual que al usar CUDA, sin embargo, en este caso la escalabilidad de la parte dedicada a transferencias de datos empeora respecto a lo visto usando CUDA. Este empeoramiento es debido principalmente al segundo de los factores que limita la escalabilidad de las transferencias de datos de la aplicación (la limitación impuesta por el canal de mínimo ancho de banda en el camino entre la memoria del host y la memoria de la GPU). En este caso, el canal más restrictivo, en términos de ancho de banda, en este

¹ Al usar rCUDA, ya sea sobre InfiniBand QDR o FDR, se observa una reducción en el tiempo de transferencia de datos (para el caso de 1 GPU) con respecto a los resultados obtenidos al usar CUDA, esto que a priori parece imposible (con rCUDA se accede a GPUs remotas) es un resultado bien conocido y explicado en [23].

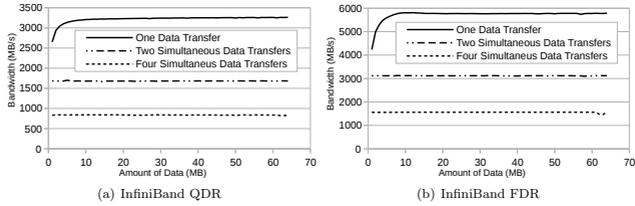


Fig. 6. Ancho de banda medio alcanzado en copias desde la memoria del host a la memoria de la GPU usando rCUDA sobre QDR y FDR. Se muestran los anchos de banda medios para cada una de las transferencias individuales en los siguientes casos: una única transferencia, dos transferencias simultáneas y cuatro transferencias simultáneas.

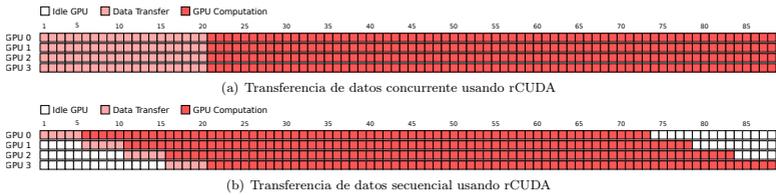


Fig. 7. Ciclo de vida de la ejecución de la aplicación de estimación de riesgos financieros usando el framework rCUDA utilizando la red de interconexión InfiniBand FDR y 4 GPUs remotas.

camino es la red de interconexión, ésta ofrece un ancho de banda máximo de 5000 MB/s en el caso de QDR y 7000 MB/s en el caso de FDR.

En la Figura 6 puede apreciarse cómo a medida que se incrementa el número de transferencias simultáneas el ancho de banda medio para cada una de ellas desciende drásticamente. Esto es debido a que todas las transferencias, aunque sean realizadas a distintas GPUs, tienen que compartir el mismo canal, la red de interconexión InfiniBand.

D. Mejora de la escalabilidad de la aplicación

Cuando ejecutamos la aplicación usando CUDA poco podemos hacer para mejorar la escalabilidad. Recordemos que los dos factores que limitan la escalabilidad son: el aumento de los datos transmitidos al aumentar el número de GPUs y la limitación que impone el canal de mínimo ancho de banda existente en el camino entre la memoria central y la memoria de la GPU. El primero de estos factores viene dado por los requisitos de implementación de la aplicación y el segundo dependerá del hardware existente en el nodo donde están instaladas las GPUs y su configuración.

Al usar rCUDA sí que tenemos un poco más de libertad para intentar paliar el problema de la escalabilidad, ya que usamos GPUs remotas virtuales y esto permite entre otras cosas instanciar una o más GPUs virtuales sobre una misma GPU física. Esta técnica, a la que podemos referirnos como *tenencia múltiple* nos permitirá poder solapar cómputo y transferencia de datos.

D.1 rCUDA: envío concurrente vs envío secuencial

La Figura 7(a) muestra el ciclo de vida de la ejecución de la aplicación utilizando rCUDA con cuatro GPUs remotas e InfiniBand FDR. Cada celda temporal representada en la figura es equivalente a 35 milisegundos de tiempo de ejecución. Este diagrama corresponde a la ejecución de la aplicación de cálculo de riesgos financieros usando rCUDA con un total de cuatro GPUs remotas mostrada en la Figura 5(b). La misma cantidad de datos es enviada a las cuatro GPUs de forma simultánea al iniciarse la aplicación. Las diferentes transferencias de datos van intercalándose en la red y las GPU remotas reciben todos los datos y comienzan los cálculos aproximadamente el mismo instante de tiempo. Sin embargo, a partir de lo visto en la Figura 6(b) se deduce que el ancho de banda alcanzado por cada una de las transferencias de datos es inversamente proporcional a la cantidad de transferencias simultáneas lo que se traduce en una degradación del rendimiento. Un método alternativo se muestra en la Figura 7(b). Los datos enviados a cada una de las GPUs son transmitidos sin compartir el canal con el resto de transferencias. Puesto que no hay competencia por el ancho de banda sólo se necesita una cuarta parte del tiempo en realizar la transmisión respecto al tiempo que se requiere cuando todos los datos se transfieren concurrentemente (mostrado en la Figura 7(a)). Por lo tanto, el cómputo en la primera GPU empieza antes, mientras el resto de datos son transmitidos a la segunda y restantes GPUs. De esta forma, la copia de datos se realiza utilizando todo el ancho de banda de la red para cada una de las transferencias individua-

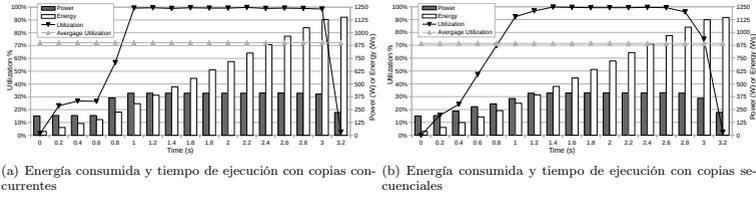


Fig. 8. Utilización de GPU, potencia, consumo de energía y tiempo de ejecución para transferencias concurrentes y secuenciales con rCUDA sobre InfiniBand FDR usando 4 GPUs remotas.

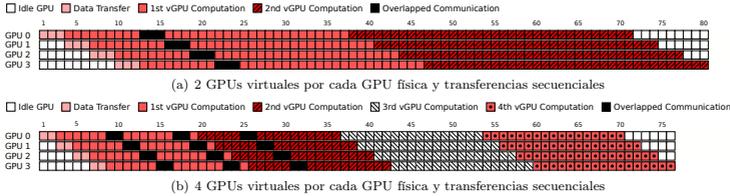


Fig. 9. Copias secuenciales con varias GPUs virtuales por cada GPU física.

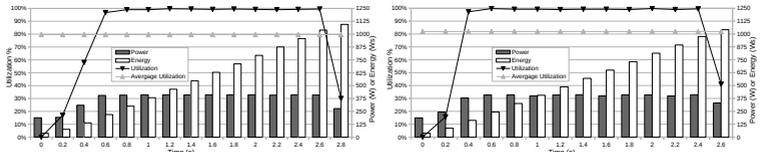
les. A partir de ahora, nos referiremos a este método de transferencia de datos como secuencial. Los datos se transfieren más rápidamente a cada una de las GPUs y existe solapamiento entre el cómputo en GPU y las transferencias de datos. Sin embargo, no se observa reducción en el tiempo de ejecución ya que la copia de datos a la cuarta GPU termina exactamente en el mismo instante que las copias de datos simultáneas realizadas en el enfoque concurrente. Las Figuras 8(a) y 8(b) muestran la utilización media de GPU, la potencia y el consumo energético de todas las GPUs usando los enfoques concurrente y secuencial para las transferencias de datos. Tal como puede apreciarse, no existe prácticamente ninguna diferencia en todos estos parámetros al usar un enfoque u otro.

Aunque en este punto el lector pueda pensar que ambos enfoques, concurrente y secuencial, son equivalentes, tal como veremos a continuación el enfoque secuencial es fundamental para la explotación de la técnica de tenencia múltiple de GPUs virtuales sobre una misma GPU física.

D.2 Tenencia múltiple de GPUs virtuales usando rCUDA

El concepto clave que motiva la instanciación de varias GPUs virtuales sobre una misma GPU física se basa en el hecho de que las GPUs actuales son capaces de ejecutar kernels y operaciones DMA (Direct Memory Access) de forma concurrente. Siendo posible, por tanto, el mover datos a una GPU y al mismo tiempo ejecutar cómputo mejorando de esta forma las prestaciones de la aplicación que se está ejecutando. Esto permite que las transferencias de datos y el cómputo de diferentes GPUs virtuales se solapen dentro de una misma GPU física. La Figu-

ra 9 muestra este concepto de tenencia múltiple de GPUs virtuales sobre una misma GPU física. Cuando 2 GPUs virtuales son instanciadas sobre cada GPU física, tal como muestra la Figura 9(a), la aplicación tiene disponibles 8 GPUs virtuales pero tan solo 4 GPUs físicas. Los datos de entrada son repartidos entre las 8 GPUs para después realizar el cómputo. A la transferencia de datos inicial nombrada como *Data Transfer* le sigue la computación en la primera GPU virtual, etiquetada como *1st vGPU Computation*. Después de transmitir los datos a las 4 primeras GPUs virtuales (celda 12), quedarán 4 GPUs virtuales a las que transferir datos. Todas estas copias de datos restantes se solaparán con el cómputo que realizan las primeras GPUs virtuales reduciendo de este modo el tiempo de ejecución de la aplicación. Una vez todos los datos han sido copiados a las diferentes GPUs virtuales, el cómputo de dos GPUs virtuales instanciadas sobre una misma GPU física no puede ejecutarse en paralelo porque cada GPU virtual posee su propio contexto dentro de la GPU física. Por lo tanto el driver de NVIDIA ejecutará estos cálculos de forma secuencial (usando los recursos requeridos por cada kernel). Así, la ejecución del cómputo de la segunda GPU virtual deberá esperar a que termine el cómputo de la primera. Podemos extraer dos conclusiones del mapeado de múltiples GPUs virtuales sobre una misma GPU física. Primera, el tiempo de ejecución se reduce, tal como muestra el ciclo de vida presentado en la Figura 9(a), aún usando los mismos recursos hardware. La aplicación con dos GPUs virtuales por GPU física acaba en la ranura temporal 80 mientras que con una instanciación 1 a 1 termina en la 88. El tiempo destinado a cómputo por cada GPU física es exactamente el mismo en ambos casos, por lo que el tiempo ahorrado



(a) Energía consumida y tiempo de ejecución del ciclo de vida de la aplicación usando 2 GPUs virtuales por cada GPU física (b) Energía consumida y tiempo de ejecución del ciclo de vida de la aplicación usando 4 GPUs virtuales por cada GPU física

Fig. 10. Utilización de GPU, potencia, consumo de energía y tiempo de ejecución para transferencias secuenciales con rCUDA sobre InfiniBand FDR variando el número de GPUs virtuales por GPU física. (a) 8 GPUs virtuales residentes en un total de 4 GPUs físicas, (b) 16 GPUs virtuales residentes en un total de 4 GPUs físicas.

se debe al solapamiento entre cómputo y transferencias de datos de diferentes GPUs virtuales. Segundo, el tiempo de copia de datos global se incrementa al aumentar el número de GPUs virtuales, en la Figura 9(a) la última copia acaba en la celda 24, mientras en el diagrama de la Figura 7(b) las copias terminan en la celda 20. La causa de este efecto ya ha sido mostrada en este documento y se debe al primero de los factores que afectan a la escalabilidad de la aplicación mostrado en la Sección V-B. La Figura 9(b) muestra el ciclo de vida de la aplicación con una instanciación de 4 GPUs virtuales sobre cada GPU física. Un análisis similar al que acabamos de realizar nos permitirá extraer las mismas conclusiones. Cabe destacar que en este caso logramos reducir el tiempo de ejecución respecto a la instanciación 2 a 1.

La ejecución de la aplicación con tenencia múltiple de GPUs virtuales sobre las GPUs físicas también ha sido analizada desde un punto de vista de consumos de potencia y energía. Las Figuras 10(a) y 10(b) muestran la utilización media de GPU, la potencia y el consumo energético de todas las GPUs usando un mapeado 2 a 1 y 4 a 1 de GPUs virtuales por GPU física. A la vista de estas figuras, la conclusión es clara, el uso de múltiples GPUs virtuales sobre una misma GPU física reduce el consumo de energía y aumenta la utilización de las GPUs. Por el contrario, el consumo de potencia instantáneo no se incrementa.

D.3 Resultados experimentales

El análisis de prestaciones de la ejecución de la aplicación de cálculo de riesgos financieros usando rCUDA con tenencia múltiple de GPUs virtuales es presentado en este apartado. Los nodos del cluster que hemos usado para realizar la evaluación experimental poseen un total de 12 cores (hasta 24 threads con hyper-threading) y por lo tanto podemos ejecutar la aplicación con hasta un máximo de 24 GPUs virtuales (para evitar añadir ruido debido a la sobrecarga por CPU). Por tanto realizaremos la evaluación experimental usando hasta 12 GPUs físicas sobre las que mapearemos diferente número de GPUs virtuales.

Las Figura 11 muestra el tiempo de ejecución de la aplicación, este tiempo de ejecución a su vez está dividido en el tiempo que la aplicación dedica a copia de datos, tiempo que dedica a cómputo y tiempo en el que tenemos copias de datos y ejecución de cómputo

de forma simultánea. Además, la Figura 11(a) muestra los resultados obtenidos al usar rCUDA sobre InfiniBand QDR y la Figura 11(b) los resultados de rCUDA sobre FDR. Cuando usamos el sistema de interconexión InfiniBand QDR el tiempo dedicado a copia de datos que no está solapado se reduce un 70%, 84%, 66% y 42% cuando usamos GPUs virtuales mapeadas sobre 1, 2, 4 y 6 GPUs físicas respectivamente. En el caso de FDR, la reducción es del 65%, 77%, 57% y 56%. En consecuencia, el consumo de energía también se reducirá, tal como hemos mostrado anteriormente. Nótese que cuando usamos 12 GPUs físicas, el uso de la tenencia múltiple de GPUs virtuales no reduce el tiempo de ejecución. Esto es debido principalmente a que la reducción en el tiempo de ejecución que ofrece el solapamiento de cómputo y copia de datos es menor que el incremento, en tiempo de ejecución, debido al aumento de la cantidad de datos a enviar, propiciado por el uso de GPUs virtuales.

VI. CONCLUSIONES

En este trabajo hemos demostrado los beneficios que la virtualización remota de GPUs puede reportar a la ejecución de aplicaciones pertinentes al ámbito de la industria de cálculo de riesgos financieros. Hemos analizado y detectado los problemas de escalabilidad de este tipo de aplicaciones. Estos problemas de escalabilidad derivan en una costosa y difícil implantación de este tipo de aplicaciones en entornos de tiempo real. El uso de técnicas de tenencia múltiple de GPUs virtuales sobre GPUs físicas junto con la secuencialización de las copias de datos entre la memoria central y la memoria de la GPU solucionan en gran medida estos problemas permitiendo, de esta forma, el uso de este tipo de aplicaciones en los entornos de tiempo real. Además, los resultados experimentales muestran que se aumenta la utilización de las GPUs, se reduce el tiempo de ejecución y también el consumo de energía.

AGRADECIMIENTOS

El presente trabajo ha sido financiado por la Generalitat Valenciana mediante el proyecto PROMETEOII/2013/009 del programa PROMETEO fase II. Asimismo, los autores agradecen también a Mellanox Technologies el generoso apoyo recibido.

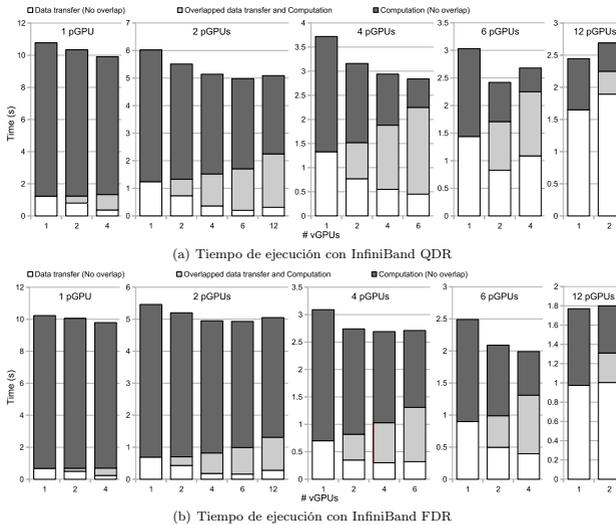


Fig. 11. Tiempo de ejecución de la aplicación para diferentes combinaciones de GPUs virtuales sobre GPUs físicas. Los tiempos de ejecución mostrados han sido obtenidos usando rCUDA sobre InfiniBand QDR y sobre InfiniBand FDR.

REFERENCIAS

- [1] Kuen Hung Tsoi and Wayne Luk, "Axel: A heterogeneous cluster with fpgas and gpus," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2010, pp. 115–124.
- [2] Fengguang Song and Jack Dongarra, "A scalable framework for heterogeneous gpu-based clusters," in *Proceedings of the 24th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2012, pp. 91–100.
- [3] Y. Jiao, H. Lin, F. Balaji, and W. Feng, "Power and performance characterization of computational kernels on the gpu," in *Proceedings of the 2010 IEEE/ACM Int'L Conference on Green Computing and Communications & Int'L Conference on Cyber, Physical and Social Computing*, 2010, pp. 221–228.
- [4] Michela Becchi, Kittisak Sajjapongse, Ian Graves, Adam Procter, Vignesh Ravi, and Srimat Chakradhar, "A virtual memory based runtime to support multi-tenancy in clusters with gpus," in *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, 2012, pp. 97–108.
- [5] Dipanjan Sengupta, Raghavendra Belupure, and Karsten Schwan, "Multi-tenancy on gpgpu-based servers," in *Proceedings of the 7th International Workshop on Virtualization Technologies in Distributed Computing*, 2013, pp. 3–10.
- [6] Blesson Varghese, Javier Prades, Carlos Reaño, and Federico Silla, "Acceleration-as-a-service: Exploiting virtualised gpus for a financial application," in *Proceedings of the 11th IEEE International Conference on eScience*, 2015, pp. 47–56.
- [7] S. Iserte, A. Castelló, R. Mayo, E.S. Quintana-Ortí, F. Silla, J. Duato, C. Reaño, and J. Prades, "Slurm support for remote gpu virtualization: Implementation and performance study," in *Computer Architecture and High Performance Computing (SAC-PAD)*, 2014 *IEEE 26th International Symposium on*, 2014, pp. 318–325.
- [8] A. Srinivasan, "Parallel and distributed computing issues in pricing financial derivatives through quasi monte carlo," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, 2002.
- [9] K. Huang and R.K. Thulasiram, "Parallel algorithm for pricing american asian options with multi-dimensional assets," in *High Performance Computing Systems and Applications, 2005. HPCS 2005. 19th International Symposium on*, 2005, pp. 177–185.
- [10] C. Bekas, A. Curioni, and I. Fedulova, "Low cost high performance uncertainty quantification," in *Proceedings of the 2nd Workshop on High Performance Computational Finance*, 2009.
- [11] D. Daly, Kyung Dong Ryu, and J.E. Moreira, "Multi-variate finance kernels in the blue gene supercomputer," in *High Performance Computational Finance, 2008. WHPCF 2008. Workshop on*, 2008, pp. 1–7.
- [12] V. Agarwal, Lung-Kuo Liu, and D.A. Bader, "Financial modeling on the cell broadband engine," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 2008, pp. 1–12.
- [13] Ciprian Docan, Manish Parashar, and Christopher Marty, "Advanced risk analytics on the cell broadband engine," pp. 1–8, 2009.
- [14] A. Irturk, B. Benson, N. Laptev, and R. Kastner, "Fpga acceleration of mean variance framework for optimal asset allocation," in *High Performance Computational Finance, 2008. WHPCF 2008. Workshop on*, 2008, pp. 1–8.
- [15] D.B. Thomas, "Acceleration of financial monte-carlo simulations using fpgas," in *High Performance Computational Finance (WHPCF)*, 2010 *IEEE Workshop on*, 2010, pp. 1–6.
- [16] L.A. Abbas-Turki, S. Vialle, B. Lapeyre, and P. Mercier, "Pricing derivatives on graphics processing units using monte carlo simulation," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 9, pp. 1679–1697, 2014.
- [17] Duy Minh Dang, Christina C. Christara, and Kenneth R. Jackson, "An efficient graphics processing unit-based parallel algorithm for pricing multi-asset american options," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 8, pp. 849–866, 2012.
- [18] Tyng Yeu Liang and Yu Wei Chang, "Gridcuda: A grid-enabled cuda programming toolkit," in *Advanced In-*

- formation Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on, 2011, pp. 141–146.*
- [19] Lin Shi, Hao Chen, and Jianhua Sun, “vCUDA: GPU accelerated high performance computing in virtual machines,” in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, 2009*, pp. 1–11.
- [20] Vishakha Gupta, Ada Gavrilovska, Karsten Schwan, Harshvardhan Kharache, Niraj Tolia, Vanish Talwar, and Parthasarathy Ranganathan, “GVim: GPU-accelerated virtual machines,” in *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing, 2009*, pp. 17–24.
- [21] Giulio Giunta, Raffaele Montella, Giuseppe Agrillo, and Giuseppe Coviello, *A GPGPU Transparent Virtualization Component for High Performance Computing Clouds*, pp. 379–391, 2010.
- [22] Minoru Oikawa, Atsushi Kawai, Kentaro Nomura, Kenji Yasuoka, Kazuyuki Yoshikawa, and Tetsu Narumi, “DS-CUDA: A Middleware to Use Many GPUs in the Cloud Environment,” in *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, 2012*, pp. 1207–1214.
- [23] A. J. Peña, C. Reaño, F. Silla, R. Mayo, E. S. Quintana-Ortí, and J. Duato, “A complete and efficient cuda-sharing solution for hpc clusters,” *Parallel Computing*, vol. 40, pp. 574–588, 12/2014 2014.
- [24] A.K. Bahl, O. Baltzer, A. Rau-Chaplin, and B. Varghese, “Parallel simulations for analysing portfolios of catastrophic event risk,” in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion*, 2012, pp. 1176–1184.
- [25] Blesson Varghese, “The hardware accelerator debate: A financial risk case study using many-core computing,” *Computers & Electrical Engineering*, vol. 46, pp. 157–175, 2015.

Parallelization Capabilities of Open-Source HEVC Codecs

David García-Lucas¹, Gabriel Cebrián-Márquez¹ and Pedro Cuenca¹

Abstract — The High Efficiency Video Coding (HEVC) standard achieves double compression efficiency compared to H.264/Advanced Video Coding at the cost of huge computational complexity. Parallelizing HEVC encoding is an efficient way of fulfilling this computational requirement. The parallelization algorithms considered in HEVC, such as Tiles or Wavefront Parallel Processing (WPP), rely on creating picture partitions that can be processed concurrently in a multi-core architecture. Since its standardization, several open-source HEVC video codecs have been developed with parallel encoding capabilities. This paper presents a performance evaluation analysis of the open-source HEVC codecs using objective measures of assessment in order to analyze their real parallelization capabilities. The experimental results show that for a high-quality encoding scenario, Divx265 and x265 obtain, on average, the best parallelization performance. In the case of a high-speed encoding scenario, x265 is the one that best exploits this parallelism. In both scenarios, with negligible impact of average BD-rate with respect to the sequential version of the codecs.

Index Terms — HEVC, Open-Source, Evaluation, Parallelization, Computational Cost

I. INTRODUCTION

HIGH Efficiency Video Coding (HEVC) was developed by the *Joint Collaborative Team on Video Coding (JCT-VC)* to replace the current H.264/Advanced Video Coding (AVC) standard [1], which has dominated digital video services in all segments of the domestic and professional markets for over ten years. Established by the JCT-VC in early 2013, HEVC [2] roughly doubles the *Rate Distortion (R-D)* performance of H.264/AVC. This improvement in coding efficiency comes, however, at the expense of an extremely high computational complexity [3]. HEVC compensates this complexity overhead by specifying two principal, mutually exclusive, data-level parallelization strategies [4]: 1) Tiles [5] and 2) *Wavefront Parallel Processing (WPP)* [6]. They are by far the most popular schemes for implementing parallel processing in both software and hardware HEVC encoders.

With the aim of exploiting these parallelization strategies, several HEVC video codecs have been developed by the industry, but most of them are commercial products whose features and operating principles are kept confidential. Therefore, only open-source encoders are considered in this paper. Among the existing open-source HEVC encoders, *HEVC Test*

Model (HM) [7] as an HEVC reference codec is able to achieve the best coding efficiency among the existing HEVC encoders, but its object-based C++ implementation results in poor performance in terms of computational complexity. Hence, it is targeted for research and conformance testing rather than practical encoding. The commercially funded x265 [8] is the most well-known practical open-source HEVC encoder. It is based on HM C++ source code, which has been enhanced by extensive assembly optimizations, multithreading, and techniques from the open-source x264 encoder. Kvazaar [9] is an academic open-source HEVC encoder initiated and coordinated by [10]. It is licensed under GNU GPLv2 license. Kvazaar uses a reverse design approach compared with x265. In addition, Kvazaar is developed in C. *HEVC Open MPEG Encoder (HomerHEVC)* [11] is an open-source, real-time, multipatform HEVC video encoder under LGPL license. Finally, DivX also presents its DivX HEVC encoder known as Divx265 [12].

Except for HM, the rest open-source HEVC encoders support parallel processing. x265 offers WPP, but not Tiles. Kvazaar and HomerHEVC offers both Tiles and WPP. HomerHEVC and x265 also accelerate further parallel coding with picture-level parallel processing. WPP and Tiles are realized in practice by multithreading the software encoder on a multi or many-core processor where data processing flows are identical to each core.

Many previous works have been done to evaluate the R-D performance of HEVC and H.264/AVC standards. All the related works presented in the literature can be classified into two categories: 1) the comparison was performed between different encoders under the same coding standard, and 2) the comparison was conducted between HEVC and other coding standards like H.264/AVC using different coding modes. Note that almost all the papers and research are based on the reference encoders, e.g. HM and JM, which are only suitable for theoretical research. However, as many optimizations have been done since the HEVC standard was released in 2013, people urgently need to know about the performance (both compression and speed) of practical encoders at their current state. Moreover, many of these comparisons focus on non-parallel versions of these encoders. For this reason, this paper focuses on a comparative evaluation of the open-source HEVC codecs using objective measures of assessment in order to analyze their real parallelization capabilities. The comparison was done using settings provided by the developers of each codec.

The remainder of this paper is organized as follows. Section 2 includes some technical background to the new HEVC standard. Section 3 presents the open-source HEVC video codecs under study, and then the

¹ Instituto de Investigación en Informática. Universidad de Castilla-La Mancha, Campus Universitario, 02071, Albacete, España. e-mail: David.Garcia72@alu.uclm.es, Gabriel.Cebrian@uclm.es, Pedro.Cuenca@uclm.es

experimental results are given in Section 4. Finally, Section 5 concludes the paper.

II. TECHNICAL BACKGROUND

HEVC can be considered an evolution of the current H.264/AVC standard, since it maintains the same block-based hybrid approach used in all previous video compression standards. In addition, new tools have been introduced in HEVC that increase its coding efficiency compared with H.264/AVC [13].

One of the most important changes affects picture partitioning [14]. HEVC defines a new flexible *Coding Tree Unit* (CTU) structure which is a replacement of the *Macroblocks* (MBs), 16×16 pixel blocks, used in previous standards. With the aim of achieving an optimal adaptation to the content details, CTUs can vary from a size of 64×64 pixels, to something much smaller, as it can be iteratively partitioned into four square sub-blocks of half resolution, named *Coding Units* (CUs), with a minimum allowable size of 8×8 pixels. Therefore, a CTU can be further partitioned into four depth levels, from $d=0$ for 64×64 CU to $d=3$ for 8×8 CUs, having 4^d CUs in each depth level. Thus, a CU in depth level d can be denoted as $CU_{d,k}$ ($k=0,1,\dots,4^d-1$), and the four sub-CUs pending on $CU_{d,k}$ are denoted as $CU_{d+1,kk+i}$ ($i=0,1,\dots,3$). In a CTU of 64×64, it can be observed that the maximum number of available CUs is $\sum_{d=0}^3 4^d$.

HEVC increases even more the flexibility of the CTU by defining two tree structures containing new unit types: the *Prediction Units* (PUs), and the *Transform Units* (TUs). For intra-picture prediction, a PU uses the same $2N \times 2N$ size as for the $CU_{d,k}$ to which it belongs, allowing it to be split into four $N \times N$ PUs only for CUs at the minimum depth level. Therefore, the PU size can range from 64×64 to 4×4 pixels. For inter-picture prediction, several non-square rectangular block shapes are available in addition to square ones, allowing eight different PU sizes ($2N \times 2N$, $2N \times N$, $N \times 2N$, $N \times N$, $2N \times nU$, $2N \times nD$, $nL \times 2N$, $nR \times 2N$).

The prediction residual obtained in each PU is transformed using the *Residual Quad Tree* (RQT) structure, which supports various TU sizes from 32×32 to 4×4. For the transform of intra 4×4 PU residuals, an integer approximation of the *Discrete Sine Transform* (DST) is used instead.

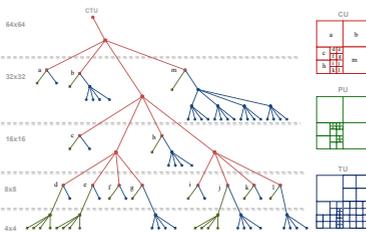


Figure 1. Partitioning of CTU into CUs, PUs and TUs.

In Figure 1, an example of the partitioning is shown, depicting how a CTU is structured in a hierarchical tree where each CU branch ends in a leaf ($CU_{d,k}$) that is the root for the two new prediction and transform trees.

It should be noted that a CTU can be split into 341 different PUs ($\sum_{d=0}^3 4^d$), and each of these available PUs has to be evaluated for all intra/inter prediction modes available, and each of the obtained residual blocks can be transformed into up to three TU sizes. HEVC checks most of the PUs (inter and intra modes) to decide whether it should split a CU or not by choosing the best R-D case. Furthermore, in the case of inter prediction, for each of these PU partitions a motion estimation algorithm is called. This wide range of possibilities makes HEVC much more computationally expensive than its predecessor, H.264/AVC. HEVC introduces changes in other modules too, such as intra prediction (where a total of 35 different coding modes can be selected), new image filters or new transform sizes [13], among others. As expected, the selection of the optimal partitioning for each CU/PU/TU is an intensive time consuming process due to the huge number of combinations that have to be evaluated.

The above analysis evidences the need to reduce the *Rate-Distortion Optimization* (RDO) complexity for the HEVC intra/inter prediction, in order to make real time HEVC video codecs with the best possible performance. With the aim of reducing this huge RDO complexity, HEVC places special emphasis on hardware friendly design and parallel-processing architectures. These parallelization approaches are Tiles [5] and WPP [6]. Basically, these parallelization approaches considered in HEVC rely on creating picture partitions and, thus, coding losses may appear due to breaking dependencies for prediction, CABAC context modeling, and/or slice header overhead. As a result, coding losses may appear.

A. Parallelization techniques in HEVC

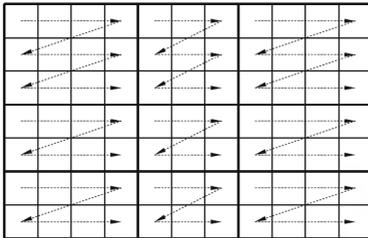
Previous video codecs, in particular H.264/AVC, have been parallelized using either frame-level, slice-level, or MB-level parallelism. Each of these approaches, however, has some limitations such as limited scalability, significant coding losses, or high memory requirements. To overcome these limitations, the new HEVC codec includes new parallelization techniques such as Tiles [5] and WPP [6] among slices.

On the one hand, Tiles are square or rectangular-shaped partitions where dependencies are broken across tile boundaries, entailing, hence, substantial coding losses, while making it possible to process them independently. Nevertheless, the deblocking and *Sample Adaptive Offset* (SAO) filters can still cross these boundaries. The number of tiles and their location can be defined for the whole sequence or changed from picture to picture.

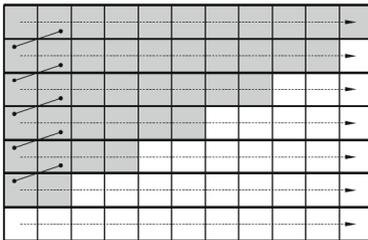
On the other hand, WPP allows the pictures to be partitioned in rows, each of which can be processed in parallel, whereas entropy encoding and prediction are allowed to cross partitions to minimize coding losses. However, coding dependencies make it necessary to have a delay of, at least, two CUs between consecutive rows in a similar way to segmentation in a computer architecture. For this reason, not all the processes can

start encoding these rows at the same time, which introduces the so-called “ramping inefficiencies” at the beginning and at the end of a frame, not allowing a parallel encoder/decoder to reach its peak performance. Both techniques, Tiles and WPP, are depicted in Fig. 2.

Tiles and WPP have different merits and disadvantages. While WPP is generally well suited for the parallelization of the encoder and the decoder due to its high number of picture partitions with low compression losses, the amount of parallelism with Tiles is not fixed, as the number of regions in which a frame is divided may vary. Additionally, WPP does not introduce artifacts at partition boundaries as is the case for Tiles. WPP has almost no effect on the analysis and compression of each CTU and so it has a very small impact on compression efficiency relative to Tiles. The compression loss from WPP has been found to be less than 1% in most of the tests. To simplify the implementation, it is not possible to use Tiles and WPP simultaneously in the same compressed video sequence.



(a) Tiles partitioning



(b) WPP partitioning

Figure 2. Partitioning and processing order of Tiles (a) and WPP (b).

III. OPEN-SOURCE HEVC VIDEO CODECS

With the aim of reducing this high computational complexity, several suboptimal (from the coding efficiency point of view) open-source HEVC encoders have been developed by the industry.

Among the existing open-source HEVC encoders, **HM** [7] as an HEVC reference codec is able to achieve the best coding efficiency among the existing HEVC encoders. The HM software includes both encoder and

decoder functionality. Reference software is useful in aiding users of a video coding standard to establish and educate users, and demonstrate the capabilities of the standard. However, its object-based C++ implementation results in poor performance in terms of computational complexity. HM encoder does not support parallel processing. Hence, it is targeted for research and conformance testing rather than practical encoding.

x265 [8] is a commercially funded open-source implementation of the H.265/HEVC compression standard. The x265 codec is available under the GNU GPL v2 license. The x265 team has licensed the rights to port and adapt x264 for use in x265. Most of the video encoding algorithms developed for x264 have been incorporated in x265, including rate control functions, the look-ahead function, macroblock tree, b-pyramid and adaptive quantization. It is based on HM C++ source code which has been enhanced by extensive assembly optimizations and multithreading techniques. For parallel encoding, x265 offers two schemes: WPP and frame-threading processing. In x265, WPP is enabled by default since it not only improves performance at encode but it also makes it possible for the decoder to be threaded. Frame threading is the act of encoding multiple frames at the same time. x265 works around this problem by limiting the motion search region within the reference frames to just one CTU row below the coincident row being encoded. Thus a frame could be encoded at the same time as its reference frames as long as the latter are at least one row ahead. By default, frame parallelism and WPP are enabled together. The number of frame threads used is auto-detected from the (hyperthreaded) CPU core count.

Kvazaar [9] is an academic open-source HEVC encoder initiated and coordinated by [10] with the following primary goals: 1) coding efficiency close to HM, 2) modular structure to ease its parallelization and portability, and 3) excellent source code readability and documentation. To obtain these objectives with the highest possible encoding speed and with minimal computation/memory resources, Kvazaar has been developed from scratch in C. This unique approach uses the object-based C++ implementation of HM as a reference for its encoding scheme and individual algorithm implementations, but it adopts completely new data and function call tree structures. For parallel encoding, Kvazaar offers three schemes: Tiles, WPP, and picture-level parallel processing. Picture-level parallel processing can be utilized jointly with tiles and WPP. Kvazaar parallelization is implemented using a thread pool with a single CTU as the smallest work unit. The CTUs are put in a queue in the order in which they would be processed in a single threaded case, and the free worker threads always select the first CTU with no dependencies for processing.

HomerHEVC [11] is an open-source, real-time, multiplatform H.265/HEVC video encoder under LGPL license. The current features in its version 2.0 are: multiplatform (Linux, Windows), 8 bit-depth, intra and baseline profile, multiple references for IPP, all intra prediction modes, $2N \times 2N$ and $N \times N$ inter prediction modes, all prediction sizes, all transform sizes, half pixel

and quarter pixel precision motion estimation, rate control, deblocking filter and SAO. Several SSE42 optimizations are also implemented. For parallel encoding, HomerHEVC offers two schemes: WPP and frame based parallelization. By default frame parallelism and WPP are enabled together. The number of default WPP and frame threads used are ten and three, respectively.

Finally, DivX and NeuLion solutions are powering the professional creation, secure distribution, and multi-screen playback of high-quality video for leading companies around the world [12]. The latest version of DivX265 has improved both encoding speed and efficiency dramatically. It provides four different encoding modes (presets) and wavefront parallel processing is employed to accelerate its coding speed.

IV. PERFORMANCE EVALUATION ANALYSIS

A. Encoding Parameters

This section aims to evaluate the real parallelization capabilities of the open-source HEVC codecs presented in this paper. The parameters used for each encoder are listed in Table I. Our experiments rely on the default configuration of HM 16.6, which is used as an anchor for the obtained results. With regard to the rest of the open-source HEVC codecs under study, the evaluated presets are *veryslow* (high-quality encoding) and *ultrafast* (high-speed encoding). In practical applications, both high compression and fast coding are desired for video coding. They represent the slowest and fastest practical presets of them, respectively.

It should be noted that as not all encoding parameters, can be specified, and the default parameters of HM may be quite different from those of x265, HomerHEVC, Kvazaar and DivX265, the evaluation can only draw a rough picture of current open-source HEVC encoders. However, we believe the R-D complexity evaluation can still provide a helpful reference for practical applications.

TABLE I: ENCODER COMMAND LINE PARAMETERS.

Encoder	Parameters
HM	-c (encoder configuration file) -c (sequence configuration file) --IntraPeriod=(intra period) --QP=(qp)
HomerHEVC	--widthxheight (resolution) -n (frames) (frames) -frame_rate (fps) -bitrate_mode 0 -qp (qp) -gop_size (0 for AI, 1 for LP, 2 for RA) -intra_period (intra period) -n_cengines 3 -n_wpp_threads 10 -performance_mode (preset)
Kvazaar	--input-res (resolution) -n (frames) --input-fps (fps) -qp (qp) -p (intra period) --gop (8 for RA, lp-g4d4r4t1 for LP) --threads (2 for 2 cores, 4 for 4 cores, 8 for 8 cores) --wpp --preset (preset)
X265	--input-res (resolution) -f (frames) --fps (fps) -qp (qp) --keyint (intra period) --bframes (0 for LP) --pools (2 for 2 cores, 4 for 4 cores, 8 for 8 cores) --frame-threads (1 for 2 cores, 2 for 4 cores, 3 for 8 cores) -p (preset)
Divx265	-s (resolution) -n (frames) -fps (fps) -l 1 -qp (qp) -aqo (preset)

The hardware platform used in the experiments is composed of an Intel® Xeon® E5-2630L v3 octa-core CPU running at 1.80 GHz, 16 GB of main memory. The

encoders were compiled with GCC 4.8.5-4 and executed on CentOS 7 (Linux 3.10.0-327). Turbo Boost was disabled to achieve the reproducibility of the results.

Due to Tiles strategy not being implemented in all open-source HEVC codecs under study, this paper focuses mainly on WPP and frame-thread parallelization strategies as benchmark.

In order to ensure a common framework for the simulations, the experiments were conducted under the *Common Test Conditions and Software Reference Configurations* recommended by the JCT-VC [15] for *Random-Access* (RA) mode configuration. That recommendation specifies the use of four *Quantization Parameter* (QPs) (22, 27, 32 and 37) and a set of 18 test sequences classified in five classes, A to E, which cover a wide range of resolutions from the largest (2560×1600 pixels) to the smallest (416×240 pixels), and frame rates from 60 fps to 24 fps. All the sequences use 4:2:0 chroma subsampling and a bit-depth of 8 bits:

- Class A (2560×1600 pixels): *PeopleOnStreet*.
- Class B (1920×1800 pixels): *Kimono*, *ParkScene*, *Cactus*, *BQTerrace* and *BasketballDrive*.
- Class C (832×480 pixels): *RaceHorsesC*, *BQMall*, *PartyScene* and *BasketballDrill*.
- Class D (416×240 pixels): *RaceHorses*, *BQSquare*, *BlowingBubbles* and *BasketballPass*.
- Class E (1280×720 pixels): *FourPeople*, *Johnny* and *KristenAndSara*.

B. Metrics

The rate-distortion/complexity analysis was evaluated in terms of *Computational Complexity Reduction* (CCR) and R-D performance for all open-source HEVC codecs under study, and both of them were compared to the HM codec results used as reference. For the CCR measure, the *Time Saving* (TS) metric was computed following the Equation (1):

$$T.Saving(\%) = \frac{EncTime(HEVCCodec) - EncTime(HM_{16.6})}{EncTime(HM_{16.6})} \cdot 100 \quad (1)$$

Regarding the R-D performance, the average *Peak Signal to Noise Ratio* (PSNR) metric was calculated for each luma (PSNR_Y) and chroma components (PSNR_U, PSNR_V) and for the full number of frames of each sequence. In order to obtain a global quality measure, the average PSNR of the three components, denoted as PSNR_{YUV}, was also computed. As mentioned above, the chroma subsampling format of the test sequences was 4:2:0, so the well-known *PSNR_{YUV}* according to Equation (2) was used, which applies weight pondering to the PSNR obtained for each component. Those weights are recommended by the JCT-VC in [16] and they are considered a fair representation of the visual quality, which is more sensitive to the luminance stimulus than the chrominance.

$$PSNR_{YUV}(dB) = \frac{6 \cdot PSNR_Y + PSNR_U + PSNR_V}{8} \quad (2)$$

The PSNR_{YUV} for the four QPs were used for the computation of the R-D performance by using the

Bjontegaard Delta Rate (BDR) metric defined by ITU [17, 18] and recommended by the JCT-VC. The BDR provides the average difference between the R-D curves measured as a percentage of bit rate that is necessary to increase or decrease to achieve the same PSNR quality in both curves. In our simulation, a positive BDR means the encoded bit rate using the open-source HEVC codec under study is higher than the bit rate obtained with HM, thus that is denoted as the penalty in terms of bit rate.

Finally, in order to measure the real parallelization capabilities, the sequential algorithm of each open-source HEVC codec has been used as the reference algorithm, to calculate the corresponding speed-up and coding efficiency values when WPP and frame-threading processing are enabled for 2, 4 and 8 cores in each codec.

C. Simulation Results

Table II shows the experimental results of the sequential open-source HEVC codecs under study compared with the HM 16.6 reference software for the high-quality encoding scenario. It can be observed that HomerHEVC codec obtain the highest time savings (98.21%) while increasing excessively the bit rate penalty by around 123%. On the contrary, Divx265 codec obtains the lowest bit rate penalty (19%) with a considerable time saving (80%). Table III shows the corresponding speed-up and coding efficiency values when WPP and frame-threading processing are enabled for the high-quality encoding scenario. As can be seen, the HEVC codecs can reach speed-up values close to the ones from a parallel efficient algorithm (i.e. threads are almost fully utilized), providing that the frame size is large enough to exploit the available parallelism (see Class A, B and E). Accordingly, WPP obtains lower speed-up results with lower resolutions (see Class C and D). Generally, we have much less frame parallelism to exploit in lower resolutions than in higher resolution due to the number of CTU rows available. For instance, Class A video sequences has 25 CTU rows available each frame while Class D video sequences only has 4 CTU rows. It should be noted that no bit rate penalty is paid when parallelization strategies are enabled in all HEVC codecs under study. Among the HEVC codecs, Divx265 and x265 obtain, on average, the best parallelization performance.

Table IV shows the experimental results of the sequential open-source HEVC codecs under study compared with the HM 16.6 reference software for the high-speed encoding scenario. It can be observed that x265 codec obtain the lowest bit rate penalty (117%) with similar time saving (98.5%) compared with the rest of HEVC codecs, which show an excessive bit rate penalty of over 150% or even 250%. High time savings are obtained by the open-source HEVC codecs using several suboptimal and reduced set of prediction modes in the picture partitioning. Table V shows the corresponding speed-up and coding efficiency values when WPP and frame-threading processing are enabled for the high-speed encoding scenario. In this scenario x265 obtain, on average, the best parallelization performance. With regard to coding efficiency, this speed-up has a negligible impact of average BD-rate with respect to the sequential version of the codecs.

V. CONCLUSIONS

This paper presents a rate-distortion/complexity analysis of the several open-source HEVC codecs in order to analyse their real parallelization capabilities. Among the HEVC codecs, for the high-quality encoding scenario, Divx265 and x265 obtain, on average, the best parallelization performance. In the case of the high-speed encoding scenario, x265 obtains, on average, the best parallelization performance. With regard to coding efficiency, the speed-up obtained have a negligible impact of average BD-rate with respect to the sequential version of the codecs.

ACKNOWLEDGEMENTS

This work was jointly supported by the Spanish MINECO Ministry and the European Commission (FEDER funds) under the project TIN2015-66972-C5-2-R, and by the grant FPU13/04601.

REFERENCES

- [1] ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), "Advanced Video Coding for Generic Audiovisual Services", Feb. 2012.
- [2] ISO/IEC and ITU-T, "High Efficiency Video Coding (HEVC). ITU-T Recommendation H.265 and ISO/IEC 23008-2 (version 3)", April 2015.
- [3] J.-R. Ohm, G. J. Sullivan, H. Schwarz, Thiwu Keng Tan, and T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards - Including High Efficiency Video Coding (HEVC)", IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1669-1684, Dec. 2012.
- [4] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel Scalability and Efficiency of HEVC Parallelization Approaches", IEEE Trans. Circuits Syst. Video Technol. vol. 22, no. 12, Dec. 2012, pp. 1827-1838.
- [5] K. Misra, A. Segall, M. Horowitz, X. Shilin, A. Fuldseth, and Z. Minhua, "An Overview of Tiles in HEVC" IEEE J. Select. Topics in Signal Process., vol. 7, no. 6, Dec. 2013, pp. 969-977.
- [6] F. Henry and S. Pateux, "Wavefront Parallel Processing" Document JCTVC-E196, Geneva, Switzerland, Mar. 2011.
- [7] Joint Collaborative Team on Video Coding Reference Software, ver. HM 16.6. [Online]. [https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/]
- [8] X265. [Online]. Available: <http://x265.org/>
- [9] Kvazaar HEVC encoder. [Online]. Available: <https://github.com/ultravideo/kvazaar>
- [10] Ultra video group [Online]. Available: <http://ultravideo.es.tut.fi/>
- [11] HomerHEVC encoder. [Online]. Available: <https://github.com/jcasal-homer/HomerHEVC>
- [12] DivxHEVC encoder. [Online]. Available: <http://labs.divx.com/divx265>
- [13] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard" IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1649-1668, Dec. 2012.
- [14] I. K. Kim, J. Min, T. Lee, W. J. Han, and J. Park, "Block Partitioning Structure in the HEVC Standard" IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1697-1706, Dec. 2012.
- [15] F. Bossen, "Common Test Conditions and Software Reference Configurations" document JCTVC-L1100, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC), 12th Meeting: Geneva, CH 14 - 23 Jan. 2013.
- [16] Gary Sullivan, Koobhyar Minoo, "Objective Quality Metric and Alternative Methods for Measuring Coding Efficiency", document JCTVC-H0012, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC), 8th Meeting: San Jose, CA, USA, 1 - 10 Feb. 2012.
- [17] G. Bjontegaard, "Calculation of Average PSNR Differences between RD-Curves", ITU-T SG16 Q.6 Document, VCEG-M33, Austin, US, April 2001.
- [18] G. Bjontegaard, "Improvements of the BD-PSNR Model," ITU-T SG16 Q.6 Document VCEG-A111, 35th VCEG Meeting, Jul. 2008.

TABLE II: TIME SAVINGS (TS) AND CODING EFFICIENCY (BDR) RESULTS WITH RESPECT TO HM.
 HIGH-QUALITY ENCODING SCENARIO

Classification	Sequence	Frames	HomerHEVC		Divx265		Kvazaar		x265	
			TS (%)	BDR (%)	TS (%)	BDR (%)	TS (%)	BDR (%)	TS (%)	BDR (%)
Class A (2560x1600)	Traffic	150	-98,08	105,8	-79,81	19,8	-87,39	60,4	-54,89	28,1
	PeopleOnStreet	150	-98,31	51,1	-82,15	10,0	-68,76	45,7	-35,51	22,0
	Kimono	240	-98,20	80,3	-83,56	20,8	-71,75	59,4	-44,94	38,1
Class B (1920x1080)	ParkScene	240	-98,06	100,5	-78,85	23,0	-82,48	57,6	-48,83	33,3
	Cactus	500	-98,22	107,2	-79,99	16,6	-74,98	77,3	-49,20	44,7
	BasketballDrive	500	-98,33	82,3	-82,19	15,2	-63,45	76,3	-36,67	52,3
	BQTerrace	600	-98,17	182,0	-78,56	25,2	-82,89	120,7	-48,87	43,5
Class C (832x480)	BasketballDrill	500	-98,29	100,0	-79,98	17,3	-65,96	45,4	-47,70	39,1
	BQMall	600	-98,20	115,5	-79,82	19,7	-69,54	73,5	-44,93	45,5
	PartyScene	500	-98,11	164,9	-74,90	21,9	-77,93	64,7	-44,51	38,6
	RaceHorses	300	-98,40	56,3	-81,03	14,3	-57,30	51,3	-29,21	23,5
Class D (416x240)	BasketballPass	500	-98,36	63,6	-80,12	10,3	-57,54	49,0	-36,94	26,0
	BQSquare	600	-97,94	334,4	-73,98	33,0	-86,23	126,7	-47,66	45,4
	BlowingBubbles	500	-98,04	145,5	-73,56	22,0	-81,61	63,7	-46,82	39,1
	RaceHorses	300	-98,34	60,9	-79,50	13,5	-56,11	51,0	-27,19	24,6
Class E (1280x720)	FourPeople	600	-98,21	124,7	-81,78	17,3	-91,43	50,2	-65,38	37,9
	Johnny	600	-98,28	197,1	-83,53	22,1	-94,56	76,7	-62,50	48,2
	KristenAndSara	600	-98,29	155,1	-83,50	19,4	-91,94	67,9	-58,00	45,1
Class A		-98,19	78,43	-80,98	14,88	-78,07	53,05	-45,20	25,07	
Class B		-98,20	110,46	-80,63	20,16	-75,11	78,24	-45,70	42,38	
Class C		-98,25	109,20	-78,93	18,28	-67,68	58,75	-41,59	36,69	
Class D		-98,17	151,11	-76,79	19,69	-70,37	72,58	-39,65	33,78	
Class E		-98,26	158,94	-82,93	19,58	-92,64	64,93	-61,96	43,73	
AVERAGE			-98,21	123,73	-79,82	18,95	-75,66	67,64	-46,10	37,51

TABLE III: SPEED-UP AND CODING EFFICIENCY (BDR) RESULTS (2, 4 AND 8 CORES).
 HIGH-QUALITY ENCODING SCENARIO

Class	Sequence	HomerHEVC			Divx265			Kvazaar			x265		
		Speed-Up			Speed-Up			Speed-Up			Speed-Up		
		2	4	8	2	4	8	2	4	8	2	4	8
A	Traffic	1,95	3,89	7,79	2,01	4,02	8,04	1,93	3,74	7,63	1,96	3,94	7,72
	PeopleOnStreet	1,95	3,91	7,81	2,00	4,01	8,03	1,91	3,82	8,05	1,96	3,98	7,79
B	Kimono	1,95	3,91	7,60	1,99	3,99	7,95	1,90	3,51	6,59	1,96	3,92	7,58
	ParkScene	1,95	3,89	7,60	1,99	3,99	7,97	1,86	2,98	5,73	1,96	3,90	7,51
	Cactus	1,96	3,91	7,63	2,00	3,99	7,99	1,90	3,38	6,11	1,96	3,99	7,74
C	BasketballDrive	1,94	3,91	7,62	2,00	4,00	7,95	1,90	3,48	6,93	1,96	3,98	7,72
	BQTerrace	1,97	3,89	7,62	2,00	4,00	7,99	1,89	3,39	6,19	1,96	3,96	7,70
	BasketballDrill	1,83	3,55	4,37	1,96	3,61	4,81	1,87	3,52	3,75	1,90	3,70	5,27
D	BQMall	1,82	3,50	4,14	1,97	3,59	4,83	1,89	3,50	3,67	1,90	3,71	5,29
	PartyScene	1,84	3,59	4,48	1,97	3,62	4,89	1,76	3,13	3,32	1,88	3,73	5,29
	RaceHorses	1,84	3,58	4,62	1,97	3,65	4,89	1,86	3,74	4,47	1,89	3,83	6,04
	BasketballPass	1,52	1,50	1,34	1,69	2,23	2,28	1,81	1,98	2,04	1,67	2,22	2,86
E	BQSquare	1,53	1,63	1,40	1,73	2,37	2,44	1,85	2,07	1,97	1,68	2,30	3,10
	BlowingBubbles	1,53	1,67	1,41	1,74	2,33	2,42	1,82	2,02	2,02	1,66	2,23	2,91
	RaceHorses	1,55	1,71	1,42	1,76	2,34	2,44	1,87	2,16	2,37	1,69	2,46	3,08
E	FourPeople	1,92	3,80	7,10	1,99	3,95	6,68	1,88	3,12	3,22	1,95	3,91	7,11
	Johnny	1,91	3,79	6,83	1,99	3,97	7,15	1,92	3,72	4,87	1,91	3,87	7,42
	KristenAndSara	1,92	3,80	6,81	1,99	3,97	7,07	1,83	3,43	3,91	1,91	3,88	7,71
A		1,95	3,90	7,80	2,00	4,01	8,03	1,92	3,78	7,84	1,96	3,96	7,76
B		1,95	3,90	7,61	1,99	3,99	7,97	1,89	3,35	6,31	1,96	3,95	7,65
C		1,83	3,55	4,41	1,97	3,62	4,86	1,85	3,47	3,80	1,89	3,74	5,47
D		1,53	1,63	1,39	1,73	2,32	2,40	1,84	2,06	2,10	1,68	2,30	2,99
E		1,92	3,79	6,91	1,99	3,96	6,97	1,88	3,42	4,00	1,93	3,89	7,28
AVERAGE		1,84	3,36	5,63	1,94	3,58	6,04	1,87	3,22	4,81	1,88	3,57	6,23
BDR (%)													
A		78,72	78,64	78,65	14,88	14,88	14,88	53,05	53,05	53,28	25,07	25,11	25,11
B		110,50	110,48	110,50	20,16	20,16	20,16	78,24	78,24	78,24	42,38	40,41	40,41
C		109,25	109,29	109,24	18,28	18,28	18,28	58,75	58,75	58,75	36,69	36,36	36,36
D		151,10	151,18	151,16	19,69	19,69	19,69	72,39	72,39	72,39	33,78	33,64	33,64
E		159,13	159,07	159,04	19,58	19,58	19,58	64,93	65,00	65,00	43,73	44,42	44,42
AVERAGE		123,82	123,82	123,81	18,95	18,95	18,95	67,59	67,61	67,63	37,51	36,98	36,98

TABLE IV: TIME SAVINGS (TS) AND CODING EFFICIENCY (BDR) RESULTS WITH RESPECT TO HM.
 HIGH-SPEED ENCODING SCENARIO

Classification	Sequence	Frames	HomerHEVC		Divx265		Kvazaar		x265	
			TS (%)	BDR (%)						
Class A (2560x1600)	Traffic	150	-99.12	135.9	-99.77	155.2	-99.42	214.5	-98.71	92.9
	PeopleOnStreet	150	-99.19	62.5	-99.74	113.7	-99.27	135.6	-98.15	82.5
	Kimono	240	-99.13	112.6	-99.69	99.2	-99.31	178.1	-98.29	63.9
Class B (1920x1080)	ParkScene	240	-99.11	120.3	-99.71	139.2	-99.30	203.3	-98.50	84.7
	Cactus	500	-99.21	136.4	-99.68	147.6	-99.33	259.9	-98.47	117.9
	BasketballDrive	500	-99.23	102.4	-99.65	124.8	-99.25	257.1	-98.34	96.5
	BQTerrace	600	-99.19	235.2	-99.69	312.1	-99.28	574.0	-98.50	189.2
Class C (832x480)	BasketballDrill	500	-99.25	121.6	-99.75	142.8	-99.35	158.7	-98.58	114.6
	BQMall	600	-99.16	140.5	-99.71	198.9	-99.21	280.8	-98.58	139.8
	PartyScene	500	-99.23	201.1	-99.72	238.0	-99.12	313.9	-98.41	133.9
	RaceHorses	300	-99.29	72.8	-99.70	154.1	-99.14	202.0	-98.14	115.9
Class D (416x240)	BasketballPass	500	-99.19	75.5	-99.69	128.4	-99.18	157.8	-98.32	98.3
	BQSquare	600	-99.04	421.2	-99.68	495.5	-99.01	641.4	-98.48	222.2
	BlowingBubbles	500	-99.15	181.4	-99.69	209.6	-99.11	268.7	-98.53	132.5
	RaceHorses	300	-99.20	73.8	-99.68	156.9	-99.09	186.8	-98.07	115.9
Class E (1280x720)	FourPeople	600	-99.20	169.9	-99.74	111.1	-99.58	177.6	-99.10	83.0
	Johnny	600	-99.19	272.9	-99.70	165.3	-99.61	293.4	-99.16	120.9
	KristenAndSara	600	-99.18	208.0	-99.72	140.9	-99.60	252.0	-99.08	106.8
Class A		-99.16	99.20	-99.75	134.47	-99.34	175.04	-98.43	87.69	
Class B		-99.17	141.35	-99.68	164.59	-99.29	294.49	-98.42	110.45	
Class C		-99.23	134.00	-99.72	183.44	-99.21	238.85	-98.43	126.03	
Class D		-99.14	187.98	-99.68	247.59	-99.10	313.69	-98.35	142.23	
Class E		-99.19	216.93	-99.72	139.11	-99.60	241.00	-99.11	103.60	
AVERAGE			-99.18	157.99	-99.71	179.63	-99.29	264.20	-98.52	117.30

TABLE V: SPEED-UP AND CODING EFFICIENCY (BDR) RESULTS (2, 4 AND 8 CORES).
 HIGH-SPEED ENCODING SCENARIO

Class	Sequence	HomerHEVC			Divx265			Kvazaar			x265		
		Speed-Up			Speed-Up			Speed-Up			Speed-Up		
		2	4	8	2	4	8	2	4	8	2	4	8
A	Traffic	1.95	3.85	7.65	1.69	3.27	5.43	1.97	3.69	8.25	1.98	3.98	7.87
	PeopleOnStreet	1.95	3.90	7.76	1.82	3.54	6.33	1.94	3.77	8.22	2.01	4.02	7.95
B	Kimono	1.97	3.90	7.27	1.83	3.46	5.97	1.96	3.57	7.92	1.96	3.94	7.77
	ParkScene	1.94	3.83	7.08	1.78	3.37	5.82	1.95	3.42	7.89	1.94	3.92	7.70
	Cactus	1.95	3.85	7.08	1.84	3.48	6.29	1.96	3.48	7.80	1.96	3.93	7.71
	BasketballDrive	1.99	3.93	7.26	1.87	3.60	6.49	1.96	3.56	7.98	1.94	3.94	7.75
C	BQTerrace	1.95	3.83	7.12	1.84	3.45	6.29	1.91	3.48	7.85	1.90	3.91	7.66
	BasketballDrill	1.83	3.29	3.44	1.75	3.01	4.86	1.87	3.81	5.57	1.88	3.88	7.38
	BQMall	1.81	3.25	3.41	1.77	3.03	4.94	1.90	3.96	5.42	1.87	3.89	7.45
	PartyScene	1.81	3.29	3.77	1.78	3.09	5.00	1.87	3.88	6.08	1.89	3.88	7.42
D	RaceHorses	1.85	3.40	4.02	1.81	3.20	5.25	1.85	3.89	6.05	1.91	3.87	7.47
	BasketballPass	1.41	1.20	1.19	1.73	2.65	3.23	1.91	2.55	2.57	1.81	3.38	5.02
	BQSquare	1.41	1.21	1.19	1.69	2.57	3.05	1.93	2.73	2.79	1.75	3.44	4.70
	BlowingBubbles	1.41	1.21	1.18	1.68	2.56	3.00	1.96	2.88	3.05	1.76	3.40	4.63
E	RaceHorses	1.46	1.24	1.21	1.76	2.67	3.19	1.89	2.76	3.16	1.83	3.68	5.48
	FourPeople	1.91	3.64	6.43	1.64	3.04	4.95	1.91	4.02	6.00	1.87	3.86	7.49
	Johnny	1.91	3.66	6.11	1.74	3.09	5.18	1.87	3.94	7.32	1.85	3.85	7.51
E	KristenAndSara	1.90	3.64	6.04	1.74	3.09	5.19	1.78	3.77	6.84	1.85	3.85	7.46
	A	1.95	3.88	7.71	1.75	3.40	5.88	1.96	3.73	8.23	1.99	4.00	7.91
B	1.96	3.87	7.16	1.83	3.47	6.17	1.95	3.50	7.89	1.94	3.93	7.72	
C	1.83	3.31	3.66	1.78	3.08	5.01	1.87	3.88	5.78	1.89	3.88	7.43	
D	1.42	1.21	1.19	1.71	2.61	3.12	1.92	2.73	2.90	1.79	3.47	4.96	
E	1.91	3.65	6.19	1.71	3.07	5.11	1.86	3.91	6.72	1.85	3.85	7.49	
AVERAGE		1.81	3.18	5.18	1.76	3.13	5.06	1.91	3.55	6.30	1.89	3.83	7.10
BDR (%)													
A	99.77	99.69	99.72	134.47	134.47	134.47	175.04	175.04	175.19	87.69	87.69	87.69	
B	141.74	141.57	141.75	164.59	164.59	164.59	294.49	294.49	294.11	110.45	110.46	110.46	
C	136.97	136.98	137.01	183.44	183.44	183.44	238.85	238.99	238.99	126.03	126.04	126.04	
D	189.63	188.80	188.39	247.59	247.59	247.59	313.48	313.48	313.48	142.23	142.23	142.23	
E	216.16	216.00	215.97	139.11	139.11	139.11	241.00	240.80	240.80	103.60	103.60	103.60	
AVERAGE		159.06	158.80	158.76	179.63	179.63	179.63	264.16	264.15	264.07	117.30	117.31	117.31

Sintonización de la cuantización uniforme en compresores perceptuales de imagen basados en la transformada wavelet

Martínez-Rach, M. O.; López Granado, O.; Piñol Peral, P.; Pérez Malumbres, M.
Departamento de Física y Arquitectura de Computadores de la Universidad Miguel Hernández
{mmrach, otoniel, pablop, mels}@umh.es

Resumen — Cuando se usan esquemas de cuantización uniforme con dead zone, el rendimiento R/D final puede verse afectado por dos parámetros, el tamaño del dead zone y la ubicación del punto de reconstrucción en cada intervalo de cuantización. En este trabajo analizamos cómo afecta el ancho del dead zone a la calidad de la imagen reconstruida en codificadores wavelet que han sido mejorados perceptualmente usando la función de sensibilidad al contraste del sistema visual humano. Puesto que la aplicación de la CSF altera la distribución de los coeficientes wavelet en los intervalos de cuantización y afecta a los coeficientes que entran en el dead zone, una elección apropiada de estos parámetros de cuantización puede mejorar el rendimiento R/D. Tras un completo estudio de los efectos de estos parámetros del cuantizador, observamos que ajustándolo de manera óptima para cada imagen, se pueden obtener ahorros en tasa de bits de hasta el 7,7% respecto a la utilización de esquemas tradicionales de cuantización uniforme o con dead zone fijo.

Palabras clave—Cuantización perceptual; Codificación perceptual de imágenes; análisis del rendimiento R/D; Contrast Sensitivity Function; Codificadores de Imagen basados en wavelets.

I. INTRODUCCIÓN

EN un codificador de imagen o video con pérdida, el cuantizador es la etapa en la que se produce la pérdida de información. Obviamente la pérdida de información está estrechamente relacionada con la calidad de la imagen reconstruida y con la tasa de bits objetivo. Por tanto el cuantizador debe diseñarse con cuidado para obtener la mejor calidad posible para una determinada tasa de bits o para obtener la mínima tasa de bits posible para una misma calidad de la imagen reconstruida, es decir, debe ser diseñado para obtener la mejor relación rate-distortion (R/D).

Los esquemas de cuantización más utilizados en los estándares de codificación de imagen y video son: (a) Uniform Scalar Quantizer (USQ) usado en el JPEG, SPITH, MPEG-2, MPEG-4 y JPEG2000 Part I entre otros; (b) Uniform Scalar Dead Zone Quantizer (USDZQ) usado en los codificadores H.263, H.264/AVC y en HEVC; (c) Universal Coded Trellis Quantizer (UTCQ) usado en JPEG2000 Part II y finalmente (d) Uniform Variable Dead Zone Quantizer (UVDZQ) usado también en el codificador JPEG2000 Part II.

Todos estos esquemas de cuantización eliminan la información de la imagen que aportan los coeficientes localizados en la zona alrededor del cero, conocida como Dead Zone (DZ). La diferencia entre USQ y USDZQ es básicamente el Dead Zone Size (DZS) como se muestra en la Fig. 1, donde además se puede observar que el tamaño de los intervalos de cuantización, permanece constante o uniforme, con valor Δ . Estos cuantizadores ubican el punto de reconstrucción en el centro del intervalo. Por ejemplo, en la Fig. 1, el punto de reconstrucción para cada coeficiente dentro del primer intervalo, entre d_1 y d_2 , es r_1 .

Un cuantizador UVDZQ, variable o parametrizado, puede actuar como un USQ o como un USDZQ. Se pueden modelar variando el paso de cuantización Δ y la ubicación del punto de reconstrucción, denotado por δ . El DZS se expresa normalmente como múltiplo de Δ y por tanto un USQ tiene un DZS DE 1Δ y un USDZQ tiene un tamaño fijo de DZ de 2Δ .

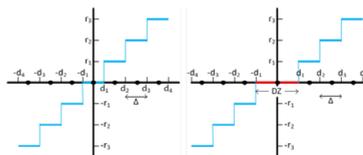


Fig. 1. Esquemas de cuantización uniforme. Izquierda USQ: DZS= 1Δ ; Derecha USDZQ: DZS= 2Δ

Para determinar qué coeficientes deben anularse, es decir, entrar en el dead zone, hay que establecer un compromiso entre calidad y rate. Variaciones en el DZS y en Δ afectan a la calidad de la reconstrucción y al rate obtenido, sin embargo, variaciones en δ sólo afectan a la calidad de reconstrucción y no a la tasa de bits, ya que este parámetro se usa sólo en el decodificador. Elegir una combinación óptima de estos tres parámetros al codificar una determinada imagen es difícil y más aún si queremos obtener un estimador de éstos que obtenga resultados similares al óptimo para cualquier imagen. La principal motivación de este trabajo es estudiar y analizar el papel que tiene el dead zone y el punto de reconstrucción en el rendimiento R/D de codificadores de imagen basados en wavelets y mejorados perceptualmente.

Otros trabajos en la literatura se han propuesto y analizado el diseño y el rendimiento de diferentes cuantizadores uniformes. En [1] los autores compararon el rendimiento de los esquemas USQ, USDZQ y UTCQ con codificadores wavelet sin codificación perceptual. Sus resultados muestran que aunque los errores de reconstrucción fueron menores con el UTCQ, cuando se combina la cuantización uniforme con codificadores entrópicos, el USDZQ fue la mejor opción habiendo seleccionado con cuidado el DZS. Los resultados muestran que el USDZQ puede reducir efectivamente la entrada al codificador entrópico mejorando por tanto la relación R/D. Por tanto, los autores mostraron que un USDZQ parametrizado, es decir, un UVDZQ es apropiado para sistemas de compresión de imagen basados en transformadas wavelet o DCT.

Está muy extendido en la literatura usar el centro del intervalo de cuantización para ubicar δ [2], o usar el centroide de la distribución de coeficientes en cada intervalo de cuantización. Sin embargo algunos autores recomiendan otras ubicaciones de δ para cuantizadores basados en la transformada DCT [3].

Algunos trabajos analizan específicamente la importancia de estos parámetros (DCZ y δ). En el estándar H.264/AVC se propone un parámetro de redondeo f que controla la ubicación del punto de reconstrucción dentro del intervalo de cuantización, especificando $f=\Delta/3$ cuando se usa codificación intra y $f=\Delta/6$ cuando se usa codificación inter. En [4] los autores aplican un esquema de cuantización con dead zone variable para el H.264/AVC utilizando un parámetro de desplazamiento (offset) adicional para mejorar simultáneamente el tamaño del dead zone y la ubicación del punto de reconstrucción. De esta forma, argumentan que δ se ajusta mejor a la distribución de coeficientes dentro de los intervalos de cuantización.

En [5] se realizaron estudios analíticos para obtener un DZS óptimo para un rango de tasas de bits de hasta 1bpp. Los autores propusieron un algoritmo para obtener el DZS y Δ óptimos. Un cuantizador uniforme con esos óptimos minimiza el error cuadrático medio de la fuente cuantizada. Usaron una GGD (Generalized Gaussian Distribution) para probar el algoritmo con diferentes tipos de distribuciones de coeficientes, como gaussiana, laplaciana y otras con colas más extensas. En todos los casos el autor mantiene sin embargo δ en el centro del intervalo de cuantización.

También Marcellin et al. en [2,6] mostraron la influencia del DZS en el rendimiento R/D del codificador JPEG2000. También usaron una GGD con distribuciones gaussianas, laplaciana y otras para cubrir la variabilidad observada en las distribuciones de coeficientes wavelet para las imágenes comúnmente usadas. Los autores proponen un DZS de 1.5Δ , el cual puede proporcionar un ligero descenso en el MSE generando sin embargo una mejor percepción de la imagen reconstruida en zonas con poca textura. Como no hay un δ óptimo para todas las imágenes, el estándar JPEG2000 permite la elección del valor de δ entre 0 y 1, usando $\delta=1/2$ como valor recomendado para la mayoría de las imágenes.

El rendimiento del codificador puede aumentarse utilizando cuantizadores con dead zone y ajustando su

tamaño. En [7] el autor hizo un experimento con una única imagen y un codificador de imagen basado en wavelets para determinar cuál era el DZS que obtenía mejor rendimiento. Midiendo la ganancia en calidad al reemplazar un USQ por un USDZQ en el codificador wavelet. El estudio midió el rendimiento R/D utilizando el PSNR como métrica de distorsión. Para dicha imagen obtuvieron un DZS óptimo de 1.9Δ que proporcionaba un incremento de calidad de 0,5 db.

En los estudios mencionados se han utilizado diferentes cuantizadores uniformes, remarcando la influencia de ambos parámetros (DZS y δ) en el rendimiento R/D de codificadores wavelet. Como se observa, estos estudios se hicieron con condiciones heterogéneas que proporcionan diferentes propuestas de cuál es el DZS óptimo, el punto de reconstrucción sugerido se sitúa en el centro del intervalo y no consideran ninguna aproximación de codificación perceptual.

En este trabajo realizaremos un extenso estudio para determinar cómo afecta el tamaño del dead zone DZS y la ubicación del punto de reconstrucción δ , en el rendimiento R/D de codificadores wavelet que han sido mejorados perceptualmente con la inclusión de la curva de sensibilidad al contraste, Contrasts Sensitivity Function (CSF) del sistema visual humano.

Cuando se aplican técnicas de codificación perceptual como la CSF los coeficientes transformados se ponderan conforme a la importancia perceptual de la información que aportan. Se puede incluir la CSF en los codificadores de diversas formas como se expone en [8] pero la idea fundamental es que aquellos coeficientes localizados en las frecuencias espaciales para las que el sistema visual humano es más sensible sean elevados o ponderados en mayor amplitud que otros menos importantes perceptualmente.

La aplicación de la CSF modifica la distribución de los coeficientes wavelets proporcionando uniformidad perceptual, de forma que cuando posteriormente se aplique un cuantizador uniforme, éste afecte perceptualmente a todos los coeficientes en igual medida. Para maximizar el rendimiento R/D para un amplio rango de tasas de bits es importante elegir la relación óptima entre del DZS y Δ , conjuntamente con el mejor valor de δ .

Para determinar la magnitud de mejora del rendimiento R/D usaremos un UVDZQ en un codificador wavelet que incluye la CSF midiendo el rendimiento en términos de la métrica perceptual PSNR-HVS puesto que como ciertos estudios sugieren [9,10,11] las comparaciones de rendimiento de codificadores que usen técnicas de codificación perceptual deben ser hechas con métricas objetivas de valoración perceptual de la calidad (perceptual quality assessment metrics). Cubriremos un amplio rango de tasas de bit, hasta 3 bpp, es decir, desde muy baja calidad hasta niveles de calidad perceptual sin pérdida (visually lossless), incrementando el valor de Δ .

El resto del artículo está organizado como sigue: en la sección II se repasa brevemente la formulación de los diferentes esquemas de cuantización y cómo se pueden relacionar entre sí. En la sección III se expone la metodología utilizada para realizar el trabajo. En la

sección IV presentamos los resultados de las comparaciones y en la sección V abordamos las conclusiones y líneas futuras.

II. ESQUEMAS DE CUANTIZACIÓN

En esta sección repasamos brevemente la formulación para los cuantizadores USQ, USDZQ y UVDZQ y cómo el UVDZQ puede ser considerado un cuantizador uniforme universal al ser posible funcionar como un USQ o un USDZQ ajustando apropiadamente los parámetros de cuantización.

Cualquier cuantizador puede ser descompuesto en dos etapas distintas, referidas como la etapa de clasificación (o cuantización directa) y la etapa de reconstrucción (o cuantización inversa). Las ecuaciones (1) y (2) corresponden a la cuantización directa e inversa para un USQ. Las ecuaciones (3) y (4) representan esas etapas para un USDZQ y finalmente las ecuaciones (5) y (6) corresponden a un UVDZQ.

$$C' = \text{sign}(C) \left\lfloor \frac{|C|}{\Delta} + \frac{1}{2} \right\rfloor \quad (1)$$

$$\hat{C} = \Delta C' \quad (2)$$

$$C' = \text{sign}(C) \left\lfloor \frac{|C|}{\Delta} \right\rfloor \quad (3)$$

$$\hat{C} = \text{sign}(C')(|C'| + \delta)\Delta \quad (4)$$

$$C' = \begin{cases} \text{sign}(C) \left\lfloor \frac{|C| + \xi\Delta}{\Delta} \right\rfloor & \text{if } |C| \geq -\xi\Delta \\ 0 & \text{if } |C| < -\xi\Delta \end{cases} \quad (5)$$

$$\hat{C} = \begin{cases} \text{sign}(C')(|C'| - \xi + \delta)\Delta & \text{if } C' \neq 0 \\ 0 & \text{if } C' = 0 \end{cases} \quad (6)$$

Donde C es el coeficiente transformado antes de la cuantización. C' es el coeficiente cuantizado justo tras la etapa directa y \hat{C} es el valor reconstruido después de la etapa de cuantización inversa. El USQ recupera el valor del coeficiente en el centro del intervalo. La constante δ , usada en los otros esquemas, establece la ubicación del punto de recuperación. Los valores permitidos para δ están en el rango $[0, 1]$. La constante ξ define el tamaño del dead zone, permitiéndose valores en el rango $(-\infty, 1]$, y finalmente el paso de cuantización Δ determina la cantidad de cuantización y por tanto el nivel de compresión.

El parámetro ξ tal que ($\xi \leq 1$), determina el tamaño del dead zone, el DZS, en un cuantizador UVDZQ. Dependiendo de su valor el dead zone se fija de la siguiente manera:

- $\xi < 0$ incrementa el tamaño típico de un USDZQ, es decir, $DZS > 2\Delta$
- $\xi = 0$ produce un dead zone exactamente de $DZS = 2\Delta$, siendo Δ el primer punto de decisión o umbral de cuantización, es decir d_i en Fig.1.
- $0 < \xi < 1$ reduce el tamaño del dead zone por debajo del típico, es decir, $DZS < 2\Delta$, donde un valor de $\xi = 0.5$ corresponde con un USQ con $DZS = 1\Delta$.
- Conforme ξ se aproxima a 1 el DZS se reduce hasta 0, por tanto cuando $\xi=1$ entonces $DZS=0$.

Para poder ajustar un UVDZQ que funcione como un USQ tenemos que usar $\xi = 0.5$ y $\delta=0.5$, y para que funcione como un USDZQ con el punto de reconstrucción en el centro del intervalo de cuantización tenemos que usar $\xi = 0.0$ y $\delta=0.5$.

III. MÉTODOS

Como mencionábamos, el objetivo de este estudio es analizar cómo afecta el tamaño del dead zone, DZS, y la ubicación del punto de reconstrucción δ , al rendimiento R/D de codificadores wavelet con mejoras perceptuales, concretamente la inclusión de la CSF.

Por tanto en este estudio utilizamos un codificador de imagen descrito en [12] y una métrica perceptual o QAM, en concreto la PSNR-HVS, para medir la calidad perceptual de cada uno de los esquemas de cuantización [13] por medio de la implementación en C++ de dicha métrica hecha en la herramienta Video Quality Measurement Tool (VQMT) [14]. El codificador utiliza una matriz de pesos optimizada que implementa uno de los modelos más utilizados de la CSF [15].

Las variaciones en estos dos parámetros, DZS y δ , producen diferentes rendimientos o curvas R/D. Analizaremos el rendimiento R/D desde dos perspectivas. La primera desde la ganancia de calidad perceptual, es decir, comparamos la calidad para la misma tasa de bits, y en la segunda comparamos las tasa de bits para un mismo nivel de calidad en la imagen reconstruida.

Para proporcionar un valor de ahorro de tasa de bits y ganancia en calidad usamos el método Bjontegaard [16] para representar los resultados como porcentajes de ganancia para distintos rangos de tasa de bits y distintos intervalos de calidad. Adicionalmente proporcionaremos valores absolutos de la tasa de bits expresados en bits por pixel (bpp).

Hemos definido un conjunto representativo de imágenes de entrenamiento (768x512) compuesto por las 23 imágenes del Kodak Set. Para cada imagen del conjunto de entrenamiento obtendremos el par de parámetros ξ y δ que maximizan el área de la curva R/D. Para hacer esto creamos un espacio 2D de valores (ξ, δ) para los rangos que se muestran a continuación. Para cada combinación de estos parámetros codificamos y decodificamos la imagen en valores crecientes de stepsize, Δ . Medimos la calidad en PSNR-HVS dBs y la tasa de bits en bpp.

- $-0.250 \leq \xi \leq 1$ en saltos de 0.010 ξ obteniendo 126 valores diferentes. Este rango produce DZS que varían desde 2.5Δ a 0 en saltos de -0.02Δ
- $0 \leq \delta \leq 1$ usando saltos de 0.1 δ para obtener 11 posiciones diferentes variando de izquierda a derecha en el intervalo de cuantización.

Se han calculado un total de 1.386 permutaciones (ξ, δ) . Para cada una de ellas hemos creado una curva R/D al usar 13 valores de Adiferentes, distribuidos uniformemente en el rango de tasa de bits. Esto produce un total de 18.018 imágenes comprimidas para cada una de las 23 imágenes del conjunto de entrenamiento.

La Fig. 2 muestra el comportamiento R/D para la imagen '01' del conjunto de entrenamiento. Cada curva corresponde con uno de los valores de ξ dentro del rango

de evaluación, pero sólo la curva con mejor rendimiento de δ se ha dibujado. La curva con mejor rendimiento (ξ, δ) se ha remarcado con una línea más gruesa. Se ha elegido un rango tan grande de valores de ξ para evitar máximos locales en el cálculo del área de las curvas y para tener una perspectiva general de cómo es el comportamiento R/D de las curvas en el rango.

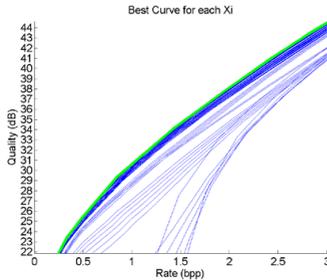


Fig. 2. Mejor curva R/D para cada ξ del conjunto de entrenamiento. En esta figura sólo el mejor δ es mostrado para cada ξ

TABLA I

RANGO DE TASAS DE BITS BAJO ESTUDIO

Rate Ranges	Rate (bpp)		Quality (dB)	
	low	high	low	high
ALL	0.0	3.0	25.0	50.5
L	0.0	0.5	25.0	32.1
M	0.5	1.0	32.1	37.6
H	1.0	1.5	37.6	41.7
VH	1.5	3.0	41.7	50.5

TABLA II

ξ Y δ ÓPTIMOS PARA EL CONJUNTO DE ENTRENAMIENTO

Image	Optimum ξ	Optimum δ	DZS
1	0.460	0.4	1.08 Δ
2	0.400	0.4	1.20 Δ
3	0.300	0.5	1.40 Δ
4	0.450	0.4	1.10 Δ
5	0.450	0.5	1.10 Δ
6	0.390	0.5	1.22 Δ
7	0.350	0.5	1.30 Δ
8	0.390	0.5	1.22 Δ
9	0.380	0.4	1.24 Δ
10	0.400	0.4	1.20 Δ
11	0.380	0.5	1.24 Δ
12	0.450	0.4	1.10 Δ
13	0.450	0.4	1.10 Δ
14	0.490	0.4	1.02 Δ
15	0.380	0.4	1.24 Δ
16	0.370	0.5	1.26 Δ
17	0.400	0.4	1.20 Δ
18	0.450	0.4	1.10 Δ
19	0.350	0.5	1.30 Δ
20	0.350	0.5	1.30 Δ
21	0.380	0.5	1.24 Δ
22	0.460	0.4	1.08 Δ
23	0.380	0.4	1.24 Δ
Min(ξ, δ)	0.300	0.40	1.40 Δ
Mean(ξ, δ)	0.403	0.44	1.19 Δ
Max(ξ, δ)	0.490	0.50	1.02 Δ

Medimos el área de cada una de estas curvas respecto al eje de abscisas pudiendo, por tanto, determinar que par de valores (ξ, δ) obtienen la mayor área. La curva con mayor área es la que tiene el mejor rendimiento R/D dentro del rango estudiado. Dicho rango va desde 0 bpp hasta 3.0 bpp. La calidad, expresada en PNSR-HVS dBs varía para todo el conjunto de entrenamiento desde 17.3 dBs hasta 57.7 dBs, es decir, desde muy mala calidad respecto al original hasta visualmente imperceptible. El rango dinámico de calidades para cada imagen varía dependiendo de su contenido.

Para cada imagen escogemos los parámetros (ξ, δ) con mejor rendimiento, los que nos proporcionan la curva R/D óptima. Llamamos a esa curva C_{OPT} y la comparamos con las curvas correspondientes a los parámetros para USQ y USDZQ, llamadas C_{USQ} y C_{USDZQ} con ($\xi=0.5, \delta=0.5$) y ($\xi=0.0, \delta=0.5$) respectivamente. En esta comparación medimos la ganancia en tasa de bits y el método para C_{OPT} respecto a C_{USQ} y C_{USDZQ} utilizando el método Bjontegaard.

Analizamos estas diferencias para todo el rango de tasa de bits y también para diferentes subrangos que etiquetamos como bajos, medios, altos y muy altos. La tabla I muestra estos rangos con una aproximación de sus correspondientes intervalos de calidad. Estos rangos son ALL para todo el rango, L para tasas bajas, M para medias, H para altas y VH para tasa de bit muy altas. Dependiendo del contenido de la imagen, la tasa de compresión es diferente para cada imagen, por tanto, las columnas para calidad en la tabla I muestran valores promedios de todas las imágenes para los límites de calidad alto y bajo.

Se calcula la ganancia en calidad o tasa de bits para C_{OPT} respecto a C_{USQ} y C_{USDZQ} en cada uno de estos rangos.

De esta forma tenemos la ganancia máxima para cada imagen y rango de tasa. Las curvas C_{OPT} , C_{USQ} y C_{USDZQ} se han estimado con un proceso automático de ajuste de curvas (curve fitting) que busca el mejor ajuste usando modelos polinómicos y racionales proporcionados por la herramienta Matlab Curve Fitting Toolbox. Una vez tenemos los parámetros que modelan la curva podemos obtener las diferencias absolutas en tasa de bits y calidad para cualquier rango. Se obtienen también los valores absolutos medios de ahorro en tasa de bits.

Tras analizar el conjunto de entrenamiento usaremos un nuevo conjunto de test compuesto por las siguientes imágenes: (512 x 512): Balloon, Barbara, Boat, Goldhill, Horse, Lena, Mandrill y Zelda; (2048 x 2560): Bike, Cafe y Woman. Como el par (ξ, δ) óptimo es diferente para cada imagen, obtenemos un valor representativo, es decir un par (ξ, δ) subóptimo basado en los parámetros del conjunto de entrenamiento.

IV. RESULTADOS

La tabla II muestra los parámetros (ξ, δ) óptimos para cada una de las imágenes del conjunto de entrenamiento. Como podemos ver, el parámetro δ , que fija la ubicación del punto de recuperación en el intervalo, permanece casi constante cerca del centro de éste con un valor medio de $\delta = 0.44$. Por otro lado, el valor óptimo de ξ varía en un rango desde 0.30 a 0.49 unidades que

corresponde a dead zones de 1.02A a 1.40A respectivamente.

Como hemos mencionado anteriormente, valores menores de ξ producen mayores DZS, por tanto en la tabla II podemos ver que para el menor ξ (0.30) obtenemos el DZ más ancho, 1.40A. Para el todo el conjunto de entrenamiento obtenemos un DZS medio de 1.19A que corresponde con un dead zone un 19% más ancho que el de un cuantizador USQ y un 40.5% más estrecho que el de un USDZQ. Por tanto, una primera conclusión es que puesto que el valor óptimo de DZS es más próximo al de un USQ que al de un USDZQ y por tanto un USQ proporcionará mejor rendimiento R/D que un USDZQ.

TABLA III

RESULTADOS PROMEDIO PARA EL CONJUNTO DE ENTRENAMIENTO

Rate Ranges	Quality Gain (dB)		% of Rate Saving	
	USQ	UDZQ	USQ	UDZQ
ALL	0.15	0.36	1.48	3.31
L	0.12	0.25	2.01	4.12
M	0.12	0.32	1.55	3.96
H	0.17	0.38	1.74	3.76
VH	0.17	0.40	1.38	2.96

TABLA IV

GANANCIAS MÁXIMAS PARA EL CONJUNTO DE ENTRENAMIENTO

Rate Ranges	Quality Gain (dB)		% of Rate Saving	
	USQ	UDZQ	USQ	UDZQ
ALL	0.30	0.53	3.23	4.55
L	0.26	0.41	5.30	6.64
M	0.25	0.60	3.53	7.67
H	0.34	0.60	4.54	5.92
VH	0.35	0.59	3.07	4.27

TABLA V

RESULTADOS PROMEDIO PARA EL CONJUNTO DE TEST.
 PSNR-HVS dBs PARA CALIDAD Y % DE AHORRO EN TASA DE BITS

Rate Ranges	Quality Gain (dB)		% of Rate Saving	
	USQ	USDZQ	USQ	USDZQ
512 x 512				
ALL	0.22	0.34	2.17	3.28
L	0.17	0.17	2.90	3.02
M	0.16	0.32	2.17	4.02
H	0.23	0.36	2.52	3.69
VH	0.25	0.38	2.02	3.00
2048 x 2560				
ALL	0.16	0.41	1.43	3.62
L	0.08	0.25	1.29	3.97
M	0.11	0.40	1.30	4.53
H	0.15	0.40	1.48	3.75
VH	0.19	0.45	1.45	3.36

Como los resultados dependen del contenido de la imagen, mostramos los valores medios y máximos para el conjunto de entrenamiento. En la Tabla III se muestran los valores medios de ganancia en tasa de bits y calidad de las imágenes, y en la tabla IV se muestran las correspondientes ganancias máximas. Como se ve, con el UVDZQ optimizado se obtienen ahorros de tasa de bits de hasta 6.64% en el rango L y hasta un 7.67% en el rango M. Respecto a las ganancias en calidad, se

obtienen incrementos de calidad de 0.41dBs y 0.60dBs en los rangos L y M respectivamente.

La Fig. 3 muestra la comparación de las curvas R/D para la imagen '01' del conjunto de entrenamiento, considerando un rango de tasa de bits desde 0 hasta 1.5 bpp (rangos L,M y H). Como se muestra, el ahorro de tasa se mantiene para todo el rango. La Fig. 4 se centra sólo en las curvas R/D en el rango H, en este caso para la imagen '14' del conjunto de entrenamiento. Se obtiene una ganancia de calidad de 0.60dB al usar el óptimo respecto a USDZQ. Esto corresponde con un ahorro en tasa de bits de un 5.9%.

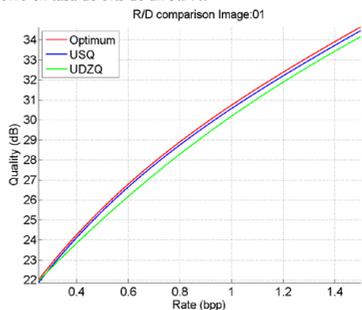


Fig. 3. Comparación R/D entre los parámetros óptimos, USQ y USDZQ para la imagen '01' en los rangos de tasa de bits L,M y H

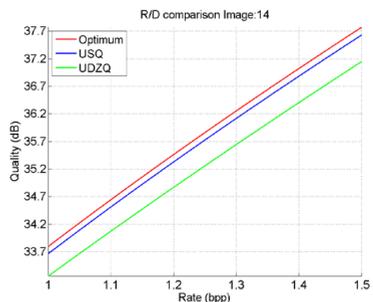


Fig. 4. Comparación R/D entre los parámetros óptimos, USQ y USDZQ para la imagen '14' en el rango de tasa de bits H

Ahora procedemos a evaluar el rendimiento R/D del UVDZQ optimizado con las imágenes del conjunto de test. Como describimos anteriormente, para ello deberíamos calcular el par de parámetros óptimo (ξ, δ) para el nuevo conjunto. Sin embargo, para poder aplicar en la práctica sin necesidad de repetir el proceso, proponemos una aproximación simple eligiendo el centroide del óptimo DSZ y δ obtenidos del conjunto de entrenamiento (ver Tabla II). El par de parámetros resultante estimado (\overline{DSZ}, δ) será el usado en las imágenes del conjunto de test.

Utilizando esta aproximación simple, calculamos para el conjunto de test las curvas para USQ, USDZQ y el

UVDZQ optimizado con los parámetros (\widehat{DSZ}, δ) . En la tabla V mostramos los valores medios de ganancia en calidad y ahorro de tasa para las imágenes del conjunto de test. Como puede verse se obtienen resultados similares a los obtenidos con el conjunto de entrenamiento. En particular podemos obtener hasta un 4,02% de ahorro de tasa en el rango M para las imágenes a una resolución de 512x512 y hasta un 4,53% de ahorro para el rango M y las imágenes de resolución 2048x2560.

TABLA VI

RESULTADOS ABSOLUTOS PARA ALGUNOS RANGOS DE TASA DEL CONJUNTO DE TEST.
 PSNR-HVS dBS PARA CALIDAD Y % DE AHORRO EN TASA DE BITS

Img. Size	Image	Quality Gain (dB)		% of Rate Saving	
		USQ	USDZQ	USQ	USDZQ
Rate Range: H (1.0 to 1.5 bpp)					
512x512	balloon	0.25	0.31	2.25	2.70
	barbara	0.15	0.36	1.41	3.43
	boat	0.15	0.54	1.36	5.10
	goldhill	0.13	0.47	1.43	5.14
	horse	0.26	0.41	2.59	4.08
	lena	0.32	0.23	4.04	2.97
	mandrill	0.31	0.39	3.30	4.11
	zebra	0.27	0.14	3.79	1.98
Rate Range: M (0.5 to 1.0 bpp)					
2048x2560	bike	0.16	0.28	1.83	3.18
	cafe	0.05	0.54	0.46	5.68
	woman	0.13	0.39	1.61	4.73
Rate Range: L (0.0 to 0.5 bpp)					
2048x2560	bike	0.08	0.18	1.15	2.75
	cafe	0.04	0.29	0.71	4.55
	woman	0.12	0.29	2.01	4.60

En la tabla VI mostramos resultados específicos para diversos rangos de compresión. El UVDZQ optimizado obtiene hasta un 5,14% de ahorro en rate en la imagen 'goldhill' en el rango de rate H y 5,68% para la imagen 'cafe' en el rango M. Además mejora el rendimiento perceptual hasta 0,54 PSNR-HVS dBS para algunas imágenes del conjunto.

V. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo hemos usado un UVDZQ para analizar cómo afectan el ancho del dead zone y la ubicación del punto de reconstrucción al rendimiento R/D de un codificador wavelet mejorado perceptualmente. De este estudio se observa que cada imagen tiene un par de parámetros (ξ, δ) óptimos para los cuales el rendimiento R/D de la imagen se maximiza en un amplio rango de tasa de bits, desde 0 bpp a 3.0 bpp.

Para cuantificar los beneficios por usar el par óptimo de parámetros de cuantización, el rendimiento R/D del UVDZQ optimizado se compara con el obtenido con cuantizadores USQ y USDZQ puesto que son los más comunes en compresión de imagen. Los resultados muestran que se pueden obtener ahorros de tasa de hasta un 4,55% y un 3,23% de media para todo el rango de tasa de bits cuando se compara con el USDZQ y el USQ respectivamente. Sin embargo, como la curva R/D tiene un comportamiento no lineal, el UVDZQ óptimo puede obtener mayores ahorros de rate: 6,64% respecto USDZQ y 5,30% respecto a USQ en el rango de 0 a 0,5

bpp y 7,67% y 3,53% respectivamente en el rango de 0,5 a 1,0 bpp.

El esquema USQ obtiene mejores resultados que el USDZQ porque su ancho de dead zone es más próximo al del óptimo que se ha obtenido en ambos conjuntos de imágenes, entrenamiento y test y para todos los rangos de tasa. En otras palabras, un dead zone de 1Δ proporciona mejores resultados que uno de 2Δ cuando los coeficientes de la transformada se han ponderado con una matriz de elevación optimizada para la CSF. La CSF proporciona a cada coeficiente su peso o importancia perceptual correcta, de forma que un simple cuantizador uniforme aplicado con posterioridad afecte perceptualmente en la misma medida a todos ellos.

Con los resultados obtenidos podemos asegurar que el uso de un cuantizador UVDZQ optimizado puede en general mejorar el rendimiento R/D de cuantizadores wavelet optimizados perceptualmente. Proponemos una aproximación al par de parámetros óptimo eligiendo el centroide de los DSZ y los δ obtenidos del conjunto de imágenes de entrenamiento, como par sub-óptimo para ser usado en otras imágenes.

Los resultados muestran que incluso con este par sub-óptimo se obtienen buenos resultados para una variedad de imágenes. El UVDZQ obtiene con él hasta 5,68% y 4,73% de ahorro de rate respecto el USDZQ para las imágenes 'cafe' y 'woman' respectivamente en el rango entre 0,5 y 1,0 bpp o hasta 4,5% y 4,6% respectivamente en el rango de 0 a 0,5 bpp.

Aunque se debe realizar más investigación para obtener un estimador del par de parámetros de cuantización óptimo para cualquier imagen basándose en propiedades estadísticas de la imagen o de los coeficientes u otros parámetros como por ejemplo la entropía, los resultados con esta simple aproximación son suficientemente buenos y cercanos al óptimo.

I. AGRADECIMIENTOS

El presente trabajo ha sido financiado por el Ministerio de Economía y Competitividad mediante el proyecto TIN2015-66972-C5-4-R cofinanciado con fondos MINECO/FEDER UE.

II. REFERENCIAS

- [1] Jinhua Yu, "Advantages of uniform scalar dead-zone quantization in image coding system," Communications, Circuits and Systems, 2004. ICCAS 2004. 2004 Int. Conference on, 2004, pp. 805-808 Vol.2.
- [2] Michael W. Marcellin, Margaret A. Lepley, Ali Bilgin, Thomas J. Flohr, Troy T. Chinen, and James H. Kasner. "An overview of quantization in JPEG2000". Signal Processing: Image Communication, 17:73-84, 2002
- [3] S. Notebaert, J. De Cock, K. Vermeirsch, P. Lambert and R. Van de Walle, "Leveraging the quantization offset for improved requantization transcoding of H.264/AVC video." Picture Coding Symposium, 2009. PCS 2009, Chicago, IL, 2009, pp. 1-4
- [4] T. Wedi and S. Wittmann, "Quantization offsets for video coding," 2005 IEEE Int. Symposium on Circuits and Systems, 2005, 324-327 Vol. 1.
- [5] Bo Tao, "On optimal entropy-constrained deadzone quantization," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 11, no. 4, pp. 560-563, Apr 2001.
- [6] Michael W Marcellin. "JPEG2000: image compression fundamentals, standards, and practice", volume 1. springer, 2002
- [7] Jacob Ström. "Dead zone quantization in wavelet image compression" - mini project in ece 253a, 1996.

- [8] Marcus J. Nadenau, Julien Reichel, and Murat Kunt. "Wavelet-based color image compression: Exploiting the contrast sensitivity function." *IEEE Transactions on image processing*, 12(1), 2003.
- [9] M. Martínez-Rach, O. Lopez, P. Piñol, J. Oliver, and M. Malumbres. "A study of objective quality assessment metrics for video codec design and evaluation," in *Eight IEEE International Symposium on Multimedia*, vol. 1, ISBN 0-7695-2746-9. San Diego, California: IEEE Computer Society, Dec 2006, pp. 517–524
- [10] H. R. Sheikh, M. F. Sabir, and A. C. Bovik, "A statistical evaluation of recent full reference image quality assessment algorithms," *IEEE Trans. on Image Processing*, vol. 15, no. 11, pp. 3440–3451, 2006.
- [11] Z. Wang, A. Bovik, H. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, 2004.
- [12]
- [13] Martínez-Rach, Miguel Onofre. "Perceptual image coding for wavelet based encoders". PhD thesis, Universidad Miguel Hernández. <http://hdl.handle.net/11000/1764> December 2014.
- [14] Egiazarian, K., Astola, J., Ponomarenko, N., Lukin, V., Battisti, F., & Carli, M. (2006, January). New full-reference quality metrics based on HVS. In *CD-ROM proceedings of the second international workshop on video processing and quality metrics*, Scottsdale, USA (Vol. 4).
- [15] Multimedia Video Processing Group, "VQMT: Video Quality Measurement Tool", <http://mmspg.epfl.ch/vqmt>
- [16] J. Mannos and D. Sakrison, "The effects of a visual fidelity criterion of the encoding of images," *IEEE Transactions on Information Theory*, vol. 20, no. 4, pp. 525 – 536, Jul. 1974.
- [17] G. Bjontegaard. "Calculation of average psnr differences between rdcurves (vceg-m33)". Technical report, VCEG Meeting (ITU-T SG16 Q.6), Austin, Texas, USA, April 2001.

Uso de aceleradores remotos para mejorar las prestaciones de la librería FFTW

Santiago Mislata¹ y Federico Silla¹

Resumen—Los aceleradores hardware han impuesto un cambio en la computación de altas prestaciones. Su uso ha producido una mejora en las prestaciones de los centros de datos. Por esta razón, se ha decidido portar software de varias aplicaciones relacionadas con diferentes campos científicos, como biología o química, a aceleradores hardware como GPUs (unidades de procesamiento gráfico). Sin embargo, no todas las aplicaciones pueden portarse a aceleradores, bien por la dificultad que conlleva este proceso, bien porque el coste de desarrollar dicha portabilidad no compensa esta mejora de prestaciones. En este trabajo se presenta un middleware que hace uso de aceleradores remotos para llevar a cabo las partes computacionalmente pesadas de librerías científicas, que inicialmente iban a ejecutarse localmente en la CPU. El movimiento de cómputo a aceleradores remotos se hace de una forma totalmente transparente a la aplicación, sin necesidad de modificar el código fuente. La implementación de este middleware se centra en la librería FFTW [1]. Se presenta una detallada evaluación de prestaciones, considerando distintas configuraciones de nodos y redes InfiniBand. Como aceleradores, se usan las GPUs K40 de NVIDIA. Los resultados muestran que la librería estudiada experimenta una aceleración superior a 25x, a pesar de tener que enviar los datos hacia y desde el servidor remoto con el acelerador.

Palabras clave—InfiniBand, GPUs, CUDA, FFTW.

I. INTRODUCCIÓN

LOS aceleradores hardware, como las GPUs (unidades de procesamiento gráfico) han llamado la atención de los investigadores. Gracias a su gran potencia de cálculo han conseguido que los programadores empiecen a desarrollar aplicaciones para ser ejecutadas en ellos además de portar las ya existentes. Sin embargo, hay algunas razones por las cuales no todas las aplicaciones pueden portarse a aceleradores hardware.

En primer lugar, puede ser debido a que hay aplicaciones que fueron desarrolladas hace mucho tiempo, y realizar su portabilidad implica un proceso complejo. Segundo, la propia naturaleza del problema no tiene un alto grado de paralelismo. Por ello, portar la aplicación puede resultar muy costoso para una baja reducción en el tiempo de ejecución, haciendo que este proceso no sea rentable.

Muchas de estas aplicaciones pertenecen a distintos campos como biología o química. Estas hacen uso de librerías matemáticas para llevar a cabo su parte de cómputo, principal responsable de su tiempo de ejecución. Por ejemplo, diversas aplicaciones usadas para mecánica molecular no hacen uso de GPUs para su parte de cómputo, como TINKER Molecular Modeling Software [2], COSMOS Software [3], YA-

¹Dpto. de Informática de Sistemas y Computadores, Univ. Politécnica de Valencia, e-mails: sanmisva@gap.upv.es, fsilla@disca.upv.es

SARA [4] o RedMDSStream [5].

En conclusión, la existencia de aplicaciones no portadas a aceleradores hardware se debe a que el coste de realizar la portabilidad no suele compensar la mejora en las prestaciones. El hecho de que la aplicación no pueda ejecutarse en un acelerador no implica que alguna de sus partes no pueda hacerlo, especialmente aquellas que implican la parte de cómputo. Por ello, en este trabajo se presenta un middleware que permite que las partes de una aplicación computacionalmente pesadas de CPU se trasladen a aceleradores remotos. Este traslado se realizará a nivel de cluster, y los aceleradores se ubicarán en nodos en los que no se ejecuten las aplicaciones. Este proceso se desarrollará de forma totalmente transparente a las aplicaciones, sin tener que modificar su código.

Dado que la aplicación no está siendo reprogramada, la evaluación de prestaciones de este middleware mostrará peores resultados en comparación a la aplicación específicamente programada para el acelerador correspondiente. Por tanto, es importante establecer un compromiso entre la pérdida de prestaciones y el coste de portar la aplicación para decidir qué hacer con la aplicación: realizar su portabilidad o ejecutarla con el nuevo middleware.

La principal contribución de este artículo es doble: (1) usando una simple combinación de técnicas, se presenta un nuevo middleware centrado en optimizaciones sobre la librería FFTW para mejorar aún más sus prestaciones; (2) se muestra una amplia comparativa para medir las prestaciones conseguidas usando este middleware, sin tener en cuenta aplicaciones específicas, obteniendo una aceleración de hasta 25x, en función del tamaño del problema.

Con la intención de presentar adecuadamente nuestra propuesta, el resto del artículo se organiza de la siguiente manera. En la Sección II se presenta la idea general de este nuevo middleware en detalle. La Sección III muestra los principales detalles de implementación del middleware, centrándose en la librería FFTW. Las prestaciones del nuevo middleware son evaluadas en la Sección IV haciendo uso de ejecuciones reales considerando distintas configuraciones de sistemas. Posteriormente, la Sección V analiza el impacto en las prestaciones del nuevo middleware al usar una red de interconexión de bajas prestaciones y una capa de comunicaciones ineficiente. La Sección VI discute algunos aspectos adicionales de la propuesta. Después, el middleware se compara frente a otras propuestas en la Sección VII. Finalmente, la Sección VIII presenta las principales conclusiones de nuestro trabajo.

II. TRASLADANDO CÓMPUTO DE CPU A ACCELERADORES REMOTOS

El concepto que implementa nuestro middleware es sencillo. En pocas palabras, mueve a aceleradores remotos las partes de una aplicación que hacen un uso intensivo de la CPU. Esto se hace de forma transparente a la aplicación, sin necesidad de modificar su código fuente.

El nuevo middleware mueve las partes más pesadas, computacionalmente hablando, de librerías matemáticas, inicialmente escritas para ser ejecutadas localmente en una CPU, a aceleradores situados en otros nodos del cluster. Una vez la parte de cómputo ha finalizado, los resultados se devuelven a la aplicación. De esta forma, la aplicación no es consciente de si los cálculos se han hecho local o remotamente.

Uno de los objetivos de diseño del nuevo middleware es que se integre con las aplicaciones automáticamente y de forma totalmente transparente. Para ello, el framework reemplaza la librería matemática por otra que contiene un conjunto de wrappers. Esta nueva librería será llamada siempre que una de las funciones originales se ejecute, y sus argumentos serán los mismos que los de la original. La librería mandará los argumentos al nodo del cluster con el acelerador, donde realmente tendrá lugar la fase de cómputo. La interfaz del nuevo middleware es la misma que la de las librerías matemáticas. Esto se consigue creando un wrapper para cada una de las funciones en las librerías.

El nuevo middleware se organiza siguiendo una arquitectura distribuida cliente-servidor, como se muestra en la Figura 1. La aplicación y la parte cliente se ejecutan ambas en la misma máquina. La parte cliente se conecta automáticamente con la aplicación cada vez que se hace una llamada a una de las funciones de la librería matemática reemplazada. En este punto, el wrapper asociado en la parte cliente del middleware, el cual tiene el mismo nombre y parámetros que la función invocada, se ejecuta. Este wrapper envía el comando apropiado y los datos de entrada al servidor remoto, que a su vez se encarga de interpretar la petición y ejecutar los cálculos en el acelerador. Una vez el acelerador ha completado la ejecución de la función requerida, se envían de vuelta los resultados a la parte cliente, entregándoseles a la aplicación demandante.

El conjunto de wrappers en la parte cliente están contenidos en un archivo de librería dinámica. Este archivo deberá ser copiado a la máquina que ejecuta la aplicación para poder hacer uso del middleware. A continuación, hay que establecer algunas variables de entorno para saber dónde se aloja esta librería, además de la dirección IP del nodo servidor. La variable de entorno `LD_LIBRARY_PATH` deberá fijarse de acuerdo a la ubicación final del archivo del middleware del cliente. Tras esto, la variable de entorno `RCPU_SERVER` deberá inicializarse con la dirección IP del servidor de nuestro middleware, así como del número de puerto usado por él para la recepción de peticiones. La sintaxis para inicializar la variable es

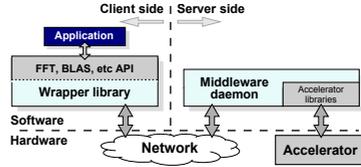


Fig. 1. Arquitectura del middleware propuesto.

"`RCPU_SERVER=dirección_IP@puerto`". En el nodo remoto, los binarios de la parte servidor del middleware se ejecutarán. Este servidor está configurado como un demonio y espera peticiones de cómputo en un número de puerto configurable.

La versión actual de nuestro middleware solo da soporte a una arquitectura con un único cliente y GPUs de NVIDIA como aceleradores remotos. Además, se da soporte completo para las librerías FFTW y BLAS [6]. Para futuras versiones, se considera la implementación de un servidor procesando varios clientes, el uso de aceleradores Intel Xeon Phi así como soporte a otras librerías como LAPACK [7]. Finalmente, indicar que el nuevo middleware solo se encuentra disponible para Linux.

III. IMPLEMENTACIÓN

La librería FFTW es un paquete relativamente pequeño con alrededor de 40 funciones [8]. Se encarga de computar la transformada discreta de Fourier (DFT) de datos complejos y reales y la transformada discreta de Hartley (DHT) de datos reales. El trabajo del presente artículo se centra en la DFT.

Respecto a FFTW, cabe destacar que esta librería puede usar diferentes algoritmos para computar distintas transformadas DFT. Este algoritmo se selecciona cuidadosamente dependiendo del hardware en el que va a ejecutarse, para así maximizar las prestaciones de la DFT. De esta manera, el proceso de cómputo de una FFTW se hace en dos fases: primero, el planificador de la FFTW analiza la forma más rápida de ejecutar la DFT requerida en el hardware subyacente considerando el tamaño de los datos de entrada, y genera una estructura opaca llamada *plan* con el resultado de este análisis. Tras esto, el plan se ejecuta para aplicar la DFT a los datos de entrada. Como consecuencia de esta división de trabajo, las funciones de la librería FFTW pueden clasificarse en dos categorías: las que crean el plan (representadas por la función `fftw_plan`, aunque existen otras variantes dependiendo del tipo de datos usado) y las que lo ejecutan (representadas de forma genérica por la función `fftw_execute`).

Dividir el trabajo entre crear el plan y ejecutarlo supone un problema importante a la hora de mover la parte de cómputo de la librería FFTW a aceleradores remotos. Básicamente, el objetivo consiste en cómo lidiar eficientemente con las dos funciones involucradas en hacer una transformada DFT (`fftw_plan`

y `fftw_execute`) *sin tener que mover por partida doble los datos de entrada al servidor remoto*. Hay que destacar que no se puede asumir que ambas funciones vayan a llamarse secuencialmente. Una aplicación puede crear varios planes para el mismo conjunto de datos de entrada, y tras ello seleccionar uno de acuerdo a ciertos criterios. También existe la posibilidad de tener varios conjuntos de datos de entrada de diferentes tamaños, y crear los planes para estos conjuntos y ejecutarlos intercaladamente. Por tanto, hay que diseñar cuidadosamente una estrategia para poder trasladar la parte de cómputo de la librería FFTW, teniendo en cuenta estos dos aspectos.

En primer lugar, hay que tener en cuenta que los planes se crean para un hardware específico. Por ello, deben ser creados en el servidor remoto, de acuerdo a sus características de hardware. Segundo, para algunas variantes de la función `fftw_execute`, el plan es su único parámetro. Los punteros de memoria a los datos de entrada y salida para computar la transformada están insertados en el plan. Así, estos punteros deberían almacenarse en algún lugar donde el servidor pueda devolver adecuadamente el resultado de la transformada al cliente.

La estrategia seguida consiste en crear un wrapper para la función `fftw_plan` que se encargará de almacenar sus parámetros de entrada en una estructura de datos interna. Dicha estructura se aloja en la parte cliente dentro de la librería de wrappers, y durante esta fase, no se enviará nada de información al servidor remoto. Como la función original devuelve un plan tras su invocación, el wrapper de `fftw_plan` se encargará de crear un falso plan, inicializado con un identificador interno, que será usado por el wrapper `fftw_execute` para localizar los parámetros almacenados en la estructura interna. De esta manera, al invocar la función `fftw_execute`, ésta utilizará sus parámetros para localizar y enviar los datos de entrada al servidor remoto. Una vez aquí, se seguirá el proceso de computación de DFTs (creación del plan y su posterior ejecución) y tras ello, se enviarán de vuelta los resultados a la parte cliente.

Nótese que esta propuesta es compatible con la creación de múltiples planes. El wrapper `fftw_plan` almacena en la estructura interna los parámetros para todos los falsos planes. Por lo tanto, cuando finalmente se ejecuta uno de ellos, el wrapper `fftw_execute` buscará el plan exacto dentro de la estructura interna y enviará los parámetros adecuados junto con los datos de entrada al servidor remoto. Además, esta estrategia hace un uso eficiente de la memoria de la GPU, almacenando solo los datos de entrada requeridos por la transformada.

Otra estrategia considerada consistía en enviar los datos de entrada al servidor al invocarse la función `fftw_plan`. Así, además de crear el plan, los datos de entrada y los punteros de entrada y salida se almacenan en el servidor. Una vez se crea el plan, se envía de vuelta al cliente para ser usado como argumento de entrada del wrapper `fftw_execute`. En el caso en que varios planes se computasen para los mismos

datos de entrada, solo la primera llamada al wrapper `fftw_plan` debería enviar los datos al servidor. Y además, cuando se llamase a la función `fftw_execute` en la parte cliente, el plan se reenviaría al servidor para computar la transformada.

Aunque factible, esta propuesta presenta dos problemas. El primero está relacionado con la posibilidad de computar varios planes para diferentes conjuntos de datos de entrada con distintos tamaños. En este caso, los datos de entrada, que se deben de almacenar en el servidor, podrían no caber en la memoria de la GPU y entonces deberían moverse de vuelta a la memoria principal del servidor. Esto se traduce en un incremento del tiempo. Una solución podría ser almacenar los datos de entrada en la memoria principal en lugar de hacerlo en la memoria de la GPU. Sin embargo, esta posibilidad nos lleva al segundo problema: si los datos se almacenan en memoria principal, una vez recibida la petición del wrapper `fftw_execute` en el servidor, los datos de entrada deben de moverse desde memoria principal a memoria de GPU, requiriendo para ello una cantidad de tiempo nada despreciable. Por ello, estos dos problemas hacen que esta propuesta sea más complicada y menos escalable que la anterior.

Para este trabajo se ha considerado también una optimización para la aproximación escogida. Esta optimización se basa en el hecho de que para crear el plan solo se necesita el tamaño de los datos de entrada y no el valor exacto de dichos datos. Por ello, es posible mejorar la estrategia seguida. Una vez se ejecuta el wrapper `fftw_execute`, el cliente se encargará de enviar los parámetros del plan escogido al servidor. La creación del plan en el servidor empezará una vez se reciban los parámetros. Simultáneamente, el cliente enviará los datos de entrada. Así, crear el plan y enviar los datos al servidor se convierten en dos tareas solapadas, y por ello, el tiempo de ejecución se verá reducido. Esta optimización requiere que el plan se haya creado antes que la transformada DFT empiece a ejecutarse. Sin embargo, esta comprobación es directa y no generará sobrecarga adicional. Por el contrario, la estructura interna de la parte servidor debe mejorarse haciendo uso de hilos. Esta implementación ha sido incluida en este trabajo, y los resultados presentados en las siguientes secciones seguirán esta aproximación.

Finalmente, hay que tener en consideración como el servidor remoto lidia con el cálculo de la DFT. La implementación presentada en este trabajo tiene un servidor con una GPU NVIDIA, inicializada antes de la llegada de los datos de entrada. A este respecto, después de recibir los datos de entrada, se transfieren a la GPU haciendo uso de copias de memoria a GPU tradicionales. Tras esto, se ejecuta la función apropiada de la librería cuFFT de NVIDIA, usada para computar la DFT. Una vez la transformada se ha completado, los resultados son devueltos desde la memoria de la GPU a la memoria principal del servidor y de ahí al cliente.

IV. EVALUACIÓN DE PRESTACIONES

En esta sección se presenta una completa evaluación de prestaciones del nuevo middleware basada en ejecuciones reales. Con respecto al banco de pruebas usado para los experimentos, el nodo cliente ejecutando la aplicación original será tanto un sistema basado en sistemas Atom como uno basado en sistemas Xeon. La razón de incluir los sistemas Atom en esta evaluación de prestaciones se debe a recientes propuestas que consideran procesadores de bajo consumo, tales como Atoms o ARMs, para ser usados en centros de datos. El X10 MicroBlade server de Supermicro [9] es un ejemplo de esta tendencia.

Las características exactas de los sistemas usados en el estudio son las siguientes. El sistema ATOM es un servidor Supermicro 5018A-TN4 conteniendo un procesador Atom C2750 de 8 núcleos funcionando a 2.4 GHz y con 16 GB de memoria DDR3 a 1600 MHz. Incluye un conector PCI 2.0 x8. El sistema Xeon es un servidor Supermicro 1027GR-TRF que contiene dos procesadores Intel Xeon E5-2620v2 de 6 núcleos (Ivy Bridge) funcionando a 2.1 GHz y con 32 GB de memoria DDR3 a 1600 MHz. Para la parte servidor hemos escogido un servidor Supermicro 1027GR-TRF con 2 procesadores Xeon E5-2620v2 de 6 núcleos funcionando a 2.1 GHz y con 32 GB de memoria DDR3 a 1600 MHz. El servidor también incluye una GPU Tesla K40 de NVIDIA con 12 GB de memoria GDDR5.

Respecto a la infraestructura de red utilizada para conectar cliente y servidor, para los sistemas basados en procesadores Xeon, se han considerado adaptadores de red FDR y EDR de InfiniBand (IB), alcanzando un ancho de banda teórico máximo de 56 y 100 Gbps, respectivamente. Para el sistema basado en el ATOM solo se ha tenido en cuenta InfiniBand QDR (40 Gbps) debido a la falta de un conector PCIe 3.0 en el sistema. Para este caso, el nodo servidor ha sido equipado también con un adaptador InfiniBand QDR, que reemplaza los adaptadores FDR y EDR con intención de ubicar la interfaz de red y la GPU en la misma raíz PCIe (mismo socket). Esto evitará los efectos secundarios de las máquinas NUMA, como teniendo que usar un enlace QPI conectando ambos sockets CPU con objeto de mover datos entre la red y la GPU. Finalmente, se ha hecho uso de la API de los InfiniBand Verbs (incluyendo las comunicaciones basadas en RDMA) en aras de las prestaciones junto con una eficiente arquitectura de comunicaciones segmentada. Esta la capa de comunicaciones implementada divide cada transferencia de datos en pequeños trozos, que concurrentemente avanzan hacia el destino, aumentando así sus prestaciones.

Con respecto a la configuración de software, se ha usado la distribución Linux CentOS release 6.4 junto con el OFED de Mellanox 2.1-2.0.0 (drivers y herramientas administrativas de InfiniBand) y CUDA 6.5 con NVIDIA driver 340.29. Esta misma configuración de software es la usada en los nodos de la parte cliente (excepto por el software de NVIDIA).

La Figura 2 muestra las prestaciones del nuevo

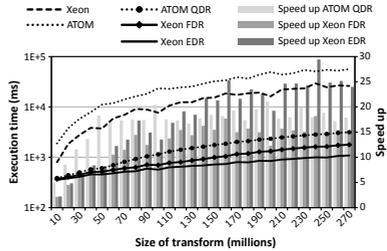


Fig. 2. Prestaciones de la DFT 1D para diferentes configuraciones de sistemas.

middleware al computar la DFT 1D. Los datos de entrada y de salida son de tipo double-complex (16 bytes). También se ha considerado el uso de datos de entrada y salida de tipos reales, aunque los resultados obtenidos son similares. La figura muestra, como referencia, el tiempo de ejecución de la DFT 1D de manera tradicional (usando la CPU local). Las curvas “Xeon” y “ATOM” hacen referencia a las ejecuciones de la FFTW usando los procesadores Xeon y Atom, respectivamente. Además, las curvas “ATOM QDR”, “Xeon FDR”, y “Xeon EDR” hacen referencia a las ejecuciones de la FFTW usando el acelerador remoto desde el sistema Atom con InfiniBand QDR, y desde el sistema Xeon con InfiniBand FDR y EDR, respectivamente. La aceleración con respecto a las ejecuciones locales también se muestra.

La Figura 2 muestra que se pueden conseguir notables reducciones en el tiempo de ejecución al usar el middleware. Por ejemplo, para tamaños de transformada superiores a 70 millones de elementos, el uso del nuevo middleware reporta, en general, una aceleración superior a 10x usando InfiniBand QDR desde el sistema Atom. Esta aceleración se incrementa a al menos a 15x cuando las redes usadas son InfiniBand FDR y EDR desde el sistema Xeon. Específicamente, al usar InfiniBand EDR, la aceleración aumenta hasta 20x o 25x, dependiendo del tamaño de la transformada. En el caso del sistema Atom usando InfiniBand QDR, la aceleración puede incrementarse hasta 15x para tamaños de problema mayores. Esta aceleración es mayor que la del sistema Xeon con InfiniBand FDR debido a la gran diferencia de prestaciones entre el sistema Atom y la GPU K40. La aceleración máxima, como era de esperar, es menor que la provista por la librería cuFFT de NVIDIA (30x-35x) al ser usada localmente sin la intermediación del middleware.

Las Figuras 3(a), 3(b), y 3(c) muestran el desglose del tiempo total de ejecución. “To GPU” es el tiempo invertido para transferir datos desde memoria principal en el cliente a memoria de GPU en el servidor (la capa de comunicaciones optimizada que se está usando no almacena los datos de entrada en la memoria principal del servidor), mientras que “From GPU” es el tiempo invertido en transferir los datos de salida

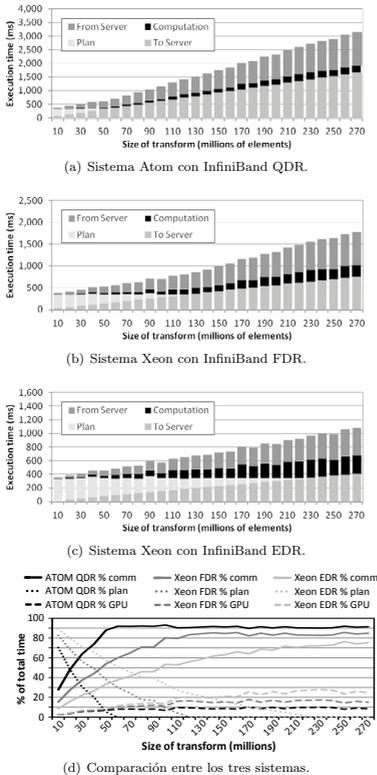


Fig. 3. Desglose del tiempo de ejecución para la DFT 1D.

en la opuesta. Por otra parte, el tiempo destinado a completar la creación del plan en el servidor viene denotado por “Plan”, mientras que el tiempo consumido en GPU para computar la transformada DFT viene etiquetado como “Computation”. Además, la Figura 3(d) muestra una comparación porcentual de los tiempos mostrados en las Figuras 3(a), 3(b), y 3(c). Las curvas etiquetadas con “comm” en la Figura 3(d) hacen referencia al tiempo total de comunicaciones (englobando a los tiempos “To GPU” y “From GPU”). En general, puede extraerse de las cuatro figuras que, conforme el tiempo de transferencia de los datos de entrada decrece con una red más rápida, se reduce el solapamiento entre éste y el tiempo de creación del plan. Adicionalmente, crear el plan en el servidor remoto requiere en torno a 330 ms, el cual no puede ocultarse, en general, para pequeños tamaños de transformada. Por otra parte, la Figura 3(d) muestra que conforme mejora la comunicación mejor, requiriendo menos tiempo para transferir datos

desde memoria del cliente a la memoria de la GPU en el servidor remoto, el porcentaje de tiempo dedicado a la parte de cómputo incrementa en torno a un 10% en caso de QDR a cerca del 30% para InfiniBand EDR.

Al considerar transformadas multidimensionales, también se consiguen notables reducciones en el tiempo de ejecución, como puede observarse en la Figura 4. Para los mayores tamaños de transformada, se consigue una aceleración de entre 10x y 30x. Nótese que, en general, la aceleración obtenida para transformadas multidimensionales es ligeramente menor a la conseguida con la DFT 1D. Eso es debido principalmente a dos razones.

La primera razón es debida al tiempo requerido para crear el plan, el cual varía en función del número de dimensiones de la DFT. En este caso, el tiempo destinado a la creación del plan de la DFT 1D en el sistema Xeon va desde los 130 a los 3485 ms, dependiendo del tamaño de la transformada, mientras que hacerlo en la GPU requiere una cantidad constante de tiempo (en torno a 330 ms). De esta manera, al trasladar la DFT 1D, habrá una reducción en el tiempo requerido a crear el plan y causando un incremento en la aceleración. Por el contrario, crear el plan para las transformadas 2D y 3D localmente en el sistema Xeon requiere de unos pocos milisegundos¹, causando así que la aceleración se reduzca al mover las DFTs multidimensionales a la GPU remota.

La segunda razón de esta menor aceleración es que el ratio *cómputo por elemento* decrementa conforme aumenta el número de dimensiones. La causa de este

¹El tiempo destinado a crear el plan depende principalmente del número de elementos por dimensión. Así, para un tamaño de transformada similar, las DFTs 2D y 3D tienen un menor número de elementos por dimensión que las DFT 1D.

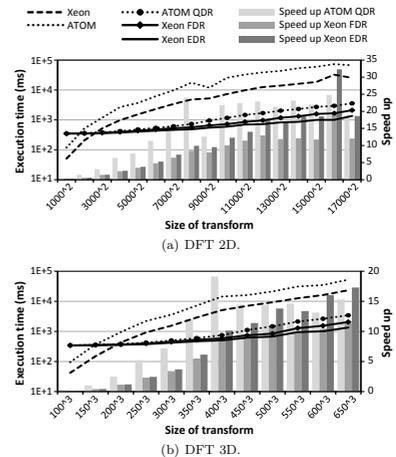


Fig. 4. Prestaciones de las DFT multidimensionales.

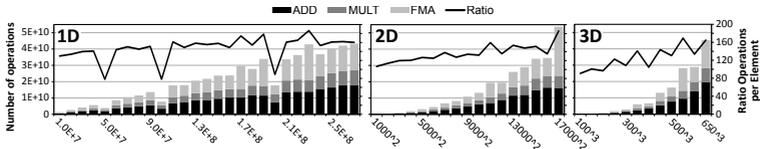


Fig. 5. Operaciones en coma flotante realizadas por las DFT 1D, 2D y 3D.

comportamiento tan diferente entre las DFTs 1D y multidimensionales puede verse en la Figura 5. En ella se muestra la cantidad de operaciones en coma flotante ejecutadas para cada tipo de transformada, desglosada en los tres tipos de operaciones usadas: sumas (ADD), multiplicaciones (MULT) y fused multiply-adds (FMA). Las FMAs se componen de una multiplicación y una suma ejecutadas en un único paso, y dependen del hardware usado. Las CPUs Intel Haswell y las GPUs de NVIDIA de arquitectura Kepler soportan estas operaciones fusionadas, por lo que se ejecutarán en un paso (las plataformas que no las soportan generarán dos operaciones para una FMA). Los números mostrados en la Figura 5 se han obtenido haciendo uso de una herramienta incluida en la librería FFTW. Los resultados muestran que el número total de operaciones se incrementa con el tamaño de la transformada.

Sin embargo, resulta más interesante analizar el ratio entre el número de elementos de la DFT y la cantidad de operaciones ejecutadas, mostrado en la Figura 5 con la línea negra etiquetada como "Ratio". Este ratio no es tan dependiente del tamaño de la transformada y señala que, para tamaños pequeños, la cantidad de operaciones por elemento se reduce conforme aumenta el número de dimensiones de la DFT. De esta manera, para los tamaños más pequeños, la DFT 1D presenta el ratio más alto, después la DFT 2D y, finalmente, la DFT 3D. Por contra, para tamaños más grandes, todas las transformadas presentan ratios similares.

Interesantemente, las transformadas pequeñas no experimentan una ganancia en las prestaciones hasta que el ratio no alcanza un valor superior a 120, aproximadamente, cuando el nodo cliente es un sistema Xeon. Esto puede verse al compararse los resultados de la Figura 5 con los valores de la Figura 4, donde para las transformadas pequeñas el uso de aceleradores remotos produce tiempos de ejecución superiores. A este respecto, en el caso de la DFT 1D, la transformada más pequeña tiene un ratio de 129, obteniendo mejoras en las prestaciones. Este resultado se alinea con el espíritu general del middleware, que acelera la parte de cómputo a costa de mover datos hacia y desde el servidor remoto. De ahí que la ganancia en prestaciones del middleware pueda ser vista como un compromiso entre el tiempo ahorrado al acelerar los cálculos con una CPU y el tiempo extra invertido en mover los datos hacia y desde esta GPU remota. Este efecto puede verse claramente en la Figura 4, que muestra la degradación en el tiempo de ejecución para las transformadas pequeñas. Las ejecuciones locales en los sistemas Xeon y Atom requieren

de un menor tiempo que las ejecuciones remotas. Estos resultados sugieren que el nuevo middleware no debería mover ciegamente todos los cómputos, sino que debería considerar el tamaño de los datos y el ancho de banda de la red para determinar si la parte de cómputo se hace local o remotamente. Esto puede conseguirse fácilmente realizando una fase de configuración automática justo después de la instalación del middleware en un cluster dado.

Como previamente se discutió, se ha incluido la optimización que crea el plan a la vez que se transfieren los datos de entrada. La inclusión de esta optimización en el middleware ha incrementado su complejidad. De este modo, uno puede preguntarse si esta mejora reporta beneficios significativos. Para responder a esto, se ha llevado a cabo una serie de experimentos con una versión no optimizada del nuevo middleware, que espera a que los datos de entrada se hayan recibido en el servidor para empezar con la creación del plan. La Figura 6 muestra el porcentaje de reducción en el tiempo de ejecución entre ambas versiones del middleware con las tres interconexiones InfiniBand consideradas en el estudio junto con la DFT 1D. Los resultados para las DFTs 2D y 3D son similares, dado que el tiempo de creación del plan en GPU requiere en torno a 330 ms independientemente del número de dimensiones.

La Figura 6 muestra que el ahorro de tiempo debido a la optimización oscila entre el 10% y el 30%, dependiendo del número de elementos de la transformada y de las prestaciones de la red. Esta dependencia tiene el comportamiento esperado, dado que la optimización se aprovecha del tiempo necesario para transferir los datos para, en paralelo, crear el plan.

En resumen, los resultados provistos claramente muestran que el nuevo middleware es capaz de reducir notablemente el tiempo de ejecución de la librería FFTW usando una GPU remota. Esta reducción se incrementa proporcionalmente con el tamaño del problema.

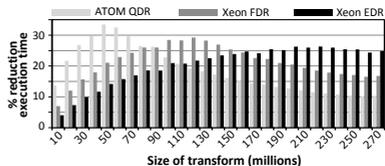


Fig. 6. Reducción en el tiempo de ejecución en la DFT 1D debido a solapar la creación del plan con la transferencia de datos de entrada.

V. USO DE COMUNICACIONES NO OPTIMIZADAS

En la sección anterior se presentaron las prestaciones obtenidas por el nuevo middleware, haciendo uso de redes InfiniBand y la API de los InfiniBand Verbs para transferir los datos. A este respecto, los datos se mueven directamente desde la memoria principal del nodo cliente a la memoria de la GPU en el servidor remoto, no haciendo solo uso de un gran ancho de banda, si no también de una arquitectura de comunicaciones mejorada basada en el uso de transferencias RDMA segmentadas. Sin embargo, la API de los InfiniBand Verbs (o motores de DMA remoto en general) no está disponible en toda red. Por ejemplo, muchas de las implementaciones de Ethernet de altas prestaciones no incluyen este tipo de comunicaciones. Por tanto, en esta sección se analizarán las prestaciones del nuevo middleware cuando se usa la interfaz de sockets TCP/IP. La razón de usar esta interfaz es debida a su amplia disponibilidad.

Nótese, sin embargo, que el uso de sockets TCP/IP implica el uso de una nueva arquitectura de comunicaciones. A este respecto, nos movemos a la arquitectura más simple en la que no se ha implementado ningún tipo de optimización en la capa de comunicaciones del nuevo framework. En esta nueva arquitectura, la parte cliente envía todos los datos de entrada de la función objetivo a la memoria principal del servidor remoto usando la API de los sockets TCP/IP. Una vez todos los datos involucrados en los cómputos requeridos se han recibido en el servidor, estos son copiados a la memoria de la GPU usando las órdenes CUDA apropiadas. De este manera, no hay paralelismo ni segmentación en el movimiento de datos desde memoria del cliente a memoria de la GPU en el servidor. Como puede verse, esta es la peor situación posible para el nuevo middleware dado que estas comunicaciones no optimizadas causan el mayor overhead posible. En consecuencia, es importante destacar que en esta sección los datos se mueven al servidor siguiendo una aproximación `stop&wait`, es decir, de memoria del cliente se envían a memoria del servidor, y de aquí a la memoria de la GPU. Al contrario, en la sección anterior, los datos evitaban esta parada intermedia en la memoria principal del servidor, transfiriéndose directamente a memoria de la GPU. Adicionalmente, la transmisión de datos estaba segmentada. Por ello, las prestaciones estaban mejoradas debido al gran ancho de banda alcanzado de la red, gracias al uso de la API de los InfiniBand verbs y la transmisión de los datos siguiendo una aproximación `cut-through`. La latencia de comunicaciones también estaba reducida.

La Figura 7 muestra las prestaciones obtenidas por el nuevo framework usando una arquitectura de comunicaciones no optimizada. Con el propósito de comparación de resultados, se ha incluido en el estudio un sistema adicional conteniendo 2 procesadores Intel Xeon E5-2637 v2 de 4 núcleos (Ivy Bridge) funcionando a 3.5 GHz y con 64 GB de memoria DDR3 a 1866 MHz. El tiempo de ejecución del sistema ejecutado localmente se ha incluido como referencia. Las

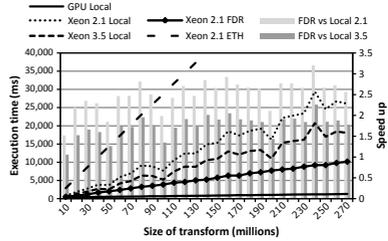


Fig. 7. Prestaciones de la DFT 1D en el sistema Xeon con las comunicaciones no optimizadas.

curvas etiquetadas con “Xeon 2.1 Local” y “Xeon 3.5 Local”, muestran estos resultados. Las prestaciones de las ejecuciones cuFFT en la GPU (sin intervención del middleware) se muestran también con propósitos comparativos (curva “GPU Local”) e incluye los tiempos para crear el plan además del movimiento de datos entre memoria principal y memoria de GPU. Además, cuando se usa el nuevo middleware, el nodo cliente consta de un sistema basado en procesador Xeon funcionando a 2.1 GHz. Dado que se usan comunicaciones TCP/IP, se incluye la red Ethernet 1 Gbps (curva “Xeon 2.1 ETH”), además de las ya usadas InfiniBand QDR, FDR y EDR, usadas en la sección anterior. Para estas últimas, se ha usado TCP/IP sobre InfiniBand en lugar de transferencias basadas en RDMA. Sin embargo, dado que las tres generaciones de InfiniBand ofrecen resultados similares², solo se muestran los resultados para FDR (curva “Xeon 2.1 FDR”). La aceleración obtenida al usar FDR se muestra. Las barras etiquetadas como “FDR vs Local 2.1” y “FDR vs Local 3.5” hacen referencia a la aceleración con respecto a la ejecución local en los procesadores Xeon a 2.1 GHz y Xeon a 3.5 GHz, respectivamente.

Los resultados en la Figura 7 muestran claramente que el uso de una red más lenta, como Ethernet 1 Gbps, no ofrece una mejora de prestaciones, a pesar de acelerar el cómputo de la DFT usando una potente GPU. Sin embargo, el uso de una red más rápida, como InfiniBand, ofrece una aceleración de 3x con respecto al Xeon a 2.1 GHz y 1.75x cuando la referencia es el Xeon a 3.5 Gbps. No obstante, estas aceleraciones están lejos de la aceleración máxima ofrecida por la librería cuFFT al usar la GPU local (entre 30x y 35x, no mostrado en la Figura 7) con respecto a la ejecución de la librería FFTW en la

²Después de obtener estos resultados para las tres generaciones de InfiniBand, decidimos hacer uso de la herramienta `iperf` [10] para recopilar resultados de ancho de banda TCP (dado que se hace uso de TCP/IP sobre InfiniBand). Esta herramienta muestra que, para el sistema con el Xeon 2.1GHz, todas las conexiones proveen el mismo ancho de banda, en torno a 1190 MB/s. Curiosamente, cuando se pasa a los procesadores a 3.5 GHz, la herramienta `iperf` presentaba una prestaciones notablemente mejores, mostrando que la productividad depende ampliamente de las características exactas del sistema, dado que no se usan mecanismos RDMA.

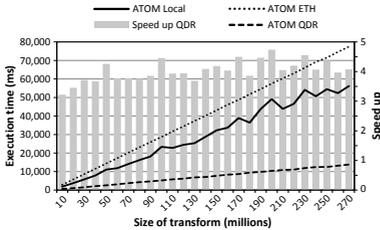


Fig. 8. Prestaciones de la DFT 1D en el sistema Atom.

CPU. Estos resultados confirman que, con intención de acelerar las ejecuciones de la librería FFTW, las prestaciones de la red son claves, como era de esperar. Sin embargo, estos resultados también confirman que para esas configuraciones hardware que constan del uso de procesadores de gama media con redes de altos anchos de banda, se pueden obtener aceleraciones nada despreciables. En la Figura 8 se muestra el tiempo de ejecución al usar el nuevo middleware en el sistema con el procesador Atom con redes Ethernet e InfiniBand QDR. Se puede observar que, debido a la baja capacidad de cómputo del sistema Atom con respecto al sistema Xeon, la aceleración obtenida es mayor. Estas redes de alta velocidad están siendo usadas con la ampliamente disponible interfaz de los sockets TCP/IP. Por lo tanto, estos resultados pueden ser aplicable a diversos despliegues de sistemas, incluso sin incluir características RDMA. Además, los resultados se han obtenido haciendo uso de la peor arquitectura de comunicaciones posible. A este respecto, podría ser posible crear algún tipo de sistema segmentado, incluso usando TCP/IP.

La Figura 9 presenta el desglose del tiempo de ejecución usando InfiniBand FDR. Nótese que en este caso el desglose de tiempo es ligeramente diferente al mostrado en la sección anterior, dado que en este caso los datos de entrada no se transfieren directamente entre la memoria principal del cliente y la memoria de la GPU en el servidor, sino que se almacenan temporalmente en la memoria principal del servidor antes de ser transferida a la memoria de la GPU. De esta forma, el tiempo de ejecución se divide en seis componentes: (1) tiempo requerido para mover los datos de entrada de la función de la FFTW desde memoria principal del cliente a memoria principal del servidor remoto (“Client To Server”), (2) tiempo empleado en el servidor en copiar los datos de entrada desde memoria principal a la memoria de la GPU (“Server to GPU”), (3) tiempo requerido en completar la creación del plan en el servidor (“Plan”), (4) tiempo empleado por la GPU para computar la transformada DFT (“Computation”), (5) tiempo requerido para mover los datos desde memoria de la GPU hasta memoria principal en el servidor (“GPU To Server”), y (6) tiempo para devolver los resultados de vuelta al cliente (“Server to Client”). Se puede observar que

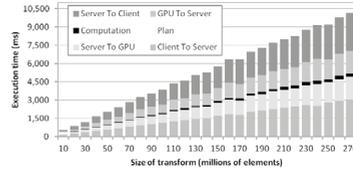


Fig. 9. Desglose del tiempo de ejecución para la DFT 1D al usar InfiniBand FDR.

casí el 60% del tiempo es empleado en mover datos entre el cliente y el servidor, mientras que el 37% del tiempo es usado para moverlos entre la memoria del servidor y de la GPU. Esto muestra claramente los beneficios de usar una capa de comunicaciones segmentada. El tiempo real de realizar la transformada tan solo requiere el 3% del total. Sin embargo, a pesar de dedicar la mayor parte del tiempo a mover datos entre cliente y servidor y la memoria de la GPU del servidor remoto, el nuevo middleware todavía ofrece una aceleración no despreciable, como puede observarse en la Figura 7. Finalmente, nótese que el tiempo para crear el plan queda oculto para todos los tamaños de transformadas excepto para los más pequeños, donde queda parcialmente solapado con el tiempo de comunicaciones.

En el caso de mover DFTs multidimensionales, la Figura 10 muestra el tiempo de ejecución para la DFT 2D. Como se pudo ver en la sección anterior, las pruebas realizadas para los datos de entrada con diferente número de elementos en distintas dimensiones proveen resultados similares. Las ejecuciones se han llevado a cabo en los mismos escenarios considerados en la Figura 7. La Figura 10 muestra que el uso del nuevo middleware para acelerar las DFTs multidimensionales sigue, en general, la misma tendencia ya analizada para la DFT 1D. A este respecto, el uso de una red de bajas prestaciones como Ethernet provee una notable degradación en el tiempo de ejecución. Al contrario, el uso de una red de altas prestaciones como InfiniBand reduce el tiempo total de ejecución. Se consigue una aceleración de entre 1.5x y 2.5x en comparación a las ejecuciones locales en el Xeon a 2.1 GHz. Al compararlo frente al Xeon a 3.5 GHz, la aceleración obtenida está en torno a 1.5x. Los resultados obtenidos para la DFT 3D (no mostrados) son similares, aunque las aceleraciones son ligeramente inferiores: entre 1.5x y 2x con respecto a usar el Xeon a 2.1 GHz, y en torno a 1.25x al compararlo con el Xeon a 3.5 GHz. Sin embargo, al contrario que la tendencia mostrada en la Figura 7, donde se conseguía una aceleración para todos los tamaños de transformada considerados, en el caso de las DFTs 2D y 3D no se obtiene ganancia. Por ejemplo, para la DFT 2D, las transformadas de hasta 3000² no proveen aceleración con respecto a las ejecuciones locales en CPU. A partir de este tamaño, la aceleración empieza a aumentar de forma propor-

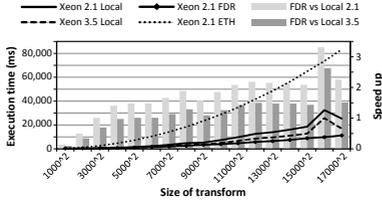


Fig. 10. Prestaciones de la DFT 2D en el sistema Xeon.

cional al número de elementos, hasta que alcanza un estado estable para los tamaños más grandes. Este efecto ya ha sido analizado en la sección anterior.

Como resumen de esta sección, es importante remarcar que el uso de redes de bajo ancho de banda, como es el caso de Ethernet 1 Gbps, provoca que el nuevo middleware no mejore las prestaciones. Sin embargo, el uso de redes de mayor ancho de banda resulta en mejoras en las prestaciones a pesar de usar comunicaciones no segmentadas basadas en TCP/IP y no una capa de comunicaciones optimizada.

VI. DISCUSIÓN

El nuevo middleware presentado en este artículo requiere una discusión adicional. Primero, los tamaños de transferencia evaluados oscilan entre los 10 y los 270 millones de elementos para DFTs 1D. Sin embargo, ¿cómo de grandes deberían ser los tamaños de transformadas a manejar? Nótese que la memoria disponible en la GPU es el factor limitante. Por tanto, dado que la GPU Tesla K40 incluye 12 GB de GDDR5, entonces se podrían considerar transformadas de hasta 400 millones de elementos. Para DFTs de tamaños mayores, la operación puede dividirse en DFTs más pequeñas [11] que pueden computarse independientemente para después combinarse. Por lo tanto, si el tamaño del problema excede la memoria de la GPU, el nuevo middleware dividirá automáticamente los datos de entrada en trozos más pequeños que individualmente sí cabrán. Los resultados se combinarán posteriormente antes de ser entregados a la aplicación ejecutando la función `fftw_execute` original. Este proceso debería ser transparente para la aplicación.

Otro aspecto de discusión es que para muchos casos de uso de la DFT 3D en computación científica implican métodos de descomposición del dominio. Esto permite el uso de DFTs 1D para resolver problemas con DFTs 3D. Además, estos métodos permiten el manejo de problemas de gran tamaño que vengan limitados por la memoria de la GPU. El middleware debe ser capaz de gestionar las DFTs multidimensionales usando DFTs 1D, haciéndolo de forma transparente a la aplicación, que únicamente hace una llamada a la librería FFTW para la DFT. Por contra, en el caso en que esta descomposición ya sea gestionada por la aplicación, el middleware solo debería manejar y direccionar DFTs 1D si caben en la memoria de la

GPU, como se explicó anteriormente.

Finalmente, nótese que además de la GPU, el servidor remoto también incluye uno o más sockets de CPU, que no han sido usados en esta implementación del nuevo middleware con objeto de acelerar más la ejecución de la DFT. Sin embargo, el nuevo middleware no se limita al uso de la librería cuFFT de NVIDIA, sino que pueden usarse otras. Por ejemplo, en caso de disponibilidad, se podrían usar otras librerías que incluyan varias GPUs (en caso que el servidor remoto tenga conectada más de una) o que distribuyan el cómputo de la DFT entre la GPU y los cores de CPU disponibles.

VII. TRABAJO RELACIONADO

En esta sección, el middleware se pone en la perspectiva correcta y en un contexto adecuado. Respecto a esto, ya se han propuesto e implementado técnicas para la eliminación de carga de cómputo de una máquina para ser ejecutada en un servidor o hardware especializado en una amplia variedad de contextos, como sistemas multicore heterogéneos en chip, GPUs (esto es lo que NVIDIA hace con CUDA), y planificación de trabajos en nodos especializados en infraestructuras de computación grid. La diferencia de estos con nuestro middleware es su funcionamiento a nivel de cluster en lugar de nodo.

Sin embargo, a nivel de cluster, nuestro middleware puede parecerse al mecanismo de llamada a procedimiento remoto (RPC) [12][13], que es una técnica arraigada que permite a la función ejecutar código en nodos remotos usando esqueletos de funciones. Sin embargo, existen importantes diferencias entre RPC y nuestro middleware. En primer lugar, con el fin de usar llamadas a procedimiento remoto, el programador tiene que programar explícitamente su uso mientras que nuestro middleware no requiere de ninguna modificación en el código de las aplicaciones. Segundo, al contrario que con RCP, nuestro middleware no requiere de la intervención del sistema operativo para mover datos al servidor remoto (por ejemplo, si se usa una capa de comunicaciones optimizada usando la API de InfiniBand Verbs, todo el código se ejecutará a nivel de usuario). Además, nuestro middleware no hace uso de sus procesos cliente o servidor ni en el origen ni en el destino, que es como RPC se comporta, ni tampoco se requieren de rutinas RPC: nuestro middleware está integrado en la aplicación en la parte cliente y, en la parte servidor, este se ocupa directamente de las peticiones de cómputo, sin necesidad de que intervenga ninguna otra entidad. Asimismo, nuestro middleware no requiere el uso de compiladores especiales como `rpgen` o el uso de funciones de la API de RPC como `rpc_reg()` o `rpc_call()`. Por último, RPC es un mecanismo *sin estado* (cada llamada RPC es independiente de otras), mientras que el nuevo middleware requiere almacenar varias variables de estado entre distintas llamadas desde la aplicación, tal y como se hace con los falsos planes descritos en la Sección III.

Otra de las propuestas previas que podrían tener

cierta similitud con nuestro middleware es ORB [14], que básicamente consiste en un proceso broker ejecutado en la red el cual pone en contacto a un cliente requiriendo un servicio con un servidor que se encarga de proveerlo. Usando la aproximación de ORB, el programador o bien usa un lenguaje de definición de interfaces (IDL) para declarar las interfaces públicas del servidor, o bien el compilador del lenguaje de programación empleado traduce las sentencias del lenguaje en sentencias apropiadas para IDL. Por el contrario, nuestro middleware no necesita un proceso centralizado para conectar solicitantes con servidores ni modificar el código de la aplicación o compiladores especializados. Además, en la parte cliente de ORB, los objetos son creados e invocados, sirviendo como la única parte visible y usada en la aplicación del cliente. Al contrario, nuestro middleware no requiere ningún tipo de proceso adicional.

En resumen, las principales diferencias entre estas propuestas y nuestro middleware son que (1) nuestro middleware no precisa realizar ninguna modificación en el código fuente de las aplicaciones (incluso no requiere que las aplicaciones sean recompiladas nada más que para el uso de librerías dinámicas) y (2) no necesita procesos o sistemas en tiempo de ejecución adicionales para su funcionamiento.

Finalmente, nótese que nuestro middleware actualmente usa GPUs en el servidor remoto, como hacen los frameworks de virtualización de GPUs. Ejemplos de estos frameworks son rCUDA [15], gVirtus [16] o DS-CUDA [17]. Sin embargo, hay una diferencia importante entre ambos: los frameworks de virtualización de GPUs redirigen las llamadas CUDA desde su destino original (una GPU local) a una GPU remota, mientras que nuestro middleware mueve la parte de cálculo de las librerías matemáticas, inicialmente previstas para ser ejecutadas en una CPU local, a un acelerador remoto. Así, se transforma la naturaleza de la llamada con objeto de ejecutar una instancia diferente del código, proveyendo exactamente el mismo resultado. Además, nótese también que trasladar a servidores remotos la computación de librerías matemáticas con nuestro middleware puede reportar ahorros en el tiempo de ejecución. Al contrario, los frameworks de virtualización de GPU remota no aceleran los cálculos enviados al servidor. Estos cálculos se ven ligeramente penalizados debido al camino más largo hacia la GPU remota con respecto al caso de usarla localmente.

VIII. CONCLUSIONES

Hemos presentado la implementación de un nuevo middleware que permite trasladar la parte de cómputo destinada a ser ejecutada en CPU de librerías científicas hacia aceleradores remotos situados en otros nodos del cluster. Además, hemos realizado una detallada evaluación de prestaciones de este middleware, que ofrece soporte a la librería FFTW y es compatible con GPUs compatibles con CUDA. Los resultados demuestran que la mejora de prestaciones es proporcional a la productividad de la red

subyacente.

Aunque la versión actual del middleware ofrece soporte para las librerías FFTW y BLAS y para un solo cliente, se espera dar soporte en futuras versiones a la librería LAPACK, múltiples clientes y aceleradores Intel Xeon Phi. Una vez este trabajo esté realizado, se evaluarán los beneficios que el nuevo middleware podría aportar a aplicaciones reales, en términos de reducción del tiempo de ejecución. Como hecho destacable, indicar que aunque el uso de GPUs se ha considerado para muchas aplicaciones, hay otras que posteriormente han decidido dejar de continuar su soporte (como por ejemplo, la aplicación DL.POLY [18]). Para otras aplicaciones nunca portadas a aceleradores, este middleware podría resultar atractivo de cara a reducir el tiempo de ejecución. De hecho, el uso del nuevo middleware con estas aplicaciones podría retrasar la necesidad de adaptarlas para su uso con GPUs u otro tipo de aceleradores. Otros campos de investigación en relación a este trabajo podrían consistir en el balanceado automático de carga entre servidores.

AGRADECIMIENTOS

El presente trabajo ha sido financiado por la Generalitat Valenciana mediante el proyecto PROMETEOII/2013/009 del programa PROMETEO fase II.

REFERENCIAS

- [1] *FFTW (Fast Fourier Transform)*, <http://www.fftw.org/>
- [2] *TINKER Molecular Modeling Package*, <http://dasher.wustl.edu/tinker/>
- [3] *COSMOS: Computer Simulation of Molecular Structures*, http://www.cosmos-software.de/ce_intro.html
- [4] *YASARA: Yet Another Scientific Artificial Reality Application*, <http://yasara.org/>
- [5] *Reduced Molecular Dynamics Streamlined*, <https://bionano.cent.uw.edu.pl/software/redmd/>
- [6] *BLAS (Basic Linear Algebra Subprograms)*, <http://www.netlib.org/blas/>
- [7] *LAPACK (Linear Algebra PACKage)*, <http://www.netlib.org/lapack/>
- [8] M. Frigo et al., *The design and implementation of FFTW3*, Proceedings of the IEEE, vol. 93 number 2, 2005
- [9] *Supernicro, X10 MicroBlade with MBI-6418A-T7H modules*, <http://www.supernicro.nl/products/MicroBlade, 2015>
- [10] *iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool*, <https://github.com/esnet/iperf>
- [11] J. W. Cooley et al., *An algorithm for the calculation of complex fourier series*, Mathematics of Computation, April 1965.
- [12] A. D. Birrell and B. J. Nelson, *Implementing Remote Procedure Calls*, ACM Trans. Comput. Syst., 1984
- [13] Y. Tanaka et al., *Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing*, Journal of Grid Computing, 2003
- [14] F. Nammour et al., *Comparative evaluation of object request broker technologies*, ICCSA 2003
- [15] A. J. Peña et al., *A complete and efficient CUDA-sharing solution for HPC clusters*, PARCO Journal, 2014
- [16] G. Giunta et al., *A GPGPU Transparent Virtualization Component for High Performance Computing Clusters*, EuroPar10
- [17] M. Oikawa et al., *DS-CUDA: A Middleware to Use Many GPUs in the Cloud Environment*, SCC, 2014
- [18] *The DL.POLY Molecular Simulation Package*, <http://www.ccp5.ac.uk/DL.POLY/>

Dynamic load balancing for an enhanced energy efficiency on heterogeneous systems

Alberto Cabrera, Alejandro Acosta, Francisco Almeida, and Vicente Blanco¹

Resumen—New energy efficient architectures, powerful graphic processing units and manycore processors, are present in current HPC systems. They provide different performance capabilities and different energy efficiencies. Being energy efficiency a major issue in this area, all the available alternatives require the adaptation of existing codes and libraries in order to achieve good performances both in time and energy efficiency. This requires a deep knowledge of programming and architectural details to achieve good performance, which are frequently interdependent. We have developed a new library, ULL-Calibrate.Energy, a dynamic load balancing library that allows load balancing to enhance the overall energy efficiency of a parallel application in an heterogeneous environment, with low additional effort for the programmer. The overhead introduced by our system is also minimal. The calibration library has been successfully used to solve problems over heterogeneous platforms. To validate it, we present an analysis of different Dynamic Programming problems over different hardware configurations.

Palabras clave—Dynamic Load Balancing; Energy Efficiency; Iterative Algorithms

I. INTRODUCTION

There is a major concern in high performance computing regarding to the increasing energy consumption of the top end systems. Current Petaflop systems draw a huge amount of energy at full performance, as much as 17.8 MW of power without considering the energy used for cooling the facility that holds the most consuming computer in the Top500 list. With the current growth, by 2020, exascale systems will start to appear, and the main challenge will be to have them limited to 20MW to have a bearable infrastructure cost.

From now on, we denote by energy efficiency, to the amount of computational work that can be performed using one watt of power, i. e., units of work developed per units of energy.

Highly heterogeneous environments are nowadays available as computational resources for institutions [1]. Traditional multicore architectures now coexist with multiGPU systems, energy efficient ARM systems and Intel Phi processors.

This allows the use of these multiple computing resources as a unique computational platform. However, applications usually has a performance penalty when they are not tuned explicitly for the heterogeneous platform. There is a strong dependence between parallel code and target architectures [2].

The usage of GPUs as coprocessing units, with the new architectures designed to increase the energy

efficiency has incremented the heterogeneity level in current MultiGPU HPC platforms.

These new architectures are affected by the same well known problems that affect conventional parallel programming, including the load balancing problem while solving irregular problems, an aggravated problem when the systems used have different efficiencies in terms of energy and time.

We investigate the load balancing problem appearing when parallel problems are executed in multi-GPU heterogeneous systems and how the energy consumption is related to imbalance problems. The heterogeneous system may have different node setups and GPUs with different computing capabilities. We designed a simple mechanism to balance the work load between different parallel processes dynamically with few changes to the source code of an application. We present the ULL_multiobjective.library, that allows dynamic workload balancing inside a parallel program running on a heterogeneous system, adapting to the conditions of the system during the execution of the application. The library provides a programmer a series of functions to be used where the load balancing operations should be applied. This software, uses different libraries as modules to allow calibration for different objectives, using ULL-Calibrate.lib for the calibration techniques, and EML[3] for the energy measurement. EML has been proven to have a negligible overhead, and provide all the tools needed to perform the measurement for the calibration phase of the dynamic load balance operations. Now it is possible to load balance using time as reference unit, performing a standard load balancing, or to allow balancing using the energy factor to enhance energy efficiency.

The main contributions of this paper are:

- Applying the ideas and methodologies developed for the load balancing of heterogeneous clusters to energy consumption.
- A new library design, capable of being extended for calibrating based on a user objective. We have developed calibration for energy consumption, but it is possible to implement other types of objective functions.
- Achieving dynamic load balancing using energy efficiency as reference unit to enhance energy usage in heterogeneous systems, concretely, in a multiGPU environment.

Before the exascale objective, all efforts were put into improving performance of parallel systems. Load balancing was made to improve time and that would reduce energy consumption, but as new archi-

¹HPC Group. ETS de Ingeniería Informática. Universidad de La Laguna, ULL. La Laguna. 38270 Tenerife. Spain, e-mail: Vicente.Blanco@ull.es.

techniques are developed, good performances per watt are achieved by sacrificing performance, allowing distinction between time and energy efficiency. We prove that it is possible to perform a different approach for balancing, to achieve good results when optimizing energy efficiency.

To validate our proposal, we have executed several Dynamic Programming optimization problems where the portability of the single GPU code to a multiGPU system is compromised due to load imbalance inefficiencies. The chosen problems correspond to an iterative scheme, so this technique can be applied to many other iterative problems like Jacobi, GaussSeidel, Longest Common Subsequence, Matrix Parenthesization and to some classes of stencil codes. The efficiency level obtained, considering the minimum code intrusion, makes this methodology a useful tool in the context of the energy efficiency in heterogeneous platforms.

This paper is structured as follows: Section II covers the related work in the field of load balancing workload on heterogeneous systems. In Section III the library usage is explained among the advantages of this approach. Section IV describes the experimentation performed and which problems were used for this purpose. Section V shows the results obtained after multiple executions are performed with our energy balancing methodology. Finally, Section VI offers our conclusions and future research possibilities.

II. RELATED WORK

Extensive work has been done in the field of load balancing in heterogeneous systems. The most direct approach to address the problem requires to develop code specifically for the architecture in order to maximize performance. This requires a deep understanding on parallel programming for the target systems and also a manual task allocation according to the capability of every processing unit. A different approach is to develop applications using skeleton programming. This concept implies finding a skeleton that covers your implementation, so you can develop the particularities of your exact problem. Once done, this skeletons that are optimized to obtain an optimal load balance for maximum time efficiency, could be modified to obtain the best configuration for energy efficiency. Frameworks like SkelCL[4] and Marrow[5] are suitable to use these techniques. There is also a well established computational model that uses DAGs as its representation together with the dynamic task scheduling that have gradually made their way into academic dense linear algebra packages. The model is currently used in shared memory codes, such as PLASMA [6] and FLAME [7], and has been ported to hybrid multicore+GPUs architectures like in MAGMA [8], [9] and FlameGPU [10]. Following this approach, systems like SuperMatrix [11] or StarSs [12] uses runtime support for the dynamic task scheduling (the last one applied to Multi-GPU systems). Our ap-

proach to balance the workload of iterative algorithms on heterogeneous systems is closer to the work presented by Martínez et al. in ADITHE [13]. This environment and ULLCalibrate.lib follows a similar strategy. First, it uses an homogeneous distribution of the workload on the heterogeneous system, the speed of every node is estimated during the first iterations of the algorithm. According to the speed of every node, a new workload distribution is carried out. Finally, the remaining iterations of the algorithm are executed

A similar approach can be used to balance workloads in terms of energy consumption instead of performance. Energy-aware scheduling algorithms can be found in datacenters with virtual machines deployments. As an example, HEROS[14], [15] proposes a load balancing algorithm for energy-efficient resource allocation in heterogeneous systems; or Takouna et al.[16] who proposes peer VMs aggregation to enable dynamic discovery of communication patterns and reschedule VMs based on the determined communication patterns using VM migration. These techniques minimize network traffic and deal with an energy efficient scheduling. Other techniques uses DVFS processor capabilities to implement energy-aware schedulers for HPC applications like PAAS[17] (Power Aware Algorithm Scheduler), or to balance energy in iterative algorithms[18]. Further strategies try to compensate directly intrinsic algorithmic imbalances with DVFS like in irregular parallel divide-and-conquer algorithms[19]

In our approach we start with an homogeneous distribution of work at the beginning of the application, and depending on the performance of each process during a few iterations, the workload gets dynamically redistributed to obtain load balance for time efficiency. Our new contribution is to be able to do the approach in terms of energy efficiency, to enhance energy usage in parallel applications.

III. DYNAMIC TASK ALLOCATION

A. Description

We have developed a library that allows load balancing using multiple approaches. We developed an interface that allows calibration in heterogeneous systems, with the help of Ull Calibrate lib for the calibration operations, and EML to provide the energy data. The Ull multiobjective library, is an interface that allows load balance using as much objectives as the user desires. Besides the standard load balancing, executed when time is chosen as the unique objective to fulfill, and, in addition, it also possible to chose energy efficiency as the target function. The overhead introduced by our system is minimal, but we had to modify the function headers in order to allow an easier usage of the new library capabilities. ULLCalibrate.lib was previously tested and validated over various regular and irregular problems over different heterogeneous systems, including clusters of homogeneous and heterogeneous multicore systems and multiGPUs. EML is an energy

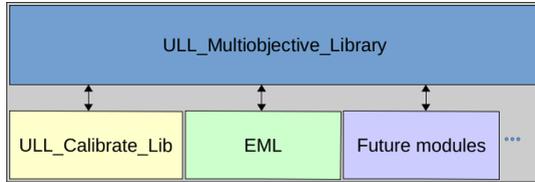


Fig. 1: Library internal structure

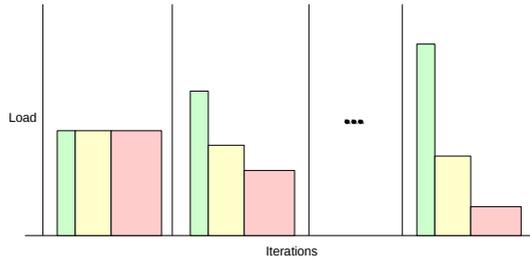


Fig. 2: Library calibration for energy. Each colored area represents the energy consumption of a process in a given iteration. The library unbalances the load in order to improve energy efficiency, minimizing the total energy.

measurement library developed to abstract energy measurement that addresses the difficulties involved in the differences between the available measurement APIs and different measurement tools. Every measurement tool has its own interface and procedure for measurement and every hardware architecture can also have specific details that affect how the measurement is done. Both EML and Ull Calibrate have been proven to introduce minimal overhead and this will be confirmed in Section V.

```

// nprocs = Number of processors ;
// id = ProcessID; n = Problem size
count[id] = nprocs;
displ[id] = id * count [id];
for (i=0 ; i<n; i++) {
    for (j = displ[id];
        j < displ[id] + count[id];
        j++) {
        // ... Work ...
        Process_data(j);
    }
    Collective_Operation ();
}
    
```

Listing 1: "Example code of an iterative scheme"

The new calibration library allows to unify the different interfaces to access the power and energy making it suitable for the purposes of calibration. The library internal structure is shown in Figure 1. Observe that both Ull Calibrate and EML are shown as independent modules. The user could, if desired,

change the calibration functions or the energy measurement library to utilize their own, or the ones that helps better to accomplish its needs. It is also possible to attach a new library that measures a completely different criteria, and balance according to that.

As mentioned, the technique applied before has now an added feature for multiobjective balancing, which includes energy consumption. The design of the library was originally meant to balance time consumptions for the different available processes, based on an iterative scheme such as the one shown in Listing 1. The code shows a loop that executes a number N of iterations performing a computing task in each iteration, which can be subdivided in multiple smaller tasks. Each process then executes the task with the distributed workload, which has a size proportional to the capabilities of the processor or GPU that is executing the task. Afterwards, a collective communication operation is performed, so all the processors are synchronized recollecting the updated dataset before continuing with the following iteration. This iterative scheme appears in various parallel algorithms such as a classic matrix product, Jacobi, dynamic programming, and shortest path problems.

Figure 2, shows a general vision on how the library modifies the workload. Every area color represents an energy consumption for a process. The wider the color, more energy is consumed per unit of work,

representating different energy efficiencies.

The energy efficiency metric can be defined as the amount of work developed per unit of energy consumed by a process. A high value for the energy efficiency means that a process performs more work with a given amount of energy than a different processor with a lower value. This approach is necessary to prevent assigning energy, since we have to take in account that, if we increase the execution time, even if the energy consumed in between measurements is lower the overall energy could increase, ruining our aim.

$$E_{E_i} = \frac{1}{\frac{E_i}{T_i \cdot M_i}} \quad (1)$$

Equation 1 introduces a metric that allows to measure Energy Efficiency E_E for a processor i . E_i is the energy consumed by the processor i for a given workload, T^i is the time measured in such interval and M_i is the number of units of work solved. This gives us a metric that reflects the energy required to solve one unit of work, thus allowing to redistribute load accordingly.

If the code in Listing 1 is executed on an heterogeneous system made up of three processes that:

- System 0 has an energy efficiency of E_f
- System 1 has an energy efficiency of $\frac{1}{4} \cdot E_f$
- System 2 has an energy efficiency of $\frac{1}{8} \cdot E_f$

the result of an execution for calibrating energy would distribute the work to the most efficient system, System 0, rendering the rest of the systems idle. Since this criteria is not enough for parallel programming, we will use a multiobjective approach. This should allow to tune the energy usage of an execution, while deciding how much time sacrifice while doing so, by reducing the usage of the least efficient systems.

If the previously mentioned systems also meet the following criteria:

- System 0 has a compute power of c
- System 1 has a compute power of $\frac{1}{4} \cdot c$
- System 2 has a compute power of $\frac{1}{8} \cdot c$

being c the amount of work per second a processor is capable of computing, then the distribution of workload would be w for System 0, $\frac{1}{4} \cdot w$ for System 1 and $\frac{1}{8} \cdot w$ for System 2, the same result we would obtain with a standard load balancing algorithm.

For our case, the multiobjective workload distribution would behave as a weighted function, as shown in Equation 2.

$$f_i = w_c \cdot \frac{E_{E_i}}{\sum_{i=0}^{p-1} E_{E_i}} + w_t \cdot \frac{c_i}{\sum_{i=0}^{p-1} c_i} \quad (2)$$

For this Equation, $f(i)$ represents a proportional value of work that has to be assigned to process i , calculated using a weighted function, where w_c and w_t are user weights, between 0 and 1, that specify the importance of the energy metric and the time metric, independently, assuming there are p processors. To

distribute the work, Equation 3 is applied to obtain a normalized value fn_i from f_i , to finally the workload that has to be assigned to processor i .

$$fn_i = \frac{f_i}{\sum_{i=0}^{p-1} f_i} \quad (3)$$

By adjusting the weights, it is possible to enhance energy efficiency with a conservative approach, with the maximum energy efficiency being the case where the work is distributed to the most energy efficient machines.

As the time used to complete the task influences the energy efficiency of this task, energy used by being idle should be taken into account to properly distribute energy consumption and increase the efficiency of the whole system.

```

// nprocs = Number of processors;
// id = ProcessID; n = Problem size
count[id] = Problem_size / nprocs;
despl[id] = id * count[id];
cal_calibration.t* calib;
cal_mpi_init(); // Manages EML initialization
cal_mpi_setup(&calib, count, displ,
              efficiency_weight,
              THRESHOLD, ROOT, MPI_COMM_WORLD);
for (i=0 ; i<n; i++) {
    cal_mpi_start(calib);
    for (i = displ[id];
         i < displ[id] + count[id];
         i++) {
        // ... Work ...
        Process_data(j);
    }
    cal_mpi_stop(calib, count, displ);
    Collective_Operation();
}
// Manages EML shutdown process
cal_mpi_shutdown();
// Deallocation of the memory
cal_mpi_free(calib);
    
```

Listing 2: "Example code of calibration usage"

B. Library Usage

The library we developed provides a datatype *calibration.t* to simplify the library calls. This variable stores:

- *count*: Array with the sizes of the problem assigned to every process.
- *despl*: Array of offset that indicates where is the data located for every process in the input.
- *energy_efficiency_weight*: {0..1} value that corresponds to the energy weight w_e .
- *time_efficiency_weight*: {0..1} value that corresponds to the time weight w_t .
- *threshold*: Corresponds to a percentage that indicates whether to balance or not. When the differences for the load balancing efficiency weight associated to the work loads, expressed in percentages, among all the processes involved in the parallel computation are under this value, the system is considered to be balanced, though there is no need for rebalancing the iteration.

Problem Name	Recurrence Equation
KP	$f_{i,j} = \{f_{i-1,j}, f_{i-1,j-w} + p_i\}$
RAP	$f_{i,j} = p_{i,j}$ if $i = 1$ and $j > 0$ $f_{i,j} = \max_{0 \leq k < j} \{f_{i-1,j-k} + p_{i,k}\}$ if $(i > 1)$ and $(j > 0)$
TCP	$f_{i,j} = cost_i \cdot cost_{i+1} \cdot cost_{i+2}$ if $i = (j-2)$ $f_{i,k} = \min_{i < j < k} \{f_{i,j} + T_{k,j} + (cost_i \cdot cost_j \cdot cost_k)\}$
CSP	$f_{i,j} = \max \begin{cases} \max_{0 \leq k < object} \{profit_k\} \\ \max_{0 \leq z \leq i/2} \{f_{z,j} + f_{i-z,j}\} \\ \max_{0 \leq y \leq j/2} \{f_{i,y} + f_{i-j,y}\} \end{cases}$

Table I: Dynamic programming problems description

If E_i is the efficiency weight associated to process i to execute the assigned task, E_{max} the maximum weight and E_{min} the minimum efficiency calculated, then the balance occurs if $1 - \frac{E_{max}}{E_{min}} < threshold$ the balancing phase does not take place.

After setting up the parameters, to perform the load balancing it is only needed to envelope the section of the parallel code with a `ULL_multiobjective_start` and `ULL_multiobjective_stop` calls. As shown in Listing 2, the library's ease of use and the minimum code intrusion. The balancing algorithm used by the library proved to be very efficient and the overhead introduced is negligible [20]. Additionally, we provide the functions required to initialize and finalize all the modules used by the library.

IV. EXPERIMENTATION

To validate our proposal we performed a series of experiments using an heterogeneous system composed by 2 nodes, each one with a Nvidia GPU GPUs. Both nodes have two Intel Xeon E5-2660 @ 2.20GHz CPU, with 8 cores each, 16 cores total in each node. The heterogeneous multiGPU configuration is provided using a Nvidia M2090, with 512 CUDA cores and 6GB of DDR5 memory, in one node (Node1) and a Nvidia Tesla K40m, with 2880 cores and 12GB of memory, in the second node (Node2). Every experiment was compiled and executed using the Intel C compiler 12.1.0 with Openmpi 1.6.4.

In order to validate the load balancing library, we tested the software using 4 dynamic programming problems: the 0/1 Knapsack Problem (KP), the Resource Allocation Problem (RAP), the Triangulation Convex Polygons (TCP) and the Cutting Stock Problem (CSP). In Table I, we show the recurrence functions of these problems. Data dependencies are different in most of the considered cases, hence the solving aren't the same for all cases. RAP and KP are solved accessing the table row by row, while TCP and CSP are solved diagonally. Also, TCP and CSP start from the opposite diagonals: TCP starts from the main diagonal upward, and CSP moves diagonally downward traversing all diagonals of the table. In order to simplify, we have chosen three different problem sizes to visualize the balancing behaviour for various cases, composed by square matrices of 1000,

2000 and 5000 rows.

Also, for reference purposes Figure 3 shows the energy consumption of every problem executed sequentially in each GPU. Node 1 GPU (Nvidia Tesla M2090) has a lower energy consumption on average, 22.97% more energy expensive than the Node 2 GPU (Nvidia K40m). The KP is 46.72%, TCP is 12.42% and CSP is 9.76% more costly in terms of energy.

V. RESULTS

The problems are executed for various sizes: 6 for small problems, the same size as their sequential counterpart; 5 for medium problems, and 4 for large problems. The experiments have been done using various weights for time and energy calibration, to validate the best approach to reduce energy consumption. The chosen weights are: only calibrate using time optimization ($W_e = 0, W_t = 1$), three steps in between ($W_e = 0.25$ and $W_t = 0.75, W_e = 0.50$ and $W_t = 0.50, W_e = 0.75$ and $W_t = 0.25$) and only taking energy consumption in account ($W_e = 1$ and $W_t = 0$).

The case for the KP, in Figure 4, differs depending on the problem size. For the small and medium problem sizes, optimizing for energy and time yields similar results of decreasing energy consumption, obtaining a 20.12% and a 21.97% of energy consumption reduction respectively. For the larger size, however the best result is achieved calibrating for time only, reducing 7.91%, whilst only reducing 1.56% for the energy consumption.

The case for the RAP is particular, since is a problem of irregular nature, and very data dependent. In this problem the load associated to each state increases when the state is more on the right of the table. This makes dynamic partitioning required, as if it was done statically, two pieces of the same size the GPU-1 would imply a load lower than the load assigned to GPU-2. When balancing the load among the different GPUs the piece of work assigned to GPU-1 will be higher in size than the piece of work assigned to GPU-2, getting a more balanced execution that reduces the running time per iteration. The results indicate that for this problem, calibrating using energy is better than not using calibration, reducing consumption by almost 60%, but in this case, due to how different the capabilities are for the GPUs as shown before, the optimal energy calibra-

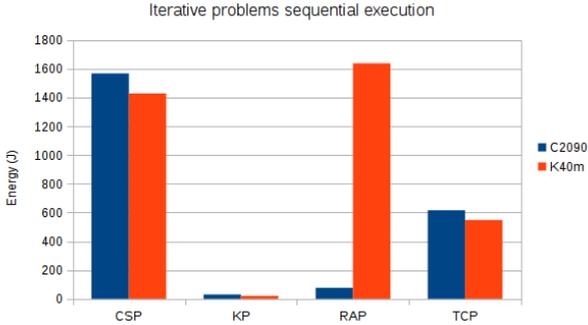


Fig. 3: Energy consumption in Joules for sequential executions of the iterative problems.

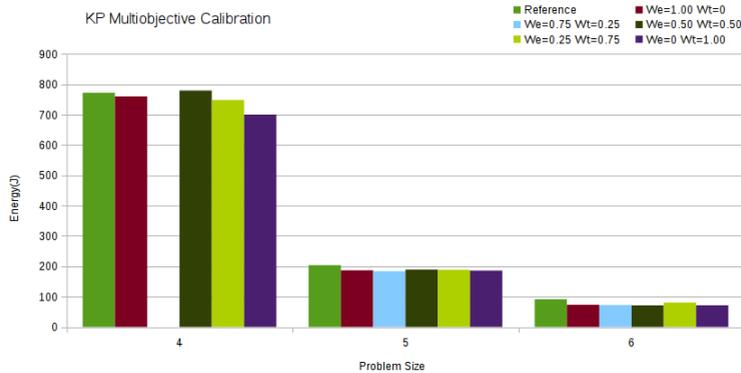


Fig. 4: Energy consumption in Joules for the KP problem. Reference times distributes equally the workload between GPUs. Shows different configurations for the parameters W_e and W_t . Lower is better.

tion is achieved by optimizing energy consumption, leading to the redistribution of all the work to the fastest GPU.

Finally, in the CSP7 we can observe the best reduction in energy consumption for the energy only calibration part with a 4.01% decrease of consumption for the largest problem size. In this case, the time only calibration doesn't reduce energy consumption, but increases it by 1%. This problem has an irregular section where the load of work changes abruptly at each iteration, in this case the library can't calibrate the load until the system becomes more stable, thus the little change of performance (compared to the other problems by using calibration).

VI. CONCLUSION

We have developed a library that allows to perform dynamic load balancing for energy consumption in heterogeneous systems. Our library has been proven to work using iterative problems, but can be easily applied to a wider range of problems with little effort required by the programmer, adding a few lines to the available codes. In the future we will extend the library to allow users a multi-objective approach where energy and time are taken into account so that minimizing energy can be feasible without increasing execution time too much. This will also be mixed with the MPI-2 capabilities of adding processes dynamically to further extend our knowledge in this field.

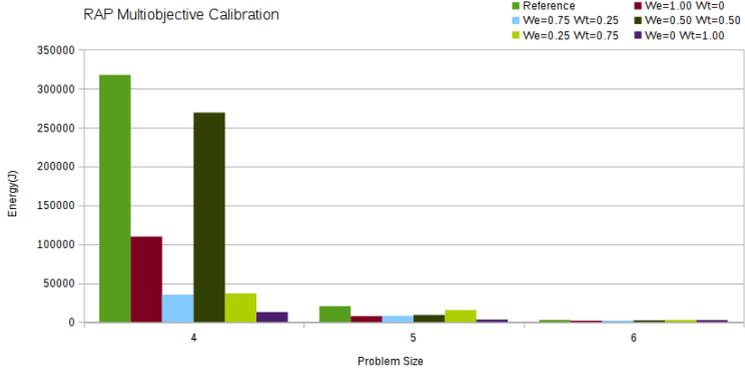


Fig. 5: Energy consumption in Joules for the RAP problem. Reference times distributes equally the workload between GPUs. Shows different configurations for the parameters W_e and W_t . Lower is better.

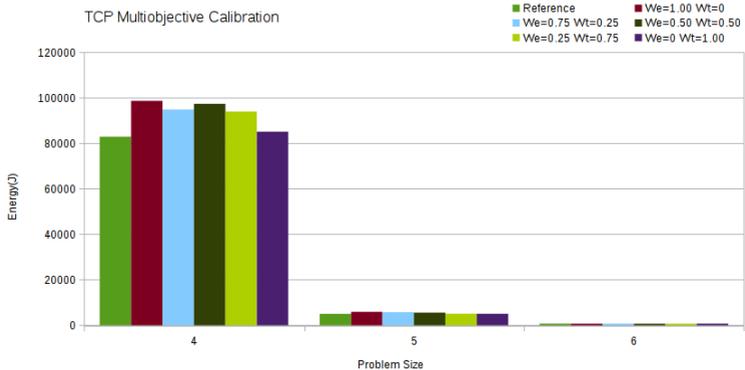


Fig. 6: Energy consumption in Joules for the TCP problem. Reference times distributes equally the workload between GPUs. Shows different configurations for the parameters W_e and W_t . Lower is better.

ACKNOWLEDGMENT

This work was supported by the Spanish Ministry of Education and Science through the TIN2011-24598 project, the Spanish network CAPAP-H4, and the European COST Actions NESSUS and CHIPSET.

REFERENCIAS

[1] Hans Meuer, Erich Strohmaier, Jack Dongarra, and Horst Simon, "Top500 list," <http://www.top500.org/>.
 [2] Jack Dongarra, George Bosilca, Zizhong Chen, Victor Eijkhout, Graham E. Fagg, Erika Fuentes, Julien Langou,

Piotr Luszczek, Jelena Pjesivac-Grbovic, Keith Seymour, Haihang You, and Sathish S. Vadhivar, "Self-adapting numerical software (SANS) effort," *IBM Journal of Research and Development*, vol. 50, no. 2/3, pp. 223–238, March-May 2006.
 [3] Alberto Cabrera, Francisco Almeida, Javier Arteaga, and Vicente Blanco, "Measuring energy consumption using eml (energy measurement library)," *Computer Science - Research and Development*, vol. 30, no. 2, pp. 135–143, 2014.
 [4] Michel Steurer and Sergei Gorlatch, "Skelcl: a high-level extension of opencl for multi-gpu systems," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 25–33, 2014.
 [5] Ricardo Marqu es, Herv e Paulino, Fernando Alexandre, and Pedro D. Medeiros, "Algorithmic skeleton framework

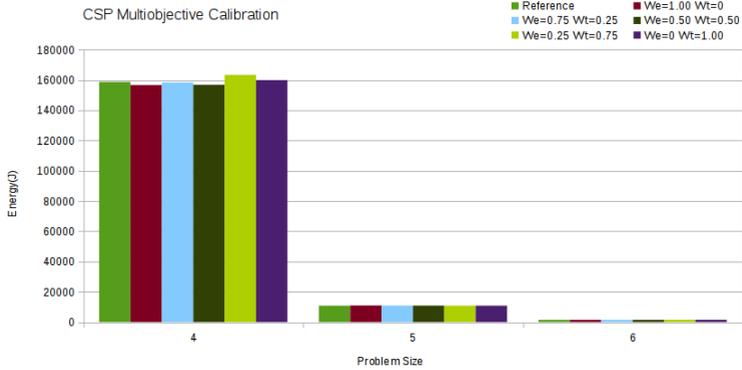


Fig. 7: Energy consumption in Joules for the CSP problem. Reference times distributes equally the workload between GPUs. Shows different configurations for the parameters W_e and W_t . Lower is better.

for the orchestration of GPU computations,” in *Euro-Par 2013 Parallel Processing - 19th International Conference, Aachen, Germany, August 26-30, 2013. Proceedings*, Felix Wolf, Bernd Mohr, and Dieter an Mey, Eds. 2013, vol. 8097 of *Lecture Notes in Computer Science*, pp. 874–885, Springer.

[6] Innovative Computing Laboratory. Univ. Tennessee, “The parallel linear algebra for scalable multi-core architectures (PLASMA) project,” <http://icl.cs.utk.edu/plasma/>, 2011.

[7] The FLAME Project, “Flame: Formal linear algebra methods environment,” <http://z.cs.utexas.edu/wiki/flame.wiki/FrontPage>, 2011.

[8] Innovative Computing Laboratory. Univ. Tennessee, “Matrix algebra on gpu and multicore architectures,” <http://icl.cs.utk.edu/magma/>, 2011.

[9] Emmanuel Agullo, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julien Langou, Hatem Ltaief, Piotr Luszczek, and Stanimire Tomov, “Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects,” *Journal of Physics: Conference Series*, vol. 180, no. 1, pp. 012037, 2009.

[10] Paul Richmond and Daniela Romano, “FLAME: Flexible large-scale agent modelling environment on the GPU,” <http://www.flamegpu.com/>, 2010.

[11] Ernie Chan, Field G. Van Zee, Paolo Bientinesi, Enrique S. Quintana-Ortí, Gregorio Quintana-Ortí, and Robert A. van de Geijn, “Supermatrix: a multithreaded runtime scheduling system for algorithms-by-blocks,” in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming PPOP 2008*, Siddhartha Chatterjee and Michael L. Scott, Eds., Salt Lake City, UT, USA, 2008, pp. 123–132, ACM.

[12] Eduard Ayguadé, Rosa M. Badia, Francisco D. Igual, Jesús Labarta, Rafael Mayo, and Enrique S. Quintana-Ortí, “An extension of the StarS programming model for platforms with multiple GPUs,” in *Euro-Par ’09: Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Berlin, Heidelberg, 2009, pp. 851–862, Springer-Verlag.

[13] J.A. Martine, E.M. Garzón, A. Plaza, and I. García, “Automatic tuning of iterative computation on heterogeneous multiprocessors with aditve,” *The Journal of Supercomputing*, pp. 1–9, 2009, 10.1007/s11227-009-0350-1.

[14] Mateusz Guzek, Dzmitry Klinzovich, and Pascal Bouvry, “HEROS: energy-efficient load balancing for heterogeneous data centers,” in *8th IEEE International Conference on Cloud Computing, CLOUD 2015, New York City, NY, USA, June 27 - July 2, 2015*, Caeton Pu and Ajay Mohindra, Eds. 2015, pp. 742–749, IEEE.

[15] Mateusz Guzek, Sébastien Varrette, Valentin Plugaru, Johnatan E. Pecero, and Pascal Bouvry, “A holistic model of the performance and the energy efficiency of hypervisors in a high-performance computing environment,” *Concurrency and Computation: Practice and Experience*, vol. 26, no. 15, pp. 2509–2590, 2014.

[16] Ibrahim Takoua, Roberto Rojas-Cessa, Kai Sachs, and Christoph Meinel, “Communication-aware and energy-efficient scheduling for parallel applications in virtualized data centers,” in *IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013, Dresden, Germany, December 9-12, 2013*, 2013, pp. 251–255, IEEE.

[17] H. V. Raghun, Sumit Kumar Saurav, and Bindhumadhava S. Bapu, “PAAS: power aware algorithm for scheduling in high performance computing,” in *IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013, Dresden, Germany, December 9-12, 2013*, 2013, pp. 327–332, IEEE.

[18] Edson L. Padoin, Márcio Bastos Castro, Laércio Lima Pilla, Philippe Olivier Alexandre Navaux, and Jean-François Méhaut, “Saving energy by exploiting residual imbalances on iterative applications,” in *21st International Conference on High Performance Computing, HiPC 2014, Goa, India, December 17-20, 2014*, 2014, pp. 1–10, IEEE.

[19] Yu-Liang Chou, Shaoshan Liu, Eui-Young Chung, and Jean-Luc Gaudiot, “An energy and performance efficient DVFS scheme for irregular parallel divide-and-conquer algorithms on the intel SCC,” *Computer Architecture Letters*, vol. 13, no. 1, pp. 13–16, 2014.

[20] A. Acosta, R. Corujo, V. Blanco, and F. Almeida, “Dynamic load balancing on heterogeneous multicore/multi-gpu systems,” in *The 2010 International Conference on High Performance Computing & Simulation, (HPCS 2010)*, Caen, France, June 2010, pp. 467–476, IEEE Computer Society, ISBN: 978-1-4244-6827-0.

Una herramienta de benchmarking para compiladores de OpenACC

Daniel Barba¹, Arturo González-Escribano² y Diego R. Llanos³

Resumen.— OpenACC es un modelo de programación paralela para aceleradores de tipo GPU y Xeon Phi que lleva en desarrollo algunos años. Durante este tiempo han aparecido distintos compiladores, tanto comerciales como de código abierto, que se encuentran aún en un estado temprano de desarrollo. Dado que tanto el estándar como sus implementaciones son relativamente recientes, disponer de una suite de benchmarks diseñada para soportar varios compiladores puede facilitar enormemente el análisis comparativo de rendimiento de los distintos códigos generados. En este artículo presentamos *TORMENT OpenACC*, una suite de benchmarks preparada para comparar tanto arquitecturas como el código generado por diferentes compiladores. Junto a esta herramienta hemos desarrollado unas métricas adecuadas para la comparación del rendimiento de los pares compilador-máquina y que hemos denominado *TORMENT_ACC Score*. Aquí se presenta la versión 0.91 de la suite, desarrollada como prueba de concepto, y que incluye dos benchmarks. La versión 1.0, considerada estable, incluirá seis benchmarks y estará disponible próximamente.

Palabras clave.— OpenACC, compiladores, benchmarking.

I. INTRODUCCIÓN

OpenACC es un estándar abierto que define una serie de directivas de compilación o pragmas para ejecución de fragmentos de código en aceleradores de tipo GPU y Xeon Phi. Su objetivo es facilitar la paralelización de código secuencial en este tipo de aceleradores reduciendo el tiempo necesario tanto en la programación como en el aprendizaje [1]. La especificación se encuentra en su versión 2.5 [2].

La responsabilidad de la paralelización automática del código secuencial recae sobre los diferentes compiladores que soportan OpenACC: PGI Compiler [3], de *The Portland Group*, empresa ahora perteneciente a Nvidia, es (según los estudios que hemos realizado) el compilador con un grado de madurez más alto. Se encuentra disponible como parte del *Nvidia OpenACC Toolkit*, gratuito durante tres meses y con la posibilidad de adquirir licencias comerciales o de investigación. OpenUH [4], de la Universidad de Houston y accULL [5] de la Universidad de La Laguna son dos alternativas de código abierto desarrolladas en el ámbito académico y que pueden descargarse libremente para su uso.

Además de los compiladores señalados, existen otros compiladores que soportan OpenACC, pero debido a que en la fecha de redacción de este artículo

fue imposible obtener licencias de prueba o académicas, han sido dejados fuera de nuestros estudios. Estos compiladores están siendo desarrollados por *CRAY Inc.* y *Pathscale Inc.*

Nuestro trabajo ha consistido en el desarrollo de una herramienta de análisis de rendimiento del código generado por estos compiladores, que hemos denominado *TORMENT OpenACC* (Trasgo performance and EvaluatioN Tool for OpenACC). El motivo que nos ha movido a comenzar este trabajo es la escasez de herramientas similares para OpenACC y las carencias que presentan las existentes. Estas herramientas son, a fecha de redacción de este artículo, EPCC OpenACC Benchmark Suite [6] y Rodinia [7]. El primero ha sido desarrollado por el *Edinburgh Parallel Computing Centre* y consiste en una serie de microbenchmarks destinados a medir el overhead de la implementación de los distintos pragmas, y benchmarks para medir el rendimiento del código generado. Rodinia [7] es una traducción de los benchmarks originales de la suite del mismo nombre [8] para OpenACC desarrollado por *Pathscale*.

Nuestro trabajo busca dar respuesta a la necesidad de análisis del rendimiento y comparación del código generado por los distintos compiladores, intentando mantener un equilibrio entre las fortalezas y debilidades de los diferentes compiladores y tratando de obtener una herramienta que sea posible utilizar con los compiladores disponibles para la comunidad académica. Debido a que los compiladores se encuentran aún en fase de desarrollo, es imprescindible mantener el código de los benchmarks lo más sencillo posible.

Además del desarrollo de la herramienta como tal, hemos desarrollado también una métrica para poder ofrecer al usuario de esta herramienta una puntuación relativa que permita la comparación entre diferentes sistemas y compiladores analizados. Esta métrica ha sido denominada *TORMENT_ACC Score*.

El resto del artículo se organiza del siguiente modo: La sección II describe con más detenimiento las herramientas de benchmarking existentes. La sección III describe los compiladores actualmente soportados por *TORMENT OpenACC*. En la sección IV se describe la herramienta, sus objetivos y características a medir y enumera los benchmarks actualmente implementados, así como la métrica utilizada. Finalmente, la sección V concluye el artículo.

II. HERRAMIENTAS EXISTENTES

Como se indicaba en la Introducción, a fecha de redacción de este artículo existen dos herramientas

¹Dpto. de Informática, Universidad de Valladolid, España. e-mail: daniel@infor.uva.es.

²Dpto. de Informática, Universidad de Valladolid, España. e-mail: arturo@infor.uva.es.

³Dpto. de Informática, Universidad de Valladolid, España. e-mail: diego@infor.uva.es.

de análisis de rendimiento para OpenACC. A continuación describimos más en profundidad las mismas y las carencias detectadas.

A. EPCC OpenACC Benchmark Suite

La suite de benchmarks desarrollada por el EPCC ha sido diseñada específicamente para OpenACC. Se compone de tres partes diferenciadas y que se denominan *Nivel 0*, *Nivel 1* y *Nivel de Aplicaciones*.

En el *Nivel 0* se pueden encontrar una serie de microbenchmarks cuyo objetivo es medir el overhead generado por las implementaciones de los pragmas. Estos microbenchmarks dan un resultado que es la diferencia de dos tiempos de ejecución en función del pragma que están analizando, o bien el tiempo de transferencia de datos en una directiva de tipo `#pragma acc data`. Estos resultados son interesantes para el desarrollo de los compiladores, pero tal y como están diseñados no ofrecen una idea clara de rendimiento y los resultados pueden ser difíciles de interpretar.

En el *Nivel 1* se encuentran un conjunto de benchmarks típicos de tipo BLAS basados en *Polybench* y *Polybench/GPU* [9]. Los resultados ofrecidos por estos benchmarks son tiempos de ejecución de los diferentes códigos, por lo que son un buen indicador del rendimiento del código generado por los diferentes compiladores.

En el *Nivel de Aplicaciones* hay tres benchmarks de mayor entidad que los anteriores. Estos benchmarks también ofrecen tiempos de ejecución, pero al no ser problemas sintéticos dan resultados más interesantes.

En general, la idea del *EPCC OpenACC Benchmark Suite* está bien planteada y diseñada. No obstante, los resultados obtenidos en los microbenchmarks no son adecuados para un análisis de rendimiento. Los demás benchmarks, si bien podrían ser útiles, siempre han dado problemas en nuestros estudios. En unos casos no podían compilarse con algunos de los compiladores utilizados, en otros casos daban error de ejecución. Uno de los mayores problemas ha sido la limitación en los tamaños de los datos a utilizar debido a problemas en la implementación que llevaban a que se intentase asignar mucha más memoria de la deseada, originando errores en la ejecución y limitando a 10MB el tamaño de los datos.

B. Rodinia para OpenACC

La empresa *Pathscale Inc.* ha desarrollado una versión [7] de la suite de benchmarks Rodinia [8], [10] para su uso con compiladores de OpenACC. Con la versión existente en GitHub a día 25 de Abril de 2014 (versión más reciente durante el desarrollo de nuestro trabajo), en general los benchmarks disponibles no compilan con ninguno de los compiladores existentes, salvo contadas excepciones. El compilador de PGI es el único que logra compilar un número significativo de benchmarks.

La suite se compone de benchmarks que devuelven el tiempo de ejecución, por lo que serían un buen

punto de partida para un análisis de rendimiento. Por desgracia, la escasa madurez del código y la poca compatibilidad con los compiladores disponibles hacen imposible su uso para establecer una comparativa de rendimientos.

III. COMPILADORES SOPORTADOS

En la Introducción hemos enumerado brevemente los compiladores disponibles. Su elección está motivada por la existencia de licencias gratuitas o académicas, o bien por ser de código abierto. A continuación explicaremos algunos detalles de los compiladores soportados en la versión preliminar de *TORMENT OpenACC*.

A. PGI Compiler

El compilador de PGI [3], desarrollado por *The Portland Group* y Nvidia, es a fecha de redacción de este artículo el compilador con un grado de madurez más alto. Su uso está muy extendido en los diferentes talleres y conferencias que se realizan sobre OpenACC. Actualmente está disponible como parte del *Nvidia OpenACC toolkit*, que incluye una licencia gratuita de tres meses y la posibilidad de adquirir una licencia comercial o académica. En nuestros estudios, el compilador de PGI ha demostrado ser el más sólido y el que más alto grado de madurez tiene. Se ajusta bien a la especificación de OpenACC. Su instalación es simple y dispone de abundante documentación.

B. OpenUH

El compilador OpenUH, desarrollado por la Universidad de Houston, es una alternativa de código abierto. Comparado con el compilador de PGI, su madurez y solidez son menores y carece de algunas funcionalidades, como reducciones sobre mínimos o máximos, lo cual dificulta ligeramente la programación de aplicaciones. Su instalación, a fecha de redacción de este artículo, no es tan sencilla como cabría esperar. La versión pre-compilada disponible en su web [4] carece de algunas librerías que deben ser compiladas aparte u obtenidas de otra forma.

C. accULL

El compilador accULL [5], desarrollado por la Universidad de La Laguna, es (al igual que OpenUH) un compilador de código abierto. De los tres compiladores soportados, es el que tiene una menor robustez, teniendo problemas para la traducción fuente a fuente de algunas características de C, como punteros a función, o de funcionalidades de OpenACC, como algunos tipos de reducciones. La instalación de la versión disponible en su web [11] es bastante simple, siguiendo las instrucciones que pueden encontrarse adjuntas al compilador en ficheros de texto.

IV. TORMENT OPENACC

Nuestra propuesta en desarrollo se denomina *TORMENT OpenACC* y es el acrónimo para *Trasgo performance and EvaluationN Tool for OpenACC*.

El proyecto ha nacido para intentar dar una solución a la necesidad de dar respuesta a los intentos de realizar comparativas de rendimiento de código OpenACC.

A. Motivación

Las herramientas actuales han demostrado en nuestros estudios previos no ser suficientes para dar una respuesta sencilla a la evaluación del código generado por los diferentes compiladores de OpenACC.

El primero de los problemas es la escasa compatibilidad del código de los benchmarks entre los diferentes compiladores. Es cierto que si el código se ajusta al estándar debería compilarse y ejecutarse correctamente en cualquier implementación del estándar OpenACC, pero en la actual etapa en la que se encuentran tanto el estándar como los compiladores esto no se cumple. Debido a esto, hemos desarrollado *TORMENT OpenACC* teniendo en cuenta tanto la especificación del estándar como las capacidades actuales de los diferentes compiladores.

Otro problema detectado es que muchos de los benchmarks no ofrecen una imagen clara de rendimiento, como es el caso de los microbenchmarks del *EPCC OpenACC Benchmark Suite*. Si bien es cierto que estos microbenchmarks son útiles para el desarrollo de los compiladores, creemos que el overhead de la implementación de los distintos pragmas no es realmente relevante en una comparativa de rendimientos. Por este motivo, en nuestra propuesta dejamos fuera ese tipo de pruebas.

Finalmente, hemos observado también que, en muchos casos, benchmarks que parecen compilar correctamente luego generan errores en tiempo de ejecución o resultados incorrectos. Esta última situación puede suponer que, si no se detecta el resultado erróneo, los resultados de rendimiento obtenidos sean también incorrectos. Para evitar esto, además de incorporar comprobación de resultados a los benchmarks, también hemos analizado los resultados durante el desarrollo de nuestra propuesta, para que los benchmarks que incorporamos en nuestra herramienta compilen y ejecuten correctamente con cualquiera de los compiladores.

Otro aspecto importante que hemos tenido en cuenta a la hora de llevar a cabo el desarrollo de *TORMENT OpenACC* es la ofrecer a la comunidad una herramienta que, además de facilitar el análisis de rendimiento del código generado por los compiladores de OpenACC ejecutados en distintas máquinas, ofrezcan una medida del rendimiento que permita una comparación fácil entre máquinas y compiladores. Esta idea nos ha llevado al desarrollo de las métricas *TORMENT_ACC Score* que describiremos posteriormente.

B. Objetivos

El objetivo principal de *TORMENT OpenACC* es la de permitir un análisis de rendimiento de código OpenACC generado por los distintos compiladores de forma sencilla, generando un resumen de resulta-

dos fácilmente analizable y ofreciendo unas métricas, denominadas *TORMENT_ACC Score*, que permiten la comparación de los pares máquina-compilador.

Nuestra propuesta pretende facilitar a la comunidad una herramienta preparada específicamente para OpenACC que sea consciente del estado de desarrollo temprano de los compiladores existentes, de forma que se eviten problemas en compilación o ejecución como sucede con otras herramientas de benchmarking existentes. *TORMENT OpenACC* estará preparado para compilarse y ejecutarse con la menor intervención posible del usuario. Los scripts que acompañan a la suite recopilan la información del proceso y ofrecen finalmente un informe en formato HTML con los datos relevantes.

Además, *TORMENT OpenACC* utiliza los compiladores GCC y NVCC para obtener datos de ejecuciones de código secuencial y código CUDA respectivamente. De este modo, nuestra propuesta ofrece al usuario información del *speedup* con respecto al código secuencial y CUDA ejecutado en la misma máquina.

C. Estructura de la herramienta

TORMENT OpenACC se compone de una serie de *scripts* que se encargan de todo el proceso de compilación, ejecución y obtención de resultados, eliminando esta carga al usuario. Estos *scripts* se dividen en tres categorías. Los scripts de configuración guían al usuario en la obtención de las rutas correctas a librerías CUDA (necesarias por algunos de los compiladores), rutas de compiladores y comandos de ejecución (en caso de que el usuario utilice, por ejemplo, sistemas de colas tipo *slurm*). El *script* de ejecución se encarga de la compilación y ejecución usando todos los compiladores que hayan sido hallados en el sistema del usuario. Finalmente, el *script* de generación de resultados procesa los resultados obtenidos y genera un fichero HTML con el informe final.

Cada benchmark está desarrollado de modo que sea lo más sencillo posible, para evitar problemas de compilación. Por este motivo se procura evitar el uso de niveles de indirección que en otras situaciones serían aconsejables. Cada benchmark está definido en su propia unidad de compilación y el código OpenACC y CUDA está incorporado en el mismo fuente, utilizando compilación condicional y evitando tener de forma simultánea ficheros *.c* y *.cu* con código duplicado. Los ficheros *.cu* son necesarios para que el compilador NVCC genere el código CUDA correspondiente, pero esto se ha resuelto mediante el uso de enlaces simbólicos a los ficheros *.c* correspondientes.

El programa principal se encarga del lanzamiento de los benchmarks. Cada uno se lanza con diez repeticiones, con una repetición extra al comienzo cuyos resultados son desechados. Al finalizar estas diez repeticiones se obtienen los valores denominados *peak* y *average* y que corresponden a la mejor de las ejecuciones y a la media aritmética de todas ellas. Estos valores servirán para calcular la métrica de la que

```

CUDA_ATTR__ int getRandom(unsigned int* seed)
{
    unsigned int next = *seed;
    int result;

    next += 1103515245;
    next += 12345;
    result = (unsigned int) (next/65536) % 2048;

    next += 1103515245;
    next += 12345;
    result <= 10;
    result ^= (unsigned int) (next/65536) % 1024;

    next += 1103515245;
    next += 12345;
    result <= 10;
    result ^= (unsigned int) (next/65536) % 1024;

    *seed = next;
    return result;
}
    
```

Fig. 1

CÓDIGO PARA LA GENERACIÓN DE NÚMEROS ALEATORIOS.

hablaremos posteriormente.

D. Benchmarks implementados

La versión preliminar de *TORMENT OpenACC* contiene únicamente dos benchmarks. El desarrollo de la herramienta sigue en proceso y otros benchmarks se irán añadiendo progresivamente.

D.1 MonteCarloPi

Este benchmark consiste en una aproximación de Pi por el método de Monte Carlo, que se basa en la generación de puntos aleatorios en un cuadrado de lado unitario. Se comprueba si estos puntos se encuentran dentro de un cuarto de círculo de radio unitario y se acumula el total de puntos que cumplen dicha condición. Finalmente, se aplica la siguiente fórmula:

$$\pi \approx \frac{4 * P}{T}$$

Donde P es el número de puntos dentro del cuarto de círculo y T es el total de puntos generados.

MonteCarloPi es un benchmark que no tiene prácticamente transferencias de memoria y que realiza un cálculo computacional muy simple, pero puede ser optimizado en CUDA haciendo que cada hilo calcule varios puntos y utilizando la *shared memory* de los bloques para evitar accesos a memoria global. Un buen resultado de los compiladores en este benchmark dependerá de estos factores.

Dado que no se puede hacer uso de la función `rand` de C en el código ejecutado en la GPU, y el uso de la librería `curand` se limita a código CUDA, hemos decidido replicar la función `rand` para permitir su ejecución en la GPU, como se indica en la Fig. 1.

El código utilizado para las versiones de OpenACC y CUDA se muestra en las Figs. 2 y 3.

```

#pragma acc parallel loop \
    private(i, d, x, y, seed) \
    reduction(+:count)
for(i = 0; i < COORD_NUM; ++i){
    seed = 1987 ^ i*27;
    x = (double)getRandom(&seed)/(double)RAND_MAX;
    y = (double)getRandom(&seed)/(double)RAND_MAX;

    d = sqrt(x*x + y*y);
    if(d <= 1.0){
        ++count;
    }
}
    
```

Fig. 2

CÓDIGO DE MONTECARLOPI PARA OPENACC.

```

__CUDA_GLOBAL__ void piKernel(
    const unsigned long int triesPerThread,
    unsigned long int* hits)
{
    unsigned int seed;
    int gid, tid, bid;
    int lhits;
    float x, y;
    extern __shared__ unsigned long int sdata[];

    gid = (blockIdx.x*blockDim.x) + threadIdx.x;
    tid = threadIdx.x;
    bid = blockIdx.x;
    lhits = 0;

    seed = 1987 ^ gid*27;

    for (int i = 0; i < triesPerThread; ++i){
        x = (float)getRandom(&seed)/(float)RAND_MAX;
        y = (float)getRandom(&seed)/(float)RAND_MAX;

        float d = sqrt(x*x + y*y);
        if (d <= 1.0f){
            ++lhits;
        }
    }
    sdata[tid] = lhits;
    __syncthreads();
    if (tid == 0){
        for (int i = 1; i < blockDim.x; ++i){
            lhits += sdata[i];
        }
    }

    hits[bid] = lhits;
}
    
```

Fig. 3

CÓDIGO DE MONTECARLOPI PARA CUDA.

benchmarks [13].

Nuestra propuesta sigue una idea similar a la de SPEC, pero con algunas variaciones. En primer lugar, la ejecución de los benchmarks que componen *TORMENT OpenACC* devuelven tres valores. *Peak Time* es el mejor tiempo de ejecución obtenido, medido en segundos. *Average Time* es el tiempo medio de todas las ejecuciones del benchmark, también en segundos. Finalmente, *Standard Deviation* es la desviación estándar del conjunto de medidas e indica la variabilidad obtenida en las ejecuciones del benchmark. Con los tiempos *peak* y *average* se calcula un ratio con respecto a los tiempos de referencia suministrados con la herramienta, y que son los tiempos de ejecución secuencial del benchmark en una máquina de referencia. Una vez ejecutados todos los benchmarks, se obtiene la media armónica de todos los ratios, tanto para *peak time* como *average time*. Estos valores que se obtienen son *TORMENT_ACC Peak Score* y *TORMENT_ACC Average Score*.

La principal diferencia entre *TORMENT OpenACC* y SPEC, aparte de la metodología de toma de tiempos de ejecución, consiste en el uso de la media armónica en lugar de la media geométrica. Esta decisión se fundamenta en el hecho de que, para los objetivos de *TORMENT OpenACC*, la media armónica es más adecuada que la media geométrica. En primer lugar, aunque la media geométrica siempre produce una ordenación consistente, no necesariamente produce la ordenación correcta [13], ya que esta media no es inversamente proporcional al tiempo de ejecución. En cambio, la media armónica si que es inversamente proporcional al tiempo de ejecución, lo que hace que sea una media correcta para expresar ratios. Estas afirmaciones son compartidas por otros trabajos, como por ejemplo [14].

Como ejemplo de uso, las figuras 6 y 7 muestran dos informes generados para sistemas con diferente configuración. En el primero puede verse que el compilador de PGI obtiene el *TORMENT_ACC Score* más alto, tanto *Peak* como *Average*, con un valor de 28.56 y 25.71 respectivamente. Los resultados con respecto a la implementación en CUDA de los benchmarks, muestran que los tres compiladores generan código cuyo rendimiento queda muy lejos del conseguido usando CUDA. En el segundo puede observarse que a pesar de utilizar una GPU mucho más modesta en comparación, los resultados son consistentes con los del primer informe. En ambos resultados se aprecia que el compilador de PGI obtiene los valores más grandes de la desviación típica. Esto parece indicar que al generar el código se está modificando la semántica del programa y la primera ejecución de cada benchmark que debería ser desechada se está incluyendo en las medidas.

V. CONCLUSIONES Y TRABAJO FUTURO

TORMENT OpenACC es una herramienta de análisis y comparación de rendimientos de código generado por compiladores de OpenACC, teniendo

en consideración el nivel de madurez de tanto el estándar OpenACC como de los diferentes compiladores. *TORMENT OpenACC* desarrolla una suite de benchmarks específicamente diseñados para OpenACC y manteniendo la máxima portabilidad entre compiladores, permitiendo su compilación y ejecución en todos ellos.

Los resultados ofrecidos por *TORMENT OpenACC* incluyen la denominada *TORMENT_ACC Score*, diseñada para la comparación de pares máquina-compilador. Además, se ofrece un resumen de la ejecución de los benchmarks con una tabla de tiempos y ratios, tanto mínimos como medios e incluyendo la desviación estándar para un análisis de variabilidad de los tiempos de ejecución del código generado.

Junto con los resultados de los compiladores de OpenACC se incluyen también resultados de ejecución de código generado por los compiladores GCC y NVCC para código secuencial y CUDA. De este modo, se puede ofrecer al usuario una comparativa a nivel de máquina del rendimiento del código generado por los compiladores de OpenACC con respecto a versiones secuenciales y CUDA de los benchmarks.

El trabajo futuro a desarrollar consiste en la ampliación de la suite de benchmarks, para lograr resultados que sean verdaderamente relevantes. Aquí se presenta la versión 0.91 de la suite, desarrollada como prueba de concepto, y que incorpora dos benchmarks. La versión 1.0, considerada estable, incluirá seis benchmarks y estará disponible próximamente. Tratando de cubrir los aspectos más interesantes de la ejecución de código en las GPUs. Además, una parte importante del trabajo restante consiste en mantener la compatibilidad entre compiladores y asegurar el correcto funcionamiento de la herramienta en distintas máquinas.

AGRADECIMIENTOS

En memoria de Agustín de Dios Hernández.

Esta investigación ha sido parcialmente financiada por el MICINN y el programa ERDF de la Unión Europea: proyecto HomProg-HetSys (TIN2014-58876-P), la red CAPAP-H5 (TIN2014-53522-REDT) y el COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

REFERENCIAS

- [1] OpenACC-standard.org, "About OpenACC."
- [2] OpenACC-standard.org, "OpenACC 2.5 draft for public comment," oct 2015.
- [3] PGI, "Pgi accelerator compilers with OpenACC directives," <https://www.pgroup.com/resources/accel.htm>, nov 2015.
- [4] University of Houston, "Open-source UH compiler," <http://web.cs.uh.edu/~openmh/download/>, nov 2015.
- [5] Ruymán Reyes, Iván López-Rodríguez, Juan J Fumero, and Francisco de Sande, "accULL: an OpenACC implementation with CUDA and OpenCL support," in *Euro-Par 2012 Parallel Processing*, pp. 871–882. Springer, 2012.
- [6] EPCC, "Epc OpenACC benchmark suite," <https://github.com/EPCCed/epcc-openacc-benchmarks>, sep 2013.
- [7] Pathscale, "Rodinia benchmark suite 2.1 with OpenACC

- port," <https://github.com/pathscale/rodinia>, apr 2014.
- [8] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. (HISWC), 2009 IEEE International Symposium on*. IEEE, 2009, pp. 44–54.
- [9] Louis-Noël Pouchet, "Polybench: The polyhedral benchmark suite," URL: [http://www.cs.ucla.edu/~pouchet/software/polybench/\[cited July,\]](http://www.cs.ucla.edu/~pouchet/software/polybench/[cited July,]), 2012.
- [10] Shuai Che, Jeremy W Sheaffer, Michael Boyer, Lukasz G Szafaryn, Liang Wang, and Kevin Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *Workload Characterization (HISWC), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–11.
- [11] Universidad de La Laguna, "accÜLL," <http://cap.pcg.u11.es/es/accÜLL>, nov 2015.
- [12] Kaivalya M Dixit, "The spec benchmarks," *Parallel computing*, vol. 17, no. 10, pp. 1195–1209, 1991.
- [13] David J Lilja, *Measuring computer performance: a practitioner's guide*, Cambridge University Press, 2005.
- [14] John R Mashey, "War of the benchmark means: time for a truce," *ACM SIGARCH Computer Architecture News*, vol. 32, no. 4, pp. 1–14, 2004.

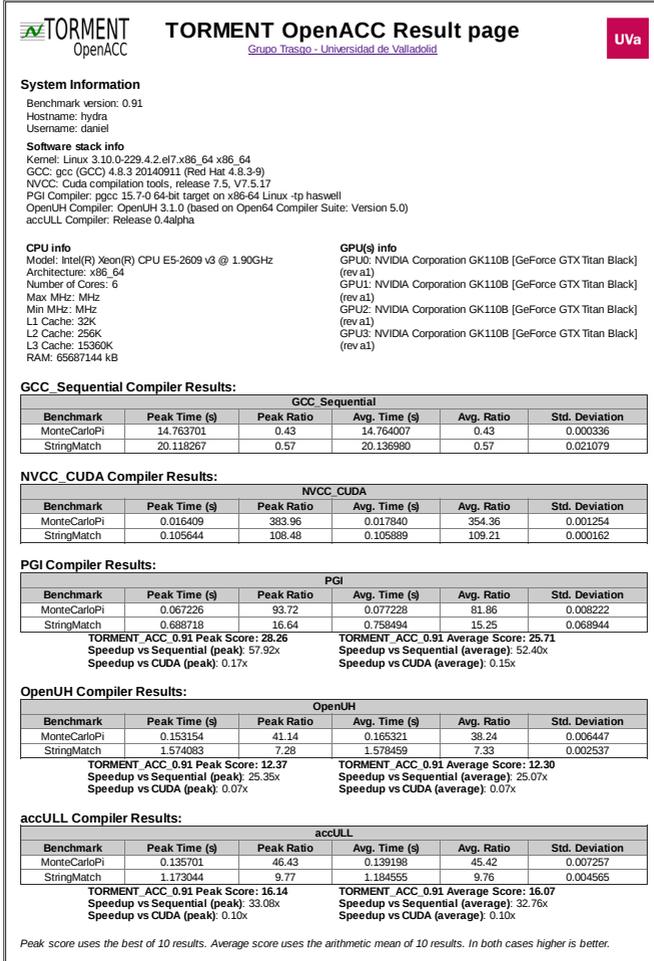


Fig. 6

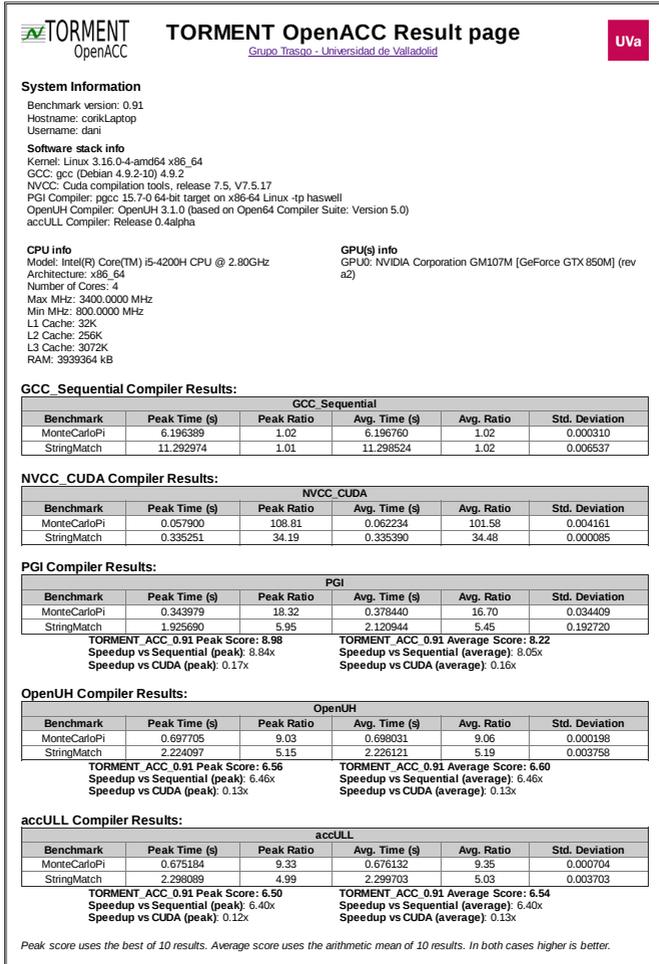


Fig. 7

Propuesta arquitectónica para la ejecución de tareas en Apache Spark para entornos heterogéneos

Estefanía Serrano¹, Javier García Blas¹, Jesús Carretero¹ y Mónica Abella^{2, 3}

Resumen— Las desventajas presentes en las plataformas de computación actuales y la fácil migración a la computación en la nube, han logrado que cada vez más aplicaciones científicas se adapten a los distintos *frameworks* de computación distribuida basadas en flujo de tareas. Sin embargo, muchas de ellas ya han sido optimizadas para su ejecución en aceleradores tales como GPUs. En este trabajo se presenta una arquitectura que facilita la ejecución de aplicaciones tradicionalmente basadas en entornos HPC al nuevo paradigma de computación Big Data. Además, se demuestra como gracias a una mayor capacidad de memoria, el reparto automático de tareas y a la mayor potencia de cálculo de los sistemas heterogéneos se puede converger a un nuevo modelo de ejecución altamente distribuido. En este trabajo se presenta un estudio de la viabilidad de esta propuesta mediante la utilización de GPUs dentro de la infraestructura de cómputo Spark. Esta arquitectura será evaluada a través de una aplicación de tratamiento de imagen médica. Los resultados demuestran que aunque nuestra arquitectura sobre un nodo no produce resultados absolutos mejores que la aplicación original, según se aumenta el número de GPUs y por lo tanto la ocupación de estas influye más, la aplicación basada en Spark se acerca al rendimiento del simulador original. Finalmente, realizamos un estudio de la ocupación de las GPUs empleadas para las distintas políticas propuestas, demostrando que al tener en cuenta las características dinámicas de las GPUs (número de tareas en ejecución) podemos tener una mayor ganancia de rendimiento.

Palabras clave— GPU, Spark, pyCUDA, imagen médica.

I. INTRODUCCIÓN

La utilización cada vez más frecuente de *frameworks* de computación distribuida creados originalmente para problemas Big Data en aplicaciones científicas, está llevando a la adaptación de estas aplicaciones al paradigma MapReduce [1], el cual es, actualmente, el paradigma más usado por este tipo de *frameworks* en las plataformas cloud. La dificultad que a veces conlleva esta adaptación y, en ocasiones, la pérdida de eficiencia debido a las dependencias entre los datos, hace que muchas aplicaciones no sean migradas a este nuevo paradigma.

En el caso de aplicaciones científicas, como en el área de trabajo de la imagen médica, tradicionalmente se ha optado por la utilización de aceleradores de cómputo como las GPUs (de las siglas en inglés, Graphics Processing Unit). Estos aceleradores, debi-

do a su arquitectura SIMD (de las siglas en inglés, Single Instruction Multiple Data) permiten la ejecución rápida de la mayoría de los algoritmos de reconstrucción y proyección, debido a que la generación de cada uno de los píxeles finales es totalmente independiente. El principal problema relacionado con el uso de este tipo de hardware es la falta de memoria dentro del dispositivo, debido a que la mayoría de tarjetas gráficas de bajo y mediano coste poseen alrededor de 3GB de memoria, típicamente GDDR5. Aunque es cierto que en estos momentos muchas tarjetas dedicadas al cómputo, como por ejemplo la gama Tesla de NVidia, pueden llegar a tener más de 10 GB de memoria, esto puede ser insuficiente para la futura generación de imágenes en alta resolución, la cual podría llegar a requerir 32 GB de almacenamiento en memoria principal. Una posible solución a este problema de limitación de memoria es el particionamiento del problema para su resolución independiente en diferentes GPUs. Sin embargo, la tarea de particionamiento no es sencilla, ya que en muchas ocasiones se requiere de modificaciones significativas en el código original.

Las ventajas de la computación distribuida en términos de capacidad de memoria son evidentes. La memoria en estos entornos se multiplica, así como la capacidad computacional. Adicionalmente, en clústeres de computadores altamente heterogéneos es posible disponer de nodos con múltiples GPUs. Aunque existen entornos de computación distribuida como MPI, en los que ya se ha estudiado la viabilidad de ejecución de estos algoritmos [2], estos están principalmente orientados a aplicaciones científicas. Sin embargo, cuando hablamos de problemas científicos altamente paralelizables, la aplicación del paradigma MapReduce puede ser de gran ayuda. Este paradigma facilita el particionamiento automático de los datos, la ejecución basada en tareas y el aumento de la localidad de datos.

La principal contribución de este trabajo es el estudio de la viabilidad de la migración de aplicaciones inicialmente basadas en plataformas HPC a plataformas Big Data. Como caso de uso, se partirá de una aplicación de imagen médica, llamada Fux-Sim [3]. Las contribuciones de este trabajo se aplican no sólo a la posibilidad de ejecutar aplicaciones dentro del ámbito de la imagen médica en entornos Big Data, sino también a la generalización del proceso llevado a cabo aquí a cualquier campo de aplicación que haga un uso intensivo de GPUs. En resumen, las contri-

¹Dpto. de Informática, Univ. Carlos III de Madrid, e-mail: esserran@inf.uc3m.es.

²Dpto. de Ingeniería Biomédica y Aeroespacial, Univ. Carlos III de Madrid

³Instituto de Investigación Sanitaria Gregorio Marañón (IISGM), Madrid

buciones de este trabajo son las siguientes: primero, se presenta una solución para proporcionar acceso a GPUs dentro de las tareas generadas por Apache Spark; segundo, se detalla la metodología seguida para la migración de una aplicación inicialmente implementada en una plataforma HPC a un sistema Big Data; por último, se realiza una evaluación comparativa de la solución propuesta.

Este documento se organiza de la siguiente manera. En la Sección II se explicarán los fundamentos básicos de Apache Spark, se fundamentará la elección de las tecnologías utilizadas y se definirá el procedimiento llevado a cabo con nuestra aplicación de forma generalizada. La Sección IV detalla la funcionalidad del caso de estudio así como los algoritmos contenidos en sus kernels. A continuación, se llevará a cabo la evaluación de la propuesta en la Sección V. Finalmente, se proporcionará una relación de los trabajos relacionados y las conclusiones de este trabajo.

II. CONCEPTOS PREVIOS

Esta sección introduce los conceptos y herramientas en las que se sustenta la propuesta. Para ello se describirá brevemente la arquitectura del *framework* Apache Spark y la interfaz de acceso a GPU, PyCUDA.

A. Apache Spark

Apache Spark [4] es un *framework* de computación distribuida de propósito general. Utilizado en diversos ámbitos, su mayor éxito lo ha tenido en aplicaciones de análisis de datos en entornos Big Data. Posee APIs para programar en distintos lenguajes como Scala, Java, Python y además cuenta con librerías enfocadas al aprendizaje automático.

Basado en el paradigma MapReduce y con una ejecución enfocada a tareas, Spark es compatible con varios gestores de recursos (por ejemplo Yarn) y tiene conectores para distintos sistemas de ficheros y bases de datos distribuidas. A diferencia de Hadoop, no sólo se reduce a implementar el paradigma MapReduce sino que ofrece muchas más posibilidades de manipulación de datos. Todas las operaciones disponibles se dividen en transformaciones y acciones sobre unas estructuras de datos llamadas *Resilient Distributed Datasets* (RDDs). Estas estructuras contienen los datos sobre los que se va a trabajar y están distribuidas sobre los diversos nodos favoreciendo la localidad de datos a la hora de computar, ya que pueden mantenerse dentro de la memoria principal del nodo.

La arquitectura de Spark incluye dos actores fundamentales: controlador (*driver*) y trabajador (*worker*). El *driver* es el encargado de lanzar la aplicación y de su manejo dentro del clúster de Spark. Para ello se comunicará con el gestor de recursos elegido. El *worker* es el proceso principal de cada nodo, encargado de lanzar los diversos ejecutores (*executors*), que serán los que lleven a cabo las distintas tareas.

B. PyCUDA

En este trabajo, se ha considerado Python como lenguaje de programación por su utilización en computación científica para prototipado. Además se tiene en cuenta la gran cantidad de módulos científicos disponibles. Estos módulos poseen la mayoría de funciones matemáticas utilizadas en este tipo de aplicaciones, con el añadido de que están altamente optimizadas gracias a la utilización del lenguaje C. Aún así, el proceso descrito posteriormente podría aplicarse a cualquier otro lenguaje siempre que posea *bindings* para controladores de GPUs. En este caso, para la conexión con los aceleradores GPU, se ha utilizado PyCUDA, un módulo que posee diversas funciones que ponen a disposición del programador de Python la API del driver CUDA [5]. Su mayor ventaja es la posibilidad de reutilización de las funciones ya programadas para CUDA, de ahora en adelante *kernels*, previamente programados en C/C++.

III. PROPUESTA DE ARQUITECTURA PARA UN CLÚSTER HETEROGÉNEO DE SPARK

Como se puede ver en las Figuras 1 y 2, desde el punto de vista de la ejecución de la aplicación hay tres componentes principales: el *driver* de la aplicación, los *workers* en cada uno de los nodos del clúster y su correspondiente *GPU scheduler*.

El *driver* es el actor encargado de ejecutar el código principal de la aplicación incluyendo la gestión de los datos. En el caso de usar un sistema de ficheros distribuido como HDFS, cada uno de los nodos será el encargado de adquirir aquellos datos más cercanos, favoreciendo la localidad de datos en el proceso de ejecución. El *driver* es independiente de la ejecución de las tareas, por lo que no es necesaria la presencia de GPUs en el nodo desde el cual se ejecuta. Esto permite la ejecución de tareas que hacen un uso exclusivo de la GPU en todos aquellos nodos que la posean, pudiendo aprovechar nodos no heterogéneos para el lanzamiento del *driver*.

En el *executor* se lleva a cabo la ejecución de las tareas, que incluye la ejecución de los kernels de GPU. Para ello, primero se comprueba dentro de la tarea la compatibilidad del nodo al cual ha sido asignada. Si existe dicha compatibilidad, entonces procederá a la compilación y preparación de argumentos y datos para el kernel, así como a la conexión con el *scheduler* presente en el mismo nodo. De esta manera, primero ejecutará el kernel correspondiente en la GPU más adecuada y, después, deberá obtener los datos de la GPU para poder terminar la ejecución de la correspondiente tarea.

El *scheduler* es un servicio independiente encargado de la planificación y selección de los dispositivos en los cuales se ejecutarán los kernels. Debido a su característica de servicio independiente, es capaz de planificar las GPUs incluso entre aplicaciones independientes de Spark que se estén ejecutando dentro de un mismo nodo. Para la planificación se han elegido diversas políticas que tendrán en cuenta factores como los recursos disponibles en el dispositivo, el

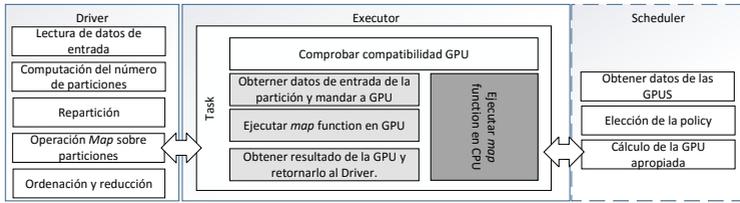


Fig. 1. Flujo general de una aplicación de GPU dentro de Spark.

rendimiento ofrecido, número de tareas en ejecución, etc. Esta aproximación permite aumentar el número de kernels concurrentes en la plataforma.

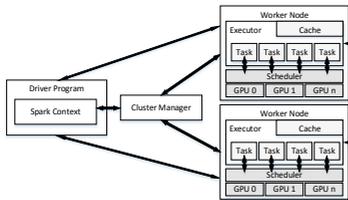


Fig. 2. Flujo de la retroproyección en Spark del simulador.

A. Planificador de dispositivos GPU para tareas MapReduce

Una de las principales limitaciones para la migración del paradigma MapReduce a entornos heterogéneos es la gestión eficiente de los aceleradores. Esto es especialmente relevante en despliegues con múltiples dispositivos, dado que, actualmente, las tareas generadas por el motor de flujos de trabajo son ajenas a los recursos disponibles. Para resolver este problema, nuestra solución incorpora un planificador que facilita dicha integración. Para ello, se ha diseñado e implementado una solución software que permite asignar GPUs a cada una de las tareas de forma eficiente, basándonos en la invocación de procesos remotos (RPC).

El sistema está compuesto por una biblioteca invocada por las tareas y un proceso servidor para cada uno de los nodos de cómputo. El funcionamiento es el siguiente. En primer lugar, al inicio de cada tarea, se solicita al servicio de planificación qué GPU está disponible dados unos requisitos de capacidad de memoria dados. En segundo lugar, el planificador resuelve qué dispositivo está disponible, teniendo en cuenta el estado (por ejemplo, memoria disponible, procesos en ejecución, etc.) de cada dispositivo. Es importante destacar que en caso de no contar con los recursos de ejecución disponibles, esta invocación quedará bloqueada. En tercer lugar, una vez obtenido el dispositivo disponible, en la tarea se selecciona

el contexto de ejecución para la GPU dada. Finalmente, una vez concluida la ejecución de la tarea, se comunica al planificador que libere los recursos anteriormente reservados.

Para lograr un uso eficiente de los dispositivos, el planificador cuenta con distintas políticas de ejecución ya implementadas:

- Aleatoria (*random*): se elige una GPU de forma aleatoria de aquellas disponibles.
- Dispositivo con menos procesos (*least processes*): se elige aquella GPU con menos tareas en ejecución.
- Round-robin: se asigna una GPUs de forma consecutiva a cada tarea que llega.

IV. CASO DE USO EN EL CAMPO DE LA IMAGEN MÉDICA

Debido al aumento de la resolución en los dispositivos actuales y a las nuevas técnicas de análisis de imagen, el volumen de datos producidos por este tipo de aplicaciones se ha incrementado notablemente. La obtención de las imágenes finales pasa por, en muchos casos, la ejecución de algoritmos con una gran carga computacional. La mayoría de estos algoritmos han sido optimizados para adaptarlos a aceleradores de cómputo tales como Intel Xeon Phi o GPUs. Sin embargo, estos dispositivos no poseen la cantidad de memoria necesaria para generar imágenes en 3D en alta resolución, las cuales pueden llegar a ocupar más de 10 GB.

Un ejemplo de este tipo de problemas es la reconstrucción y los procesos de simulación en el campo de la imagen médica, la cual utiliza algoritmos cada vez computacionalmente más complejos. Es el caso del simulador de rayos-X Fux-Sim, que utiliza varios métodos de reconstrucción de imagen de Tomografía Axial Computerizada (TAC) y simula distintas geometrías de adquisición de datos.

En el presente trabajo, se ha migrado una parte del simulador al *framework* de computación distribuida Apache Spark: la retroproyección. La retroproyección es la parte esencial de la reconstrucción, ya que se encarga de calcular los valores de cada uno de los vóxeles que forman la imagen final en 3D. En nuestro caso, el algoritmo de reconstrucción elegido realiza la integral de cada una de las líneas de proyección y tiene su origen en un algoritmo tradicionalmente

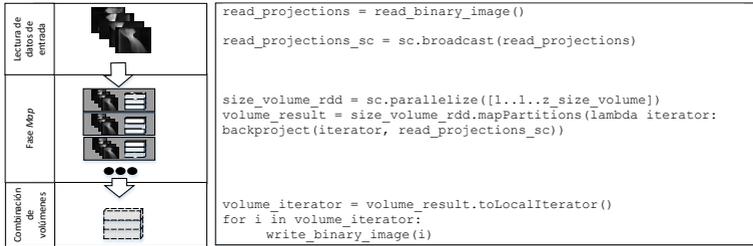


Fig. 3. Flujo de la retroproyección en Spark del simulador.

utilizado, el FDK [6]. Estas líneas de proyección comienzan en la fuente de Rayos-X y pasan por cada uno de los vóxeles a calcular para terminar en las imágenes de entrada, donde se produce una interpolación lineal.

El cómputo de cada una de las líneas de proyección es completamente independiente, lo que permite la paralelización del algoritmo. Debido a la gran cantidad de valores a calcular y a la facilidad de utilización de los algoritmos de interpolación, tradicionalmente este tipo de algoritmos han sido implementados y optimizados para aceleradores de tipo SIMD como las GPUs. Utilizando la arquitectura definida en la sección anterior se puede reutilizar los kernels existentes junto con la distribución de tareas que ofrece Spark para proporcionar una ejecución acelerada gracias a la arquitectura GPU. Las operaciones llevadas a cabo así como un pseudocódigo pueden verse en la Figura 3. Los datos de entrada son leídos en el *driver* y retransmitidos a cada uno de los *executors* para su procesamiento. Durante la fase *map*, cada partición tiene asignadas un determinado número de rodajas del volumen a reconstruir, éstas serán las que se devuelvan posteriormente. El RDD resultante será el que finalmente se escriba, bien en un sistema de ficheros local en el *driver* o de forma distribuida a través de HDFS. Esta última característica permite el tratamiento de grandes estudios.

V. EVALUACIÓN

La evaluación se ha realizado en un nodo de cómputo compuesto por dos procesadores Intel(R) Xeon(R) CPU E5-2620 0 a 2.00GHz y 3 GPUs, 1 Tesla K40c (12 GB de memoria GDDR5) y 2 Geforce GTX Titan (6GB de memoria GDDR5). En las pruebas en las que así se especifique se utilizarán estas GPUs con la memoria limitada a 2GB.

El sistema está supervisado mediante Cloudera 5.7. La versión de Spark sobre la que se ha trabajado es 1.6 en modo de ejecución *stand-alone*, los ficheros de entrada se encuentran en un SSD local y se escriben los ficheros resultantes en un directorio almacenado en HDFS, también sobre SSDs y una red Ethernet 10 Gbps. La versión de python utilizada es 2.7 y de pyCUDA 1.3. Los datos utilizados en la eva-

luación consisten en 360 imágenes (radiografías) de 1024x1024 píxeles. El volumen total reconstruido es de 1024x1024x1024 vóxeles, formando un total de 4 GB de datos de salida.

A. Evaluación del tiempo de ejecución en GPUs

Debido a la sobrecarga de la ejecución del runtime asociado a Apache Spark los tiempos de ejecución para volúmenes estándar de 1024x1024x1024 vóxeles (4 GB) de la aplicación de retroproyección distribuida en un nodo no son totalmente competitivos con la ejecución del simulador en Fux-Sim para el caso del mismo nodo y número de GPUs. En la Figura V izquierda, mostramos los tiempos de ejecución para el simulador y la aplicación en Spark para distintas GPUs. La configuración de Spark consistía en 3 threads y 3 particiones, lo más parecido a la configuración del simulador en modo multiGPU. En todos los casos Spark requiere más tiempo para completar la ejecución sin embargo se puede apreciar que según aumenta el número de GPUs utilizadas esta diferencia se reduce. Esto es debido al mejor aprovechamiento de los recursos por nuestra arquitectura gracias al planificador, como veremos posteriormente.

En la Figura V derecha mostramos los tiempos de ejecución de la aplicación en Spark para 3 hilos, distinto número de particiones y las 3 políticas implementadas. Estos experimentos se hicieron estableciendo un límite de memoria, con el objetivo de emular dispositivos de bajo coste (familia GTX). A partir de los resultados, se puede observar que para el caso de un sólo nodo el aumento del número de particiones influye negativamente en los tiempos de ejecución, debido principalmente al aumento del uso de la memoria. De las tres políticas la única que se diferencia claramente del resto negativamente es la política aleatoria mientras que Round Robin y menos procesos se mantienen igualadas, aunque como veremos posteriormente, con un mayor número de hilos paralelos y un estudio de la ocupación sí se aprecian diferencias significativas.

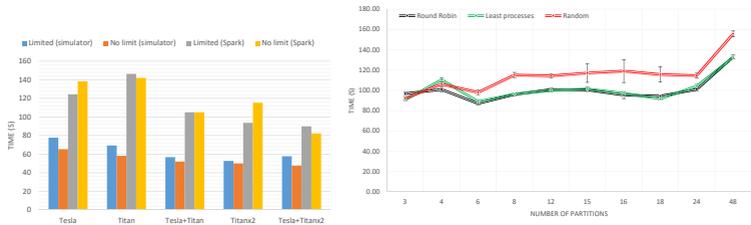


Fig. 4.A la izquierda: tiempos de ejecución para las distintas combinaciones de GPUs en un nodo, con limitación de memoria, sin limitación, en el simulador original y en su versión en Spark. A la derecha: resultados de la evaluación de la aplicación con 3 hilos y diferente número de particiones para cada una de las políticas y su correspondiente error estándar.

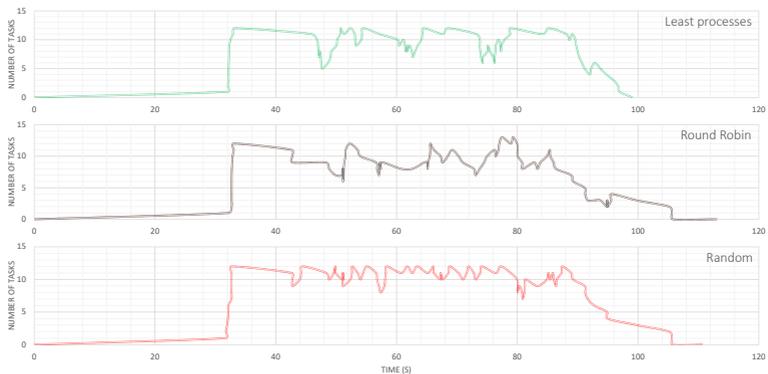


Fig. 5.Resultados de la evaluación de la aplicación con 12 hilos y 48 particiones para cada una de las políticas. Se muestra el número de tareas ejecutadas en cada momento, en segundos, por las GPUs.

B. Ocupación de las GPUs en las distintas políticas

El objetivo de la capa *scheduler* es el aprovechamiento de las GPUs de los nodos en los cuales se está ejecutando la aplicación. En la Figura 5 mostramos la ejecución de la aplicación sobre 12 threads y 48 particiones para cada una de las políticas implementadas. Para emular un sistema homogéneo, la memoria total de las GPUs ha sido limitada a 2GB. Como se puede ver, la política que planifica en función del número de procesos resulta en un mejor tiempo de ejecución, tal y como habíamos confirmado anteriormente. Una de las razones para este menor tiempo de ejecución es el aprovechamiento regular de los recursos disponibles en el nodo. Tanto Round Robin como la política aleatoria poseen numerosos picos de ocupación, mientras que con la primera política se mantiene estable para la ejecución de 12 tareas. Este valor resulta ser el número ideal ya que es el número máximo de threads que ejecutan en paralelo. Aún así, se aprecian tres caídas regulares de la ocupación, las cuales corresponden con la finalización de los pa-

quetes de tareas formados por 12 cada uno. Por ello, en un sistema homogéneo, la política que favorece los dispositivos con menos procesos es la que lleva a un mejor reparto de tareas.

VI. TRABAJOS RELACIONADOS

La unificación de arquitecturas heterogéneas con frameworks de computación distribuida (por ejemplo Apache Hadoop) ha sido tratado en diversos trabajos desde distintos puntos de vista. En Hadoop existen varios ejemplos de esta unificación, bien centrados en modelos de programación concretos basados en GPU como HadoopCL [7] o centrados en la programación de los aceleradores para ofrecer soporte tanto a OpenCL como CUDA, como es el caso de Hadoop+ [8]. En cuanto a Apache Spark, está actualmente en desarrollo una mejora del framework llamada HeteroSpark [9] basada en Java RMI, sin embargo no se encuentra disponible actualmente. La mayoría de estos frameworks se basan, al igual que en nuestro trabajo, en la compilación de kernels programados, aunque la planificación de los trabajos mandados a

la GPU no está incluida, reduciendo la capacidad de mejora.

Sin embargo, otros trabajos relacionados han optado por no modificar el framework distribuido y añadir GPUs a nivel de aplicación. Ejemplos de este procedimiento son, por citar algunos, el presentado por Zheng et al. [10] que utiliza GPUs dentro de Hadoop para acelerar el algoritmo *kmeans* o [11] que hace uso de Spark un servidor separado para las GPUs para el análisis de imágenes de resonancia magnética. Nosotros hemos optado por una solución similar, pero dentro del framework de Apache Spark.

Más allá del uso de aceleradores GPUs, en estos entornos distribuidos algunos trabajos en el campo de la imagen médica se han centrado en su evaluación en entornos sin presencia de aceleradores como por ejemplo [12], que implementa el algoritmo de tomografía computerizada FDK en su versión 4D utilizando el paradigma MapReduce en Hadoop. Otra solución [13], implementa una deconvolución en Spark y lo compara con otras alternativas como el uso de GPUs o multicore CPUs.

VII. CONCLUSIONES

Este trabajo ha presentado una prueba de concepto del uso de GPUs dentro del *framework* Apache Spark a través de la adaptación de una aplicación de reconstrucción de imagen médica. Se han detallado las características de la solución y de las herramientas necesarias para llevarlo a cabo esta tarea. El prototipo se basa en el conector PyCUDA, aunque se puede generalizar a otros modelos de programación en aceleradores de cómputo, así como otros lenguajes de programación que tengan disponibles estos conectores. Además se ha provisto de un planificador adicional intranado en la plataforma Spark para el mejor aprovechamiento de los recursos disponibles. Este planificador posee diversas políticas que han sido evaluadas sobre un sólo nodo, demostrando su utilidad cuando comparamos con la ejecución de la aplicación original. Cuando se aumenta el número de GPU a utilizar, nuestra arquitectura aprovecha de mejor manera los recursos de cómputo disponibles. Esto último se aprecia especialmente cuando se hace uso de una política que tiene en cuenta parámetros dinámicos de la GPU, como el número de procesos en ejecución.

El siguiente paso de este trabajo consistirá en la evaluación sobre varios nodos, así como en ampliar el planificador de GPU a nivel de clúster para un mejor reparto de los trabajos. Además se prevé utilizar el mismo procedimiento en otras aplicaciones de procesamiento de imagen médica, como algoritmos iterativos de reconstrucción, para su posterior combinación con flujos de análisis de las imágenes resultantes.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente subvencionado por el Ministerio de Economía y Competitividad, bajo el proyecto TIN2013-41350-P, *Scalable Data Ma-*

agement Techniques for High-End Computing Systems. También queremos agradecer a Nvidia por proporcionar la tarjeta Tesla K40 con la que se han realizado los experimentos.

REFERENCIAS

- [1] Felix García-Carballeira Jesus Carretero Perez Silvina Caino-Lores, Alberto García Fernandez, "A cloudification methodology for multidimensional analysis: Implementation and application to a railway power simulator," vol. 55, pp. 46–62, June 2015.
- [2] Javier García Blas, Mónica Abella, Florin Isaila, Jesus Carretero, and Manuel Desco, "Surfing the optimization space of a multiple-gpu parallel implementation of a x-ray tomography reconstruction algorithm," *Journal of Systems and Software*, vol. 95, pp. 166 – 175, 2014.
- [3] Claudia Molina Ines Garcia Jesus Carretero Manuel Desco Estefanía Serrano, Javier García Blas and Mónica Abella, "Design and evaluation of a parallel and multiplatform cone-beam x-ray simulation framework," July 2016.
- [4] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.
- [5] Andreas Klöckner, Nicolas Pinto, Yunsup Lee, B. Catanzaro, Paul Ivanov, and Ahmed Fasih, "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Runtime Code Generation," *Parallel Computing*, vol. 38, no. 3, pp. 157–174, 2012.
- [6] LA Feldkamp, LC Davis, and JW Kress, "Practical cone-beam algorithm," *JOSA A*, vol. 1, no. 6, pp. 612–619, 1984.
- [7] Max Grossman, Mauricio Breremitz, and Vivek Sarkar, "Hadoopcl: Mapreduce on distributed heterogeneous platforms through seamless integration of hadoop and opencl," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International. IEEE*, 2013, pp. 1918–1927.
- [8] Wenting He, Huimin Cui, Binbin Lu, Jiacheng Zhao, Shengmei Li, Gong Yuan, Jingling Xue, Xiaobing Feng, Wensen Yang, and Youliang Yan, "Hadoop-*r*: Modeling and evaluating the heterogeneity for mapreduce applications in heterogeneous clusters," in *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 2015, pp. 143–153.
- [9] Peilong Li, Yan Luo, Ning Zhang, and Yu Cao, "Heterospark: A heterogeneous cpu/gpu spark platform for machine learning algorithms," in *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 347–348.
- [10] HuanXin Zheng and JunMin Wu, "Accelerate k-means algorithm by using gpu in the hadoop framework," in *Web-Age Information Management*, pp. 177–186. Springer, 2014.
- [11] Roland N Boubela, Klausius Kalcher, Wolfgang Huf, Christian Nasel, and Ewald Moser, "Big data approaches for the analysis of large-scale fmri data using apache spark and gpu processing: A demonstration on resting-state fmri data from the human connectome project," *Frontiers in neuroscience*, vol. 9, 2015.
- [12] Bowen Meng, Guilem Pratz, and Lei Xing, "Ultrafast and scalable cone-beam ct reconstruction using mapreduce in a cloud computing environment," *Medical physics*, vol. 38, no. 12, pp. 6603–6609, 2011.
- [13] Lianyu Cao, Penghui Juan, and Yinghua Zhang, "Real-time deconvolution with gpu and spark for big imaging data analysis," in *Algorithms and Architectures for Parallel Processing*, pp. 240–250. Springer, 2015.

Methodology for the efficient execution of applications on multi-core HPC environments

Carlos Rangel¹, Álvaro Wong¹, Dolores Rexachs¹ and Emilio Luque¹

Abstract— This paper describes a methodology for the efficient execution of parallel scientific applications in multi-core HPC environments. Our proposed methodology is based on the analysis and characterization of the behavior of the application on a target machine, which is made in a bounded time relative to the total runtime of the complete application. The analysis and the characterization of the target machine are performed using a signature of the application. This signature represents the application core and is made up with phases; the phases are segments of code delimited by communications events, which are repeated throughout the application. To conclude we have conducted experiments to corroborate our hypothesis that there inefficiencies in the application that can be reduced through mappings mechanisms, and we prove that the process mapping between the compute nodes and within a compute node is an important factor in order to reduce the communication time between phases.

Keywords— Application Signature, Performance Analysis, efficient execution, Multi-core machines characterization, Application Mapping Policies.

I. INTRODUCTION

HIGH Performance Computing (HPC), is based on using parallel processing to implement advanced application programs in an efficient, reliable and fast way. In HPC environments predicting performance and benchmarking are some of the relevant activities in the area. In order to efficiently use the HPC systems, it has to be taken into consideration the importance of the efficient use of resources, an application can be inefficient in a machine for various reasons, among which may be mentioned the congestion in communications, which can be given by the communication pattern of the network or congestion in the output of the interconnection network; others reasons of inefficiency is the increased in computing times originated by failures in shared caches memories, bottlenecks in the internal communication buses, among others.

Modify the source code of the application to improve efficiency is not always possible, most of the time the source code is not available, a first proposal would be improve efficiency without the need to modify the source code of the application.

This paper proposes a methodology which will allow analyze, predict, and select an appropriate mapping policy for a scientific application with message passing for a number of processes assigned by the user in HPC environments, in order to improve the application performance. That is, to reduce runtime

and improve efficiency in the execution of the application by executing a signature of the application in the target machine in order to obtain the predicted execution time (PET) of the application.

The application signature is a characterization of the relevant phases form an application, and the repetition rates of each phase, which we will call weights. As said before, a phase is defined as a segment of code delimited by communications events, which are repeated throughout the application.

With the intention of improving the efficiency of parallel scientific application, with a fixed number of processes it has been detected some initial strategies that can be performed without touching the source code of the application, and that affect the performance of the same in HPC environments. Such strategies are listed below.

One of the possible strategies we propose is in a cluster at the level of compute nodes, the possibility to modify the global mapping, how the application processes are distributed in the compute nodes. The modification of this mapping strategy may results in a time reduction of the application execution, due to the decrease in the congestion pattern in the communication network, or a communication congestion pattern at the input/output of the nodes to the network.

Another strategy will be within a compute node-level multicore, it can be changed the internal mapping of each node at the level of the cores. That is, leaving free some of the cores within the compute node, reconsidering more nodes to perform the computation. This may results in a reduction in execution times due to a decrease in the caches memory misses shared between cores, or a system's congestion within the compute node.

The proposed methodology is based on the fact that scientific message passing applications have been developed with the goal of being scalable due to the complexity of the problem to be solved. With the aim of being more efficient these models will be performed: detection of inefficiencies at the level of application phases; detection of target machine inefficiencies in the communication pattern; models for the development of new mapping policies; and models to detect inefficient use of resources. All these models are done so that when the application is run, it can be run with a new mapping or resources policies which were designed for the application in order to run more efficiently on a target machine, taking into account the communication patterns in this target machine and the efficient use of the resources.

¹Department of Computer Architecture and Operating System (CAOS), University Aut3noma de Barcelona, Spain: carlostramon.rangel@uab.cat, alvaro.wong@caos.uab.es, dolores.rexachs@uab.cat and emilio.luque@uab.cat

This paper is organized as follows: Section II presents the background and some related works. Section III presents the proposed methodology. Section IV presents the experimental validation and finally section V the conclusion and future works.

II. BACKGROUND

So far several studies and tools have been developed for predicting performance and efficiency in HPC environments [1-5, 11], but none of these take any action if the prediction of the application execution time is efficient or not. Below these studies are described.

A. Studies aiming at mapping improving of an application for a specific machine

The work presented in [17] samples several techniques and algorithms that efficiently address the issue of mapping the application taking into account its communication pattern onto the physical topology. Using such strategy it enables to improve the application overall execution time significantly. This study concludes by listing a series of open issues and problems.

The topology mapping problem can be expressed as a simplification of various metrics. In [17] they discuss the static topology mapping problem and later generalize to a dynamic version that is relevant for task-based environments. The topology network can be modeled by a weighted graph $H = (VH, \omega H, RH)$ where the set of vertices, $VH \in \mathbb{N}$, represents the execution units and the weighted edges $\omega H(u, v) \in \mathbb{R}$ with $u, v \in VH$ represent the weight of the edge between the two vertices u and v . Non-existing edges can be modeled by the weight zero.

Also the static application graph is modeled in [17] as a weighted graph $A = (VA, \omega A)$, and the topology mapping as $\sigma: VA \rightarrow VH$. Each concrete mapping σ has an associated cost metric, which is typically the target of the minimization problem.

Optimization algorithms in [17] strive to minimize two metrics: dilation and congestion. Where dilation allows a comparison of the number of times packets are transmitted over network interfaces, and congestion counts how many communication pairs use a certain link. Most specific problems of mapping arbitrary A to arbitrary H minimizing either the dilation or congestion are a NP-hard problem.

Among the various factors influencing the performance, process placement plays a key role as it impacts the communication time of the application.

Different techniques have been shown up in [17], and algorithms and tools to perform a topology-aware process placement such as [18-20]. In all cases, the problem consists in matching what represents communication pattern of the application to the physical topology of the architecture.

Some open issues that concern to the topology mapping problem are listed below.

Handle very large problems, such as high performance computing applications which feature hun-

dreds of thousands of processes. Mapping these processes onto the architecture require a huge computing power.

Reducing the communication load has a huge impact on the energy consumption; big part of the energy spent in executing an application is due to data movement. However, there is a lack of studies about the real gain of topology-aware mapping and energy savings.

The link between the different aspects of process affinity: within the compute node (cache), between compute nodes (network) and between the node and storage. Each of these aspects may incur contradictory objectives in terms of placement [17]. Therefore, it requires to find compromises or to be able to adapt, at runtime, the mapping according to the dominating factor.

Our methodology that is being proposed differs from the works presented in [17-20] because it faces the issue of topology-aware process placement, which takes a lot of time, by using a signature of the application, which as has been mentioned is a characterization of the relevant application phases and their weights. The signature runs on the target machine in a short time, compared to the total execution time of the application.

B. PAS2P (Parallel Signature Application for Performance Prediction)

Parallel computers show different rates of performance for different applications when they are executed, the accurate prediction of the performance of parallel applications has become an increasingly complex study and the time required to perform a thorough analysis is highly expensive, especially if it is wanted to predict in different systems. In clusters of production, performance and the efficiency use of resources are critical, it is important to be able to predict which system is best suited for an application, or how long it will take to run a parallel application.

In order to have the foresight to predict the execution time, it has been created a tool where the signature of parallel applications for performance prediction called PAS2P (Parallel Signature Application for Performance Prediction) is extracted to characterize the parallel message passing applications [1-3]. PAS2P instrumented and runs applications in a parallel machine, and produces a trace file. The data obtained is used to characterize the behavior of computing and communication.

In order to obtain the model of the application that is independent from the machine, it is assigned to the trace a global logical clock according to the causal relationships between the communication events through a logic sorting algorithm.

Once it has an abstract model, it identifies and extracts the most relevant events sequences called phases, and a weight is assigned to the phases from the number of times they are repeated. Subsequently, the signature, defined by a set of phases

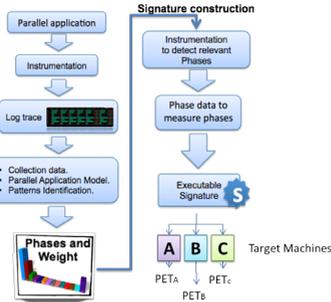


Fig. 1. PAS2P Methodology

selected according to their weight is created.

The execution of the signature in different target systems allows measuring the execution time of each phase, and therefore calculates the runtime of the entire application in each of those systems. This is done by extrapolating the execution time of each phase using the weights that were obtained. In Figure 1, it can be observed the PAS2P methodology; there is a sequence of steps that are necessary to obtain the phases and their weights.

Then with the phases information, PAS2P can move on to create the signature that is independent of the machine, this signature can be run on other machines in a short time, as the signature will always be a small fraction of the entire application which is translated in a small fraction of the application execution time.

Finally, the signature predicts the total execution time (PET) of the parallel application runtime adding all phases multiplied by their weights. The equation to obtain the PET is shown in (1) where the PhaseET_i is the estimated time for each fase i, and W_i is the weight for each phase i.

$$PET = \sum_{i=1}^n (PhaseET_i)(W_i) \quad (1)$$

What it is been proposed in this paper, differs from PAS2P, as it will step beyond of just making a prediction of execution times; using this time as one of the parameters for the detection of inefficiencies model it may characterize the target machine, so it can take actions that lead to an improvement in the efficiency of the application.

C. P3S (Prediction of Parallel Program Scalability)

In order to predict application scalability in an HPC system, P3S (Prediction of Parallel Program Scalability) methodology was conducted, which predicts the performance of the strong scalability of parallel scientific applications of message passing in a given system using a limited time analysis and a small set of system resources [4-6].

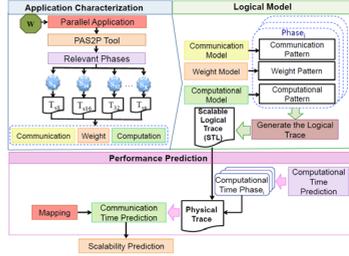


Fig. 2. P3S Overview

The P3S methodology is based on analyzing the repetitive behavior of parallel message passing applications, which are composed of a set of relevant phases that are repeated throughout the application, regardless of the number of processes of this. P3S methodology has three stages, as shown in Figure 2 the first stage is "characterization", where relevant phases of the parallel application are identified, from the information obtained from the execution of a set of signatures of the application, with a small scale of processes.

The second stage is the application modeling, in which the logic model scalability of each relevant phase of the parallel application is generated; this model is used to build the Scaled Logical Trace (SLT), which will predict the logical behavior of the application, as it increases the number of processes.

The last stage is the performance prediction of the application for a specific number of processes, in which the SLT will be set to the number of processes in which it is wanted to predict the performance of the application. The result is a scalability curve of the application, provided by the P3S methodology, in which users can select the most appropriate resources to run the application in the target machine. As shown in Figure 3 P3S manages to predict how the N-Body application with 100000 particles will scale to more than 500 processes running in only less than 100 processes.

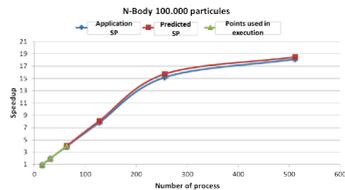


Fig. 3. P3S Scalability prediction for the N-Body application

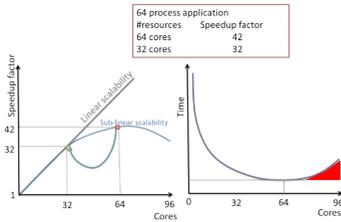


Fig. 4. Inverse scalability example

D. Inverse scalability

When the processes mapping of a parallel application in HPC environments, places two processes that have high density in messages passing between them in two different compute nodes of the machine, it may be generating delays in message passing and congestion in the output or input of each compute node to the interconnection network, as well these delays may be happening by the topology of the interconnection network of the machine. The study [11], proposes a methodology that helps the user to decide the number of resources needed for efficient mapping of an application. This study of reverse scalability uses PAS2P tool [1-3] for the behavior of the application and extract the relevant phases (signature), and then run the signature in a short time; reducing the number of compute nodes, it reduces the execution time because it reduces the remote communications to local communications. This study has been tested in cloud environments, where the efficiency improvements are very remarkable.

Inefficiencies are the conclusion of the imbalances in communication between of the processes [10], these inefficiencies are detected in the delays which are measured from a process that executes a sending message and another process which receives the message, in this work of Inverse Scalability the inefficiencies are also idle time in which a processor is not performing computation.

As shown in Figure 4, the hypothesis is valid for applications that have a sub-linear scalability, and that at some point no longer efficiently scale. Inverse scalability has been tested on cloud computing, obtaining good results for the efficient use of resources.

III. PROPOSED METHODOLOGY

In the execution of parallel applications which run in HPC environments, there are factors and system components that can be modified without changing the application code and affecting the performance of it. We propose a methodology for parallel scientific application, with a fixed number of processes; which will allows fulfilling user requirements designed to: reduce runtime, improve energy efficiency, and reduce cost, in executing the application.

There are tools available for predicting the perfor-

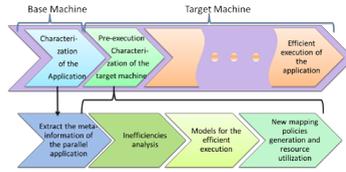


Fig. 5. Proposed Methodology

mance of a parallel scientific application of message passing in HPC environments, only predict the execution time and how would scale on the target machine. In order that an application would become more efficient, it is proposed the hypothesis in the fact that there are factors that can be modified to improve the efficiency of a parallel application, without the need to modify the source code of it, as is the mapping of this application on the target machine; we distinguish between two ways to change that mapping, the possibility to modify it in a global way, that is how the application is distributed in the compute nodes, and the possibility to modify the mapping in a local way, that is how the application is distributed inside each compute node. Using different mappings may result in different performance and therefore different efficiencies.

In addition to predicting, it is further proposed to take action based on detection models of inefficiencies, in order to improve efficiency either taking into account: the total time of execution, associated with the operation cost, energy cost, and the relationship runtime by the number of resources used.

The proposed methodology is focused on obtaining a better mapping and better selection in resource use for the application, which uses a parallel message passing in HPC environments; this is done by making a short pre-execution to characterize the target machine in order to generate an appropriate mapping for the application on that target machine. Figure 5 shows the methodology proposed in a macro level where it is observed that it consists mainly of three stages.

The first stage is the characterization of the application, in which the meta-information from the scientific application is extracted outside the destination machine (using a based machine), where the signature of the application is extracted.

The proposed methodology will be described in two sections, the first section where the characterization of inefficiencies in the communication pattern of the phases is explained; the second section discusses the characterization of the computing inefficiencies of each phase.

A. Communications inefficiencies detection

For purposes of this study, communication inefficiency is all the waiting times for communications in which the processors are not doing any work.

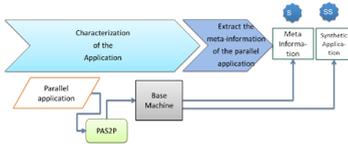


Fig. 6. First stage of the proposed methodology

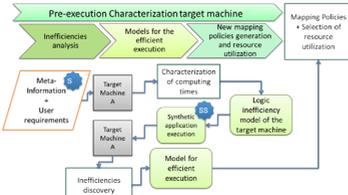


Fig. 7. Second stage of the proposed methodology

The first stage can be seen in Figure 6, which details that the parallel application will be executed on a base machine with PAS2P tool in order to obtain the meta-information of the phases, known as the signature. This signature is the input of the second stage, where first level inefficiencies from the phases are detected. In addition here in the first stage it also can be obtained a Synthetic Signature (SS) program that allows us to predict the application performance in a target machine, which can be used to emulate the behavior of the application on the target machine, for further understanding of the SS see [21].

A second stage of pre-execution and characterization of the target machine, where an analysis of the inefficiencies in the phase is carried out, which is done outside the target machine, but it requires certain information to the this target machine, followed by constructing models for efficient execution and at the end of this stage, the generation of new mapping policies and resource utilization.

Having performed the analysis of inefficiency per phases, in the second stage, all the meta information extracted in the first and second stage, are sent to the destination machine for execution as shown in Figure 7, remembering that this runtime corresponds to the pre-execution of the application submitted by the user, that time will always be significantly lower in relation to the execution time of the complete application.

Once the communication inefficiencies are characterized in each phase, the following step is to detect computing inefficiencies from the phases, which is described in the section below.

B. Computational inefficiencies detection

The final stage is the result of our methodology, which is the efficient execution of the application on

the target machine, which takes significantly longer than previous stages, which are pre-execution and characterization of the application. For purposes of this study, the computational inefficiencies are given by overruns cache failures that can be solved by changing the mapping process.

To begin the final stage, the characterization of computational times is performed, which will measure the CPU time, instructions, and cache failures on the target machine, then these are used to detect this kind of inefficiencies in each phase, which are detected using a logic model that correspond to how the application will behave on the target machine; then the model of efficient execution of the application, with the new mapping policies and the selection of the necessary resources is performed in order to execute the application in an efficient way, because a balance was found in reducing inefficiencies in all phases.

Once the new mapping policy is obtained along with the selection of resources, the application is executed on the target machine with improved efficiency, either: taking into account: the total execution time, cost related to use, energy costs, or the relationship between execution time by the number of resources used.

A series of experiments in order to test the hypothesis that there are computing inefficiencies or communication inefficiencies were conducted and are described below.

IV. CASE STUDY

Two main experiments were performed, in which was taken into consideration leaving always fixed the number of processes of the application. In the first experiment, the signature of the application was run using different number of compute nodes. In the other hand, the second experiment leaves the number of compute nodes fixed, and varies the global mapping of the application in the case of mappings A, B and C; D mapping makes a step further and changes the internal mapping in each compute node, modifying how process are map in each core of each CPU.

The performed experiments used the NAS Parallel Benchmarks which have been developed for performance evaluation of HPC [22]. Specifically it has been used the Conjugate Gradient (CG), irregular memory access and communication class E for 128 process and a 100 iteration. The experiments performed are described below.

Each compute node from Capita cluster has 4 CPUs, each CPU has 16 cores, to get a total of 64 cores per node, sharing an L3 cache memory by eight cores and the L2 each two cores. Cluster BEM has 2 CPUs per node, each CPU has 12 cores, to get a total of 24 cores per node, sharing an L3 cache memory by six cores and each core has an own L2 cache memory.

TABLE I
 MACHINES USED FOR EXPERIMENTATION

Cluster	Characteristics
Capita 8 Nodes X 64 cores	Processor: AMD Opteron 6262 HE Memory: 8 x 2 MB L2 Cache shared each 2 cores, 2 x 8 MB L3 Cache Network: Infiniband
BEM 8 Nodes X 64 cores	Processor: Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz Memory: 8 x 2 MB L2 Cache shared each 2 cores, 2 x 8 MB L3 Cache Network: Infiniband

A. First experiment, increasing the recourses

Leaving the number of processes of the application always fixed and varying the number of compute nodes where it is executed. The target machines used for this experiment are: the first one will be named Capita, and the second will be named BEM. Table I shows a description of the clusters used in the experimentations.

As shown in Figure 8a, the results is that as the compute nodes are increased in the target machine, the execution times are reduced between different numbers of selected compute nodes in the experiment, the main profit is obtained from two to three compute nodes, but further increase in the number of compute nodes does not bring benefits for this application on the target machine.

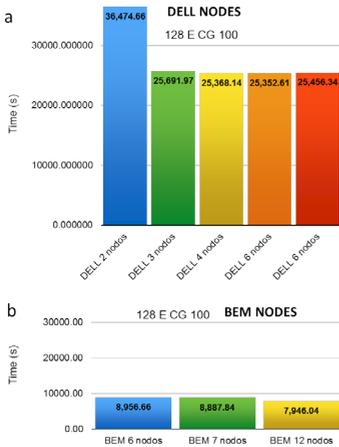


Fig. 8. First experiment in two machines

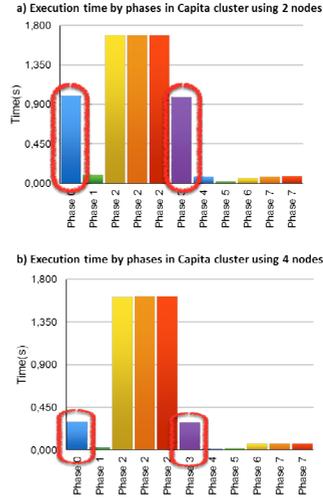


Fig. 9. First experiment Analysis by phases in Cluster Capita

In Figure 8b, the results is that as the compute nodes are increased in the target machine, the execution times are not considerable reduced from six nodes to seven or seven nodes to twelve.

A detailed analysis of what part of the application is been optimized is performed with an analysis using the signature, where it can be seen the times that each phase of the application takes, as it is shown in Figure 9 from Figure 9a to Figure 9b the phases 0 and 3 are been optimized by this selection of resources.

B. Second experiment, mapping variation

The experiment was performed leaving the number of processes of the application always fixed and the number of compute nodes to execute the application also fixed (held to four compute nodes), and changing he mapping of processes on CPUs on each compute node. The four different mapping used are described below.

Mapping A uses a round robin strategy. An example of mapping A is been shown in Figure 10, where 8 process are distributed in four compute nodes.

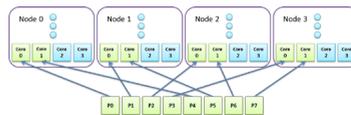


Fig. 10. Mapping A example

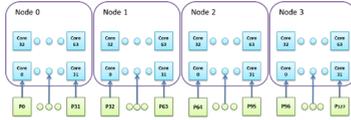


Fig. 11. Mapping B example

In mapping B the first quarter of processes was distributed on the first compute node, the second quarter of the processes in the second compute node and so on until the fourth. An example of mapping B is shown in Figure 11, where the 128 process are distributed in four compute nodes, using only the first half of cores.

The mapping C distribute the first eighth of processes in the first compute node, the second eighth in the second compute node and so on until the fourth eighth in the fourth node, then the fifth eighth of processes in the first compute node again, and so on up the last eighth in the fourth compute node.

An example of mapping C is shown in Figure 12, where Figure 12a show the distribution of the first 64 process, and Figure 12b shows the second part of the total process, which are the last 64 process, all the process are distributed in four compute nodes, using only the first half of cores.

The mapping D is somewhat similar to mapping B, with the difference that instead of placing the processes in consecutive cores on the chip, leaves an empty core means. An example of mapping D is shown in Figure 13, where the 128 process are distributed in four compute nodes, using only the cores id which are an even number, starting with number zero, using half of cores but not only the first half.

As result of this mapping variation a very significant improvement in the execution time of the application can be obtained, in Figure 14 can be seen this

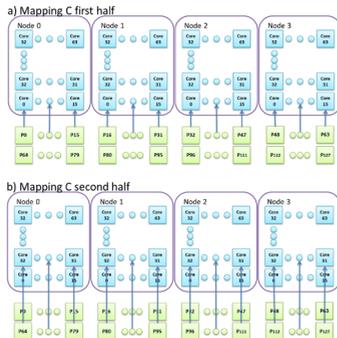


Fig. 12. Mapping C example

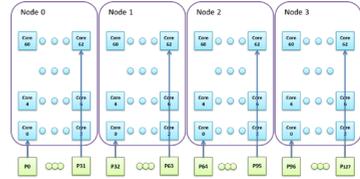


Fig. 13. Mapping D example

variation in the execution time. Between mapping A and mapping D, the execution time is been reduced halved.

Varying the mapping application in four compute nodes can improve runtime to half the initial time. It has to be noted that the mapping D is based on mapping B, which only modify global mapping, the new step in mapping D is to modify the internal mapping.

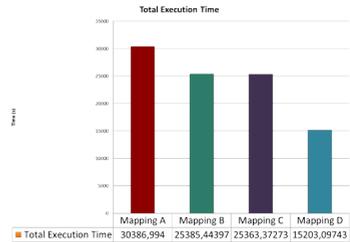


Fig. 14. Analysis by mapping variation

As shown in Figure 15, the analysis by phase is shown, a variation of mappings between the mapping A, B, and in Figure 16 between mapping C and D.

Figure 15a and 14b shows that the communication times from phases 0 and 2 are been improved using the global mapping modification technique.

On the other hand Figure 16a and 15b show that the computing time of phase 2 has been improved using the local mapping modification technique shown in mapping D.

The hypothesis proposed to find a more appropriate level analysis by mapping phase is possible, and can even generate a mapping to improve the times of all phases.

V. CONCLUSION AND FUTURE WORK

It has been proven that there are factors that can be modified and that affect the performance of the parallel message passing scientific application, without the need to modify the code of this application.

Process affinity between compute nodes (global mapping) has been proved to be an important factor in order to reduce the communication time between phases. It was demonstrated that the affinity within

compute node (local mapping) has also an important impact in the application execution time.

A future study is to identify other factors that also may affect the application performance, as it is suspected to be cache misses in CPUs shared memory and changes in its clock frequency.

As a future work we will to generate models to take action to improve the efficiency of the parallel application taking into account: the total runtime, energy cost, the relationship between the runtime by the number of resources used.

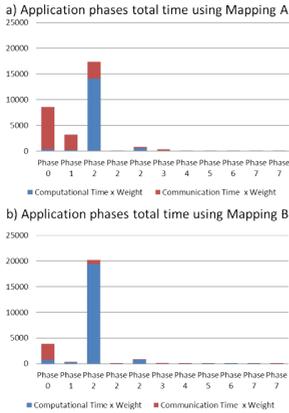


Fig. 15. Phase Analysis in local mapping variation

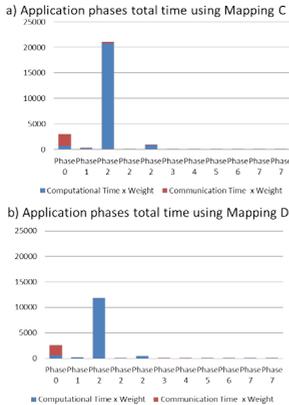


Fig. 16. Phase Analysis in local mapping variation

ACKNOWLEDGMENT

This research has been supported by the MINECO (MICINN) Spain under contracts TIN2014-53172-P.

REFERENCES

- [1] Álvaro Wong, Dolores Rexachs, and Emilio Luque. *Parallel Application Signature for Performance Analysis and Prediction*. Parallel and Distributed Systems, IEEE Transactions on 26, no. 7 (2015): 2009-2019
- [2] Álvaro Wong, Dolores Rexachs, and Emilio Luque. *Extraction of parallel application signatures for performance prediction*. Proceedings - 2010 12th IEEE International Conference on High Performance Computing and Communications, HPCC 2010, pages 223-230, 2010.
- [3] Álvaro Wong, Dolores Rexachs, and Emilio Luque. *PAS2P tool, parallel application signature for performance prediction, 2012*. Applied Parallel and Scientific Computing, Springer Berlin Heidelberg, 2010. 293-302.
- [4] Javier Panadero, Álvaro Wong, Dolores Rexachs, and Emilio Luque. *A tool for selecting the right target machine for parallel scientific applications*. Procedia Computer Science 18 (2013): 1824-1833.
- [5] Javier Panadero, Álvaro Wong, Dolores Rexachs, and Emilio Luque. *Scalability of Parallel Applications: An approach to predict the computational behavior*. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPPTA). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015.
- [6] Javier Panadero, Álvaro Wong, Dolores Rexachs, and Emilio Luque. *Scalability of Parallel Applications: An approach to predict the computational behavior*. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPPTA). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015.
- [7] Erika Abrah, Costas Bekas, Ivona Brandic, Samir Genaim, Einar Broch Johnsen, Ivan Kondov, Sauri Pillana, and Achim Streit. *Challenges and Recommendations for Preparing HPC Applications for Eascale*. arXiv preprint arXiv:1503.06974 (2015).
- [8] Bilge Acun, Nikhil Jain, Abhinav Bhatel, and Misbah Mubarak. *Preliminary Evaluation of a Parallel Trace Replay Tool for HPC Network Simulations*. Euro-Par 2015: Parallel Processing Workshops. Springer International Publishing, 2015.
- [9] Paul Bédaride, Augustin Degomme, Stéphane Genaud, and Arnaud Legrand. *Toward Better Simulation of MPI Applications on Ethernet / TCP Networks*. PMBS13-4th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, 2013.
- [10] Wolf Felix, Bernd Mohr, Jack Dongarra, and Shirley Moore. *Automatic analysis of inefficiency patterns in parallel applications*. Concurrency and Computation: Practice and Experience 19, no. 11 (2007): 1481-1496.
- [11] Saveta Gertvojlola. *A Naive methodology for looking efficient execution: inverse scalability*. Master's Theses, Universidad Autónoma de Barcelona, 2013.
- [12] Laura Carrington, Michael Laurenzano, and Ananta Tiwari. *Characterizing large-scale hpc applications through trace extrapolation*. Parallel Processing Letters, 23(04), p.1340008.
- [13] Henri Casanova, Frédéric Desprez, George S Markomanolis, and Frédéric Suter. *Simulation of MPI applications with time-independent traces*. Concurrency and Computation: Practice and Experience 2014.
- [14] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. *Versatile, scalable, and accurate simulation of distributed applications and platforms*. Journal of Parallel and Distributed Computing, 74(10):2899-2917, 2014.
- [15] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. *SimGrid: a Sustained Effort for the Versatile Simulation of Large Scale Distributed Systems*. arXiv preprint arXiv:1309.1630 (2013).
- [16] Henri Casanova, Anshul Gupta, Henri Casanova, and Anshul Gupta. *Toward More Scalable Off-Line Simulations*

- of *MPI Applications*. *Parallel Processing Letters* 25.03 (2015): 1541002.
- [17] Torsten Hoefler, Emmanuel Jeannot, Guillaume Mercier. *An Overview of Process Mapping Techniques and Algorithms in High-Performance Computing*. High Performance Computing on Complex Environments, Wiley, pp.75-94, 2014. 978-1-118-71205-4.
 - [18] Sudheer, C. D., and A. Srinivasan. *Optimization of the hop-byte metric for effective topology aware mapping*. High Performance Computing (HiPC), 2012 19th International Conference on, pp. 1-9. IEEE, 2012.
 - [19] Yin Qin, and Timothy Roscoe. *VF2x: fast, efficient virtual network mapping for real testbed workloads*. Testbeds and Research Infrastructure. Development of Networks and Communities. Springer Berlin Heidelberg, 2012. 271-286.
 - [20] Rashti, Mohammad Javad, Jonathan Green, Pavan Balaji, Ahmad Afshahi, and William Gropp. *Multi-core and network aware MPI topology functions*. Recent Advances in the Message Passing Interface. Springer Berlin Heidelberg, 2011. 50-60.
 - [21] Javier Panadero, Álvaro Wong, Dolores Rexachs, and Emilio Luque. *Synthetic Signature Program for Performance Scalability*. *Parallel Processing and Applied Mathematics*. Springer International Publishing, 2015. 345-355.
 - [22] Bailey DH, Barszcz E, Barton JT, Browning DS, Carter RL, Dagum L, Fatoohi RA, Frederickson PO, Lasinski TA, Schreiber RS, Simon HD. *The NAS parallel benchmarks*. *International Journal of High Performance Computing Applications* 5.3 (1991): 63-73.

Planificación Simbiótica de Procesos en el IBM POWER8

Josué Feliu¹, Stijn Eyerman², Julio Sahuquillo¹ y Salvador Petit¹

Resumen— Los procesadores SMT comparten la mayoría de los recursos internos entre las aplicaciones que ejecutan de forma simultánea. Debido a la competencia entre las aplicaciones por el acceso a estos recursos compartidos se pueden generar fuertes interferencias que afecten a sus prestaciones. Por ello, las prestaciones de los procesadores SMT dependen fuertemente de la complementariedad de las aplicaciones que ejecutan concurrentemente. La planificación simbiótica, es decir, la planificación de aplicaciones complementarias en un mismo núcleo puede tener un impacto considerable en las prestaciones alcanzadas por los núcleos SMT. El trabajo previo ha identificado esta situación y propone soluciones basadas en el muestreo periódico o en extensiones hardware para dar soporte a la planificación simbiótica, lo cual tiene una sobrecarga no despreciable o es imposible de realizar en sistemas actuales, respectivamente.

En este trabajo proponemos un planificador simbiótico de procesos para el IBM POWER8. El planificador utiliza un modelo de interferencia SMT basado en CPI stacks para estimar la simbiosis entre aplicaciones y poder seleccionar la combinación con mejores prestaciones. Implementamos el planificador simbiótico en Linux y lo evaluamos en un sistema IBM POWER8 con cargas multiprograma. Los resultados experimentales muestran que el planificador simbiótico mejora las prestaciones en un 10.3% i en un 4.7%, con respecto a un planificador aleatorio i el planificador de Linux, respectivamente.

Palabras clave— Planificación simbiótica, modelo de interferencia, estimación de prestaciones.

I. INTRODUCCIÓN

LA era actual de los procesadores manycore y manythread presenta un gran número de desafíos para la comunidad científica, entre los que podrían destacar la programación paralela eficiente, la gestión inteligente de la energía o la prevención de congestión en la red. El último desafío consiste en extraer las máximas prestaciones, limitando el consumo energético y garantizando una ejecución confiable. El planificador de procesos es un componente importante en los sistemas manycore y/o manythread para abordar estos objetivos, puesto que habitualmente existe una gran cantidad de formas diferentes de planificar múltiples hilos o aplicaciones, cada una con unas prestaciones diferentes debido a la comparación de recursos entre las aplicaciones. Seleccionar la planificación óptima permite obtener mejoras significativas en las prestaciones y/o consumo del sistema.

Elegir que aplicaciones deben ejecutarse en cada núcleo o contexto de ejecución afecta fuertemente a

las prestaciones, pues los núcleos comparten recursos por los que las aplicaciones compiten. Es por ello que las aplicaciones pueden interferir entre ellas provocando, generalmente, degradación en sus prestaciones. El nivel de compartición de los recursos varía en función del contexto en el que se ejecuta cada aplicación. Mientras que todos los núcleos del chip comparten habitualmente la memoria principal, las caches pueden estar compartidas por subconjuntos de núcleos, y los hilos en un núcleo con soporte para la ejecución simultánea de procesos (SMT) comparten la mayoría de los recursos internos del núcleo. Una buena planificación debe minimizar las interferencias negativas, planificando aplicaciones complementarias en los contextos de ejecución apropiados.

En la actualidad, la arquitectura prevalente para procesadores de altas prestaciones es la del procesador multinúcleo (CMP) formado por núcleos SMT. La planificación para este tipo de procesadores es particularmente desafiante, puesto que las prestaciones de los núcleos SMT son muy sensibles a las características de las aplicaciones que se ejecutan de forma concurrente. Cuando el número de aplicaciones supera el de núcleos, el planificador debe decidir que aplicaciones deben ejecutarse simultáneamente en un mismo núcleo SMT. Dado un conjunto de aplicaciones, seleccionar la planificación óptima es un problema NP-complejo [1], y predecir las prestaciones de una planificación (o combinación de aplicaciones) concreta no es una tarea trivial debido a la gran cantidad de recursos compartidos en los núcleos SMT.

En este artículo presentamos un nuevo algoritmo de planificación para el IBM POWER8 [2], un procesador multinúcleo donde cada núcleo es SMT y puede ejecutar hasta ocho hilos de forma simultánea. Con este diseño ofrece tanto altas prestaciones para un único hilo por medio de sus núcleos fuera de orden, como gran paralelismo con hasta 80 hilos de ejecución concurrentes (10 núcleos con soporte cada uno para la ejecución de 8 hilos). El sistema es escogido para este estudio por su gran número de núcleos, lo que aumenta la complejidad de la planificación, y porque dispone de una arquitectura de contadores de prestaciones muy extensa, incluyendo el soporte para la obtención de pilas de CPI o *CPI stacks* (ver Sección III-A).

El trabajo previo en planificación simbiótica de procesos utiliza el muestreo para explorar el espacio de planificaciones posibles [3], confía en soporte de extensiones hardware no disponibles en sistemas actuales [4], o realiza amplios análisis offline para predecir las interferencias entre las aplicaciones [5]. A diferencia de estos trabajos, nosotros proponemos

¹Departamento de Informática de Sistemas y Computadores (DISCA), Universitat Politècnica de València, e-mail: jofepre@gap.upv.es, {jsahuqui,spetit}@disca.upv.es

²Intel Belgium, e-mail: stijn.eyerman@intel.com. Este trabajo ha sido realizado mientras Stijn Eyerman estaba en la Universidad de Gante.

una planificación online basada en modelos, sin realizar un muestreado extenso y aplicable a procesadores comercial actuales. Para ello, explotamos la capacidad del IBM POWERS para obtener CPI stacks con las que construimos un modelo capaz de predecir la simbiosis entre aplicaciones si estas se ejecutasen simultáneamente en un núcleo SMT. Utilizando este modelo, el planificador simbiótico puede evaluar las diferentes planificaciones disponibles y, por tanto, seleccionar la planificación óptima para el siguiente quantum. Dado que el planificador simbiótico monitoriza constantemente las CPI stacks de las aplicaciones, puede adaptar rápidamente la planificación al comportamiento dinámico que presentan las aplicaciones.

Las principales contribuciones de nuestro trabajo son:

- Proponemos un planificador online para procesadores multinúcleo compuestos por núcleos SMT, que no requiere realizar muestreado extensivo de las aplicaciones ni utilizar extensiones hardware.
- Desarrollamos un modelo de interferencia comprensible para núcleos SMT basado en CPI stacks, que considera la contención en todos los recursos compartidos del núcleo: *pipeline*, unidades funcionales, cache y memoria. El planificador simbiótico utiliza este modelo para planificar las aplicaciones maximizando las prestaciones.
- Implementamos el planificador simbiótico en Linux y lo evaluamos en un sistema IBM POWERS con cargas multiprograma.

El planificador simbiótico mejora las prestaciones de un planificador aleatorio y del planificador de Linux, en promedio, en un 10.3% y un 4.7%, respectivamente, a lo largo de las cargas evaluadas, que varían entre 8 y 20 aplicaciones, y se ejecutan en el modo SMT2 (2 hilos de ejecución por núcleo). Para cargas de entre 10 y 14 aplicaciones se alcanzan las mayores mejoras, incrementado las prestaciones del planificador de Linux en un 7.0%. Gracias a los modelos de interferencia utilizados, la sobrecarga de explorar las posibles planificaciones es muy baja. Además, nuestro planificador es completamente software y podría empaquetarse sin cambios en una distribución de Linux destinada al IBM POWERS. Por último, destacar que los modelos y el planificador propuesto también podrían adaptarse a otras arquitecturas simplemente entrenando los modelos en el nuevo sistema.

II. TRABAJO PREVIO

El soporte para la ejecución simultánea de procesos (SMT) fue propuesto por Tullsen et al. [6] como una forma de aumentar la utilización y productividad de los procesadores. Implementar el soporte para la ejecución simultánea de procesos en un núcleo supone incrementar su área y consumo energético entre un 5% y un 20% [7], [8], principalmente debido a la replicación de estructuras para mantener el es-

tado de los procesos y estructuras críticas para las prestaciones, pero también puede suponer una importante mejora de su productividad. Recientemente, Eyerman y Eeckhout [9] demuestran que, además de la productividad, otro beneficio de los procesadores multinúcleo con núcleos SMT es que son muy flexibles cuando se varía el número de hilos de ejecución por núcleo. Esta flexibilidad les permite ofrecer unas altas prestaciones por hilo cuando el número de hilos en ejecución es reducido e incrementar la productividad total cuando el número de hilos en ejecución crece. De esta forma, pueden incluso llegar a considerarse como una solución mejor y más adaptable que sistemas heterogéneos, donde el conjunto de núcleos grandes y pequeños es fijo.

Tras la introducción de los núcleos SMT, la importancia de seleccionar de forma inteligente las aplicaciones que se ejecutan concurrentemente fue rápidamente identificada. La mejora de prestaciones que esta planificación puede proporcionar depende en gran medida de las características de las aplicaciones a ejecutar, puesto que algunas combinaciones pueden incluso degradar la productividad total del sistema, por ejemplo, debido a *cache trashing* [10]. Snaveley y Tullsen [3] fueron los primeros en proponer un mecanismo para decidir que aplicaciones deberían ejecutarse concurrentemente para maximizar la productividad del sistema. Al principio de cada quantum, proponían ejecutar brevemente (al menos) un subconjunto de las posibles combinaciones de aplicaciones para, posteriormente, seleccionar y ejecutar la que obtuviera mejores prestaciones. Dado que el número de combinaciones diferentes crece rápidamente con el número de aplicaciones, la sobrecarga de muestrear las posibles combinaciones pronto supone una sobrecarga muy importante. Para evitarla, Eyerman y Eeckhout [4] proponen una planificación basada en modelos de interferencia. Un rápido modelo analítico predice la degradación de prestaciones para cada aplicación en cada una de las posibles combinaciones, de forma que se puede seleccionar la combinación con mejores prestaciones en cada quantum. Sin embargo, las entradas para el modelo que proponen se generan utilizando extensiones hardware no disponibles en sistemas comerciales actuales. Nuestra propuesta emplea un modelo de interferencia similar, pero utilizando únicamente entradas que pueden obtenerse con los contadores de prestaciones actuales de la mayoría de procesadores actuales.

Otros estudios también han explorado el uso de modelos y el muestreado para estimar las mejoras de prestaciones en procesadores SMT. Moseley et al. [11] usan regresión sobre medidas de contadores de prestaciones para estimar la aceleración que se puede alcanzar al ejecutar dos aplicaciones en un núcleo SMT. Porter et al. [12] estiman las mejoras al ejecutar aplicaciones multihilo en sistemas SMT a partir de contadores de prestaciones y técnicas de aprendizaje automático. Settle et al. [13] predicen la simbiosis utilizando mapas de actividad en la cache obtenidos de forma offline. Felu et al. [14] propo-

nen equilibrar los requisitos de ancho de banda en la cache L1 a lo largo de los núcleos para reducir la interferencia y mejorar la productividad. Zhang et al. [5] presentan una metodología para predecir la interferencia entre aplicaciones en núcleos SMT. Para ello desarrollan pequeños microbenchmarks llamados “rulers” que estresan los diferentes recursos del núcleo, lo que les permite determinar la sensibilidad de las aplicaciones a la contención en cada recurso. Después combinando la sensibilidad con la utilización de recursos se guía la planificación en cada quantum. Nuestra propuesta no requiere de ninguna fase de muestreo para aplicaciones nuevas y considera la contención en todos los recursos compartidos en el núcleo, además de las caches y la memoria principal.

III. PREDICIENDO LA SIMBIOSIS ENTRE PROCESOS

Nuestro planificador simbiótico para procesadores multinúcleo SMT se basa en un modelo que estima la simbiosis entre aplicaciones. El modelo predice, para cualquier combinación de aplicaciones (o planificación), cuanta degradación de prestaciones o *slowdown* sufrirá cada una de las aplicaciones si la combinación se ejecutan concurrentemente en un núcleo SMT. El modelo es rápido, lo que permite explorar todas las posibles combinaciones de aplicaciones, y únicamente requiere como entradas valores que pueden obtenerse fácilmente utilizando los contadores de prestaciones.

A. Modelo de interferencia en núcleos SMT

El modelo utilizado en nuestro planificador simbiótico se basa en el propuesto por Eyerman y Eeckhout [4], el cual aprovecha las CPI stacks para predecir la simbiosis entre aplicaciones. Una CPI stack (o pila de CPI) divide los ciclos de ejecución de una aplicación en varios componentes o eventos, cuantificando cuanto tiempo se utiliza o pierde en cada uno de ellos, ver la figura 1 a la izquierda donde se muestra una CPI stack con tres componentes. El componente base hace referencia al CPI ideal, en ausencia de eventos de fallo o ciclos de parada. Los componentes restantes contabilizan los ciclos *perdidos*, donde el procesador no es capaz de completar ninguna instrucción debido a diferentes eventos de fallo (por ejemplo, fallos en las caches de datos o instrucciones, o errores del predictor de saltos) o ciclos de parada provocados por diferentes recursos y unidades de ejecución del núcleo (por ejemplo, las unidades aritméticas de enteros o de coma flotante).

El modelo de interferencia propuesto utiliza la CPI stack de una aplicación cuando se ejecuta en solitario en modo single-thread (ST), y predice su slowdown estimando el incremento de cada uno de los componentes debido a la interferencia, ver la figura 1 a la derecha. Eyerman y Eeckhout [4] estiman la interferencia interpretando los componentes normalizados de la CPI stack como probabilidades, y calculando la probabilidad de que los eventos de diferentes aplicaciones puedan interferir. Por ejemplo, si una aplicación pasa la mitad de sus ciclos buscando

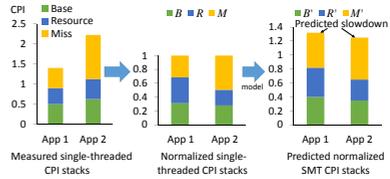


Fig. 1. Resumen del modelo: primero, las CPI stacks obtenidas se normalizan para obtener probabilidades; después, el modelo predice el incremento de los componentes y el slowdown resultante (1.32 para la App 1 y 1.25 para la App 2).

instrucciones, y la otra aplicación una tercera parte, existe una probabilidad de 1/6 de que ambas aplicaciones traten de buscar instrucciones en el mismo ciclo, lo cual provocará interferencia porque la unidad de búsqueda de instrucciones es compartida. Sin embargo, Eyerman y Eeckhout utilizan extensiones hardware novedosas [15] y no disponibles en procesadores comerciales actuales para obtener los componentes de las CPI stacks en ST durante la ejecución simultánea (SMT) de un conjunto de aplicaciones.

De forma interesante, el IBM POWER8 dispone de una unidad de monitorización de prestaciones (PMU) capaz de generar CPI stacks tanto en modo ST como en modo SMT. Sin embargo, los componentes o eventos que se pueden obtener difieren de los que proponen para la PMU de núcleos SMT Eyerman et al. [15], lo que significa que el modelo que proponen en [4] no puede utilizarse directamente. Algunos de los componentes están relacionados, pero no contabilizan exactamente el mismo evento. Otros eventos no se consideran una penalización en alguno de los mecanismos pero sí en el otro. Debido a estas diferencias, desarrollamos un modelo nuevo para estimar el slowdown provocado por las aplicaciones cuando se ejecutan simultáneamente en un núcleo SMT. El modelo propuesto utiliza regresión, por lo que es más empírico que el modelo puramente analítico propuesto por Eyerman y Eeckhout [4], pero la idea principal es similar: normalizar las CPI stacks dividiendo cada componente por el CPI total, e interpretar cada componente como una probabilidad. Después se calculan las probabilidades de que los eventos de las aplicaciones de una combinación ocurran al mismo tiempo si las aplicaciones se ejecutan simultáneamente, lo que generaría un retraso que se añade a la CPI stack como interferencia.

En [4], Eyerman y Eeckhout dividen los componentes en tres categorías: componente base, ciclos de parada provocados por recursos y eventos de fallo. Aunque el razonamiento para cada componente puede variar ligeramente, en nuestro modelo utilizamos la misma ecuación para modelar todos los componentes, y dejamos con el valor de los parámetros, el cual determinamos con regresión lineal, adapte la ecuación al comportamiento particular de cada componente. En este trabajo asumimos que disponemos

de las CPI stacks de las aplicaciones en modo ST. Las CPI stacks se normalizan dividiendo cada componente por el CPI total de la aplicación, ver la figura 1. Las CPI stacks en modo ST podrían haberse obtenido de forma off-line durante ejecuciones de las aplicaciones en solitario. No obstante, como se explica en [16], el modelo que proponemos puede invertirse para estimar las ST CPI stacks a partir de las SMT CPI stacks que se obtienen en tiempo de ejecución con los contadores de prestaciones.

B. Ecuación del modelo

El modelo de interferencia SMT que proponemos utiliza la ecuación 1 para predecir el componente SMT de una aplicación cuando se ejecuta en una combinación, es decir, cuando se ejecuta concurrentemente con otras aplicaciones en un núcleo SMT. El componente SMT de la aplicación se predice a partir de su componente ST (el componente cuando la aplicación se ejecuta en solitario) y los componentes ST del resto de aplicaciones de la combinación. Para una aplicación j , C_j representa su componente en modo ST, C'_j identifica el componente SMT de la misma aplicación, y C_k representa el componente ST del resto de aplicaciones de la combinación.

$$C'_j = \alpha + \beta C_j + \gamma \sum_{k \neq j} C_k + \delta C_j \sum_{k \neq j} C_k \quad (1)$$

Los parámetros α , β , γ y δ se determinan por medio de regresión, ver la sección III-C. Aunque el significado de cada término puede variar ligeramente en función del componente, a continuación se describen los términos de la ecuación de forma general. α refleja un potencial incremento constante del componente en ejecución SMT con respecto a su ejecución ST. El término β refleja el hecho de que el componente ST original, o parte de él, permanezca en la ejecución SMT. Sería intuitivo fijar β a uno, pero los siguientes términos de la ecuación podrían cubrir parte del componente original. Al mismo tiempo, también podría producirse un incremento relativo al componente ST por lo que β podría ser superior a 1. γ modela el impacto de la suma de los componentes ST de las otras aplicaciones de la combinación en el componente SMT de la aplicación j . Finalmente, δ modela alguna interacción extra que pueda ocurrir entre los componentes ST de la aplicación j y el resto de aplicaciones del conjunto. A pesar de que pueda parecer que no todos los términos tienen un significado claro para algunos componentes, decidimos mantenerlos en la ecuación para obtener un modelo general capaz de modelar todas las posibles interacciones. De este modo se deja que la regresión determine los parámetros que deben tener mayor o menor peso para cada componente.

C. Construcción del modelo y estimación del slowdown

Los parámetros del modelo se determinan por regresión lineal basada en datos experimentales de entrenamiento. Se trata de una aproximación menos

rigurosa que la usada por Eyerman y Eeckhout en su modelo [4], que es construido casi completamente de forma analítica, pero como hemos explicado se debe a que las CPI stacks obtenidas con el IBM POEWR8 son diferentes y además, algunos eventos no están completamente documentados. Para entrenar el modelo, en primer lugar ejecutamos todas las aplicaciones en solitario, guardando sus CPI stacks cada quantum (100ms). También guardamos el número de instrucciones en el que empieza cada quantum para poder identificar que parte de la aplicación está siendo ejecutada y normalizamos cada CPI stack respecto a su CPI total.

A continuación, ejecutamos todas las combinaciones posibles de dos aplicaciones en un único núcleo en modo SMT2. También guardamos las CPI stacks para cada aplicación junto con su cuenta de instrucciones. Después normalizamos las CPI stacks con respecto al CPI total de ejecutar las mismas instrucciones en solitario en modo ST. Normalizamos respecto al CPI en modo ST porque queremos estimar el slowdown que cada aplicación sufre con respecto a su ejecución en solitario, lo cual equivale a su CPI en modo SMT (en ejecución concurrente junto con las otras aplicaciones de la combinación) dividido por su CPI en solitario en modo ST (ver la figura 1). Estos pasos también coinciden con la metodología utilizada en [4]. El CPI en modo ST se calcula a partir del número de instrucciones y ciclos obtenidos en el muestreo de las aplicaciones en solitario. Dado que las prestaciones de las aplicaciones difieren entre los modos ST y SMT, y los quanta son de tiempo fijo, las cuentas de instrucciones no coinciden entre los quanta en modo ST y en modo SMT. Para resolverlo, interpolamos las ST CPI stacks entre dos quanta para asegurarnos que las CPI stacks en los modos ST y SMT cubren aproximadamente las mismas instrucciones.

Utilizando esta información (CPI stacks normalizadas por quantum para todas las aplicaciones en ST y para todas las combinaciones de aplicaciones en SMT), determinamos los parámetros del modelo. Existe un conjunto de parámetros (α , β , γ , δ) por componente. Los parámetros se asocian a componentes de la CPI stack y no a aplicaciones específicas. Por ello, asumiendo que el conjunto de aplicaciones de entrenamiento es lo suficientemente diverso, no es necesario volver a entrenar el modelo para incluir aplicaciones nuevas. Además, dado el gran número de casos de entrenamiento no existe riesgo de sobreajustar el modelo a alguna aplicación concreta.

Una vez construido el modelo, lo podemos utilizar para estimar las CPI stacks en modo SMT de cualquier conjunto de aplicaciones si éste se ejecutase concurrentemente en un núcleo SMT a partir de las CPI stacks en modo ST de cada aplicación. Para ello, en primer lugar se calcula el componente SMT de las aplicaciones de la combinación para todos los componentes que forman las CPI stacks utilizando la ecuación 1. Después se suman todos los componentes SMT para cada aplicación. El valor resultante será

Evento	Explicación
PM_GRP_CMPPL PM_CMPLU_STALL	Ciclos donde el hilo completa instrucciones. Éste es el componente base de las CPI stacks. Ciclos donde el hilo no puede completar instrucciones porque éstas no están finalizadas. <i>Incluye ciclos de parada debidos a unidades funcionales y fallos en la cache de datos.</i>
PM_GCT_NOSLOT_CVC	Ciclos donde el hilo no dispone de instrucciones en el ROB debido a fallos en la cache de instrucciones o errores del predictor de saltos.
PM_CMPLU_STALL_THRD	Ciclos donde el hilo no puede completar instrucciones porque el puerto para confirmarias está ocupado por otros hilo.
PM_NTCCG_ALL_FIN	Ciclos donde todas las instrucciones en el grupo han finalizado pero su confirmación aún está pendiente.

TABLA I
 RESUMEN DE LOS EVENTOS DEL IBM POWERS8 CONTABILIZADOS PARA CONSTRUIR LAS CPI STACKS.

mayor que uno e indica el slowdown que la aplicación sufrirá si se ejecuta en esa combinación. Esta información es utilizada por el planificador simbiótico para seleccionar las combinaciones con menor degradación de prestaciones (ver la sección IV).

IV. PLANIFICACIÓN CONSIDERANDO LA INTERFERENCIA EN SMT

En esta sección describimos la implementación del planificador simbiótico, que utiliza el modelo de interferencia SMT propuesto para mejorar las prestaciones del sistema. El objetivo del planificador es distribuir n aplicaciones en c núcleos, siendo $n > c$, con el objetivo de maximizar productividad. Cada núcleo puede ejecutar $\lceil \frac{n}{c} \rceil$ hilos utilizando SMT. Obsérvese que el problema de seleccionar n aplicaciones de un conjunto mayor que n no forma parte del objetivo de este trabajo. Se asume qué, o bien esta selección ya ha sido hecha, o el número de aplicaciones es igual o inferior al número de contextos de ejecución disponibles. Como se describe en la sección V, implementamos nuestro planificador en Linux y evaluamos sus prestaciones en un sistema IBM POWERS8. La implementación del planificador comprende diversos pasos que se discuten en las próximas secciones.

A. Reducción de los componentes de las CPI stacks

La CPI stack más detallada que se puede obtener con el IBM POWERS8 incluye 45 eventos. Sin embargo, su unidad de monitorización de prestaciones (PMU) solamente implementa seis contadores. Cuatro de estos contadores son programables y los dos restantes contabilizan el número de instrucciones completadas y ciclos en ejecución. Además, algunos eventos son incompatibles entre ellos y no pueden contabilizarse de forma simultánea, por lo que se requieren 19 intervalos o quanta para poder actualizar por completo una CPI stack. Esto implica que en el momento en el que se actualizan los últimos componentes de la CPI stack, otros componentes disponen de un valor obtenido 18 intervalos de ejecución antes y que por tanto, puede estar ya obsoleto. Este problema puede convertir el planificador simbiótico en poco reactivo ante cambios de comportamiento en las aplicaciones en el mejor de los casos y completamente inservible en el peor.

Una característica interesante de las CPI stacks del IBM POWERS8 es que están construidas de forma jerárquica, con un nivel superior formado por cin-

co componentes y múltiples niveles inferiores, donde cada componente se divide en diversos componentes más detallados en niveles inferiores [17]. Por ejemplo, el evento *completion_stalls* del primer nivel contabiliza los ciclos de parada causados por diferentes unidades de ejecución del núcleo. Este componente se divide en niveles inferiores en diversos componentes que contabilizan los ciclos de parada para recursos como la unidad de coma fija, la unidad vectorial-escalar o la unidad de carga y almacenamiento. Para mejorar la capacidad de reacción del planificador ante cambios de comportamiento de las aplicaciones y reducir el tiempo de cálculo del modelo, únicamente consideramos los eventos del primer nivel de las CPI stacks. Esto reduce el número de intervalos para poder actualizar por completo las CPI stacks a solamente dos. Los eventos o componentes que contabilizamos se detallan en la tabla I. Aunque la precisión del modelo puede incrementarse considerando un mayor número de eventos, el planificador alcanza peores prestaciones por tener que predecir la simbiosis entre aplicaciones con componentes de las CPI stacks obsoletos.

B. Selección de la planificación óptima

El planificador utiliza las CPI stacks obtenidas para las aplicaciones en tiempo de ejecución (SMT) y el modelo de interferencia propuesto para ubicar las aplicaciones en los núcleos disponibles. Para simplificar las decisiones del planificador se toman dos asunciones:

- Todos los núcleos son homogéneos y por tanto, tienen el mismo modelo de interferencia. El planificador podría extenderse a arquitecturas heterogéneas construyendo un modelo para cada tipo de núcleo.
- La interferencia en los recursos compartidos por todos los núcleos (caches del último nivel y memoria principal) está determinada principalmente por las características de las aplicaciones que se ejecutan y no por cómo se ubican en los núcleos. Esta observación también fue realizada por Radjoković et al. [18]. Como resultado, con un conjunto fijo de aplicaciones en ejecución, la planificación no impacta en las interferencias en estos recursos y por tanto, no es necesario considerarlos para realizar la planificación.

A pesar de estas asunciones, el número de combi-

naciones posibles es habitualmente demasiado grande como para realizar una búsqueda exhaustiva, incluso con los rápidos modelos que proponemos. El número de formas diferentes de dividir n aplicaciones en c núcleos es $\frac{n!}{c!(\frac{n}{c})^c}$ (asumiendo que n es múltiplo de c). Para dividir 16 aplicaciones en 8 núcleos, ya existen más de dos millones de combinaciones posibles. Evaluar cada combinación implicaría una sobrecarga no despreciable. Además, Jiang et al. [1] muestran que este problema es NP-completo cuando $\frac{n}{c} > 2$. Los autores proponen modelar el problema de planificación como un problema de teoría de grafos para poder abordar un número de combinaciones tan grande de forma eficiente. En concreto, el problema se modela como un grafo para el que se pretende obtener un apareamiento perfecto de mínimo peso, el cual puede resolverse en tiempo polinómico utilizando el algoritmo blossom [19].

En resumen, el planificador propuesto realiza los siguientes pasos al inicio de cada quantum:

1. Obtiene el valor de los contadores de prestaciones, con los que construye la CPI stack en modo SMT para cada aplicación a lo largo de su último quantum en ejecución.
2. Utiliza el modelo de interferencia SMT invertido [16] para obtener una estimación de las CPI stacks en modo ST de cada aplicación a partir de sus CPI stacks en modo SMT.
3. Utiliza el modelo de interferencia SMT para predecir las prestaciones de cada combinación de aplicaciones posible y selecciona la mejor combinación con el algoritmo blossom.
4. Ejecuta la combinación seleccionada durante el siguiente quantum.

V. PLATAFORMA EXPERIMENTAL

Realizamos todos los experimentos en un sistema IBM Power System S812L, que dispone en un procesador IBM POWERS de diez núcleos. Cada núcleo puede ejecutar hasta ocho hilos de ejecución de forma simultánea (SMT). Dado que evaluamos cargas multiprograma formadas por benchmarks de la suite SPEC, y la cache de L1 solamente dispone de 64KB, únicamente evaluamos el planificador simbiótico ejecutando dos hilos por núcleo (modo SMT2). Un número mayor de hilos por núcleo crea una presión excesiva en la cache L1. La posibilidad de ejecutar hasta ocho hilos de forma simultánea está diseñada para aplicación multi-hilo que comparten una gran cantidad de código y datos. Nuestro sistema utiliza la distribución de Linux Ubuntu 14.04.

Utilizamos los benchmarks de la SPEC CPU 2006 con las entradas *reference* para evaluar el planificador simbiótico. Para cada benchmark, medimos el número de instrucciones necesario para que éste se ejecute en solitario durante 120 segundos y lo guardamos como el número de instrucciones objetivo. Esto reduce la variación en el tiempo de ejecución de las aplicaciones durante los experimentos. Para las cargas multiprograma, ejecutamos todas las aplica-

ciones hasta que la última completa su número de instrucciones objetivo. Cuando una aplicación completa su número de instrucciones objetivo, guardamos su IPC y relanzamos la aplicación. Este método garantiza que comparamos la misma parte de la ejecución de cada aplicación en ejecuciones diferentes de la carga, y que la carga es uniforme durante el experimento completo. Evaluamos la productividad total del sistema (STP) por medio de la métrica *weighted speedup* [20]. En concreto, medimos el tiempo que cada aplicación requiere para ejecutar su número de instrucciones objetivo durante la ejecución de la carga. Después sumamos, para todas las aplicaciones, el cociente entre el tiempo requerido para que la aplicación ejecute las instrucciones objetivo en solitario (120 segundos) por el tiempo requerido para que las ejecute en la carga. Evaluamos 105 cargas, que varían entre combinaciones de 8 aplicaciones para su ejecución en 4 núcleos hasta combinaciones de 20 aplicaciones para su ejecución en 10 núcleos.

VI. EVALUACIÓN EXPERIMENTAL

En esta sección se evalúa el planificador simbiótico de procesos, comparando sus prestaciones contra un algoritmo de planificación aleatorio, el planificador de Linux y un algoritmo de planificación que considera la contención en el ancho de banda de L1. No obstante, antes de abordar las prestaciones de los planificadores, evaluaremos la precisión del modelo de interferencia SMT que proponemos.

A. Precisión del modelo de interferencia propuesto

La figura 2 presenta un histograma de errores para el modelo de interferencia SMT propuesto. En concreto, el histograma muestra la distribución de errores cuando se predice el slowdown que alcanzarán las aplicaciones de una combinación si se ejecutasen concurrentemente en un núcleo SMT, partir de sus CPI stacks en solitario en modo ST. Los resultados cubren todas las posibles combinaciones de aplicaciones y múltiples quanta por combinación para poder capturar el comportamiento dinámico de las aplicaciones. Se puede observar que la mayoría de los errores se encuentra por debajo del 15%, con un error medio absoluto del 12.3% y con solo un 3% de estimaciones con un error superior al 30%.

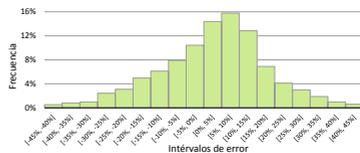


Fig. 2. Histograma de distribución de errores del modelo de interferencia SMT propuesto.

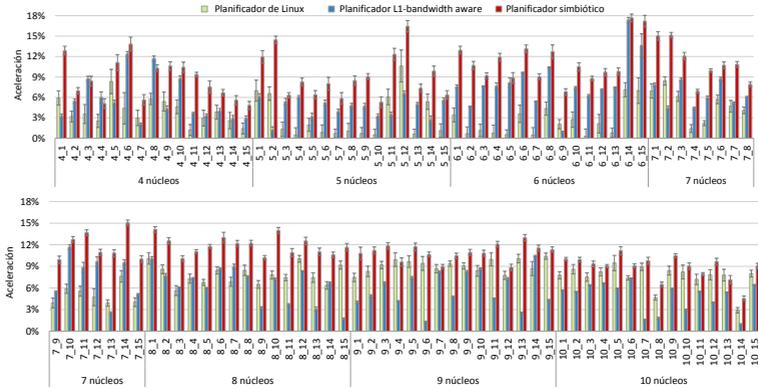


Fig. 3. Mejora de la productividad del sistema con el planificador de Linux, el planificador considerando el ancho de banda de L1 y el planificador simbiótico, con respecto al planificador aleatorio.

B. Evaluación de prestaciones

Tras comprobar que el modelo de interferencia es preciso, esta sección evalúa la mejora de prestaciones que alcanza el planificador simbiótico de procesos que hace uso de este modelo para ejecutar la mejor combinación de aplicaciones en cada quantum. También estudiamos el impacto de la planificación simbiótica en la equidad del sistema y analizamos la estabilidad de las planificaciones seleccionadas.

Para evaluar las prestaciones alcanzadas por el planificador simbiótico de procesos, comparamos cuatro planificadores diferentes:

1. Planificador aleatorio: las aplicaciones se distribuyen de forma aleatoria en los núcleos. Cada quantum se determina una nueva planificación aleatoria.
2. Planificador de Linux: el planificador por defecto de Linux, conocido como Completely Fair Scheduler (CFS).
3. Planificador considerando el ancho de banda de L1 (L1-bandwidth aware scheduler) [14]: este planificador es el más reciente y próximo a nuestra propuesta. Cada quantum trata de balancear los accesos a las caches de L1 entre los núcleos del procesador. Como el planificador simbiótico, solamente requiere de contadores de prestaciones para su funcionamiento.
4. Planificador simbiótico de procesos: nuestra propuesta.

B.1 Productividad del sistema

La figura 3 presenta la mejora de la productividad del sistema alcanzada por el planificador de Linux, el planificador considerando el ancho de banda de L1 y el planificador simbiótico, con respecto al planificador aleatorio. Cada conjunto de barras representa

una carga (o combinación de aplicaciones) diferente. Se ha evaluado un conjunto de 15 cargas para cada número de núcleos (105 cargas en total). Las cargas están formadas por dos aplicaciones por cada núcleo considerado en el experimento.

Los resultados presentados incluyen la pequeña sobrecarga de los planificadores debida a la obtención de los valores de los contadores de prestaciones y la actualizaciones de las variables de planificación. Para el planificador simbiótico también incluyen el tiempo requerido para evaluar las posibles combinaciones y seleccionar la que se ejecutará en el siguiente quantum. Las aceleraciones mostradas para cada carga y planificador representan el valor medio de 15 ejecuciones, representado también intervalos de confianza del 95%.

El planificador simbiótico claramente mejora las prestaciones del resto de planificadores independientemente del número de núcleos considerado. La mejora de la productividad media a lo largo de todas las cargas es del 10.3% con respecto al planificador aleatorio, del 4.7% con respecto al planificador de Linux y del 4.1% con respecto al planificador que considera el ancho de banda de L1. Si solamente consideramos las cargas medias (entre 5 y 7 núcleos), la mejora de la productividad con respecto a Linux crece hasta un 7.0%. En comparación con cargas más pequeñas, las cargas medias ofrecen un mayor número de combinaciones, por lo que la diferencia entre las mejores y peores combinaciones aumenta. Para las cargas más grandes, las interferencias en la cache del último nivel y la memoria principal actúan como cuello de botella, limitando las mejoras alcanzadas por el planificador simbiótico. Como explicaremos a continuación, Linux es capaz de controlar la contención en la memoria principal y por eso mejora sus prestaciones relativas en las cargas más grandes.

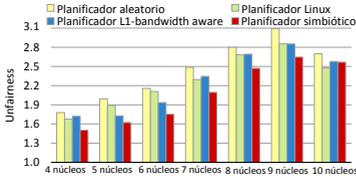


Fig. 4. Inequidad media (mejor cuanto más baja) de los cuatro planificadores evaluados.

En cuanto al planificador de Linux, se puede observar que su mejora de la productividad con respecto al planificador aleatorio tiende a incrementarse a medida que aumenta el número de núcleos. Esta mejora de prestaciones está relacionada con la capacidad de Linux de monitorizar los accesos a memoria principal para tratar de evitar la contención en el acceso a memoria. Esta observación concuerda con el hecho de que, en algunos casos, Linux decide pausar la ejecución de algunas aplicaciones cuando la contención en memoria es muy elevada. El planificador simbiótico también considera la contención en la memoria al realizar la planificación por medio de los ciclos de parada por fallos en las caches, y evita ubicar las aplicaciones con mayores requisitos de acceso a memoria en el mismo núcleo. Además, también considera la contención en el resto de recursos compartidos del núcleo, lo que explica sus mayores prestaciones.

Por último, las mejoras de prestaciones del planificador considerando el ancho de banda de L1 son similares a las que ofrece Linux, pero siguen una tendencia contraria, ya que su mejora se reduce al incrementarse el número de núcleos considerados. Con un número de aplicaciones relativamente bajo, la contención en memoria principal no es importante y la interferencia en las caches L1 es el principal limitador de las prestaciones. Sin embargo, a medida que aumenta el número de aplicaciones la contención en memoria crece, hasta convertirse en el cuello de botella. Las mejoras de prestaciones que observamos en nuestro sistema son inferiores a las observadas por Felu et al. en [14], lo cual se deba probablemente al hecho de que la cache L1 del IBM POWER8 dispone del doble de capacidad que la del procesador utilizado en sus experimentos, lo cual reduce la presión sobre este recurso.

B.2 Equidad del sistema

Aun que el objetivo principal del planificador simbiótico de procesos es maximizar la productividad, también evaluamos su impacto en la equidad del sistema. La equidad cuantifica como de equilibradas están las mejoras (o pérdidas) de prestaciones entre todas las aplicaciones de una carga. Planificaciones poco equitativas pueden llevar a inversión de prioridades o incluso inanición de algún proceso, a pesar de que el sistema alcance una mayor productividad. Estudiamos la equidad del sistema con la métrica

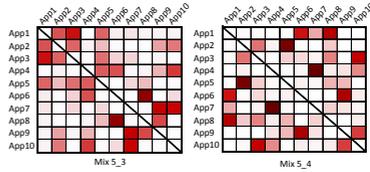


Fig. 5. Matrices de frecuencia para dos cargas de 5 núcleos.

de *unfairness* o inequidad, que se calcula como el slowdown máximo dividido por el slowdown mínimo entre todas las aplicaciones de la carga. La equidad del sistema es mejor cuanto más baja sea su inequidad. Una inequidad de 1 significa que el sistema es completamente equitativo.

La figura 4 muestra la inequidad alcanzada por los cuatro algoritmos de planificación evaluados. Las diferentes barras representan la inequidad media a lo largo de las quince cargas evaluadas para cada número de núcleos. La figura muestra que el planificador simbiótico alcanza la mejor equidad para las cargas que consideran entre 4 y 9 núcleos, seguido del planificador que considera el ancho de banda de L1 y del planificador de Linux. Como cabía espera, la mayor inequidad se alcanza con el algoritmo de planificación aleatorio. Con respecto a Linux, la mejora de la inequidad es bastante significativa. Por ejemplo, para cargas de seis núcleos, el planificador simbiótico y el planificador de Linux alcanzan una inequidad de 1.75 y 2.11, respectivamente, lo que significa que Linux es un 20% menos equitativo que nuestra propuesta. El planificador simbiótico trata de reducir las interferencias entre las aplicaciones y, como un efecto añadido, consigue reducir la inequidad del sistema.

Para diez núcleos, la inequidad se reduce para todos los algoritmos de planificación. Dado que el procesador implementa este número de núcleos, supونه que utilizar todos los núcleos lleva a una utilización más equilibrada de los recursos, lo que permite obtener en una mejor equidad. Con cargas de 10 núcleos, el planificador simbiótico es ligeramente menos equitativo que Linux, también en parte, por la capacidad de Linux para controlar la contención en memoria principal.

B.3 Patrones de simbiosis

El planificador simbiótico constantemente reevalúa la planificación óptima, lo que significa que se adapta al comportamiento dinámico de las aplicaciones actualizando, cada quantum, las combinaciones de aplicaciones que se ejecutarán concurrentemente en el mismo núcleo. Sin embargo, si las aplicaciones no presentan un comportamiento dinámico donde diferentes fases tengan diferentes características, una planificación estática sería suficiente y evitaría la sobrecarga de tener que obtener combinaciones nuevas.

La figura 5 presenta la matriz de frecuencia de las combinaciones de aplicaciones para dos cargas dife-

rentes de 5 núcleos ejecutadas con el planificador simbiótico. La matriz simétrica representa la fracción de tiempo donde cada combinación de aplicaciones se ejecuta concurrentemente en el mismo núcleo. Cuanto más oscuro es el color, más frecuentemente se ejecuta la combinación de aplicaciones de forma simultánea en el mismo núcleo SMT.

Las dos matrices representan los dos comportamientos que observamos en las ejecuciones de las cargas con el planificador simbiótico. La matriz de frecuencia de la carga 5.4 (a la derecha) muestra una carga donde dos pares de aplicaciones se planifican en el mismo núcleo de forma muy frecuente (la aplicación 2 se ejecutan concurrentemente con la aplicación 5 y la aplicación 4 con la aplicación 7, en un 66% y 70% de los quantas). Este comportamiento es representativo de aplicaciones que tienen una gran simbiosis entre ellas (por ejemplo, una aplicación que hace un uso intenso de la memoria y otra que lo hace del procesador) y poco variación del comportamiento de forma dinámica. Una matriz de frecuencia opuesta se observa para la carga 5.3. En este caso no existe ninguna combinación de aplicaciones excesivamente predominante, lo que hace especialmente importante poder adaptar la planificación al comportamiento que las aplicaciones presentan de forma dinámica en tiempo de ejecución.

VII. CONCLUSIONES Y TRABAJO FUTURO

La planificación de procesos tiene un impacto considerable en procesadores que permiten ejecutar un gran número de aplicaciones concurrentemente, debido a la interferencia que aparece entre las aplicaciones en los recursos compartidos. En este trabajo proponemos un algoritmo de planificación simbiótico para procesadores multinúcleo con núcleos SMT. El planificador utiliza un modelo de interferencia SMT basado en CPI stacks, y no requiere de un muestreo extensivo de las aplicaciones. La evaluación experimental realizada en un sistema IBM POWERS muestra que nuestro algoritmo de planificación mejora la productividad un 10.3% y 4.7%, en promedio, con respecto a un planificador aleatorio y al planificador de Linux, respectivamente, a lo largo de las 105 cargas evaluadas. Además, debido al modelo de interferencia SMT que utilizamos, la sobrecarga de encontrar la planificación óptima para cada quantum es mínima.

Aunque nuestra implementación actual está diseñada para el IBM POWERS puede adaptarse fácilmente a cualquier otra arquitectura con un procesador multinúcleo con núcleos SMT, siempre que sus contadores de prestaciones permitan obtener CPI stacks como ocurre en la mayoría de procesadores recientes. Además, el planificador también podría extenderse para considerar arquitecturas heterogéneas, utilizando un modelo diferente para cada tipo de núcleo.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad (MINECO) y por los fondos FEDER mediante subvenciones a los proyectos TIN2015-66972-C5-1-R y TIN2014-62246-EXP, así como por el programa Intel Early Career Honor Award y por la Generalitat Valenciana mediante subvención al proyecto AICO/2016/059.

REFERENCIAS

- [1] Yunlian Jiang, Xipeng Shen, Jie Chen, and Rahul Tripathi, "Analysis and approximation of optimal co-scheduling on chip multiprocessors," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008, pp. 220–229.
- [2] B. Sinharoy, J. A. Van Norstrand, R.J. Eickemeyer, H.Q. Le, J. Leenstra, D.Q. Nguyen, B. Konigsburg, K. Ward, M.D. Brown, J.E. Moreira, D. Levitan, S. Tung, D. Hruscky, J.W. Bishop, M. Gschwind, M. Boersma, M. Kroeber, M. Kaltenbach, T. Karkhanis, and K.M. Fensler, "Ibm power8 processor core microarchitecture," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 2:1–2:21, Jan 2015.
- [3] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for simultaneous multithreading processor," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Nov. 2000, pp. 234–244.
- [4] S. Eyerman and L. Eeckhout, "Probabilistic job symbiosis modeling for SMT processor scheduling," in *The International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2010, pp. 91–102.
- [5] Yunqi Zhang, Michael A. Laurenzano, Jason Mars, and Lingjia Tang, "SMITe: Precise QoS prediction on real-system SMT processors to improve utilization in warehouse scale computers," in *International Symposium on Microarchitecture (MICRO)*, 2014, pp. 406–418.
- [6] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA)*, June 1995, pp. 392–403.
- [7] James Burns and Jean-Luc Gaudiot, "SMT layout overhead and scalability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 2, pp. 142–155, 2002.
- [8] Yingmin Li, Kevin Skadron, David Brooks, and Zhigang Hu, "Performance, energy, and thermal considerations for SMT and CMP architectures," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2005, pp. 71–82.
- [9] Stijn Eyerman and Lieven Eeckhout, "The benefit of SMT in the multi-core era: Flexibility towards degrees of thread-level parallelism," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014, pp. 591–606.
- [10] Sébastien Hily and André Seznec, "Contention on 2nd level cache may limit the effectiveness of simultaneous multithreading," 1997.
- [11] T. Moseley, J.L. Kilm, D.A. Connors, and D. Grunwald, "Methods for modeling resource contention on simultaneous multithreading processors," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Oct 2005, pp. 373–380.
- [12] Leo Porter, Michael A. Laurenzano, Ananta Tiwari, Adam Jundt, William A. Ward, Jr., Roy Campbell, and Laura Carrington, "Making the most of SMT in HPC: System- and application-level perspectives," *ACM Transactions on Architecture and Code Optimization (TAACO)*, vol. 11, no. 4, pp. 59:1–59:26, Jan. 2015.
- [13] Alex Settle, Joshua Kilm, Andrew Janiszewski, and Dan Connors, "Architectural support for enhanced SMT job scheduling," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2004, pp. 63–73.
- [14] Josué Felíu, Julio Sahuquillo, Salvador Petit, and José Duato, "L1-bandwidth aware thread allocation in

- multicore SMT processors,” in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2013, pp. 123–132.
- [15] S. Eyerman and L. Eeckhout, “Per-thread cycle accounting in SMT processors,” in *The International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2009, pp. 133–144.
- [16] Josué Felíu, Stijn Eyerman, Julio Sahuquillo, and Salvador Petit, “Symbiotic Job Scheduling on the IBM POWERS,” in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2016, pp. 669–680.
- [17] “IBM Knowledge Center, *Analyzing application performance on Power Systems servers*,” 2015.
- [18] P. Radojkovic, V. Cakarevic, J. Verdu, A. Pajuelo, F.J. Cazorla, M. Nemirovsky, and M. Valero, “Thread assignment of multithreaded network applications in multicore/multithreaded processors,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 12, pp. 2513–2525, Dec 2013.
- [19] Jack Edmonds, “Maximum matching and a polyhedron with 0, 1-vertices,” *J. Res. Nat. Bur. Standards B*, vol. 69, no. 1965, pp. 125–130, 1965.
- [20] Stijn Eyerman and Lieven Eeckhout, “System-level performance metrics for multiprogram workloads,” *IEEE Micro*, vol. 28, no. 3, pp. 42–53, 2008.

Entorno de planificación para sistemas de tiempo real con un procesador SMT comercial

Clara Furió, Josué Felín, Julio Sahuquillo, Salvador Petit y Houcine Hassan¹

Resumen.— Tradicionalmente, el estudio de los sistemas de tiempo real se ha centrado en sistemas hardware lo suficientemente sencillos para facilitar el modelado analítico de la predictibilidad temporal del sistema y su planificabilidad. Esto implica que usualmente los procesadores incluidos en los sistemas de tiempo real no implementan características avanzadas que incrementen la variabilidad de las prestaciones. Sin embargo, la convergencia entre los sistemas de altas prestaciones y los sistemas móviles y empotrados implica que muchas de las características avanzadas de los primeros están siendo introducidas en los últimos.

En este contexto, el uso de modelos analíticos resulta insuficiente para estudiar el comportamiento de los sistemas de tiempo real modernos. En este trabajo, se propone y verifica un nuevo entorno para el estudio de la planificación de sistemas de tiempo real con procesadores multinúcleo SMT. El entorno propuesto funciona sobre el sistema operativo Linux y permite cambiar tanto el algoritmo de planificación como la asignación de las aplicaciones a los hilos de ejecución sin requerir la modificación del núcleo del sistema operativo, lo que facilita y acelera el desarrollo de nuevas propuestas de planificación para sistemas de tiempo real con procesadores actuales.

El entorno ha sido verificado mediante la implementación de dos algoritmos de planificación de sistemas de tiempo real: Earliest Deadline First (EDF) y Rate Monotonic Scheduling (RMS), ejecutando ambos benchmarks reales sobre un procesador Intel Xeon con núcleos de cómputo SMT.

Palabras clave.— SMT processors, Process scheduling, Real-time systems.

I. INTRODUCCIÓN

LOS sistemas de tiempo real han cobrado gran importancia recientemente debido a su amplio uso en una gran cantidad de dispositivos y aplicaciones industriales (aviónica, robótica, control de procesos, etc.) [1]. Tradicionalmente estos sistemas han utilizado procesadores simples pues son predecibles y más fáciles de modelar para acotar el tiempo de ejecución en el peor caso (*Worst Case Execution Time* o WCET) de las aplicaciones. Además, el uso de procesadores de bajas prestaciones abarata el coste de adquisición.

El estudio de los sistemas de tiempo real se ha centrado principalmente en garantizar la planificabilidad de las aplicaciones o carga a ejecutar. Para ello, habitualmente se han utilizado modelos analíticos que utilizan parámetros tales como el tiempo requerido para la finalización de las aplicaciones (también conocido como *deadline*), su periodicidad y su tiempo de cómputo [2], [3].

Los modelos analíticos son de gran utilidad para estudiar el comportamiento de un gran abanico de cargas desde el punto de vista teórico. Sin embargo, debido a la simplicidad de estos modelos, resultan poco apropiados para sistemas con procesadores reales relativamente complejos. Para atacar este problema, parte de la investigación reciente utiliza simuladores de sistemas ciclo-a-ciclo que permiten modelar la ejecución detallada de las aplicaciones o *benchmarks* representativos en sistemas de tiempo real. Estos simuladores modelan la microarquitectura del procesador y el subsistema de memoria en cada uno de los ciclos de ejecución con lo que su precisión es relativamente alta. Este enfoque es más preciso desde el punto de vista de tiempo de ejecución e interferencias entre las aplicaciones que el modelado analítico. No obstante, los simuladores pueden presentar desviaciones importantes respecto a las ejecuciones en sistemas reales. Por último, algunos estudios se han centrado en sistemas reales [4]. Sin embargo, estos sistemas utilizaban procesadores mono-núcleo sencillos y sus cargas consideraban un número reducido de tareas.

Por otro lado, actualmente existe una convergencia entre los procesadores utilizados en los sistemas de altas prestaciones y los utilizados en sistemas móviles (teléfonos inteligentes o tabletas), sistemas empotrados, sistemas de tiempo real, etc. Efectivamente, el diseño de nuevos procesadores con avanzadas características que mejoran las prestaciones; por ejemplo, la capacidad de ejecutar múltiples hilos simultáneamente en un mismo núcleo de cómputo (*Simultaneous Multithreading* o SMT), presenta inicialmente un coste relativamente alto. En consecuencia, estos diseños se implementan en primer lugar en sistemas de altas prestaciones y servidores de cómputo. Posteriormente, los avances en el sistema de fabricación y el retorno de la inversión inicial permiten reducir costes, lo que conlleva la inclusión de estas características avanzadas en sistemas menos complejos, permitiendo mejorar sus prestaciones con un coste relativamente bajo.

Los sistemas de tiempo real cada vez requieren un mayor número de funcionalidades por lo que es de esperar que los procesadores actuales de altas prestaciones, o al menos, procesadores con similares características se incorporen a estos sistemas. Cabe por tanto prever que los futuros sistemas de tiempo real incorporen procesadores multinúcleo con núcleos SMT (como ya ocurre en algunos dispositivos móviles [5]) que proporcionan mejor productividad para cargas multiprogramadas.

Sin embargo, la tecnología SMT dificulta grave-

¹Departamento de Informática de Sistemas y Computadores (DISCA), Universitat Politècnica de València, e-mail: clarafo@etsid.upv.es, jofepre@fiv.upv.es, {jsahuqui, spetit, husein}@isca.upv.es

mente el análisis de la prototipabilidad de las prestaciones, lo que supone un reto en el desarrollo de planificadores para sistemas de tiempo real que incluyan procesadores dotados de esta tecnología. Para facilitar y asistir la investigación en esta temática, en este trabajo se presenta un entorno de diseño y evaluación de planificadores para sistemas de tiempo real con procesadores multinúcleo SMT. El entorno propuesto funciona sobre el sistema operativo Linux y permite cambiar tanto el algoritmo de planificación como la asignación de las aplicaciones a los hilos de ejecución sin requerir modificaciones en el núcleo del sistema operativo. El entorno consta de 5 partes: i) algoritmo de selección de las aplicaciones, ii) asignación de aplicaciones a núcleos, iii) lanzamiento a ejecución, iv) aplicación de restricciones de tiempo real, y v) monitorización de las prestaciones.

El resto de este trabajo se organiza como sigue. En el apartado II se describen los conceptos previos. En el apartado III se presenta el entorno de planificación propuesto. En el apartado IV se describe la plataforma experimental donde se ejecuta el entorno. En los apartados V y VI se caracteriza la carga y se valida el entorno, respectivamente. Por último las principales conclusiones se resumen en el apartado VII.

II. CONCEPTOS PREVIOS

A. Sistemas de tiempo real

Los sistemas de tiempo real (STR) se pueden clasificar en STR críticos y STR acríticos. Los STR críticos (*hard real-time systems*) son aquellos donde los deadlines de las tareas han de cumplirse si no se quiere comprometer la integridad del sistema (por ejemplo, el control de la altitud de una aeronave). En cuanto a los STR acríticos, estos se caracterizan por requerir la ejecución de un número mínimo de deadlines para garantizar una determinada calidad de servicio (por ejemplo, mínimo número de frames/s en un streaming de vídeo) [6].

Un sistema de tiempo real ejecuta una carga o un conjunto de tareas $T = \{T_1, \dots, T_n\}$. Cada tarea T_i se caracteriza por tres parámetros principales, $T_i = \{WCET_i, P_i, D_i\}$. El WCET hace referencia al tiempo de ejecución de la tarea en el peor caso y es una característica crítica de las aplicaciones de tiempo real [7]. Este tiempo puede ser estimado mediante mediciones en el sistema real o bien acotado usando técnicas analíticas. En el apartado V se detalla la estimación del $WCET_i$. P_i es el período de la misma tarea, es decir, el intervalo de tiempo que transcurre entre sucesivas activaciones de la tarea. Finalmente, D_i es el deadline y representa el plazo máximo permitido para que la tarea se ejecute sin comprometer la planificabilidad del sistema. Para simplificar el análisis del sistema, típicamente se asume que $D_i = P_i$ [8].

El algoritmo de planificación RMS es un algoritmo de planificación estática que realiza una asignación fija de las prioridades de las tareas a priori. La tarea cuyo período es más pequeño adquiere la máxima pri-

Algorithm 1 Planificador EDF

```

1: while existen contextos disponibles do
2:   Earliest Deadline = ∞
3:   for cada proceso i do
4:     if  $i_{deadline} < Earliest\ Deadline$  then
5:       Earliest Deadline =  $i_{deadline}$ 
6:       proceso EDF = i
7:     end if
8:   end for
9:   Seleccionar proceso EDF
10:  Extraer proceso EDF de la cola
11: end while
12: Lanzar procesos seleccionados

```

oridad y vice-versa, durante toda la ejecución del sistema. Otro algoritmo de planificación ampliamente utilizado en sistemas de tiempo real es el EDF (*Earliest Deadline First*) [9]. Este es más adecuado para sistemas dinámicos donde puede existir mayor variabilidad de la carga real. EDF prioriza la ejecución de los procesos cuyo deadline esté más próximo. Para ello, en cada *quantum* de ejecución, este algoritmo itera sobre sobre la cola de procesos disponibles para ejecución (activos) y los selecciona de acuerdo al Algoritmo 1. Ambos algoritmos han sido implementados en el entorno que se propone en este trabajo. Como ejemplo se muestra en algoritmo 1 la implementación del EDF.

B. SMT

Los procesadores multinúcleo con núcleos SMT soportan la ejecución de varios hilos de aplicaciones en cada núcleo [10]. Para ello, implementan múltiples contextos hardware que mantienen el estado de cada hilo de manera independiente.

Desde el punto de vista del sistema operativo, un mismo núcleo físico se trata como múltiples núcleos lógicos (tantos como hilos soportados en un determinado modo SMT). Para maximizar la utilización del sistema, el planificador selecciona para su ejecución en cada quantum tantas aplicaciones como núcleos lógicos haya disponibles.

Algunos recursos del núcleo físico son privados para cada hilo (por ejemplo, el contador de programa o la tabla de renombrado de registros) mientras que otros recursos se comparten entre los distintos hilos ejecutándose en el mismo núcleo (por ejemplo, la cache de datos de nivel 1). Los distintos hilos en un mismo núcleo físico compiten entre ellos por el acceso a los recursos compartidos, por lo que se producen interferencias que incrementan el tiempo de ejecución de una tarea con respecto a su tiempo de ejecución en solitario. El impacto en las prestaciones de estas interferencias es impredecible, ya que depende de las características de las aplicaciones que se ejecutan en el mismo núcleo SMT (*co-runners*) y de su sensibilidad a la contención en cada uno de los recursos compartidos [11].

Del razonamiento expuesto se deduce que la estimación del WCET se complica en procesadores con

núcleos SMT. Como consecuencia, la planificación de tareas en procesadores con núcleos SMT para sistemas de tiempo real representa actualmente un reto.

III. ENTORNO DE PLANIFICACIÓN PROPUESTO

El entorno de planificación propuesto consta de cinco partes: i) algoritmo de selección de las aplicaciones (política de planificación), ii) algoritmo de asignación de aplicaciones a núcleos (política de ubicación), iii) lanzamiento a ejecución, iv) aplicación de restricciones de tiempo real, y v) monitorización de las prestaciones.

La política de planificación sólo se aplica cuando hay más aplicaciones activas que contextos hardware para decidir que aplicaciones se ejecutan en cada quantum. En nuestro caso, al tratarse de un procesador multinúcleo SMT, deberán seleccionarse n aplicaciones para cada quantum, siendo n el producto del número de núcleos en el procesador por el número de hilos soportados por la tecnología SMT. En este trabajo se han implementado las políticas de planificación EDF y RMS.

La política de ubicación asigna las aplicaciones a los contextos hardware. La asignación de aplicaciones cobra mayor importancia en sistemas multinúcleo SMT que en otro tipo de sistemas. Esto es debido a que la compartición de recursos entre los diversos contextos hardware no es homogénea y por tanto los efectos de la contención en el acceso a recursos compartidos puede diferir enormemente dependiendo de en qué contexto se sitúa cada aplicación. Típicamente, todos los núcleos comparten la memoria principal y la caché del último nivel. Las cachés intermedias pueden ser privadas al núcleo o compartidas entre núcleos. Y los recursos internos del núcleo se comparten entre todas las aplicaciones que se ejecutan en él.

Una vez las aplicaciones seleccionadas han sido asignadas a los contextos hardware correspondientes, estas son lanzadas para su ejecución durante el siguiente quantum.

El planificador monitoriza los contadores de prestaciones hardware del procesador para medir las prestaciones alcanzadas durante el quantum. Así, es posible analizar y comparar el comportamiento y efectividad de diferentes políticas de planificación y ubicación. Al final de cada quantum, se aplican las restricciones de tiempo real, terminando las aplicaciones que hayan cumplido su deadline y activando aquellas que inicien un nuevo periodo.

El entorno de planificación ha sido desarrollado como una aplicación de usuario en el sistema operativo Linux. Para implementar el lanzamiento y las políticas de selección de procesos se utilizan los mecanismos de serialización del sistema operativo mientras que para la ubicación se usan las llamadas al sistema que permiten determinar la afinidad de los procesos a los núcleos lógicos. Finalmente, el acceso a los contadores de prestaciones se realiza mediante la librería *libppm*. Este diseño evita la necesidad de modificar el núcleo del sistema operativo, lo que

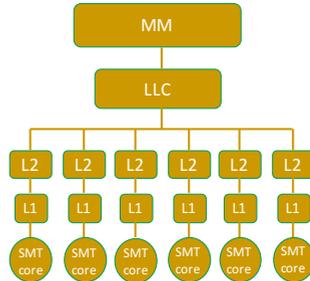


Fig. 1. Estructura del procesador Intel Xeon E5645.

permite prototipar diferentes algoritmos de selección y asignación así como evaluar diversas métricas de prestaciones mucho más fácilmente. Además, este diseño también facilita la detección y depuración de errores en la implementación de las políticas que se pretende estudiar.

IV. PLATAFORMA EXPERIMENTAL

Los experimentos se han realizado en un procesador multinúcleo SMT Intel Xeon E5645. Tal como se muestra en la figura 1, este procesador está compuesto por 6 núcleos que soportan dos hilos de ejecución cada uno. Cada núcleo incluye dos niveles de cachés privadas, una L1 de 32 KB y otra L2 de 256 KB. Una caché de tercer nivel de 12 MB es compartida por todos los núcleos del sistema. El sistema cuenta con 12 GB de DDR3 RAM y funciona a una frecuencia de 2.4 GHz.

El sistema operativo instalado es Linux Fedora Core 16, el cual incluye el núcleo 3.11.4. Se utiliza la librería *libppm* 4.3.0 para configurar y acceder a los contadores de prestaciones y obtener, para cada hilo en ejecución, los ciclos del procesador empleados, las instrucciones ejecutadas y el número de accesos a la caché de L1 y a la memoria principal. El planificador recoge estos valores al final de cada quantum de ejecución para obtener el ancho de banda y el IPC durante dicho quantum.

V. CARACTERIZACIÓN DE LA CARGA

Las suites de benchmarks típicamente utilizadas para evaluar sistemas de tiempo real no han sido actualizadas en los últimos años mientras que las prestaciones de los procesadores han aumentado considerablemente. En consecuencia, la ejecución de la mayoría de los benchmarks de estas suites dura sólo unos pocos quantums. Esto hace inadecuadas para evaluar algoritmos de planificación en el entorno propuesto, cuya granularidad temporal es de un quantum. Por ello, se ha considerado un conjunto amplio de benchmarks, incluyendo algunos de la suite SPEC CPU2006 [12]. Dado que el objetivo es evaluar algoritmos de planificación en entornos de tiempo real, los benchmarks SPEC elegi-

TABLA I
 CARACTERÍSTICAS DE LOS BENCHMARKS EN EJECUCIÓN EN SOLITARIO.

Benchmark	Descripción	Instrucciones (M)	IPC	BW L1 (trans/usec)
bzip2	Utilidad de compresión de datos.	27.303,38	1,74	1794,36
mcf	Implementa algoritmo de redes simplex para planificación del transporte público.	14.895,49	0,95	1125,42
gobmk	Aplicación de inteligencia artificial que juega a Go.	21.454,57	1,37	1118,29
hammer	Implementa modelos de Markov.	21.000,96	1,34	1739,84
sjeng	Aplicación de inteligencia artificial que juega al ajedrez.	25.566,79	1,63	1203,79
h264	Utilidad de codificación de vídeo H.264/AVC.	30.480,19	1,94	2395,59
popray	Aplicación de procesamiento de imágenes.	29.033,59	1,85	1625,06
lame	Utilidad de codificación de audio en formato MP3.	32.998,86	2,12	1240,59
x264	Otra implementación de codificación de vídeo H.264/AVC.	30.781,72	1,96	1284,98
facesim	Simulación del movimiento del rostro humano.	28.300,93	1,83	1791,02
flac	Codificación de audio en formato FLAC.	40.724,59	2,59	2222,94

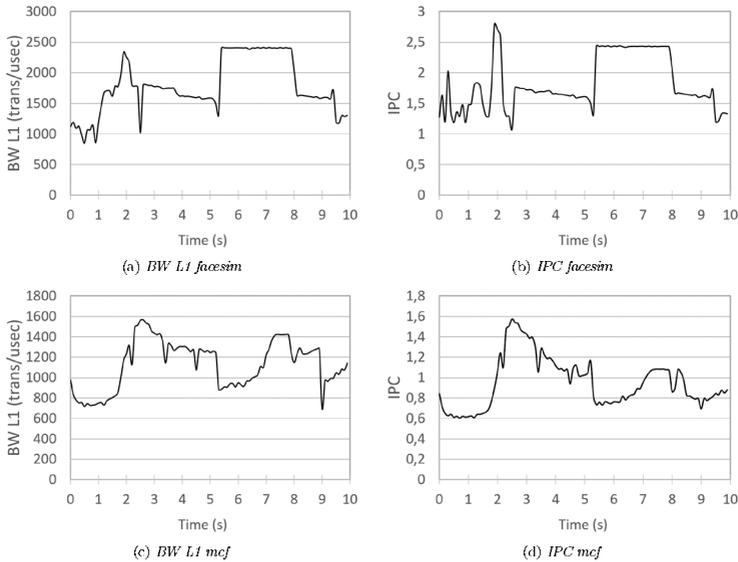


Fig. 2. Valor dinámico del IPC y el ancho de banda de L1 de los benchmarks *mcf* y *facesim*.

dos guardan cierta relación con sistemas multimedia y aplicaciones que podrían requerir de una planificación de tiempo real no crítica. La Tabla I presenta la lista de benchmarks utilizados junto con una breve descripción de su cometido. Antes de evaluar el funcionamiento del entorno de planificación propuesto, hemos realizado un estudio del comportamiento de las aplicaciones en nuestra plataforma, tanto cuando se ejecutan en solitario en el sistema como cuando se ejecutan concurrentemente con otra aplicación en un núcleo SMT. Esta caracterización ha servido para obtener el WCET de cada aplicación.

A. Ejecución en solitario

La caracterización en solitario se ha realizado tanto desde un punto de vista estático como dinámico y se ha centrado en el estudio de los requerimientos de algunos recursos del procesador que impactan en el tiempo de ejecución. Para equilibrar la gran diferencia en la duración temporal entre las cargas estudiadas se ha limitado el número de instrucciones a ejecutar por cada benchmark de tal manera que su duración sea de 10 segundos.

Respecto a la caracterización estática, la tabla I presenta el número de instrucciones, el IPC y el ancho de banda de L1 durante toda la ejecución para

TABLA II
 TIEMPO DE EJECUCIÓN (EN S) DE CADA BENCHMARK (FILA) EN FUNCIÓN DE LA PAREJA (COLUMNA) EN MODO SMT2. LA CELDA COLOREADA DE CADA FILA INDICA EL PEOR CASO.

Benchmark	T. alone	co-runner										
		lzzip2	mcf	gobmk	hmmcr	sjeng	h261	powerx	lame	x261	facesim	flac
lzzip2	9.82	15.83	15.76	14.63	16.10	14.43	17.07	15.61	14.27	14.54	15.62	15.42
mcf	9.81	14.80	15.23	13.07	13.48	14.09	15.24	13.98	12.49	14.15	13.37	14.69
gobmk	9.79	15.00	14.91	15.02	15.00	15.01	15.98	15.59	14.22	15.20	15.19	14.81
hmmcr	9.82	14.55	14.17	13.26	14.75	13.11	16.18	14.11	12.98	13.17	14.35	14.15
sjeng	9.81	15.32	15.18	16.29	16.00	15.619	16.81	16.35	15.22	15.81	15.81	15.39
h261	9.81	16.49	15.65	14.92	16.59	13.82	17.77	16.09	14.63	14.83	16.20	16.10
powerx	9.82	15.72	13.78	15.11	15.95	15.02	17.06	15.92	15.05	15.71	15.62	15.71
lame	9.73	15.37	15.42	15.38	16.09	15.03	16.98	16.36	15.35	15.31	15.69	16.09
x261	9.79	17.10	16.77	17.17	17.26	17.26	18.33	17.94	16.58	16.71	17.28	17.15
facesim	9.69	16.10	16.12	14.97	16.42	14.91	17.15	15.81	15.10	15.15	16.12	16.44
flac	9.80	17.38	15.72	16.29	18.15	15.91	18.79	17.62	16.39	15.90	16.90	17.42

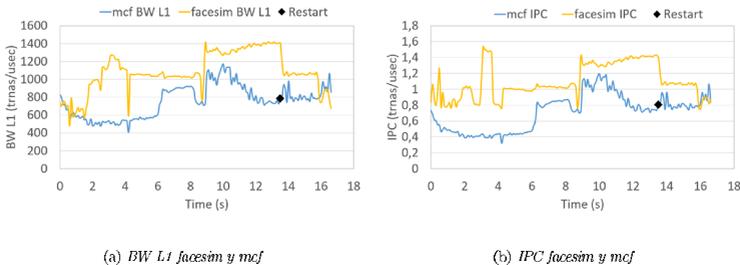


Fig. 3. Valor dinámico del IPC y el ancho de banda de L1 en la ejecución simultánea de los benchmarks *mcf* y *facesim*.

los distintos benchmarks estudiados. Como se puede observar, existe una gran variedad entre las aplicaciones respecto a estos tres parámetros. Además, no se observa correlación alguna entre IPC y consumo de ancho de banda de manera estática.

Una gran cantidad de aplicaciones exhiben varias fases de ejecución distintas con grandes variaciones en el consumo de recursos. Este hecho motiva una caracterización dinámica de la ejecución en solitario ya que las variaciones pueden afectar en gran medida el tiempo de ejecución de los benchmarks cuando se ejecutan concurrentemente en el mismo núcleo. A modo de ejemplo, la figura 2 muestra el valor dinámico que en tiempo de ejecución toman las parámetros estudiados para los benchmarks *facesim* y *mcf*. Se observa correlación entre IPC y ancho de banda tanto en el sentido como en la proporcionalidad de las variaciones a lo largo de la ejecución.

B. Análisis de interferencias y cálculo del WCET

En el procesador SMT se incrementa el tiempo de ejecución de manera impredecible en función de las aplicaciones en ejecución simultánea en el mismo núcleo. Debido a que el procesador sólo soporta dos hilos de ejecución (SMT2), en este apartado se analiza el tiempo de ejecución de cada una de las aplicaciones en ejecución concurrente con cada una de las posibles parejas. Nótese que al estudiar la eje-

cución concurrente de dos benchmarks, el de menor duración es relanzado para cubrir todo el tiempo de ejecución de la pareja de tal manera que ninguno de ellos llegue a ejecutarse en solitario, lo cual permite estudiar el efecto de las interferencias durante toda la ejecución.

En este estudio, se considera el subconjunto de benchmarks caracterizados como una muestra representativa de toda la población. Por tanto, se considera que el WCET de una aplicación (fila) está determinado por el peor tiempo de ejecución entre los tiempos obtenidos al ejecutarse concurrentemente con cada uno de los benchmarks estudiados. Los resultados de la tabla II muestran el tiempo de ejecución en segundos de cada proceso (fila) en función de su pareja (columna). Las celdas sombreadas indican el peor tiempo de ejecución, es decir, el WCET. Cabe destacar que el benchmark que más retrasos provoca a cualquier aplicación es *h261* puesto que el peor tiempo de ejecución de todas las aplicaciones es resultado de su ejecución con dicho benchmark. Los números resaltados en negrita indican el segundo peor tiempo de ejecución, en el que se observa más variación entre los benchmarks.

En la figura 3 puede observarse la evolución dinámica al ejecutarse simultáneamente las dos aplicaciones ya comentadas en el apartado anterior, *facesim* y *mcf*. Comparando estas gráficas con las de

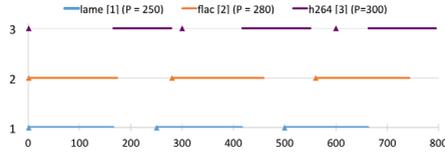


Fig. 4. Ejecución de la mezcla con 3 aplicaciones.

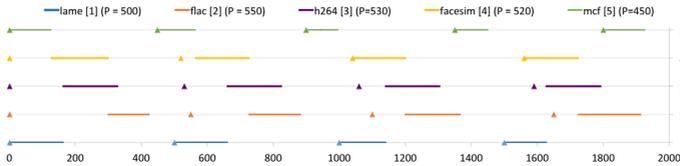


Fig. 5. Ejecución de la mezcla con 5 aplicaciones.

su ejecución en solitario, pueden observarse varias diferencias notables. Ejecutándose en solitario, la duración de ambos benchmarks apenas llegaba a los 10 segundos. En comparación, la ejecución concurrente de ambas aplicaciones se prolonga hasta casi 17 y 14 segundos para *facesim* y *mcf*, respectivamente (el punto rojo en las gráficas representa el instante cuando *mcf* se reinicia atendiendo a lo comentado anteriormente). Estos resultados demuestran la variación en el retraso en el tiempo de ejecución que cada aplicación puede sufrir en función de la parca y de la contención por recursos compartidos como L1.

Otro aspecto importante son las interferencias provocadas por la ejecución simultánea. Por ejemplo, en la gráfica 2(a) el valor del ancho de banda de *facesim* alcanza un máximo de aproximadamente 2500 transacciones/usec. Sin embargo, en ejecución concurrente, su valor máximo sólo alcanza 1400. En el caso del IPC (ver figura 2(b)), en solitario puede superar 2.5 instrucciones/ciclo, mientras que en ejecución concurrente no supera 1.6 instrucciones/ciclo. Asimismo, en el caso del benchmark *mcf* en solitario el ancho de banda y el IPC alcanzan los valores máximos de 1600 y 1.6 respectivamente, mientras que en ejecución concurrente no superan los valores 1200 y 1.2.

VI. VALIDACIÓN DEL ENTORNO

En este apartado los experimentos llevados a cabo se centran en demostrar el correcto funcionamiento del entorno de planificación. Para ello se han modelado dos planificadores del estado del arte (EDF y RMS) y se han ejecutado varias mezclas de tareas de tiempo real. La figura 4 presenta el diagrama temporal de la ejecución de la mezcla más sencilla utilizando el planificador EDF. El eje X representa el tiempo de ejecución medido en quantums. Para esta mezcla, se han seleccionado las aplicaciones *lame*, *flac*, y *h264*, con unos períodos de 250, 280, y 300

quantums respectivamente.

Puesto que se trabaja con dos hilos de ejecución, únicamente pueden ejecutarse dos benchmarks simultáneamente. Así, los dos primeros en ejecutarse son *lame* y *flac* puesto que son los que tienen el deadline más próximo, lo que corresponde al funcionamiento esperado para el algoritmo EDF. Otro aspecto que valida el correcto funcionamiento del entorno es el hecho de que aunque un proceso inicie un nuevo periodo, su ejecución no se inicia efectivamente hasta que se libera alguno de los dos núcleos.

Para estresar el entorno de planificación, también se ha experimentado con mezclas de 5 aplicaciones. En la figura 5 se muestra la evolución temporal de una de las mezclas. Como en el ejemplo de 3 aplicaciones, se aprecia que durante toda la ejecución nunca se ejecutan simultáneamente más de dos procesos. Esto es debido a que únicamente se han utilizado dos contextos.

Finalmente, se ha comparado el desempeño de EDF y RMS utilizando los mismos benchmarks ya comentados, pero con un periodo más restringido para estresar la planificación. Los resultados pueden observarse en la figura 6. En la figura 6(a) se observa el resultado de utilizar el algoritmo de planificación EDF. Pese a haberse ajustado los deadlines de las aplicaciones con respecto a las mezclas anteriores, EDF consigue cumplirlos todos. Nótese que alrededor del quantum 1500 los benchmarks *h264* y *lame* se encuentran igual de próximos a sus respectivos deadlines. Por ello, durante un breve periodo de la ejecución, ambos benchmarks se alternan en cada quantum.

En comparación, cuando se usa RMS (figura 6(b)) el benchmark *h264* pierde tres deadlines (indicados en la figura con un círculo). También cabe destacar que hacia el final de la ejecución el benchmark *h264* se ve interrumpido varias veces por aplicaciones con mayor prioridad. Ambos hechos ponen de manifiesto

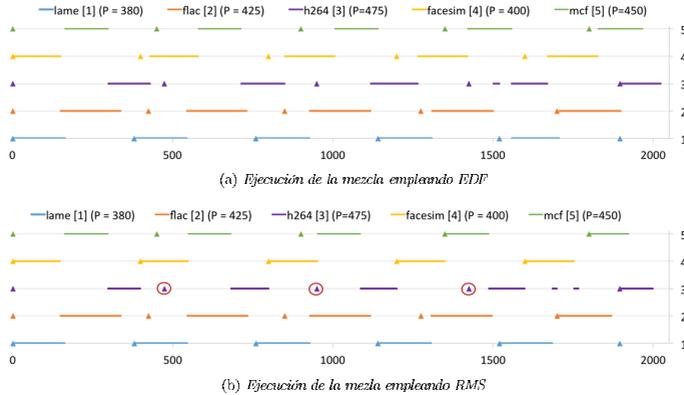


Fig. 6. Comparación de ejecución de la mezcla empleando los dos algoritmos implementados.

que la planificación dinámica EDF resulta más efectiva que la planificación estática por prioridades.

VII. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se ha presentado un nuevo entorno para el desarrollo de algoritmos de planificación de tiempo real en sistemas con procesadores SMT multinúcleo comerciales. El entorno permite el rápido prototipado de diferentes políticas de planificación y ubicación y aprovecha los contadores hardware del procesador para evaluar las prestaciones.

El entorno nos ha permitido evaluar las prestaciones de la ejecución en solitario y concurrente de los benchmarks evaluados y derivar un valor para su WCET en ejecución concurrente. Además, ha sido validado con los algoritmos de planificación EDF y RMS.

Como trabajo futuro, nos proponemos desarrollar nuevos algoritmos de planificación especialmente diseñados teniendo en cuenta las características avanzadas de los procesadores multinúcleo actuales.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad (MINECO) y por los fondos del Plan E mediante subvenciones a los proyectos TIN2015-66972-C5-1-R y TIN2014-62246-EXP así como por la Generalitat Valenciana mediante subvención al proyecto AICO/2016/059.

REFERENCIAS

[1] A. Yamaguchi, Y. Nakamoto, K. Sato, Y. Watanabe, and H. Takada, "Earl-rtstream: Earliest deadline first scheduling of preemptable data streams - issues related to automotive applications," in *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug 2015, pp. 257-267.

[2] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm," *IEEE Transactions on Software Engineering*, vol. 15, no. 10, pp. 1261-1269, 1989.

[3] S. K. Baruah and J. Coossens, "Rate-monotonic scheduling on uniform multiprocessors," in *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, May 2003, pp. 360-366.

[4] M. Asberg, T. Nolte, and S. Kato, "A loadable task execution recorder for hierarchical scheduling in linux," in *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug 2011, vol. 1, pp. 380-387.

[5] Brad Smith, "Arm and intel battle over the mobile chip's future," *Computer*, vol. 41, no. 5, pp. 15-18, May 2008.

[6] J. Ren and L. T. X. Phan, "Mixed-criticality scheduling on multiprocessors using task grouping," in *2015 27th EuroMicro Conference on Real-Time Systems*, July 2015, pp. 25-34.

[7] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE, Dec 2003*, pp. 52-62.

[8] T. Ungerer, F. J. Cazorla, P. Sainat, G. Bernat, Z. Petrow, C. Roehrig, E. Quinones, M. Gordes, M. Paolieri, J. Wolf, H. Casse, S. Uhrig, I. Galiasvili, M. Houston, F. Kluge, S. Metzloff, and J. Mische, "Memsat: Multicore Execution of Hard Real-Time Applications Supporting Analyzability," *Micro, IEEE*, vol. 30, no. 5, pp. 66-75, 2010.

[9] S. Fink and S. Baruah, "Characteristics of edf schedulability on uniform multiprocessors," in *Real-Time Systems, 2003. Proceedings. 15th EuroMicro Conference on*, July 2003, pp. 211-218.

[10] Deun M. Tulsien, Susan J. Eggers, and Henry M. Levy, "Simultaneous multithreading: maximizing on-chip parallelism," *SIGARCH Comput. Arch. News*, vol. 23, no. 2, pp. 392-403, May 1995.

[11] J. Feliu, J. Sahuquillo, S. Petit, and J. Duarte, "Bandwidth-Aware On-Line Scheduling in SMT Multi-cores," in *IEEE Transactions on Computers*, 2015.

[12] "SPEC CPU2006," <http://www.spec.org/cpu2006/Docs/readme1.htm>.

Redes y Comunicaciones

Una nueva metodología para encaminamiento tolerante a fallos en topologías KNS.

Roberto Peñaranda¹, Ernst Gunnar Gran, Tor Skeie², María Engracia Gómez y Pedro López³

Resumen— Los sistemas de computadores exascale contienen miles de nodos. La red de interconexión es un componente clave en estos sistemas. El gran número de componentes en estas redes de interconexión aumenta la probabilidad de que un fallo ocurra. Si un fallo ocurre en una red de interconexión, puede que una gran parte de la red se aisle o quede inalcanzable desde otros puntos de la red. Para que ello no pase, se necesita un mecanismo que tolere fallos de forma eficiente para mantener el sistema interconectado incluso en presencia de fallos. La recientemente propuesta topología híbrida KNS proporciona altas prestaciones y conectividad con un bajo coste en hardware. Este trabajo presenta una metodología de encaminamiento tolerante a fallos para la topología KNS, donde las prestaciones sufren una baja degradación en presencia de fallos y tolera un número más que razonable de fallos sin deshabilitar nodos sanos (no afectados directamente por los fallos). Para tolerar fallos de la red, se utiliza un simple mecanismo: Para algunos pares origen-destino, solo si es necesario, los paquetes son dirigidos hacia el nodo destino a través de uno o más nodos intermedios (sin ser eyectados de la red) lo que permite evitar los fallos. Los resultados de la evaluación muestran que la metodología tolera un gran número de fallos. Además, la metodología ofrece una afeble degradación de las prestaciones. Por ejemplo, las prestaciones sufren una degradación de solo un 1% para una red 2D de 1024 nodos y un 1% de enlaces defectuosos.

Palabras clave— Tolerancia a fallos, Topologías Híbridas, Topología KNS, algoritmo de Encaminamiento.

I. INTRODUCCIÓN

LAS redes de interconexión se utilizan para diferentes propósitos, desde pequeños dispositivos que usan redes en chip para conectar los diferentes componentes hasta grandes supercomputadores que conectan un gran número de nodos. El tamaño de los grandes supercomputadores han ido creciendo año tras año. Las mejores máquinas de la lista top 500 de supercomputadores [1] están formadas por cientos de miles de nodos de procesamiento. Esta cantidad de nodos trabajan conjuntamente para resolver problemas en un periodo de tiempo lo más corto posible.

La cantidad de componentes que se puede encontrar en una red de interconexión para máquinas de altas prestaciones afecta a la probabilidad de tener un fallo en el sistema. Cada componente puede fallar de forma independiente, y por lo tanto, la probabilidad de tener un fallo en el sistema aumenta de forma drástica con el número de elementos que lo componen. Por esa razón, es importante que los sistemas

puedan mantener el funcionamiento aunque la red tenga fallos.

Una posible solución se basa en replicar todos los elementos de la red, y usar dichos elementos como componentes de repuesto. Pero esto aumenta significativamente el coste de la red. Por otro lado, hay otra solución que se centra en modificar el algoritmo de encaminamiento para que los paquetes puedan alcanzar sus destinos usando caminos alternativos que circunvalan los fallos.

En este artículo se presenta un nuevo algoritmo de encaminamiento para las topologías KNS, ya que topologías híbridas propuestas recientemente. Este algoritmo es capaz de tolerar múltiples fallos sin sufrir una gran degradación de prestaciones. El algoritmo se basa en el uso de nodos intermedios [2] que ya se hacía en toros y mallas, donde un nodo intermedio se utiliza para un par origen-destino dado para evitar fallos en el camino a seguir.

Lo que queda del artículo se organiza de la siguiente forma: La Sección II describe de forma breve los diferentes algoritmos de tolerancia a fallos propuestos anteriormente para otras topologías. En la Sección III se describe la topología híbrida KNS, para la que se ha implementado este mecanismo. En la Sección IV se presenta el mecanismo tolerante a fallos propuesto en este artículo. En la Sección V se evalúan diferentes configuraciones del nuevo algoritmo de encaminamiento. Finalmente, en la Sección VI, Se presentan algunas conclusiones.

II. TRABAJO RELACIONADO

Hay dos categorías diferentes de técnicas de tolerancia a fallos que están basadas en la configuración del encaminamiento. La primera categoría reconfigura las tablas de encaminamiento cuando ocurre un fallo. En este caso, se tiene que actualizar las tablas de encaminamiento con una nueva topología después de un fallo [3], [4], [5], [6]. Esta técnica permite a la red tolerar cualquier número de fallos sin requerir ningún recurso extra [7], siempre y cuando la red permanezca conectada, gracias a su flexibilidad. Pero esta flexibilidad puede incidir en las prestaciones debido a la necesidad de usar algoritmos de encaminamiento agnósticos a la topología, ya que la topología de red resultante es irregular. Es decir, no se considera las características específicas de la topología, por lo que suelen ofrecer un balanceado de tráfico inferior

Por otro lado, la segunda categoría cubre los algoritmos de encaminamiento tolerantes a fallos. Se han propuesto un gran número de algoritmos tole-

¹Grupo de Arquitecturas Paralelas, Univ. Politec. Valencia, e-mail: ropeaceb@gap.up.es.

²Simula Research Laboratory

³Grupo de Arquitecturas Paralelas, Univ. Politec. Valencia, e-mail: ropeaceb@gap.up.es.

rantes a fallos para redes de interconexión, y especialmente para redes con topologías directas como toros y mallas. Algunas de estas propuestas suelen requerir recursos extras (canales virtuales), a veces dependiendo del número de fallos a tolerar [8] o del número de dimensiones de la red [9]. Otros algoritmos se basan en deshabilitar regiones con fallos [10], [11], [12], [13], [14] o nodos individuales [15], [16], [17] para encaminar los paquetes alrededor de esas regiones. Sin embargo, para hacer eso, estos algoritmos deshabilitan nodos sanos. En [2], los autores usan la técnica de encaminamiento Valiant [18] para implementar un algoritmo que usa los nodos intermedios para evitar caminos con fallos. Esta metodología requiere algunos canales virtuales pero no deshabilita nodos sanos. Hay otras alternativas que no requieren canales virtuales extra, como [19], [20] o [21]. Los dos primeros son capaces de tolerar solo unos cuantos fallos para mallas con pocas dimensiones, mientras que el tercer método puede ofrecer más tolerancia a fallos, pero necesita más canales virtuales extra en toros. Sin embargo, estos últimos algoritmos fueron específicamente diseñados para mallas y toros, y proporcionan un mal balanceo de tráfico porque redirige mucho tráfico a un solo enlace, haciendo que se sature fácilmente.

El hecho de centrarse en topologías directas es debido a la similitud con las topologías KNS.

III. PRELIMINARES

Las topologías suelen adoptar estructuras regulares para simplificar su implementación y el algoritmo de encaminamiento. Sobre las diferentes taxonomías de las topologías regulares, la más usada divide las topologías en directas e indirectas [22], [23].

Las topologías directas suelen adoptar una estructura ortogonal donde los nodos están organizadas en un espacio n -dimensional, y cada nodo de procesamiento tiene su encaminador asociado. Los nodos están conectados en cada dimensión en modo de anillo o de vector. Las topologías 2D o 3D son relativamente fáciles de realizar porque cada dimensión de la topología se puede implementar como una dimensión física o real. Pero implementar topologías directas con más dimensiones no solo implica su complejidad en el cableado, también en la longitud de sus enlaces cuando se implementan en el espacio tridimensional real. Además, el número de puertos de los encaminadores crece con el número de dimensiones (se requieren dos puertos por dimensiones). Este hecho limita el uso de dimensiones, lo que lleva a que el número de nodos por dimensión aumente, lo que incrementa la latencia de comunicación, afectando negativamente las prestaciones.

La alternativa es usar una topología indirecta. La principal diferencia, comparando con las topologías directas, es que no todos los encaminadores tienen asociado un nodo de procesamiento. La topologías indirectas más comunes son las redes indirectas multi-etapa, donde los conmutadores se organizan en un conjunto de n etapas. Estas topologías proporcionan

mejores prestaciones para un gran número de nodos que las directas. Sin embargo, esto se logra a base de usar más conmutadores y enlaces. Además, sus implementaciones físicas son más complejas debido a que la complejidad del cableado crece con el número de nodos en el sistema. No como en las topologías directas, donde la complejidad aumenta con el número de dimensiones.

Para superar limitaciones de las topologías directas e indirectas se han propuesto topologías híbridas y jerárquicas. En [24], los autores proponen una variación de la topología butterfly usando conmutadores de gran conectividad, resultando en una topología directa. Esta topología se puede ver como un hiper-cubo con concentración, ya que todos los nodos en el mismo vector lineal están conectados directamente. Es decir, hay un enlace desde cada nodo a los otros del mismo vector lineal. Como una extensión de esta topología se propuso la topología Dragonfly en [25], la cual es una topología jerárquica que se basa en agrupar encaminadores en encaminadores virtuales para incrementar la conectividad efectiva de la red. Esta topología usa dos redes diferentes, una dentro del grupo y otra entre los diferentes grupos. Para esta topología se aconseja usar un encaminamiento adaptativo global no mínimo para balancear la carga a traves de los canales globales. Estos canales globales, que enlazan los diferentes grupos, son enlaces largos, por lo que se puede esperar una gran latencia.

Para resolver las limitaciones de las topologías previas se ha propuesto la topología k -ary n -direct s -indirect (KNS) [26], una topología n -dimensional donde los anillos que conectan los nodos en cada vector lineal son reemplazados por una pequeña red indirecta. De esta forma, la latencia de comunicación en cada dimensión no crece linealmente con el número de nodos por dimensión. Por otro lado, el pequeño tamaño de estas topologías indirectas permiten una complejidad de cableado más que razonable a diferencia de las grandes topologías indirectas. Esta combinación resulta en una familia de topologías que proporciona altas prestaciones, con una latencia y una productividad cercanas o mejores que las topologías indirectas, pero con un coste reducido. Esta topología está definida por tres parámetros: el número de dimensiones n , el número de nodos por cada vector lineal k y el número de etapas en las subredes indirectas s .

En la Figura 1 podemos ver un ejemplo de esta topología. Esta tiene 2 dimensiones y 4 nodos por cada vector lineal. Podemos usar diferentes soluciones para conectar los diferentes nodos del mismo vector lineal. Por ejemplo, podemos usar un simple conmutador para conectarlos o diferentes topologías multi-etapa como las topologías k -ary n -tree o RUFT [27] si tenemos un número grande de nodos por vector lineal. El número de nodos de procesamiento viene dado por $N = k^n$. En este artículo nos centramos en las topologías k -ary n -direct 1-indirect, aunque el algoritmo propuesto se puede implementar en cualquier configuración de la topología KNS.

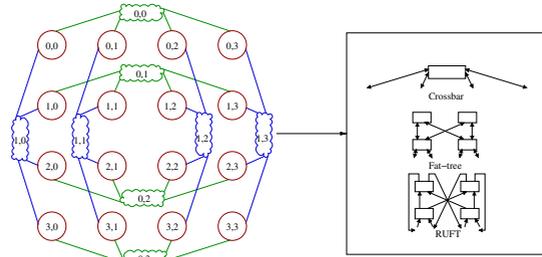


Fig. 1
 UN EJEMPLO DE LA TOPOLOGÍA KNS CON $n = 2$ Y $k = 4$.

IV. METODOLOGÍA DE ENCAMINAMIENTO TOLERANTE A FALLOS

En este artículo, se asume topologías KNS con crossbars como redes indirectas, donde se utiliza encaminamiento mínimo y determinista, Hybrid-DOR [26]. Si no hay ningún fallo, los paquetes son encaminados usando este algoritmo de encaminamiento mínimo, sin necesidad de utilizar canales virtuales extra.

Se considerarán fallos de enlaces, ya que un fallo del conmutador se puede modelar como un conmutador con fallos en todos sus enlaces. Se asume que si un canal falla, entonces el enlace falla en ambas direcciones. En este artículo, no nos centramos en como la información de un fallo se transmite a otros nodos. Se asume, por tanto, un modelo de fallos estático donde la red tiene por adelantado toda la información de los fallos de la red.

Para cada par fuente-destino sin fallos en su camino, los paquetes se encaminan usando Hybrid-DOR con caminos mínimos. Pero si hay algún fallo en el camino, la metodología usa nodos intermedios, como en [2]. El uso de los nodos intermedios fue propuesto en [18] para otras propuestas, como el balanceo de tráfico. Por lo tanto, el algoritmo de encaminamiento evita estos fallos mandando los paquetes a uno o más nodos intermedios y desde el último nodo intermedio al nodo destino. El algoritmo Hybrid-DOR se utilizará en todos los subcaminos. Cabe destacar que los paquetes no se expectan de la red cuando alcanzan nodos intermedios. La idea es evitar los fallos desviando el paquete por un nodo intermedio. El nodo intermedio se selecciona con este propósito para cada par fuente-destino que lo necesite. Si no hay ningún fallo en el camino entre el nodo fuente y el destino, no se utiliza un nodo intermedio.

La dirección de los nodos intermedios se almacena en los paquetes, junto con la dirección del nodo destino. Cuando el nodo intermedio se alcanza, su dirección se elimina del paquete. Este procedimiento se repite por cada nodo intermedio, si se utiliza más de uno. El uso de más de un nodo intermedio permite tolerar más fallos.

Por cada par fuente-destino, se comprueba si existe un camino libre de fallos. Si se requiere que se encamine mediante un nodo intermedio, se calculan y se guardan en una tabla en cada nodo fuente. Hay una entrada en la tabla por cada nodo destino que requiere encaminamiento a través de nodos intermedios.

Como en [2], denotamos al nodo fuente como S y al nodo destino como D . Para cada nodo intermedio usaremos la nomenclatura I_x , donde x representa el índice de el nodo intermedio (I_1 para el primero, I_2 para el segundo...).

Para asegurar que es libre de bloqueo necesitamos al menos tantos canales virtuales como nodos intermedios. Por ejemplo, si se usa hasta dos nodos intermedios, necesitamos al menos tres canales virtuales. Cuando un paquete alcanza un nodo intermedio, el paquete cambia de canal virtual para evitar bloqueos. Por lo tanto, siguiendo el ejemplo de dos nodos intermedios, un canal virtual se utiliza desde S a I_1 , otro desde I_1 a I_2 , y el último desde I_2 a D . De esta forma se evitan los bloqueos porque se divide la red en tres diferentes redes virtuales, y se utiliza un algoritmo de encaminamiento libre de bloqueos dentro de cada red virtual. También se podría usar un algoritmo de encaminamiento adaptativo, usando varios canales adaptativos y uno de escape [28], pero en este caso, se necesitaría un canal de escape por cada red virtual para asegurar que sea libre de bloqueos, donde los canales de escape usan Hybrid-DOR. En este artículo nos hemos centrado en el algoritmo de encaminamiento determinista.

En esta sección no se considera los casos donde un conjunto de fallos desconecta físicamente uno o más nodos de la red, ya que en este caso el paquete sería incapaz de alcanzar su destino. Porque esta metodología no añade nuevos recursos. Por lo tanto, estos casos, donde la red completa no está conectada, no son considerados en esta sección.

A continuación, se presentará como se seleccionan los nodos intermedios. Primero, nos centraremos en el uso de un solo nodo intermedio. Después, se mostrará como extender la metodología para usar múlti-

ples nodos intermedios.

A. Un Nodo Intermedio

En este caso, se usará un solo nodo intermedio cuando uno o más fallos afectan al camino mínimo proporcionado por Hybrid-DOR entre un par de nodos. Este nodo intermedio, I_1 , tiene que satisfacer dos reglas:

1. I_1 es alcanzable desde S .
2. D es alcanzable desde I_1 .

Un nodo Y es alcanzable desde X cuando hay un camino mínimo proporcionado por Hybrid-DOR entre este par de nodos, y no hay fallos en el.

Para topologías directas como el toro o la malla, la elección de este nodo intermedio es muy importante, porque el número de halos puede ser incrementado en gran medida, dependiendo del nodo intermedio seleccionado. En KNS, el número de saltos depende del número de dimensiones que el paquete debe cruzar. La metodología prioriza el uso de esos nodos intermedios que permiten alcanzar el nodo destino sin saltos adicionales, comparado con el camino mínimo entre el nodo fuente y destino.

Por ejemplo, en la Figura 2.(A) hay un ejemplo de una red 4-ary 2-direct 1-indirect donde hay un fallo en el nodo fuente (S) en el enlace de la dimensión X . Por lo tanto, no puede alcanzarse al nodo destino (D) usando Hybrid-DOR. En este caso, los posibles nodos intermedios son todos aquellos nodos de la misma columna (rodeada por la línea discontinua en la figura), pero la mejor opción es el nodo que está en la misma fila que el nodo destino, porque permite alcanzar al nodo destino mediante un camino mínimo.

Lema 1: Dado una red KNS con n dimensiones, usando un nodo intermedio, el algoritmo encaminamiento puede tolerar hasta $n - 1$ fallos.

Demostración: Para una red con n dimensiones, cada nodo tiene n enlaces. Cada enlace permite conectar a los otros nodos de la misma dimensión por una subred indirecta. Sea T_{RS} el conjunto de nodos alcanzables desde S usando Hybrid-DOR, y T_D el conjunto de nodos desde donde D es alcanzable usando Hybrid-DOR. Siempre y cuando $T_{RS} \cap T_D$ no sea un conjunto vacío para cada par fuente-destino, la red es capaz de manejar la combinación de fallos presente. Por lo tanto, la peor combinación de fallos es aquella en la que reduce el número de nodos en T_{RS} o/y T_D .

Por ejemplo, en la Figura 2.(A), mientras el conjunto T_D abarca todos los nodos de la red (excepto el nodo fuente), el fallo reduce T_{RS} a solo 3 nodos. El cardinal $T_{RS} \cap T_D$ es 3 y el fallo se puede tolerar. Sin embargo, podría haber otro fallo que reduzca T_D . El peor escenario ocurre cuando el fallo ocurre en el enlace de la dimensión Y de D (Figura 2.(B)). Entonces, $T_{RS} \cap T_D$ se reduce a un nodo (el único nodo intermedio posible), y el fallo se puede tolerar. Sin embargo, si un nuevo fallo aparece en los enlaces del único nodo intermedio posible (y por tanto, habría 3 fallos), el par fuente-destino S - D estarían desco-

nectados. Esto se puede ver en la Figura 2.(C).

Sin embargo, el peor escenario ocurre cuando el nodo fuente y el destino están en la misma fila. En este caso, con solo dos fallos (los primeros dos fallos del ejemplo, un fallo en el enlace X del fuente y otro en el enlace Y del destino) el nodo destino dejaría de ser alcanzable desde el nodo fuente y $T_{RS} \cap T_D$ estará vacío (ver Figura 3). Por tanto, la red soporta un fallo. Cabe destacar, que hay algunos casos donde 2 fallos desconectan la red físicamente (por ejemplo, un fallo en todos los enlaces de un nodo) siendo imposible alcanzarlos. Obviamente, estos casos no son soportados.

En general, para cualquier red KNS n -dimensional, el mínimo escenario no tolerado ocurre cuando el nodo fuente y destino tienen las mismas coordenadas excepto la de la primera dimensión. Si T_D se reduce debido a los fallos en todos los enlaces del nodo destino excepto el de la primera dimensión (el cual desconectaría al nodo físicamente) y hay un fallo en el enlace de la primera dimensión en el nodo fuente, el nodo fuente no alcanzará al destino.

Por lo tanto, el número mínimo de fallos necesarios para desconectar la red con un solo nodo intermedio es n fallos ($n - 1$ fallos en los enlaces del nodo destino más uno en el nodo fuente) Pro tanto, el algoritmo de encaminamiento es capaz de tolerar $n - 1$ fallos. Esto significa que el algoritmo de encaminamiento es capaz de tolerar todas las combinaciones con menos o igual número de fallos. ■

B. Múltiple Nodos Intermedios

Hay casos donde un solo nodo intermedio no es suficiente para tolerar todos los fallos. En esos casos, el algoritmo de encaminamiento puede usar más de un nodo intermedio. Asumiendo que x es el número de los nodos intermedios requeridos, I_1, I_2, \dots, I_x , estos nodos serán seleccionados siguiendo estas reglas:

1. I_1 es alcanzable desde S .
2. I_{i+1} es alcanzable desde I_i , para $0 < i < x$.
3. D es alcanzable desde I_x .

Por tanto, se puede garantizar que el paquete es capaz de llegar a su destino siguiendo el camino S - I_1 -...- I_i - I_{i+1} -...- I_x - D .

Para reducir el número de saltos, se tratará de usar un conjunto de nodos intermedios que no incrementen el número de saltos comparado con el camino mínimo. En particular, si hay caminos no mínimos usando i nodos intermedios y también hay caminos mínimos que usan j nodos intermedios, aunque i sea menor que j , usaremos los de los caminos mínimos siempre que sea posible.

Lema 2: Dada una red KNS con $n > 2$ dimensiones, usando dos nodos intermedios, el algoritmo de encaminamiento puede tolerar $2 * (n - 1) + k - 3$ fallos. Si $n = 2$, el algoritmo de encaminamiento puede tolerar $2 * k - 1$ fallos.

Demostración: Llamemos a T_{D1} al conjunto

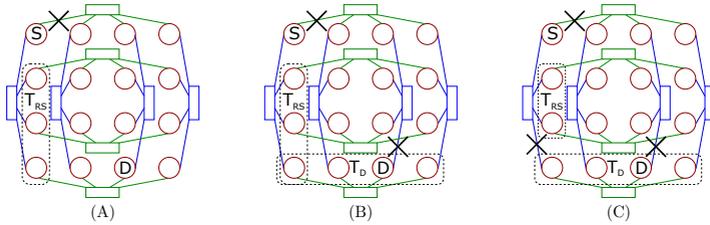


Fig. 2

UN EJEMPLO DE UN CONJUNTO DE FALLOS QUE DESCONECTAN LA RED CON UNA TOPOLOGÍA KNS CON $n = 2 \vee k = 4$.

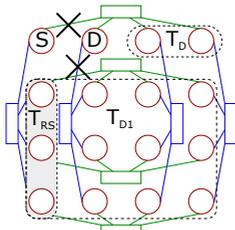


Fig. 3

UN EJEMPLO DE LA TOPOLOGÍA KNS CON $n = 2 \vee k = 4 \vee 2$
 FALLOS.

de nodos desde donde el nodo destino es alcanzable usando Hybrid-DOR y un nodo intermedio, es decir, el conjunto de nodos que pueden alcanzar el nodo destino usando un nodo intermedio desde T_D . En la Figura 3 se puede ver un ejemplo de una red 4-ary 2-direct 1-indirect con 2 fallos. El primero está en el enlace X del nodo fuente, y el segundo en el enlace Y del nodo destino. La figura muestra el conjunto de nodos alcanzables desde el nodo fuente (T_{RS} con un fondo gris), el conjunto de nodos que pueden alcanzar al destino (T_D), y el conjunto de nodos que pueden alcanzar al destino usando un nodo dentro de T_D como un nodo intermedio (T_{D1}). Por tanto, en este caso, tenemos que usar un nodo de $T_{RS} \cap T_{D1}$ como el primer nodo intermedio, y otro de T_D como el segundo nodo intermedio. Siempre y cuando $T_{RS} \cap T_{D1}$ no esté vacío para cualquier par fuente-destino, la red será capaz de tolerar la combinación de fallos. Por lo tanto, las peores combinaciones de fallos serán aquellas que reduzcan el número de nodos en T_{RS} , T_D o/y T_{D1} .

Como en el Lema 1, si hay fallos en todos los enlaces del nodo destino excepto el de la primera dimensión (para evitar desconectarlo físicamente de la red), T_D será reducido a $k - 1$ nodos, los nodos que comparten todas las coordenadas con el nodo destino, excepto la coordenada de la primera dimensión. Por otro lado, si algunos fallos aparecen en todos los en-

laces del nodo fuente, excepto el de la última dimensión, T_{RS} será reducido a $k - 1$ nodos, los nodos que comparten todas las coordenadas con el nodo fuente excepto la coordenada de la última dimensión. De esta forma, los posibles segundos nodos intermedios, T_D , son alcanzados solamente mediante el enlace de la última dimensión. Pero si se asume que esos enlaces también fallan, el nodo destino no será alcanzable. Por lo tanto, solo se necesita $n - 1$ fallos en los enlaces del nodo fuente, $n - 1$ fallos en los enlaces del nodo destino y $k - 1$ fallos en el conjunto de los posibles segundos nodos intermedios. Sin embargo, el peor escenario sigue siendo cuando el nodo fuente comparte todas las coordenadas con el destino, excepto la coordenada de la primera dimensión (ver Figura 3). En este caso, el nodo destino no será alcanzable con solo $k - 2$ enlaces con fallos en el conjunto T_D . Por tanto, el par fuente-destino estará desconectado con $2 * (n - 1) + k - 2$ fallos, ergo, la red podrá tolerar $2 * (n - 1) + k - 3$ fallos. Esto significa que el algoritmo de encañamiento es capaz de tolerar todas las combinaciones de fallos menores o iguales a ese número de fallos.

Cabe destacar que para redes de dos dimensiones, añadir fallos para cada posible segundo nodo intermedio (que componen T_D) desconectan físicamente la fila donde está el destino, como podemos ver en la Figura 3. Estos casos no son soportados por la metodología.

Para este tipo de topologías se necesitan otras combinaciones de fallos. Por ejemplo, manteniendo los fallos del nodo destino ($n - 1 = 1$) y del fuente ($n - 1 = 1$), y colocando fallos en los enlaces X de los nodos de T_{RS} excepto en uno de ellos ($k - 2$, ver Figura 3), para evitar desconectar físicamente la columna, tenemos solo un nodo en $T_{RS} \cap T_{D1}$, el cual será el primer nodo intermedio. Desde este nodo intermedio hay $k - 2$ posibles caminos hacia el destino, esto es, un camino por cada posible segundo nodo intermedio que componen T_D . Si hay fallos en estos caminos, el nodo destino no será alcanzado por el nodo fuente con $1 + 1 + (k - 2) + (k - 2) = 2 * k - 2$ fallos. Por tanto, el encañamiento es capaz de tolerar $2 * k - 1$ fallos en una red KNS de dos dimensiones. ■

C. Extensión a Otras Subredes Indirectas

En este artículo, nos hemos centrado en topologías KNS que usan crossbars como subredes indirectas. Sin embargo, la metodología se puede extender a topologías KNS que usan otras subredes indirectas como fat-trees o RUFT. Para ello, también se debe utilizar una metodología específica para tolerar fallos en cada subred indirecta. Los nodos intermedios se utilizan de forma global y la metodología específica se utiliza de forma local en cada subred.

Mientras las subredes puedan evitar los fallos, los encaminadores directos trabajaran de manera normal. Sin embargo, si un nodo se queda inalcanzable, se modelará como un fallo en ese nodo, en el enlace de la correspondiente dimensión de la subred en particular.

V. EVALUACIÓN EXPERIMENTAL

Para evaluar esta metodología, esta sección se ha dividido en dos partes. Primero, se ha analizado el número de fallos que se pueden tolerar. Un algoritmo de encaminamiento tolerante a fallos es capaz de tolerar n fallos si puede proporcionar un camino válido entre cualquier par de nodos con cualquier combinación de n fallos. Hay situaciones donde los fallos desconectan la red de forma física. Estas situaciones son consideradas como combinaciones donde no hay caminos para todos los pares de nodos.

En segundo lugar, se evalúa las prestaciones de la red usando esta metodología con diferentes números de fallos. Para ello, se ha simulado diferentes configuraciones de red con diferentes números de fallos bajo tráfico uniforme. Para cada número de fallos dado, se ha simulado 50 combinaciones aleatorias de fallos para obtener la productividad y la latencia media. En estos experimentos, las situaciones donde algún nodo está físicamente desconectado no se tienen en cuenta.

Se ha analizado esta metodología para dos configuraciones de red distintas: una topología 32-ary 2-direct 1-indirect y una topología 10-ary 3-direct 1-indirect. Tienen un número similar de nodos (1024 y 1000, respectivamente), por lo que podemos analizar el impacto de tener diferente número de dimensiones

A. Modelo de Simulación

Para ejecutar las simulaciones, se ha usado un simulador por eventos que modela las topologías KNS con enlaces bidireccionales. Este simulador usa virtual cut-through. Cada conmutador tiene un crossbar completamente conectado con colas de 4 paquetes en los puertos de entrada y salida. Se utiliza el mecanismo de control de flujo basado en créditos. El tamaño del paquete es de 16 flits. Se asume un encaminador segmentado con una latencia de 4 ciclos de reloj, y el ancho de banda del enlace y del conmutador de un flit por ciclo de reloj.

B. Análisis de Fallos

El número de posibles combinaciones de fallos se incrementa de forma exponencial con el número de

fallos. Por esa razón, no es posible explorar todas las posibles combinaciones en un tiempo razonable. Por tanto, se ha usado el análisis estadístico como herramienta.

Específicamente, se ha utilizado un subconjunto de combinaciones de fallos, donde los fallos son escogidos de forma aleatoria. Este subconjunto es lo suficientemente grande para obtener resultados con un nivel de confianza del 99% y un error menor al 1%.

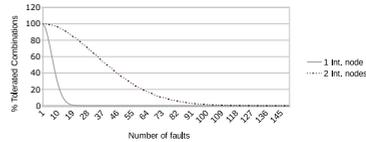


Fig. 4
COMBINACIONES DE FALLOS SOPORTADAS POR LA METODOLOGÍA CUANDO SE UTILIZA UNO O DOS NODOS INTERMEDIOS EN UNA RED 2D CON 1024 NODOS.

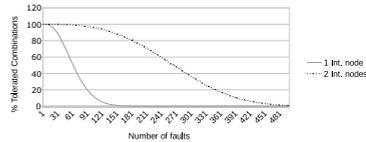


Fig. 5
COMBINACIONES DE FALLOS SOPORTADAS POR LA METODOLOGÍA CUANDO SE UTILIZA UNO O DOS NODOS INTERMEDIOS EN UNA RED 3D CON 1000 NODOS.

Las Figuras 4 y 5 muestran el porcentaje de combinaciones soportadas usando uno o dos nodos intermedios para una red 2D con 1024 nodos y una red 3D con 1000 nodos, respectivamente. Estas figuras muestran los resultados desde 1 fallo hasta 150 para redes 2D y 500 para redes 3D. Primero, como era esperado, el porcentaje de combinaciones de fallos soportadas aumenta considerablemente cuando usamos dos nodos intermedios en vez de uno. Por otro lado, en las redes 2D, el porcentaje de combinaciones de fallos toleradas disminuye considerablemente con un número relativamente pequeño comparado con las redes 3D. Sin embargo, las redes 3D tienen más recursos. Hay 3000 enlaces en las redes 3D mientras que en las 2D hay 2048 enlaces. Además, el hecho de tener más dimensiones da más posibilidades de encaminar los paquetes por diferentes caminos que no comparten recursos, siendo capaz de evitar más fallos. Por lo tanto, podemos ver que con 23 fallos en redes 2D, incluso con dos nodos intermedios, el porcentaje de combinaciones soportadas es menos que el 80%. Sin embargo, en redes 3D, se necesitan más de

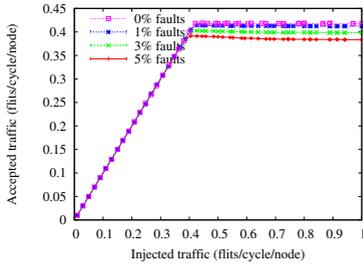


Fig. 6

TRÁFICO ACEPTADO FRENTE A TRÁFICO INYECTADO PARA UNA 32-ARY 2-DIRECT 1-INDIRECT BAJO TRÁFICO UNIFORME.

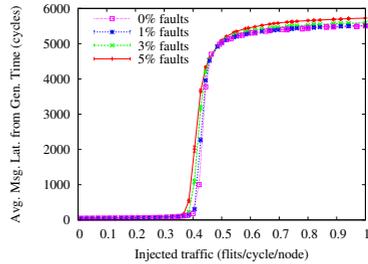


Fig. 7

LATENCIA MEDIA FRENTE A TRÁFICO INYECTADO PARA UNA 32-ARY 2-DIRECT 1-INDIRECT BAJO TRÁFICO UNIFORME.

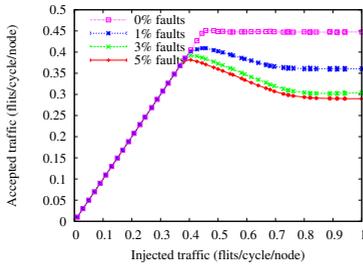


Fig. 8

TRÁFICO ACEPTADO FRENTE A TRÁFICO INYECTADO PARA UNA 10-ARY 3-DIRECT 1-INDIRECT BAJO TRÁFICO UNIFORME.

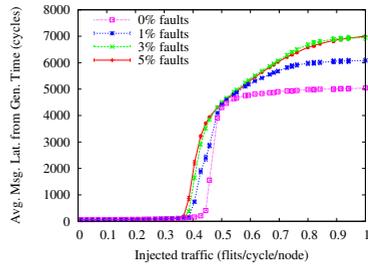


Fig. 9

LATENCIA MEDIA FRENTE A TRÁFICO INYECTADO PARA UNA 10-ARY 3-DIRECT 1-INDIRECT BAJO TRÁFICO UNIFORME.

100 fallos para alcanzar un porcentaje menor que el 80%. Recuerde que las combinaciones de fallos que desconectan la red físicamente están incluidas en este análisis. Por ejemplo, en redes 2D, hay combinaciones no soportadas con solo 2 fallos (o 3 en redes 3D).

C. Análisis de Prestaciones

En esta sección se ha analizado el comportamiento de la metodología de encaminamiento tolerante a fallos en presencia de fallos. Para ello se ha simulado varios escenarios, con un 1%, un 3% y un 5% de enlaces con fallos. Para cada escenario, se han generado 50 conjuntos de fallos aleatorios, todos soportados por la metodología. Se ha usado la metodología usando 2 nodos intermedios, lo que permite evaluar combinaciones con más fallos. Sin embargo, el hecho de usar dos en vez de un nodo intermedio no impacta de forma grave a las prestaciones de la red. Esto es por que la metodología es capaz de evitar el fallo usando solo un nodo intermedio en muchos de los casos, aunque es capaz de usar dos. Por tanto, la

metodología solo usa dos nodos intermedios en unos pocos casos, lo que no impacta a las prestaciones en gran medida.

Para medir las prestaciones, se ha obtenido el tráfico aceptado y la latencia media frente al tráfico inyectado. El tráfico aceptado se ha medido como la cantidad de datos por nodo y por unidad de tiempo que la red puede aceptar (flits/ciclo/nodo). La productividad de la red es el valor pico del tráfico aceptado. La latencia media del mensaje es la media del tiempo comprendido entre la inyección del mensaje dentro de la red en el nodo fuente hasta su eyección en el nodo destino.

Se puede ver los resultados de una red 32-ary 2-direct 1-indirect bajo tráfico uniforme en las Figuras 6 y 7. En este caso, se tiene una degradación de un 1% en productividad con un 1% de enlaces con fallos (21 enlaces) comparado con la misma configuración sin fallos. Para 3% y 5% de enlaces con fallos, la degradación de la productividad aumenta a un 3,8% y 6,5%, respectivamente. La latencia también resulta

afectada, aumentando el valor medio con respecto a la configuración libre de fallos. En particular, en el punto de saturación, la latencia aumenta en un 67% con un 1% de enlaces con fallos.

Cuando utilizamos una red 10-ary 3-direct 1-indirect (Figuras 8 y 9) se puede ver que la degradación con los mismos porcentajes de enlaces con fallos, es mayor. En particular, la productividad de la red se degrada en un 9% con un 1% de enlaces con fallos (30 fallos) comparado con la red libre de fallos, y un 13% y 15% con un 3% y 5% de enlaces con fallos, respectivamente. El incremento en latencia, para un 1% de enlaces con fallos, es 1.8 veces en el punto de saturación. En parte, esto es porque la red tiene más menos el mismo número de nodos, pero más dimensiones, por tanto más enlaces que la configuración de dos dimensiones y, por tanto, un mayor número absoluto de enlaces con fallos para el mismo porcentaje de fallos. Por lo tanto, hay más caminos afectados. Por otro lado, como se muestra en las Figuras 4 y 5, el hecho de tener un número más alto de dimensiones con el mismo número de nodos mejora la probabilidad de evitar una combinación de fallos dada. Por lo tanto, aunque la productividad se degrada a un 3.8% con un 3% de enlaces con fallos en una red 2D frente a una degradación del 13% con el mismo porcentaje de fallos en una red 3D, la probabilidad de que se soporte una combinación con ese número de fallos es del 97% en una red 3D, frente a un 16% en una 2D.

Resumiendo, la metodología propuesta ofrece una mejor tolerancia a fallos en redes con más dimensiones, pero esto no significa unas mejores prestaciones, porque eso depende de la combinación de fallos y los nodos intermedios seleccionados. Con más dimensiones, los nodos intermedios seleccionados pueden aumentar el número de saltos en gran medida, porque la distancia entre dos nodos en una topología KNS aumenta con el número de dimensiones.

VI. CONCLUSIONES

Se ha propuesto un algoritmo de encaminamiento tolerante a fallos para las topologías k -ary n -direct 1-indirect. Este algoritmo se basa en usar nodos intermedios y asume un modelo estático de fallos. Puede tolerar un gran número de fallos sin sufrir una gran degradación en las prestaciones. Este algoritmo no deshabilita nodos sanos, a diferencia de otros algoritmos, y no requiere muchos recursos. El algoritmo solo requiere de un canal virtual extra por cada nodo intermedio que se quiera utilizar. Este algoritmo ha sido evaluado mediante simulación bajo tráfico uniforme, y los resultados muestran que las prestaciones solo sufren una pequeña degradación. Por ejemplo, usando solo dos nodos intermedios (2 canales virtuales extra), los resultados muestran una degradación en prestaciones de un 1% para una red 2D de 1024 nodos, teniendo un 1% de enlaces con fallos (21 fallos). La metodología propuesta puede ser fácilmente extendida en otras configuraciones de la topología de red KNS

AGRADECIMIENTOS

El presente trabajo ha sido financiado por el Ministerio de Economía y Competitividad (MINECO), y los fondos FEDER bajo la ayuda TIN2012-38341-C04-01, y por el Programa de Ayudas de Investigación y Desarrollo (PAID) de la Universitat Politècnica de València y por la financiación de la red FP7 HiPEAC de Excelencia bajo la ayuda 287759.

REFERENCIAS

- [1] "TOP500 Supercomputer Site," <http://www.top500.org>.
- [2] María Engracia Gómez, José Duato, Jose Flich, Pedro López, Antonio Robles, Nils Agne Nordbotten, Olav Lysne, and Tor Skeie, "An Efficient Fault-Tolerant Routing Methodology for Meshes and Tori," *Computer Architecture Letters*, vol. 3, 2004.
- [3] Ruben Casado, Aurelio Bermúdez, Jose Duato, Francisco J Quiles, and José L Sánchez, "A protocol for deadlock-free dynamic reconfiguration in high-speed local area networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, no. 2, pp. 115–132, 2001.
- [4] Timothy Mark Pinkston, Ruoming Pang, and José Duato, "Deadlock-free dynamic reconfiguration schemes for increased network dependability," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 14, no. 8, pp. 780–794, 2003.
- [5] Olav Lysne, Timothy Mark Pinkston, and Jose Duato, "A methodology for developing dynamic network reconfiguration processes," in *Parallel Processing, 2003. Proceedings. 2003 International Conference on*. IEEE, 2003, pp. 77–86.
- [6] Olav Lysne, José Miguel Montañaña, Timothy Mark Pinkston, José Duato, Tor Skeie, and José Flich, "Simple deadlock-free dynamic network reconfiguration," in *High Performance Computing-HPC 2004*, pp. 504–515. Springer, 2005.
- [7] Valentín Puente, José A. Gregorio, Fernando Vallejo, and Ramón Bevidé, "Immunit: A Cheap and Robust Fault-Tolerant Packet Routing Mechanism," in *ISCA*. 2004, pp. 198–211, IEEE Computer Society.
- [8] William J. Dally and Hiromichi Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 4, pp. 466–475, 1993.
- [9] Daniel H. Linder and James C. Harden, "An Adaptive and Fault Tolerant Wormhole Routing Strategy for k -Ary n -Cubes," *IEEE Trans. Computers*, vol. 40, no. 1, pp. 2–12, 1991.
- [10] Rajendra V Boppana and Suresh Chalasani, "Fault-tolerant wormhole routing algorithms for mesh networks," *Computers, IEEE Transactions on*, vol. 44, no. 7, pp. 848–864, 1995.
- [11] Andrew A. Chien and Jae H. Kim, "Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors," in *ISCA*, Allan Gottlieb, Ed. 1992, pp. 268–277, ACM.
- [12] Suresh Chalasani and Rajendra V Boppana, "Communication in multicomputers with nonconvex faults," *Computers, IEEE Transactions on*, vol. 46, no. 5, pp. 616–622, 1997.
- [13] Chun-Lung Chen and Ge-Ming Chiu, "A Fault-Tolerant Routing Scheme for Meshes with Nonconvex Faults," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 5, pp. 467–475, 2001.
- [14] Suresh Chalasani and Rajendra V Boppana, "Fault-tolerant wormhole routing in tori," in *Proceedings of the 8th International Conference on Supercomputing*. ACM, 1994, pp. 146–155.
- [15] Chris M. Cunningham and Dimiter R. Avresky, "Fault-Tolerant Adaptive Routing for Two-Dimensional Meshes," in *HPCA*. 1995, pp. 122–131, IEEE Computer Society.
- [16] Christopher J Glass and Lionel M Ni, "The turn model for adaptive routing," in *ACM SIGARCH Computer Architecture News*. ACM, 1992, vol. 20, pp. 278–287.
- [17] José Duato, "A Theory of Fault-Tolerant Routing in Wormhole Networks," in *ICPADS*, Lionel M. Ni, Ed. 1994, pp. 600–607, IEEE Computer Society.

- [18] Leslie G. Valiant, "A Scheme for Fast Parallel Communication.," *SIAM J. Comput.*, vol. 11, no. 2, pp. 350–361, 1982.
- [19] Christopher J. Glass and Lionel M. Ni, "Fault-Tolerant Wormhole Routing in Meshes without Virtual Channels.," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 6, pp. 620–636, 1996.
- [20] Olav Lysne, Tor Skeie, and Thomas Waadeland, "One-fault tolerance arid beyond in wormhole routed meshes.," *Microprocessors and Microsystems*, vol. 21, no. 7, pp. 471–480, 1998.
- [21] Nils Agne Nordbotten and Tor Skeie, "A routing methodology for dynamic fault tolerance in meshes and tori.," in *High Performance Computing–HPC 2007*, pp. 514–527. Springer, 2007.
- [22] W.J. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.
- [23] J. Duato, S. Yalamanchili, and N. Lionel, *Interconnection Networks: An Engineering Approach*, Morgan Kaufmann Publishers Inc., USA, 2002.
- [24] J. Kim, W.J. Dally, and D. Abts, "Flattened butterfly: a cost-efficient topology for high-radix networks.," in *Proceedings of the 34th annual international symposium on Computer architecture*, New York, NY, USA, 2007, ISCA '07, pp. 126–137, ACM.
- [25] J. Kim, W.J. Dally, S. Scott, and D. Abts, "Technology-Driven, Highly-Scalable Dragonfly Topology.," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, Washington, DC, USA, 2008, ISCA '08, pp. 77–88, IEEE Computer Society.
- [26] Roberto Peñaranda, Crispín Gómez Requena, María Engracia Gómez, Pedro López, and José Duato, "A New Family of Hybrid Topologies for Large-Scale Interconnection Networks.," in *NCA. 2012*, pp. 220–227, IEEE Computer Society.
- [27] C. Gómez, F. Gilabert, M.E. Gómez, P. López, and J. Duato, "RUFF: Simplifying the Fat-Tree Topology.," in *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*, dec. 2008, pp. 153–160.
- [28] J. Duato, "A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks.," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 841–854, 1996.

Pérdida de señal en redes *on-chip* ópticas

Javier Cano¹, Juan-José Crespo¹, Franciso J. Alfaro-Cortés¹ y José L. Sánchez¹

Resumen— Las redes de interconexión en chip ópticas se proponen como una alternativa eficiente en términos de latencia y consumo. En este trabajo se analiza la viabilidad de este tipo de comunicaciones en base a la atenuación de señal sufrida por el paso de ésta a través de los diferentes elementos que componen la red de interconexión. Mediante una herramienta de simulación y con diferentes experimentos se muestra cómo determinar la escalabilidad de la red en función de las pérdidas de señal.

Palabras clave— Redes *on-chip* ópticas (ONOCs). Pérdidas de señal. Escalabilidad. Jornadas Sarteco.

I. INTRODUCCIÓN

EN la actualidad, debido a los requisitos impuestos por las aplicaciones, cada vez se demandan procesadores más potentes y con un menor consumo energético. Para lograr estas propiedades, y salvar las restricciones debidas al consumo, los arquitectos de computadores se han movido desde el incremento de la frecuencia de las unidades mono-procesador, hacia el aumento del número de núcleos por unidad de proceso, con el objetivo de incrementar el rendimiento final. El incremento en el número de unidades o nodos de procesamiento trae consigo la necesidad de contemplar aspectos propios de sistemas multiprocesador; por ejemplo, protocolos de coherencia o sistemas de interconexión que permitan la comunicación eficiente de todos los elementos.

Las redes de interconexión dentro del chip fueron introducidas para conseguir hacer un uso lo más eficiente posible de los transistores, generalmente nodos, que la ley de Moore vaticina. No obstante, estas redes presentan problemas de escalabilidad para un gran número de nodos debido a: el aumento de los tiempos de envío/recepción de los mensajes a través de la red y el alto consumo energético. Adicionalmente, este último factor se ve agravado por la limitada capacidad de disipación de calor presente en los chips sin hacer uso de costosas técnicas de refrigeración y empaquetado.

Para salvar todos estos problemas, las redes de interconexión ópticas se han empezado a estudiar como alternativa. Esto es debido a que están demostrando ser una opción eficiente, en términos de rendimiento (especialmente latencia) y consumo.

La implementación óptica de las redes de interconexión está basada en un mecanismo diferente de propagación de señales, el cual, no solamente carece de los problemas de la señalización eléctrica, sino que además permite una implementación energéticamente eficiente y escalable para un mayor número de nodos. Se ha demostrado que los componentes básicos de escala nanométrica tales como:

moduladores, filtros, guías ópticas, receptores, etc. pueden producirse de forma masiva con los procedimientos de fabricación actuales. Esto posibilita la fabricación de bajo coste necesaria para la comercialización de chips basados en esta tecnología [1].

La idea principal en este tipo de comunicaciones es la modulación de señales ópticas. La manera más simple de realizar dicha tarea es utilizando la codificación On-Off (OOK), que consiste en determinar si se transmite el valor 1 o 0 en función de la presencia o no de luz en un intervalo de tiempo. En el otro extremo del canal, el receptor normalmente utiliza un filtro y un detector para transformar las señales ópticas en señales eléctricas y poder trabajar con ellas y/o almacenarlas.

La figura 1 muestra un esquema a alto nivel de un canal de comunicación óptico, además de los componentes principales que hacen posible el funcionamiento del mismo.

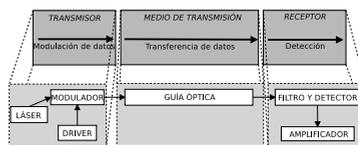


Fig. 1. Esquema general de un emisor/receptor óptico.

Las pérdidas de señales es un aspecto muy importante en los sistemas en general, y en las redes de interconexión en particular. En redes *on-chip* es un área de estudio poco explorada debido a la corta edad de estas redes. Se trata de determinar la existencia de dichas pérdidas y la intensidad de una longitud de onda al final de un recorrido determinado. Con ello, es posible conocer la distancia máxima que una onda es capaz de recorrer y, por tanto, la escalabilidad del sistema.

Por otro lado, y a partir de las investigaciones realizadas, se ha descubierto que es posible integrar tanto tecnologías eléctricas como ópticas dentro del mismo chip. Esto hace posible que se puedan aprovechar las ventajas de ambas tecnologías ayudando, por ejemplo, a una comunicación más eficiente entre los módulos de memoria.

En el caso de las redes en chip basadas en tecnología eléctrica se ha propuesto una gran cantidad de diseños, en la mayoría de los casos modelados y evaluados mediante simulación. Para su realización, se pueden encontrar múltiples herramientas muy completas como: DARSIM [2], Noxim [3], GEM5 [4] y varios proyectos para OMNet++ como HNOCS [5] entre otros muchos. Hasta la fecha, desde nuestro conocimiento, hay disponibles muy pocas

¹Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, e-mail: {juanjose.grespo, javier.cano, fco.alfaro, jose.sgarcia}@uclm.es

herramientas para la simulación de redes en chip ópticas.

Nuestro objetivo, es desarrollar una herramienta para la simulación de redes en chip que hagan uso sólo de tecnología óptica y sea capaz de obtener medidas acerca de pérdidas de señales a través de los diferentes elementos de la red.

El resto del documento está organizado de la siguiente forma: en la sección II se explican los componente básicos y más relevantes en las comunicaciones ópticas. En la sección III se tratan los detalles referentes al modelado de pérdidas de señal considerados en este trabajo. En la sección IV se presentan resultados obtenidos haciendo uso de la herramienta de simulación. En la sección V, por último, se muestran las conclusiones del trabajo.

II. TECNOLOGÍA ÓPTICA

Para una comprensión correcta de las redes dentro del chip ópticas, es necesario entender el funcionamiento de los componentes básicos de dichas redes, así como los factores que influyen en su rendimiento.

Los componentes que describiremos a continuación son: transmisor, fuente de luz, medio de transmisión y receptor. Posteriormente, en la sección III daremos una descripción más detallada en lo referente a pérdidas de señales de todos los componentes.

A. Transmisor

Los transmisores ópticos se basan en la modulación de luz para transmitir la información, directa o indirectamente. Podemos hablar de modulación directa, cuando se actúa sobre la fuente de luz; por ejemplo, encender o apagar un haz. Por otro lado, hablamos de modulación indirecta cuando se realiza de forma externa. Esto implica que la fuente luminica siempre se encuentra encendida y, de alguna forma, algo se interpone en su camino provocando que el receptor no pueda detectar el haz de luz. En este trabajo nos centraremos en la modulación indirecta haciendo uso de anillos resonadores y una fuente externa láser que actuará de fuente.

De forma simplificada, los componentes de un transmisor pueden resumirse en los 3 siguientes:

- **Fuente de luz:** Es necesario un emisor de luz (diodo o LED) para la modulación de las señales ópticas a través del medio de transmisión. En este trabajo consideramos el uso de láseres como fuente de luz. Los láseres comúnmente utilizados en telecomunicaciones son los que se encuentran en la banda C, es decir, aquellos que transmiten longitudes de onda entre 1530 y 1565 nm. Tal y como muestra la figura 2, este tipo de láseres producen un espectro de longitudes de onda uniforme y continuo en lugar de discreto. Esto supone un problema ya que las transmisiones ópticas están basadas en WDM (*Wave-length Division Multiplexing*). Dicha estrategia está basada en el uso de varias longitudes de onda para la transmisión de múltiples bits de

forma simultánea, a través de un único medio de transmisión. En la figura 2 puede verse además, un ejemplo de espectro WDM.

- **Driver:** Se trata de un componente electrónico encargado de activar o desactivar (mediante excitación eléctrica) los anillos ópticos de tal forma que se module o no, una señal en la vía óptica.
- **Modulador:** Consiste en un **anillo activo** con una frecuencia de resonancia modificable mediante excitación eléctrica. Cuando una corriente eléctrica incide sobre el anillo, éste retirará una onda determinada de la fuente de luz para transportarla al otro extremo de la guía óptica.

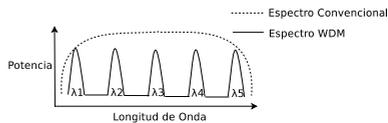


Fig. 2. Espectro convencional y WDM.

B. Medio de transmisión

En las redes de interconexión *on-chip* ópticas, existe gran cantidad de medios para transportar fotones entre un transmisor y un receptor. En este trabajo hemos asumido que el medio de transmisión forma parte de una guía dentro del chip y el material de la misma es silicio.

Los tipos de guía más comúnmente utilizadas son: canal y cresta. Dichas guías proporcionan una transmisión *mono-modo* que es preferible a la transmisión *multi-modo* ya que esta última se ve afectada por la dispersión modal¹.

Las guías ópticas pueden ser cruzadas sin que se produzcan interferencias significativas en las señales que transportan. Esto supone una gran diferencia a las guías eléctricas convencionales, en las que no puede haber cruces sin que esto suponga interferencias. En una guía óptica, una intersección perpendicular entre dos guías supondrá una atenuación mínima (aproximadamente entre 0.05dB [6] y 0.2 dB [7]).

Esta característica del medio óptico permite que las redes *on-chip* ópticas se puedan situar en una única capa. No obstante, el efecto acumulativo de múltiples intersecciones puede hacer que la disposición sobre una única capa sea inviable. Esto puede llevar a colocar las guías en capas diferentes utilizando vías ópticas para permitir la transferencia de señales entre capas.

C. Receptor

El papel de un receptor óptico es transformar las señales ópticas en señales eléctricas. Para ello, deben

¹La dispersión modal ocurre cuando la velocidad de cada modo no es la misma.

ser capaces de detectar una longitud de onda determinada y representarla en forma de señal eléctrica. Un receptor consta de dos componentes básicos:

- **Filtro y fotodetector:** Puede ser considerado el análogo al modulador. Se encuentra formado, al igual que el modulador, por anillos activos que retirarán longitudes de onda determinadas, y las enviarán al detector. El detector es el encargado de realizar la transformación del medio óptico al eléctrico.
- **Amplificador:** Completada la fase de detección, es posible (depende de la sensibilidad del detector) que la señal eléctrica tenga que ser amplificada. En este trabajo no hemos considerado este componente por simplicidad. No obstante, en un sistema real, si el detector utilizado no garantiza suficiente potencia de salida, es necesario hacer uso de un amplificador.

III. MODELADO DE PÉRDIDA DE SEÑAL

Tal y como se ha indicado en la sección II, existen múltiples elementos que se deben tomar en cuenta a la hora de modelar una red en chip óptica. En un proceso de simulación habitual, es necesario modelar el funcionamiento de los componentes de la red (por ejemplo, un switch). En este área, además, es necesario tener en cuenta las pérdidas de señales existentes entre los diferentes elementos. Esto es debido a que los receptores y transmisores tienen unos umbrales de sensibilidad (expresados en dB) entre los cuales son capaces de interpretar una longitud de onda. Por ello, en este trabajo vamos a poner atención en este aspecto.

En concreto, en esta sección describiremos los modelos de pérdida de señal en cada componente incluida en la red.

A. Transmisor

Las fuentes de pérdidas en todos los elementos se localizan en los componentes que los conforman. La principal fuente de pérdidas en el transmisor son los anillos resonadores. En la figura 3 se puede ver el esquema genérico de un transmisor. En concreto, la onda se verá atenuada en una ocasión por su modulación hacia la guía óptica y, $W - 1$ veces (siendo W el número de longitudes de onda utilizando técnicas de WDM) por su paso a través de los anillos resonadores no activos para esa longitud de onda en los moduladores fuera de resonancia de cada nodo intermedio.

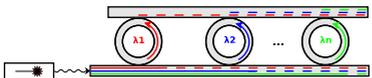


Fig. 3. Esquema genérico de un transmisor

B. Guías ópticas

Principalmente, existen cuatro fuentes de pérdidas por la propagación en las guías *on-chip*:

1. **Absorción intrínseca de luz:** Esta fuente de pérdida ocurre por la propia absorción de las señales por los materiales de las guías. Sin embargo, cuando se trata de longitudes de onda nanofotónica, la pérdida es prácticamente despreciable comparada con otras fuentes de pérdida de señal [8]. Por esto, no hemos tenido en cuenta esta fuente de pérdidas en nuestro estudio.
2. **Acoplamiento radial:** Causada por los cambios bruscos en la geometría de la guía, por ejemplo, giros pronunciados. Tal y como veremos en la sección IV, esta fuente de pérdidas no se dará en este estudio.
3. **Dispersión:** Se trata de una pérdida provocada por la aspereza de las paredes internas de la guía óptica. Esta es, generalmente, la fuente de pérdida de señal principal en las guías ópticas *on-chip*. Debido a esto, esta fuente de pérdida es tomada en cuenta en el estudio.
4. **Two-Photon Absortion (TPA):** Es provocada por una inyección excesiva de energía en una guía de silicio. Se trata, por tanto, de la menos común. Asumimos que la cantidad de energía suministrada es adecuada y, por tanto, no se producirán pérdidas por este factor.

C. Receptor

En nuestro modelo, tal y como se ha comentado en la sección II, el receptor se encuentra formado únicamente por un filtro y un fotodetector.

- **Filtro:** Este componente se encuentra formado por tantos anillos resonadores como longitudes de onda. Por tanto, la señal será atenuada una vez por la modulación hacia el fotodetector y, en el peor de los casos, $W - 1$ veces por atravesar anillos no activos para esa longitud de onda. Cabe destacar, como se puede intuir, que las últimas longitudes de onda que deseamos modular, presenten una potencia menor en función de si son moduladas al principio del *array* de anillos o al final. Nosotros hemos asumido el peor de los casos en este trabajo.
- **Fotodetector:** Compuesto por un detector y un anillo resonador que modula la longitud de onda concreta hacia el detector. En este caso, la fuente de pérdidas será la introducida por la modulación hacia el detector.

En resumen, en el peor de los casos una onda será atenuada una vez en el proceso de detección y $W - 1$ veces por atravesar un anillo fuera de resonancia en los detectores de cada nodo intermedio. A modo ilustrativo, en la figura 4 se puede ver un esquema de un receptor óptico genérico.

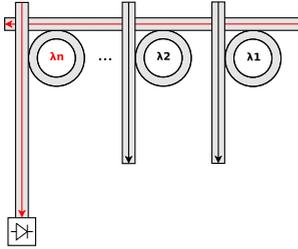


Fig. 4. Esquema genérico de un receptor

IV. EXPERIMENTOS

En esta sección vamos a describir el experimento llevado a cabo en este trabajo para el estudio de la atenuación de señal en transmisiones ópticas integradas en el silicio. En primer lugar vamos a presentar el objetivo del experimento y a continuación describiremos el diseño de la prueba, los parámetros utilizados en los diferentes casos de estudio y por último mostraremos los resultados obtenidos para cada uno.

El objetivo que persigue este experimento es medir la atenuación de la señal óptica entre un par de nodos separados entre sí por nodos intermedios. Con ello se pretende averiguar la viabilidad en la detección de dicha señal en el nodo receptor, haciendo posible la comunicación entre ambos.

La prueba en concreto consiste en una transmisión entre los 2 nodos más alejados de una red con n nodos. Es decir, el emisor y receptor están separados por $n - 2$ nodos intermedios. Dichos nodos intermedios no toman parte en la comunicación, únicamente se tendrán en cuenta como causas de atenuación de la señal.

La herramienta de trabajo que hemos utilizado para la realización de este experimento, es el simulador de redes electro/ópticas *PhotoNoCs* [9]. Utilizando los modelos proporcionados por dicha herramienta, hemos configurado el experimento y la disposición de los componentes.

En la figura 5 se puede ver un diagrama del experimento, con la disposición de los nodos.



Fig. 5. Diagrama del experimento

En el experimento se varía el número de nodos (n), el número de longitudes de onda (W) y algunos parámetros que gobiernan las fuentes de atenuación.

El número de anillos fuera de resonancia que una longitud de onda deberá atravesar en el peor de los casos, sigue el siguiente razonamiento. En el nodo origen, debe atravesar $A_t = W - 1$ anillos del modulador. El cómputo de los nodos intermedios consta

de $A_{int} = 2W(n - 2)$ tras atravesar los anillos de los detectores y moduladores fuera de resonancia. Por último, en el nodo final, debe atravesar los anillos del detector, siendo $A_f = W - 1$ en el peor de los casos.

La cantidad total de anillos fuera de resonancia que deberá atravesar se puede obtener a partir de las tres expresiones anteriores como:

$$\begin{aligned} A_{total} &= A_t + A_{int} + A_f \\ &= (W - 1) + (2W(n - 2)) + (W - 1) \\ &= 2(W(n - 1) - 1) \end{aligned} \quad (1)$$

El número de anillos en resonancia que debe atravesar una longitud de onda determinada w_i , es fijo, constando de un anillo en el nodo transmisor y uno en el receptor.

El número de nodos varía en el rango discreto [2, 40] de dos en dos. En cuanto al número de longitudes de onda, consideramos los siguientes valores: 1, 2, 4, 8, 16, 32, 64, 128. Por último, los parámetros que gobiernan las fuentes de atenuación dan lugar a dos casos de estudio diferentes, uno de ellos utilizando parámetros moderados para las fuentes de pérdidas de señal, y otro utilizando valores más optimistas de dichas pérdidas. En ambos estudios existen parámetros comunes que mostramos en la tabla I.

TABLA I
 PARÁMETROS COMUNES

Parámetro	Valor
Longitud de onda base	1550 nm
Separación entre longitudes de onda	0.8 nm
Frecuencia de operación	2.5 GHz
Distancia entre nodos	1900 μ m
Potencia del láser	10 dBm
Sensibilidad del detector	-20 dB
Ratio de extinción de anillo activo	20 dB
Ratio de extinción de anillo desactivado	25 dB

A. Parámetros moderados

A continuación, presentamos la prueba realizada considerando unos parámetros moderados. En la tabla II se pueden ver dichos parámetros.

TABLA II
 PARÁMETROS MODERADOS

Parámetro	Valor
Pérdida por propagación	$1.5 * 10^{-4}$ dB
Pérdida anillo resonador puerto <i>through</i>	0.005 dB
Pérdida anillo resonador puerto <i>drop</i>	0.5 dB

En la figura 6 mostramos la potencia de la señal (en dB) en el nodo destino tras atravesar los nodos intermedios. La línea discontinua paralela al eje X, representa la sensibilidad de los detectores. Cualquier valor de la potencia de la señal por debajo de este umbral no permite detectar dicha señal correctamente.

Dado que la potencia del láser utilizada es la misma para todos los casos de estudio, y que dicha potencia debe repartirse equitativamente entre todas las longitudes de onda, a medida que aumentamos el número de éstas, disminuye la potencia para cada una. Siendo w_i la potencia de la longitud de onda i , L_p la potencia del láser y W el número de longitudes de onda, la expresión que permite obtener w_i es:

$$w_i = L_p - 10 * \log_{10}(W) \quad (2)$$

Por otro lado, a medida que aumentamos el número de longitudes de onda, es necesario introducir más anillos resonadores tanto en los emisores como en los receptores. Esto se puede ver gráficamente en las figuras 3 y 4 descritas en la sección III.

Debido a lo anterior, en la figura 6 el punto de comienzo de cada recta es diferente y se encuentra más cercano al umbral. Por otro lado, estos factores implican una pendiente más pronunciada cuantas más longitudes de onda utilizemos.

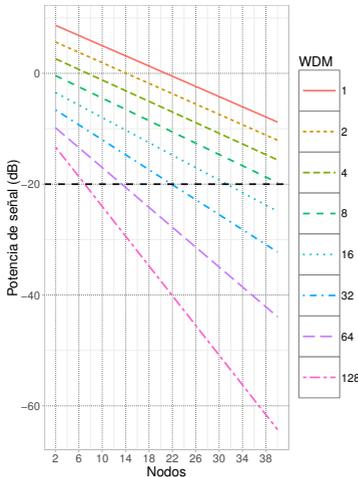


Fig. 6. Potencia de la señal en el detector con aproximación realista de parámetros.

Como se puede ver, es viable la transmisión a través del medio óptico hasta un máximo de 40 nodos si utilizamos 8 o menos longitudes de onda. La utilización de más de 8 longitudes de onda implica

no poder atravesar más de 30 nodos en el mejor de los casos, y únicamente 6 en el caso de 128 longitudes de onda.

En la figura 7 detallamos las pérdidas según la causa de éstas para los casos en los que utilizamos 1, 16, 64 y 128 longitudes de onda. Podemos ver que a medida que aumentamos el número de longitudes de onda, la pérdida por *Paso a través* (de los anillos resonadores no activos) aumenta significativamente.

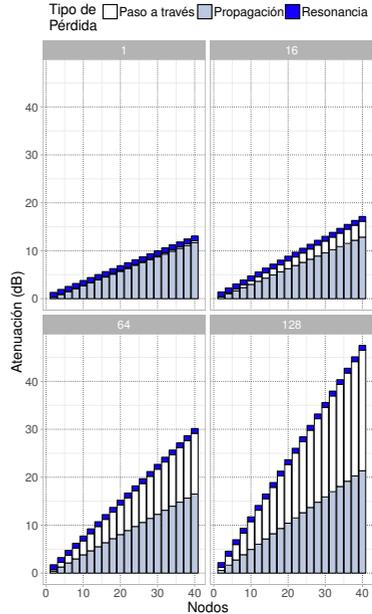


Fig. 7. Pérdidas con aproximación realista de parámetros

Para el caso de una única longitud de onda, esta fuente de pérdida es despreciable ya que moduladores y detectores únicamente constan de un anillo resonador. Por tanto el número total de anillos es $A_{total} = 2n - 4$, lo cual, multiplicado por la pérdida en un anillo resonador puerto *through* (0.005 dB) resulta un valor insignificante.

A medida que aumentamos el número de longitudes de onda, A_{total} aumenta de forma proporcional, lo cual hace que en el caso en que $W = 128$, la pérdida por paso a través de anillos fuera de resonancia sea la mayor causa de atenuación de señal.

En cuanto a la pérdida por propagación, podemos ver un aumento causado debido a una mayor cantidad de guías (cuantos más nodos más guías ópticas necesitaremos para interconectarlos entre ellos). No obstante, esta pérdida no se dispara excesivamente

a medida que aumentamos nodos y longitudes de onda. También consideramos una pequeña pérdida por propagación al paso de las señales a través de componentes ópticos como moduladores, detectores y anillos resonadores, lo cual explica el aumento de esta fuente de pérdida al aumentar W .

Por último, las pérdidas debidas a los anillos en resonancia se mantienen siempre constantes. Esto es debido a que las longitudes de onda sólo serán moduladas dos veces, en el envío y en la recepción tal y como hemos comentado anteriormente.

B. Parámetros optimistas

Por último, presentamos la prueba realizada considerando unos parámetros optimistas. En la tabla III se pueden ver dichos parámetros y sus valores.

TABLA III
 PARÁMETROS OPTIMISTAS

Parámetro	Valor
Pérdida por propagación	$0.5 * 10^{-4}$ dB
Pérdida anillo resonador puerto <i>through</i>	0 dB
Pérdida anillo resonador puerto <i>drop</i>	0.5 dB

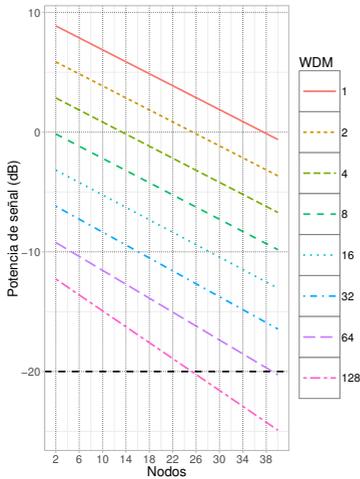


Fig. 8. Potencia de la señal en el receptor con aproximación optimista de parámetros

En la figura 8 se puede ver la potencia de la señal en el nodo destino expresada en dB. Se puede observar que ahora la señal es detectable hasta 64 longitudes de onda en todos los casos. Haciendo uso de 128 longitudes de onda, tan solo podríamos detectar la señal atravesando a lo sumo 26 nodos.

En la figura 9 podemos ver el desglose de los tipos de pérdidas. El hecho más notable es que en esta ocasión, la pérdida por paso a través es inexistente. Esto es debido a que dicho parámetro es nulo. En esta ocasión el mayor factor de pérdida es la propagación, que aumenta linealmente debido a las causas comentadas en la prueba mostrada en la sección IV-A, es decir, el aumento de la cantidad de guías ópticas al aumentar el número de nodos, y una mayor cantidad de moduladores y detectores al aumentar la cantidad de ondas utilizadas.

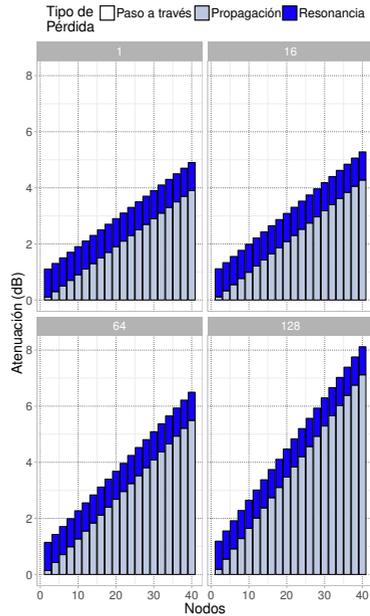


Fig. 9. Pérdidas con aproximación optimista de parámetros

V. CONCLUSIONES

Hemos estudiado la viabilidad de las comunicaciones ópticas entre un par de nodos separados entre ellos por múltiples nodos intermedios.

Dicho estudio se ha llevado a cabo utilizando la herramienta *PhotoNoCs* desarrollada tomando como base el proyecto *PhoenixSim*.

Los resultados muestran las limitaciones de este tipo de comunicaciones en términos de potencia de la señal en el receptor. Dichas limitaciones pueden ser salvadas con diseños optimizados de los componentes así como su distribución en el sistema.

Hemos analizado las principales fuentes de atenuación de señal concluyendo que la atenuación

sufrida por el paso a través de anillos resonadores fuera de resonancia tiene una gran aportación a la pérdida total sufrida. Además, es el factor que experimenta un mayor aumento al utilizar técnicas WDM.

Por último, un aumento en el número de longitudes de onda, a pesar de la mejora en el rendimiento que pueda aportar, sufre las consecuencias de la atenuación debido al aumento de componentes necesarios para su gestión y al reparto de la potencia del láser entre las mismas.

Se propone como trabajo futuro el estudio de otras alternativas en la disposición de los componentes y guías ópticas que minimice la pérdida de señal. También el estudio de otras fuentes de atenuación como por ejemplo la pérdida por cruce de guías, *crossstalk* entre otras.

AGRADECIMIENTOS

Los autores de este estudio agradecen la colaboración del Grupo de Investigación en Comunicaciones Ópticas y Cuánticas (*OQCG*) del Instituto de Telecomunicaciones y Aplicaciones Multimedia (*iTEAM*)

Este trabajo ha sido financiado conjuntamente por el Ministerio de Economía y Competitividad de España (MINECO) con el proyecto TIN2015-66972-C5-2-R, la Junta de Comunidades de Castilla-La Mancha con el proyecto PEII-2014-028-P, y en ambos casos por la Comisión Europea con fondos FEDER.

REFERENCIAS

- [1] Meisam Bahadori, Sebastien Rumley, Dessislava Nikolova, and Keren Bergman, "Comprehensive design space exploration of silicon photonic interconnects," 2016.
- [2] Mieszko Lis, Keun Sup Shim, Myong Hyon Cho, Pengju Ren, Omer Khan, and Srinivas Devadas, "Darsim: a parallel cycle-level noc simulator," in *MoBS 2010-Sixth Annual Workshop on Modeling, Benchmarking and Simulation*, 2010.
- [3] Maurizio Palesi, Davide Patti, and Fabrizio Fazzino, "Noxim-the noc simulator," *Online*, <http://noxim.sourceforge.net>, 2010.
- [4] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al., "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1-7, 2011.
- [5] Yaniv Ben-Itzhak, Eitan Zahavi, Israel Cidon, and Avinoam Kolodny, "Hnoc: modular open-source simulator for heterogeneous nocs," in *Embedded Computer Systems (SAMOS), 2012 International Conference on*. IEEE, 2012, pp. 51-57.
- [6] Ajay Joshi, Christopher Batten, Yong-Jin Kwon, Scott Beamer, Imran Shamim, Krste Asanovic, and Vladimir Stojanovic, "Silicon-photonics cros networks for global on-chip communication," in *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*. IEEE Computer Society, 2009, pp. 124-133.
- [7] Pranav Koka, Michael O McCracken, Herb Schwetman, Chia-Hsin Owen Chen, Xuezhe Zheng, Ron Ho, Kannan Raj, and Ashok V Krishnamoorthy, "A micro-architectural analysis of switched photonic multi-chip interconnects," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 153-164, 2012.
- [8] Michal Lipson, "Guiding, modulating, and emitting light on silicon-challenges and opportunities," *Lightwave Technology, Journal of*, vol. 23, no. 12, pp. 4222-4238, 2005.
- [9] Juan-José Crespo, Francisco J. Alfaro-Cortés, and José L. Sánchez, "PhotoNoCs: un simulador de redes ópticas para CMPs," *XXVI Edición de las Jornadas de Paralelismo (JP2015)*, pp. 509-516, 2015.

Redes en malla BLE para comunicaciones inteligentes y colaborativas

Diego Hortelano¹, Teresa Olivares², M. Carmen Ruiz³ y Celia Garrido-Hidalgo⁴

Resumen— En este artículo se expone la situación actual del estándar de comunicaciones Bluetooth Low Energy (BLE), también conocido como Bluetooth Smart (BS), incluido en Bluetooth a partir de la versión 4.0. En algunas situaciones como la aquí planteada, puede ser necesario el uso de topologías de red que aún no están definidas en la especificación Bluetooth, como la topología en malla.

Por ello, se muestran los resultados de evaluación obtenidos con la nueva topología en malla para el estándar BLE en escenarios de Internet de las Cosas (IoT, Internet of Things), donde los usuarios envían valores de diferentes parámetros de monitorización. Dentro de este marco, hemos descubierto cómo se pueden incrementar los beneficios de este tipo de redes si los dispositivos de los usuarios trabajan de forma colaborativa, ya que es posible ampliar el espacio cubierto por la red sin añadir nuevos nodos BLE, utilizando los dispositivos inteligentes de los usuarios.

Palabras clave— Bluetooth Low Energy, Topología en Malla, Redes Colaborativas, Smart Spaces Management.

I. INTRODUCCIÓN

LA tecnología BLE se ha convertido en una las más importantes dentro del campo de IoT. Las últimas versiones del estándar Bluetooth (desde la especificación 4.0) incluyen BLE, el cual ha reducido drásticamente el consumo de las comunicaciones. Sin embargo, su área de cobertura es limitada, y extenderla es complicado debido al escaso número de posibles topologías especificadas en el estándar: bien mediante el envío de mensajes de tipo *advertisement* con datos a todos los dispositivos dentro de su área de cobertura, sin establecimiento de conexión, en modo *broadcast*; o bien mediante el intercambio de datos entre dos dispositivos conectados en modo maestro-esclavo. Para más información sobre los modos de intercambio de datos en BLE, véase [1] y [2].

Para tratar de solucionar esta situación, la especificación de Bluetooth 4.1 elimina la restricción de que un dispositivo se encuentre en más de un modo o rol simultáneamente, permitiendo, entre otras opciones, crear redes denominadas *scatternet*, formadas por múltiples *piconets* (red en estrella donde un maestro se encuentra conectado a uno o más esclavos), al permitir a un dispositivo encontrarse en modo maestro y esclavo a la vez.

Sin embargo, y a pesar de la facilidad de comunicación directa con el usuario a través de los dispo-

sitivos móviles, aún existen inconvenientes para utilizar BLE en redes inalámbricas de sensores (WSN, Wireless Sensor Network): no existe una topología de red diseñada especialmente para este caso de uso, a diferencia de otras especificaciones como ZigBee [3], basado en el IEEE 802.15.4, que cuenta con una topología de red en malla. Esta topología permite conectar una gran cantidad de dispositivos en una misma red, permitiendo la comunicación entre todos ellos.

Para tratar de asentar definitivamente BLE en el mundo de IoT y hacerlo competitivo en el mercado como ZigBee, algunas empresas han planteado su propio protocolo de red BLE en malla. Además, el grupo de trabajo de Bluetooth ha anunciado recientemente que esta nueva topología de red será incluida en futuras versiones de la especificación [4].

Nuestro interés se centra en la administración de espacios inteligentes (como tiendas, casa, oficinas, escuelas, etc.) donde una cobertura perfecta sin zonas muertas es una prioridad, con el fin de recibir todos los paquetes y actuar en consecuencia. Por ello se han comprobado estas condiciones variando el número de nodos que conforman la red en malla, y se ha propuesto una nueva forma de obtener un área de cobertura mayor haciendo uso de los dispositivos de los usuarios sin incrementar el número de nodos estáticos en la malla.

Este artículo se estructura como sigue: la Sección 2 expone los resultados previos obtenidos con otras topologías de red que nos han motivado a adentrarnos en la red en malla; la Sección 3 repasa el estado actual de esta topología, tanto en el plano académico como en el empresarial; la Sección 4 detalla el funcionamiento de la topología BLE utilizada. La Sección 5 explica y muestra los experimentos realizados utilizando esta topología, mientras que la Sección 6 incluye nuestra propuesta de mejora y una comparativa con el caso anterior. Por último, la Sección 7 muestra las conclusiones de este artículo, así como el trabajo futuro que tenemos por delante.

II. TRABAJO PREVIO Y MOTIVACIÓN

Con la aparición de BLE, nuestro primer objetivo fue conocerlo en profundidad y comprobar si realmente esta tecnología era apropiada para utilizarse en IoT, permitiéndonos crear una red de sensores y actuadores inalámbrica (WSAN), enviando y recibiendo datos y comandos. Para ello, realizamos una serie de experimentos, algunos de los cuales se describen, de manera resumida, en esta sección.

En estos experimentos, se posicionó una serie de dispositivos Waspote de Libelium [5], equipados

¹Instituto de Investigación en Informática de Albacete, UCLM, Albacete, España, email: Diego.Hortelano@uclm.es.

²Escuela Superior de Ingeniería Informática, UCLM, Albacete, España, email: Teresa.Olivares@uclm.es.

³Escuela Superior de Ingeniería Informática, UCLM, Albacete, España, email: MCarmen.Ruiz@uclm.es.

⁴Instituto de Investigación en Informática de Albacete, UCLM, Albacete, España, email: Celia.Garrido@uclm.es.

con su módulo BLE, a lo largo de nuestro laboratorio, en el Instituto de Investigación en Informática de Albacete [6] (I3A), enviando mensajes de tipo *advertisement* [1]. Tras esto, se diseñó un itinerario de prueba, que nos permitió comprobar el comportamiento de las comunicaciones cuando el usuario se encuentra en movimiento dentro de una red con múltiples emisores BLE. La Figura 1 muestra la colocación de los dispositivos (indicados con una cruz y el número de dispositivo, desde D1 hasta D10) y el itinerario planteado (señalados con un punto y su número correspondiente, desde P1 hasta P16). Este usuario recibe los paquetes BLE en su dispositivo smartphone, el cual los envía a su vez a un servidor a través de Wi-Fi, para su procesamiento y almacenamiento.

Para comprobar si existen variaciones en los resultados obtenidos en función de la configuración de los dispositivos, se aplicaron diferentes configuraciones, las cuales se detallan a continuación, junto a los resultados obtenidos para cada uno de los casos. A la hora de evaluar los resultados se ha tenido en cuenta si los paquetes recibidos en un momento dado con más potencia se correspondían con el dispositivo más cercano, y, por otro lado, si el área de cobertura de cada dispositivo era la esperada.

A. Configuración 1

Para la primera prueba, los dispositivos BLE en estado *advertising* se han configurado con las siguientes características:

- Intervalo de *advertising* de 1 segundo.
- Potencia de transmisión (o TX Power) mínima (-23 dBm).

Los resultados obtenidos se muestran en la Figura 2. En dicha figura aparecen en un color más claro aquellos puntos del itinerario donde los paquetes recibidos con mayor potencia no pertenecen al dispositivo emisor más cercano, o bien no se ha recibido ningún paquete de dicho dispositivo. Únicamente en un 68.75% de los puntos de prueba (11 de 16) el resultado obtenido es satisfactorio.

B. Configuración 2

En este caso, los dispositivos BLE se han configurado de la siguiente manera:

- Intervalo de *advertising* de 1 segundo.
- Potencia de transmisión media (-10 dBm).

Los resultados obtenidos para esta configuración se encuentran reflejados en la Figura 3. En ella aparecen, de nuevo, en color más claro aquellos puntos donde los paquetes recibidos con mayor potencia no se corresponden con los enviados por el dispositivo emisor más cercano, o bien no se ha recibido ningún paquete del mismo. De nuevo, el porcentaje de los puntos de prueba en los que el resultado es correcto es 68.75% (11 de 16).

C. Configuración 3

Para esta última prueba, los dispositivos cuentan con la siguiente configuración:

- Intervalo de *advertising* de 0.5 segundos.
- Mínima potencia de transmisión (-23 dBm).

Los resultados obtenidos con esta configuración se muestran en la Figura 4. De nuevo, los puntos de color claro nos indican que los paquetes recibidos con mayor potencia en esos puntos no se corresponden con el dispositivo emisor más cercano, o bien no se ha recibido ningún paquete de este dispositivo. De nuevo, los resultados obtenidos se repiten aquí: solamente en un 68.75% de los puntos de prueba (11 de 16) se han recibido los datos correctos de los dispositivos más cercanos.

Como conclusiones de estos experimentos previos podemos destacar las siguientes:

- Dificultad para obtener el área de cobertura de un dispositivo BLE. En el alcance de un dispositivo influyen múltiples factores, como la presencia o ausencia de los diferentes materiales de construcción. Estos factores varían de un edificio a otro, y, por norma general, no pueden modificarse.
- Para aplicaciones en las que es necesario tener una cobertura total, sin la existencia de las denominadas zonas muertas (sin cobertura), es necesario el desarrollo de topologías de red diferentes, que eliminen las restricciones descritas.

En este contexto aparecen las redes en malla, topología con la que cuentan otros estándares de comunicación, y que es altamente recomendable para este tipo de aplicaciones. Por ello, nos centraremos ahora en ellas, describiendo su funcionamiento y mejorando su funcionamiento con nuestra propuesta.

III. ESTADO DEL ARTE

A pesar de que BLE es una tecnología muy reciente, y de que aún no cuenta con una especificación para una topología de red en malla en su estándar, ya podemos encontrar diferentes proyectos en este campo:

- BLEMesh [7] es una propuesta de topología de red en malla para BLE que aprovecha la capacidad de las transmisiones broadcast de las redes inalámbricas. Para reducir el número de paquetes enviados por la red, BLEMesh utiliza un esquema de enrutamiento oportunista.
- El grupo de trabajo Bluetooth ha anunciado oficialmente la formación del Bluetooth Smart Mesh Working Group, que será el encargado de crear la arquitectura para la estandarización de la topología de red en malla de BLE [4]. Se espera que la nueva especificación vea la luz a finales del año 2016 o a principios del 2017, pero en estos momentos no hay más información sobre cómo se especificará esta topología en el estándar.

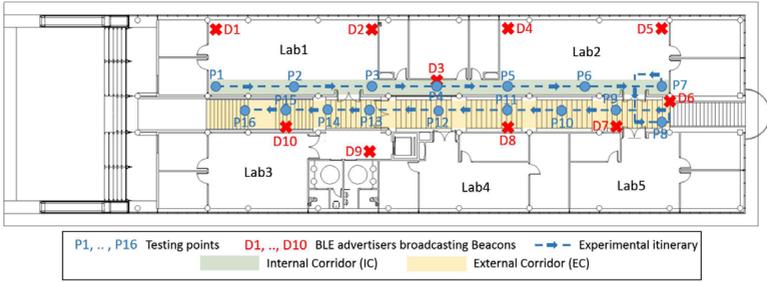


Fig. 1. Colocación de los dispositivos emisores e itinerario planteado.

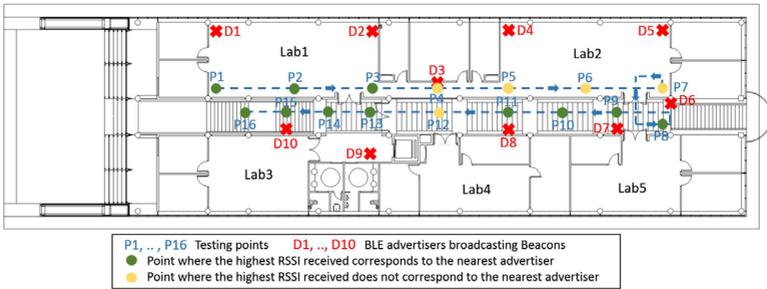


Fig. 2. Resultados obtenidos en cada punto del itinerario dentro del área de la red de emisores BLE con la configuración 1.

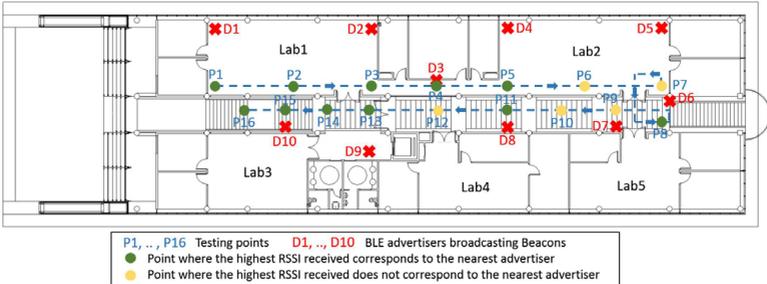


Fig. 3. Resultados obtenidos en cada punto del itinerario dentro del área de la red de emisores BLE con la configuración 2.

Además, en el mundo empresarial también existe interés para dotar de esta nueva funcionalidad a la tecnología BLE, existiendo proyectos relacionados con esta topología de red aplicada a BLE. Entre los más importantes podemos destacar los proyectos de Nordic y CSR:

- El proyecto de Nordic [8] fue creado en colaboración con la NTNU (Norwegian University of Sci-

ence and Technology) como parte de una Tesis y no forma parte del SDK oficial de Nordic Semiconductor. Se trata de un proyecto abierto, que utiliza la transmisión de datos broadcast mediante mensajes de tipo advertisement. Los dispositivos utilizados pertenecen a la serie Nordic nRF51 [9]. Estos dispositivos hacen uso de BLE en su versión 4.0, lo que provoca la pérdida de

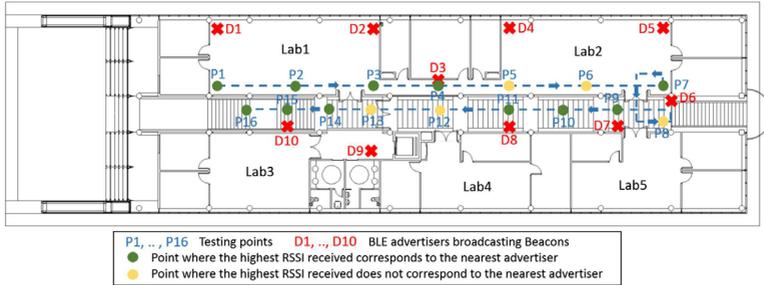


Fig. 4. Resultados obtenidos en cada punto del itinerario dentro del área de la red de emisores BLE con la configuración 3.

- paquetes en la malla, al impedir recibir mensajes cuando se están retransmitiendo los ya recibidos.
- Otra empresa que ha apostado por el desarrollo de una topología de red en malla para BLE es CSR [10]. CSR ha desarrollado dispositivos BLE con la versión 4.1, pertenecientes a la familia Bluetooth Smart CSR101x [11], así como un protocolo propietario construido sobre BLE, denominado CSRmesh. Este protocolo permite crear una topología en malla mediante el envío y recepción de mensajes de tipo *advertisement*, al igual que otros proyectos, pero la principal diferencia radica en el uso de dispositivos con la versión 4.1 de Bluetooth, que permite enviar y recibir paquetes de forma simultánea, disminuyendo en gran medida la pérdida de paquetes.

Esta nueva posibilidad dentro de BLE se adapta perfectamente a nuestro propósito de administrar espacios inteligentes, donde dispositivos como smartphones o smartbands, por ejemplo, pueden enviar datos de manera continua (pulso, temperatura, etc.) a un servidor.

IV. FUNCIONAMIENTO DE LA RED EN MALLA BLE

Para crear nuestra malla BLE hemos seleccionado los dispositivos de CSR. La razón principal de esta decisión es la versión Bluetooth de estos dispositivos: son los primeros en utilizar BLE 4.1 para este fin.

BLE cuenta en su capa de enlace con una serie de estados: *standby*, *advertising*, *scanning*, *initiating* y *connection*, los cuales se muestran en la Figura 5 [2]. Estos estados nos permiten crear dos tipos de redes BLE en la especificación 4.0: red en estrella, con conexión maestro-esclavo, ver Figura 6A; y red broadcast, mediante el envío de paquetes de tipo *advertisement*, ver Figura 6B. Para conocer más sobre los modos de funcionamiento de BLE puede consultarse [1] y [2].

A pesar de que la versión de Bluetooth 4.1 nos permite combinar los diferentes estados para utilizarlos simultáneamente, no proporciona una topología de red en malla propiamente dicha (ésta se espera que

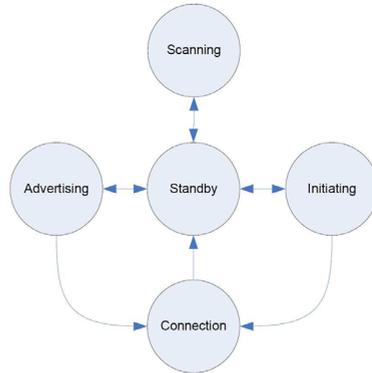


Fig. 5. Máquina de estados de la capa de enlace de Bluetooth.

aparezca en la versión 4.3). Por ello, CSR propone la siguiente configuración de malla:

- Los dispositivos de CSR envían paquetes de datos de tipo *advertisement*. Estos mensajes están codificados y tienen un formato que no sigue completamente el estándar.
- Utilizando la posibilidad que nos proporciona la versión 4.1, combinan el estado *advertising* anterior con el estado *scanning*. De esta manera pueden recibir paquetes de otros dispositivos en cualquier momento. Si los paquetes recibidos cumplen con las características de formato, son reenviados de nuevo en *broadcast*, aumentando de esta manera el alcance y la cobertura del dispositivo emisor.
- Además de los dos estados descritos anteriormente (*advertising* enviando los paquetes que ha recibido en el estado *scanning*), estos dispositivos funcionan en un tercer estado simultáneamente. Este nuevo estado es también de *adver-*

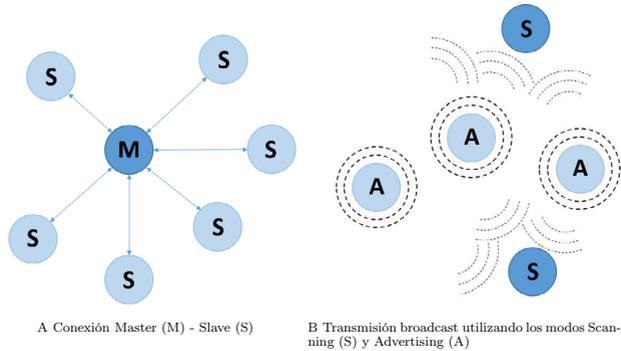


Fig. 6. Modos de transmisión de datos proporcionados por BLE en su versión 4.0

tising, pero en lugar de transmitir datos, realiza su otra función: anunciarse para permitir a otros dispositivos en estado *initiating* establecer una conexión maestro-esclavo. Al ser un protocolo parcialmente cerrado, la manera que proporciona CSR para introducir nuevos dispositivos en la malla es utilizar una conexión directa con uno de sus nodos, el cual funcionará como puente entre el nuevo dispositivo y el resto de la malla, enviando sus datos con el formato adecuado.

De esta manera, la red en malla creada tiene un protocolo de enrutamiento mediante inundación de la red. Esto provoca que haya un alto número de paquetes en la red en un momento dado. Para paliar ligeramente esta situación, se utilizan las siguientes medidas:

- Tiempo de Vida (TTL, Time To Live): implementado en un byte del paquete que indica el número máximo de reenvíos o saltos posibles. Cada vez que un dispositivo recibe y reenvía un paquete, este número es decrementado, desechando el paquete en caso de ser 0.
- Identificador del paquete: cada uno de los paquetes enviados por un dispositivo cuenta con un identificador, que será el mismo durante el tiempo que el paquete siga en la red. Este identificador permite a los dispositivos descartar paquetes que ya haya reenviado previamente, comprobando únicamente este campo.

En nuestro caso, utilizamos una red de dispositivos CSR1010 [11], el cual se muestra en la Figura 7. Estos dispositivos cuentan con una antena omnidireccional, que les permite transmitir datos en cualquier dirección, a una distancia de entre 20 y 30 metros, en función de los obstáculos existentes. Además, cuentan con una versión modificada del protocolo CSRMesh de CSR. Esta versión nos proporciona nueva funcionalidad:

- Conocer el nodo emisor y destino de cada paquete.
- Tener disponibles los últimos datos enviados desde cada uno de los dispositivos. Estos datos se almacenan en características BLE, de manera que están disponibles para su lectura en el momento en que se produce la conexión con alguno de los nodos.
- Aumentar el tamaño del campo disponible para datos en cada paquete.
- Introducir nuevos dispositivos en la malla, ya sea para enviar únicamente sus propios datos o también para retransmitir los paquetes recibidos, sin necesidad de utilizar un dispositivo como puente.



Fig. 7. Dispositivos CSR1010 utilizados.

V. EXPERIMENTOS

El objetivo final de nuestro trabajo es diseñar y construir una red BLE en malla con dispositivos reales para asegurar la cobertura total del espacio a monitorizar y de bajo costo, para que pueda llegar a ser una realidad. Esta red permitirá a los usuarios enviar los datos y comandos a un servidor BLE. Además, los usuarios podrán recibir datos del servidor, así como de los diferentes dispositivos equipados con sensores, en su smartphone, tablet o smart-

watch. Sin embargo, es posible implementar esta red de múltiples formas, por lo que nos centraremos en conocer si la red seleccionada es la que mejor se adapta a nuestro proyecto, permitiéndonos su implementación.

En primer lugar, se ha evaluado cómo funciona una topología BLE en malla, para verificar que realmente todos los paquetes enviados se reciben en su destino y que esta topología de red no implica la creación de nuevos problemas en las comunicaciones BLE. Con este propósito, se ha desplegado una red de 10 dispositivos BLE en la primera planta (680m²) de nuestro Instituto de Investigación [6]. En estos primeros despliegues, el servidor BLE es el encargado de recibir los paquetes de todos los usuarios. La Figura 8 muestra la situación de los 10 dispositivos de la red en malla.

Para verificar el correcto funcionamiento de la red en malla desplegada se ha realizado un estudio con 3 usuarios. Estos usuarios se han movido de manera aleatoria entre los laboratorios, equipados con un dispositivo BLE CSR1010 [11] que envía paquetes a la malla cada 0,5 segundos. Estos paquetes han sido retransmitidos por los dispositivos de la malla, hasta llegar al servidor BLE (conectado a la red en malla utilizando un dispositivo BLE 4.1 como puente), donde son procesados y almacenados. Los resultados de esta prueba se recogen en la Tabla I.

TABLA I
 PAQUETES RECIBIDOS DE 3 TRANSMISORES BLE EN MOVIMIENTO A TRAVÉS DE UNA RED DE 10 DISPOSITIVOS EN MALLA.

Usuario	Paquetes enviados	Paquetes recibidos	Éxito Tx
1	338	338	100%
2	346	346	100%
3	349	349	100%
Total	1033	1033	100%

Tras los buenos resultados obtenidos, el siguiente paso es reducir el número de dispositivos pero garantizando la cobertura total, reduciendo de esta manera el coste final de la red. En la Figura 9 se muestran los 4 dispositivos de la red en malla que se han mantenido: D2, D5, D8 y D10.

En este nuevo escenario se ha reproducido el experimento anterior, con 3 usuarios moviéndose aleatoriamente por los laboratorios mientras sus dispositivos envían paquetes a través de la malla, comprobando si este número de dispositivos es suficiente para cubrir el área deseada. La Tabla II muestra los resultados obtenidos.

Finalmente, para evaluar cómo la ausencia de un dispositivo afecta a la cobertura de nuestra red en malla, se ha eliminado un dispositivo (D10) de la misma. La red en malla obtenida, formada únicamente por 3 dispositivos BLE, se muestra en la Figura 10.

TABLA II
 PAQUETES RECIBIDOS DE 3 TRANSMISORES BLE EN MOVIMIENTO A TRAVÉS DE UNA RED DE 4 DISPOSITIVOS EN MALLA.

Usuario	Paquetes enviados	Paquetes recibidos	Éxito Tx
1	366	363	99.18%
2	480	475	98.96%
3	468	466	99.57%
Total	1314	1304	99.24%

La Tabla III muestra los resultados obtenidos al repetir el experimento anterior en la nueva red en malla. Aquí se aprecia una reducción importante (alrededor del 22,5%) del número de paquetes recibidos, debido a la reducción de la cobertura. Al eliminar el nodo D10, se producen zonas de sombra en partes del laboratorio denominado Lab3, por lo que no hay señal BLE en toda la zona que deseamos cubrir. Estos resultados no son suficientes para garantizar el éxito en las transmisiones, por lo que a continuación se detallará y evaluará nuestra propuesta con el fin de mejorarlos.

TABLA III
 PAQUETES RECIBIDOS DE 3 TRANSMISORES BLE EN MOVIMIENTO A TRAVÉS DE UNA RED DE 3 DISPOSITIVOS EN MALLA.

Usuario	Paquetes enviados	Paquetes recibidos	Éxito Tx
1	436	331	75.92%
2	440	337	76.59%
3	434	335	77.19%
Total	1310	1003	76.76%

En la Figura 11 se recoge, a modo de resumen de esta sección, el porcentaje de paquetes recibidos en cada uno de los 3 escenarios descritos en ella:

1. Red BLE en malla formada por 10 dispositivos.
2. Red BLE en malla formada por 4 dispositivos.
3. Red BLE en malla formada por 3 dispositivos.

Como muestra la Figura 11, con la disposición utilizada, el número mínimo de dispositivos necesario para cubrir todo el área es 4. Por ello, a partir de ahora trabajaremos con el escenario con 3 dispositivos, en el que trataremos de mejorar los resultados obtenidos sin incrementar el número de nodos.

VI. NUESTRA PROPUESTA

La solución más simple al problema planteado anteriormente podría ser incrementar el número de dispositivos estáticos de la malla (recordemos que en el caso en que la malla estaba formada por 10 dispositivos era posible recibir el 100% de los paquetes enviados). Sin embargo, hemos optado por otra solución: la colaboración de los usuarios.

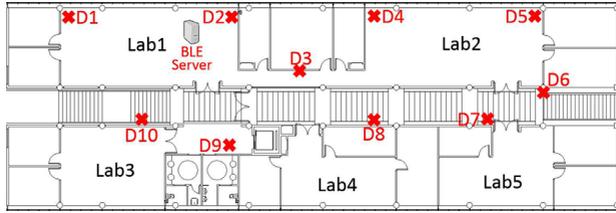


Fig. 8. Distribución de la red en malla de 10 dispositivos y servidor BLE.

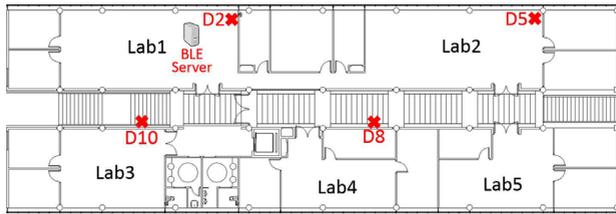


Fig. 9. Distribución de la red en malla de 4 dispositivos y servidor BLE.

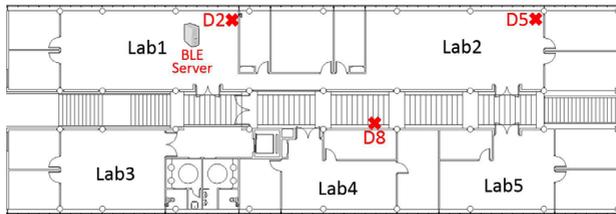


Fig. 10. Distribución de la red en malla de 3 dispositivos y servidor BLE.

Esta solución consiste en hacer que los dispositivos de los usuarios no solo envíen paquetes a la malla, sino que además actúen como los dispositivos de la misma: escuchando el medio y reenviando los mensajes recibidos con el formato adecuado. De esta manera, nuestra red en malla estará formada por dispositivos estáticos (nodos de la malla) y por dispositivos móviles: dispositivos del usuario que retransmiten los mensajes recibidos, además de enviar sus propios datos. Este tipo de malla nos permite aumentar el rango de cobertura de una manera dinámica, en función de la actividad y posicionamiento de los usuarios.

Para comprobar el impacto de esta mejora se ha repetido la última prueba, con la misma red BLE en malla de 3 dispositivos, pero en este caso los dispositivos de los 3 usuarios han sido configurados para reenviar los paquetes recibidos, además de enviar sus propios datos. Los resultados obtenidos se muestran en la Tabla IV.

TABLA IV
 PAQUETES RECIBIDOS DE 3 DISPOSITIVOS
 RECEPTORES-TRANSMISORES EN UNA RED EN MALLA BLE DE
 3 DISPOSITIVOS.

Usuario	Paquetes enviados	Paquetes recibidos	Éxito Tx
1	412	408	99.01%
2	418	367	87.80%
3	402	381	94.78%
Total	1232	1156	93.83%

La Figura 12 refleja los resultados obtenidos para los dos modos de trabajo con el mismo número de dispositivos. Los resultados de la gráfica muestran:

- Modo individual: porcentaje de paquetes recibidos de cada usuario en el servidor en función de los paquetes enviados, en una red con 3

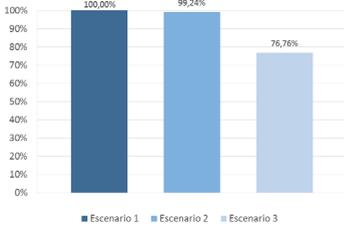


Fig. 11. Comparativa de los paquetes recibidos en los 3 escenarios descritos: malla de 10, 4 y 3 dispositivos respectivamente.

nodos en malla y 3 usuarios transmitiendo paquetes.

- Modo colaborativo: porcentaje de paquetes recibidos de cada usuario en el servidor en función de los paquetes enviados, en una red con 3 nodos en malla y 3 usuarios transmitiendo sus propios paquetes y retransmitiendo los paquetes recibidos de la red (formando parte de la malla).

Como se muestra en la Figura 12, el servidor recibe, de media, alrededor de un 17% más paquetes utilizando la misma red en malla (con el mismo número de dispositivos y usuarios) utilizando nuestro método colaborativo. Desglosado para cada uno de los usuarios, este aumento supone un 23.09%, 11.01% y 17.59% para los usuarios 1, 2 y 3 respectivamente. Por lo tanto, proponemos el uso de una red en malla que no esté formada únicamente por dispositivos estáticos, sino que incluya también los dispositivos de los propios usuarios, lo que nos permitirá incrementar el alcance y la cobertura de manera dinámica, en función del movimiento y posicionamiento de los usuarios.

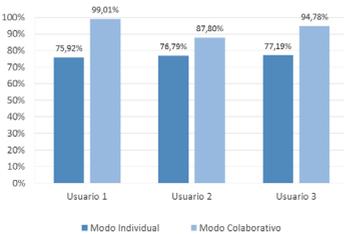


Fig. 12. Comparativa de los paquetes recibidos en una red en malla con 3 dispositivos estáticos y 3 usuarios.

VII. CONCLUSIONES

La red BLE en malla aparece como una interesante topología con un protocolo mejorado e ideal para múltiples aplicaciones relacionadas con IoT. En

estas aplicaciones, los usuarios reciben en sus dispositivos (smartphones, smartwatches, etc.) diferente tipo de información y envían información personal sobre sus propios parámetros corporales.

En este artículo se ha estudiado y evaluado una malla BLE, modificando el número de dispositivos que la conforman. Tras este estudio inicial se ha descubierto una carencia, y es que para asegurar una cobertura total es necesario incrementar el número de nodos de la malla, aumentando de esta forma el coste de la propia red. Para mejorar esta situación se ha propuesto un método de trabajo basado en el trabajo colaborativo, introduciendo en la malla los dispositivos de los usuarios. Tras probar con dispositivos reales esta propuesta se ha comprobado cómo la colaboración del usuario podría mejorar significativamente las prestaciones de una red final instalada en un espacio inteligente, sin incrementar el coste de la misma.

Nuestro trabajo futuro se centrará en convertir esta red en malla en una red heterogénea de bajo coste que cuente con diferentes dispositivos BLE (tanto con Bluetooth 4.0 como con Bluetooth 4.1), así como con dispositivos de ZigBee, utilizado en nuestro testbed ya instalado [12], o Wi-Fi, que asegura cobertura total.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el proyecto POII-2014-010P de la JCCM, por los proyectos TIN2015-66972-C5-2-R (MINECO/FEDER) y TIN2015-65845-C3-2-R, y por el plan de la UCLM de I+D+i cofinanciado por el Fondo Social Europeo.

REFERENCIAS

- [1] D. Hortelano, T. Olivares, M. C. Ruiz, and V. López, "Estudio del estándar Bluetooth para redes híbridas en IoT," in *XXVI Jornadas de Paralelismo*.
- [2] Bluetooth SIG, *BLUETOOTH SPECIFICATION Version 4.0*, 2010.
- [3] ZigBee Alliance, "Zigbee," <http://www.zigbee.org/>, 2016.
- [4] Bluetooth SIG, Inc., "Bluetooth technology adding mesh networking to spur new wave of innovation," <https://www.bluetooth.com/news/pressreleases/2015/02/24/bluetoothtechnology-adding-mesh-networking-to-spur-new-wave-of-innovation>, 2015.
- [5] Libellium Comunicaciones Distribuidas S.L., "Libellium," <http://www.libellium.com/>.
- [6] "Instituto de investigación en informática de albacete," <http://www.i3a.uclm.es/>.
- [7] H. S. Kim, J. Lee, and J. W. Jang, "Bmesh: A wireless mesh network protocol for bluetooth low energy devices," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, Aug 2015, pp. 558-563.
- [8] Trond Einar Snekvik, "nrf openmesh," .
- [9] Nordic Semiconductor, "nrf51 series soc," <https://www.nordicsemi.com/eng/Products/nRF51-Series-Soc>.
- [10] Qualcomm Technologies International, "Csrmesh," <https://wiki.csr.com/wiki/Csrmesh>.
- [11] Qualcomm Technologies International, "Bluetooth smart csr101x product family," <http://www.csr.com/products/bluetooth-smart-csr101x-product-family>.
- [12] Teresa Olivares, Fernando Royo, and Antonio M. Ortiz, "An experimental testbed for smart cities applications," in *Proceedings of the 11th ACM International Symposium on Mobility Management and Wireless Access*, New York, NY, USA, 2013, MobiWac '13, pp. 115-118, ACM.

Implementación Real de un Sistema de Gestión Eficiente de WSN basado en una Arquitectura BLE para IoT

Celia Garrido-Hidalgo¹, Teresa Olivares², M. Carmen Ruiz³, y Diego Hortelano⁴

Resumen— Este artículo propone un modelo de arquitectura centralizada como referencia para el despliegue de redes heterogéneas de sensores y actuadores basadas en la tecnología de red inalámbrica *Bluetooth Low Energy* (BLE) en el dominio de *Internet of Things* (IoT). En este contexto, y con el objetivo de verificar las aportaciones de esta tecnología, se ha desarrollado un caso de estudio basado en el despliegue de una Red Inalámbrica de Sensores y Actuadores (*Wireless Sensor and Actuator Network*, WSN) en una vivienda unifamiliar compuesta por una serie de sectores característicos. Se ha desarrollado un sistema centralizado capaz de gestionar el funcionamiento de la red, posibilitando la monitorización y el control de cualquier parámetro de la WSN.

Palabras clave— *Wireless Sensor and Actuator Network*, *Internet of Things*, *Bluetooth Low Energy*.

I. INTRODUCCIÓN

LA mejora de las técnicas de fabricación de dispositivos ha contribuido al desarrollo de las WSN [1], las cuales se han visto altamente influenciadas por la amplia diversidad de plataformas hardware que las constituyen. En este contexto, resulta imprescindible unificar el modelo comunicativo utilizado por todas ellas, posibilitando así una cooperación coordinada entre dispositivos que contribuya al ámbito de aplicación de IoT [2].

La gestión eficiente de WSN se ha convertido en un desafío perseguido en los últimos años, de modo que numerosos estándares de comunicación han evolucionado con objeto de mejorar rendimiento a partir de un reducido consumo energético. Tal es el caso del estándar Bluetooth, cuya evolución a versiones tales como la 4.0 y posteriores (BLE, [3]) ha repercutido ampliamente en el campo de aplicación de IoT.

Este artículo presenta un caso de estudio basado en el despliegue de una WSN en un entorno real, como es una vivienda unifamiliar compuesta por espacios tanto de interior como de exterior. Se han definido una serie de sectores representativos en la vivienda, situando en cada uno de ellos los nodos de la red de manera apropiada para asegurar una comunicación ininterrumpida con el sistema central de control, situado en el interior de la vivienda. De

esta forma, se han analizado algunas de las aportaciones positivas que el estándar BLE proporciona en el contexto de redes orientadas al control y la monitorización de entornos domésticos.

Se propone un modelo de sistema centralizado de control capaz de gestionar de manera apropiada las interacciones con los distintos nodos periféricos de la red. Asimismo, con objeto de verificar la compatibilidad entre distintos dispositivos, se han integrado en la WSN plataformas hardware que operan bajo un principio de funcionamiento distinto. Una vez desplegada la red, todas ellas han sido programadas de forma que su implementación física sea completamente transparente para el nodo central de control.

Para verificar la interacción de los usuarios con las funcionalidades de la WSN, se ha desarrollado una aplicación móvil que permite el control y la monitorización de parámetros relativos a la vivienda unifamiliar. Esta aplicación está basada en una comunicación constante vía TCP/IP con el sistema de control, la cual posibilita una gestión remota del sistema desplegado. De esta manera, se ha verificado la cooperación de un conjunto heterogéneo de dispositivos conectados vía BLE al sistema centralizado.

Las aportaciones más importantes de este trabajo se detallan a continuación:

- Despliegue de una WSN integrando un conjunto heterogéneo real de dispositivos.
- Implementación de un sistema centralizado de control para la WSN desplegada, capaz de gestionar de manera eficiente los distintos nodos periféricos de la red.
- Desarrollo de una aplicación móvil multi-plataforma capaz de acceder al sistema central de control para permitir a distintos usuarios interactuar con la red.

Este artículo está estructurado de la siguiente forma: las Secciones 2 y 3 se corresponden con el Estado del Arte y la motivación que ha impulsado este artículo, respectivamente. En la Sección 4 se realiza una breve descripción de la arquitectura tomada como referencia para el despliegue de la WSN. La Sección 5 argumenta en detalle el caso de estudio práctico, en el que se analiza el despliegue y la implementación física de cada una de las capas de la arquitectura. Finalmente, en la Sección 6 se definen las pruebas del sistema desplegado junto con los resultados obtenidos, encontrándose en la Sección 7 las conclusiones y aportaciones de este trabajo.

¹Instituto de Investigación en Informática de Albacete, UCLM, Albacete, España, email: Celia.Garrido@uclm.es.

²Escuela Superior de Ingeniería Informática, UCLM, Albacete, España, email: Teresa.Olivares@uclm.es.

³Escuela Superior de Ingeniería Informática, UCLM, Albacete, España, email: M.Carmen.Ruiz@uclm.es.

⁴Instituto de Investigación en Informática de Albacete, UCLM, Albacete, España, email: Diego.Hortelano@uclm.es.

II. ESTADO DEL ARTE

El despliegue de WSAN en los llamados "Hogares Inteligentes" (*Smart Homes*) es uno de los ámbitos más abordados en lo que respecta a la monitorización de entornos reales. Tanto la arquitectura de referencia como la tecnología de red utilizada para comunicar los distintos nodos en una determinada WSAN, son algunos de los factores de mayor interés que actualmente están analizándose con objeto de aumentar la versatilidad de este tipo de sistemas. BLE irrumpe con fuerza en este área de aplicación, donde ya se pueden encontrar interesantes trabajos relacionados con la mejora de las comunicaciones y automatización de entornos domésticos [5] o casos reales que benefician nuestra rutina diaria [9].

En la actualidad, se han llevado a cabo numerosos estudios comparativos entre las tecnologías de red inalámbrica más extendidas, como por ejemplo entre ZigBee y BLE [4]. La gran mayoría de estos trabajos de investigación coinciden en el análisis de estas dos tecnologías, prestando especial interés a factores tales como la velocidad de transmisión, alcance, y consumo energético. En este contexto, BLE se abre paso entre las tecnologías de red más tradicionales.

Asimismo, la optimización del ciclo de trabajo BLE de dispositivos pertenecientes a una misma WSAN ha recibido en la actualidad diversos enfoques, todos ellos basados en la alternancia de estados activos e inactivos. En cuanto a BLE, [6] aporta un algoritmo basado en ciclos de bajo consumo para los nodos pertenecientes a una misma red, en el que se analizan de manera descriptiva las aportaciones energéticas al conjunto global. Sin embargo, se trata de un análisis que no ha sido puesto a prueba en un escenario real, de manera que no se verifica la funcionalidad real del algoritmo propuesto.

III. MOTIVACIÓN

La motivación principal de nuestro trabajo son las enormes posibilidades de aplicación que presenta el estándar BLE en WSN. En concreto, nuestra línea de investigación está orientada a la mejora de la eficiencia energética así como las condiciones de operación de la propia red en un entorno real de operación, lo que conlleva un beneficio del usuario final. El despliegue y la gestión de redes en entornos reales permite obtener una serie de resultados que podrían no ser predecibles mediante estudios meramente teóricos.

Mediante el despliegue de la red en un entorno real, tal y como es una vivienda unifamiliar con una serie de sectores representativos, permite determinar cuáles son los principales desafíos en redes inalámbricas de comunicación que presenta la arquitectura propuesta.

El principio fundamental en el que se basa el despliegue de una WSN centralizada en un entorno como el propuesto, consiste en la cooperación entre dispositivos, mediante la centralización de la toma de decisiones. Por ello, resulta imprescindible que el sistema central gestione de manera adecuada el flujo de información procedente de los sensores de la red,

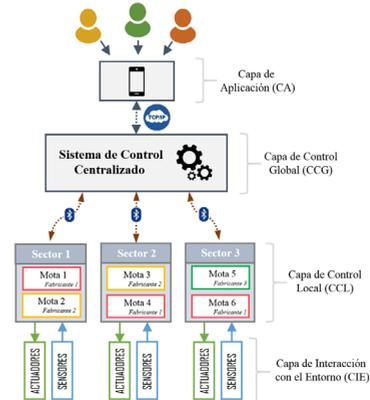


Fig. 1. Modelo genérico de arquitectura centralizada

extrayendo aquellos parámetros más relevantes que impliquen una respuesta en forma de actuación. Una vez analizada y filtrada dicha información, se proporciona directamente al usuario vía TCP/IP mediante una aplicación móvil de control y monitorización.

IV. DESCRIPCIÓN DE LA ARQUITECTURA

Esta sección aborda una descripción en detalle de la arquitectura, en la que se muestran los aspectos más importantes en el ámbito de la gestión de WSAN en entornos cotidianos mediante un sistema centralizado de control. En la Figura 1 se muestran las capas que forman parte de la arquitectura. Se pueden encontrar más detalles en [7].

A continuación se muestra un resumen de las principales capas que forman parte de la arquitectura, las cuales abarcan el tratamiento de la información desde su adquisición hasta su monitorización mediante una aplicación de usuario:

- **Capa de Interacción con el Entorno (CIE):** capa encargada de la adquisición de información del entorno así como actuación sobre determinadas magnitudes físicas, mediante el uso de una serie de sensores y actuadores (Tabla I).
- **Capa de Control Local (CCL):** se trata de una capa intermedia de pre-procesamiento de la información, homogeneizándola y estructurándola de manera eficiente. Esta capa intermedia, compuesta por un conjunto de nodos periféricos, establece comunicación cableada con el conjunto de sensores y actuadores (capa inmediatamente inferior, CIE) y comunicación inalámbrica BLE con su capa superior, CCG.
- **Capa de Control Global (CCG):** esta capa representa al sistema de control centralizado encargado de gestión de la información a nivel global en la red. Está representada por un único

TABLA I
 DESPLIEGUE DE NODOS EN LA VIVIENDA, SENSORES Y ACTUADORES

Cocina Sensores/Actuadores		Sala de Estar Sensores/Actuadores		Huerto Sensores/Actuadores		Jardín Sensores
LBB	WSP	LBB	WSP	WSP		WSP
Presencia (HC-SR501)	Inundación (STM32-ST)	Humedad, Temperatura (DHT-22)	Presencia (HC-SR501)	Humedad Suelo (OB-Soil 01)	Humedad, Temperatura (DHT-22)	Presión Atmosférica (MTL-115A2)
Gases (MQ-135)	Luminosidad (TSL-2561)	Interruptor (Luz)	Luminosidad (TSL-2561)	Caudal (Flow-meter)	Luminosidad (TSL-2561)	Precipitación (LaCrosse Tx11)
Interruptor (Luz)			Motor (Persiana)	Electroválvula (Riego)	Motor (Toldeo)	Viento (LaCrosse Tx12)

nodo, en el cual residen las funciones más importantes de la red, entre las que destacan coordinación de dispositivos y estructuración de la información.

- **Capa de Aplicación (CA):** se trata de la capa de mayor abstracción de la arquitectura, en la cual se ofrece al usuario la posibilidad de interactuar directamente con determinados nodos de la red. Para ello, se propone el utilización del protocolo TCP/IP para la transferencia de información entre CCG y AL.

V. CASO DE ESTUDIO: DESPLIEGUE DE LA RED

En esta sección se detallan las distintas fases analizadas para el despliegue de la red. En primer lugar, en cuanto al entorno de pruebas que se ha utilizado, se trata de una vivienda unifamiliar con espacios tanto de interior como de exterior, con una superficie total aproximada de 1000 m². En el contexto de esta vivienda se han definido una serie de sectores de carácter general, considerando tanto su localización como la naturaleza de las magnitudes físicas que se desean controlar y monitorizar. En particular, se han definidos los siguientes sectores:

- **Cocina (Kitchen):** en este tipo sector resulta de gran interés el despliegue de un conjunto de sensores encargados de detectar situaciones anómalas, como por ejemplo una inundación provocada por una fuga, un escape de gas, etc.
- **Sala de Estar (Living Room):** se trata de un sector interior a la vivienda cuyo carácter genérico permite monitorizar una serie de parámetros ambientales básicos, tales como la luminosidad, la temperatura, la humedad, la calidad del aire, etc. Se incluye además la posibilidad de controlar distintos actuadores con objeto de mejorar el confort de los habitantes de la vivienda.
- **Huerto (Orchard):** se engloba en este sector a un conjunto de cultivos que requieren ser sometidos a un cuidado específico en función de las condiciones ambientales. Con ello, se ha destinado este sector a la monitorización de las magnitudes físicas más representativas de los mismos.
- **Jardín (Garden):** se trata de una región exterior de la vivienda en la que es interesante extraer magnitudes físicas tales como el índice UV,

la luminosidad o el sentido y la dirección del viento. Típicamente se trata de una estación meteorológica, la cual aporta al sistema central información que puede ser utilizada para realizar pronósticos medioambientales y estimaciones, contribuyendo a un proceso de toma de decisiones eficaz.

A. Capa de Interacción con el Entorno (CIE)

Esta capa representa el nivel más bajo de la arquitectura, en ella se determinan los sensores y actuadores que forman parte de la red. De acuerdo con los sectores genéricos definidos anteriormente, se han utilizado sensores de luminosidad, temperatura, humedad, inundación, etc. En cuanto a los actuadores, han sido simulados con objeto de verificar el correcto funcionamiento de la red, sin que ésta dependa directamente de su implementación física.

En la Tabla I se encuentran definidos los sensores y actuadores seleccionados de acuerdo con cada uno de los sectores definidos. Sin embargo, de acuerdo con la arquitectura de referencia planteada, cualquier tipo de sensor y actuador podría integrarse en el sistema, siempre y cuando haya sido previamente calibrado.

B. Capa de Control Local (CCL)

La función de esta capa es transmitir (vía BLE) la información adquirida por los sensores a su capa superior (CCG), así como enviar de manera inmediata las órdenes dirigidas a los actuadores de la red, englobados en su capa inferior (CIE).

En cuanto a la comunicación CCL-CCG, se ha unificado el formato de la información transmitida, de manera que cada una de los nodos pertenecientes a la CCL se encarga de unificar y convertir la información de acuerdo con una estructura fija en todo el sistema. Para ello, la información es codificada de acuerdo con un criterio establecido por el nodo central, posibilitando la inclusión de nuevos nodos en el sistema sin necesidad de reconfigurar el sistema central de control (CCG). Asimismo, los nodos periféricos que forman parte de la CCL han sido programados bajo un ciclo parcial en el que se alternan estados activos e inactivos en función de las necesidades de la red. Por tanto, siempre que un dispositivo periférico de la red se encuentre en modo activo, enviará la información monitorizada (mediante BLE) al nodo central, y llevará a cabo las acciones requeridas.

Se han seleccionado dos plataformas diferentes para desempeñar la función de nodo periférico. De esta forma, se posibilita la unificación del flujo de información existente entre CCL-CCG con vistas a desplegar una red heterogénea que, posteriormente, pueda ser escalada a distintos entornos o funcionalidades. Se han analizado e integrado las placas controladoras LightBlue Bean [10] y Waspnote [11], mostradas en las Figuras 2(a) y 2(b), respectivamente:

- **LightBlue Bean (LBB):** plataforma hardware basada en el micro-controlador ATmega 328p [12] con una frecuencia de reloj de 8 MHz, que incluye un módulo de radio BLE integrado (LBM313, [13]). LBB cuenta la versión 4.0 del estándar BLE, pudiendo desempeñar la función de nodo periférico de la red (*slave* si existe un vínculo de conexión con el dispositivo central, o *advertiser* en el caso contrario).
- **Waspnote (WSP):** plataforma hardware basada en el micro-controlador ATmega 1281 [12] con una frecuencia de reloj de 14 MHz. WSP incluye un conjunto de sockets habilitados para el conexionado de módulos de radio de distintas tecnologías de red; ZigBee, Bluetooth, LoRa, etc. En cuanto al módulo de radio BLE utilizado para el despliegue, se trata del modelo BLE112 [14]. Este módulo trabaja bajo la versión 4.0 del estándar BLE, aunque puede desempeñar tanto roles de dispositivo periférico (*slave* o *advertiser*) como de dispositivo central (*master* en caso de existir vínculo de conexión o *scanner* en caso contrario).

En la Tabla II se muestra una comparativa de algunas de las características más representativas de las plataformas anteriormente descritas. En función de dichas características se ha establecido la posición y orientación de los nodos en la vivienda.

TABLA II
 CARACTERÍSTICAS INTRÍNSICAS DE LOS NODOS

	LBB	WSP
Programación	Inalámbrica	Cableada
Potencia de transmisión	4 dBm	(-18, 8) dBm
Velocidad de transmisión	No ajustable (250 kbps-1Mbps)	Ajustable 1Mbps
Antena	Direccional	Omnidireccional
Configuración	Periférico	Perif./Central

En cuanto a la distribución de los nodos, puede apreciarse cómo los de tipo LBB han sido principalmente utilizados para sectores de interior, debido a las limitaciones de cobertura impuestas por la implementación física de su antena. Aprovechando la ganancia de la antena que WSP aporta, este tipo de nodo se ha instalado en sectores de exterior, verificándose previamente cómo el área de cobertura abarca la superficie completa de la vivienda.

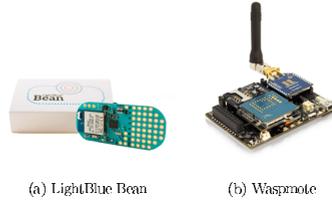


Fig. 2. Plataformas hardware para nodos periféricos



Fig. 3. Plataformas hardware para el nodo central y adaptador USB de radio BLE

C. Capa de Control Global (CCG)

El sistema central de control encargado de la toma de decisiones y gestión de la información reside en la CCG. Dicho sistema está compuesto por un único nodo que se comunica constantemente mediante BLE con los nodos periféricos de la capa inferior (CCL) vía BLE y con la capa inmediatamente superior (CA) vía TCP/IP.

Para las pruebas de la WSAN en el escenario propuesto se ha utilizado una placa computadora de propósito general Raspberry Pi 2B [15]. Dicha plataforma ha permitido el despliegue de la red con un coste reducido, y cumpliendo las especificaciones requeridas de funcionamiento en la red. A esta placa se le ha incluido un adaptador USB de radio BLE (CSR 4.0 [16]) basado en la implementación Linux de la pila Bluetooth (BlueZ, [17]). A partir de lo anterior, se ha realizado un estudio de coberturas que será posteriormente detallado (Figura 4).

La programación del nodo central de control se ha basado en el lenguaje asíncrono JavaScript. De esta forma se ha posibilitado una gestión muy eficiente de los distintos nodos periféricos del sistema. En particular, se ha utilizado Node.js [18] como entorno de ejecución. El módulo *noble.js* [19] ha proporcionado las funciones principales requeridas para un nodo central de acuerdo con el estándar BLE. Por último, la tecnología WebSocket [20] ha sido utilizada en el contexto de la comunicación vía TCP/IP, de forma que toda la información de la red es transmitida a la aplicación móvil.

En cuanto a la funcionalidad del nodo central de control, se han considerado dos modos de operación de acuerdo con las exigencias del entorno en que la red ha sido desplegada, los cuales se describen a continuación:

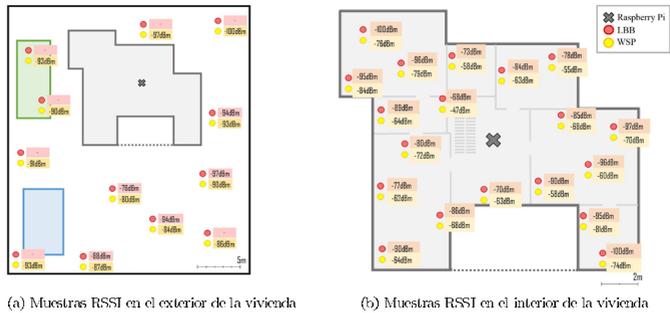


Fig. 4. Mapa de señal RSSI medido en la vivienda

- Modo autónomo: basado en un proceso de toma de decisiones en el que a partir de unos límites, preestablecidos por el usuario, el sistema central genera actuaciones orientadas a la gestión de la vivienda. Algunos de los parámetros a gestionar por el sistema son la iluminación, la temperatura, o incluso el suministro de agua, el cual podría ser parcialmente interrumpido en caso de detección de fugas.
- Modo usuario: este modo está basado en solicitudes de actuación específicas del usuario, en cuyo caso preponderarán sobre las decisiones autónomas del sistema central hasta que el usuario no indique lo contrario.

C.1 Estudio de cobertura en el entorno de pruebas

Con objeto de localizar cada una de los nodos en los sectores definidos, se ha realizado un estudio de cobertura BLE en el escenario de pruebas para el despliegue. Para ello, se ha realizado muestreo de señal RSSI (*Received Signal Strength Indication*) para BLE en coexistencia con Wi-Fi, de manera que las condiciones de operación de la red BLE sean lo más realistas posibles. Los resultados del estudio de coberturas realizado, mostrados en la Figura 4, han sido utilizados para la localización de los nodos en la vivienda.

Se ha llevado a cabo un análisis de muestras BLE-RSSI, comprobando la intensidad de señal proveniente de cada una de los nodos. Para ello, se ha preestablecido una situación céntrica para el nodo controlador y se ha variado la posición de los nodos WSP y LBB. En la Figura 4(a) se han representado las muestras tomadas en los espacios exteriores a la vivienda. A su vez, Figura 4(b) se muestra el mapa de valores RSSI medidos desde el nodo central para distintas localizaciones de los periféricos en la planta inferior. Las principales conclusiones obtenidas se exponen a continuación:

- El rango de alcance de LBB resulta significativamente menor que el de los nodos WSP, ya que estos últimos cuentan con una antena que aporta una ganancia adicional de 5 dB.

- Considerando el alcance y la funcionalidad de cada una de las plataformas estudiadas se ha determinado la utilización de nodos tipo WSP en los sectores de exterior, asegurando así una comunicación fluida con el sistema de control central. Asimismo, los sectores de interior de la vivienda se ha desplegado un conjunto heterogéneo de nodos periféricos (LBB y WSP), debido a que ambos posibilitan la cobertura de la superficie propuesta, aportando cada uno de ellos diferentes ventajas, más o menos significativas en función de los requerimientos del sector en cuestión.
- A partir de un conjunto heterogéneo de nodos periféricos, se ha verificado un área de cobertura suficiente para abarcar los distintos sectores (tanto de interior como de exterior) desde la localización del nodo central. Lo anterior implica, de acuerdo con la descripción de la vivienda, un rango de alcance aproximado de 15 metros en presencia de distintos elementos constructivos tales como tabiques, muros, puertas o ventanas.

D. Capa de Aplicación (CA)

La CA es la encargada de proporcionar un mecanismo de interacción entre la red y los usuarios así como otros sistemas externos. Por ello, esta sección centra su atención en el desarrollo de una aplicación para la monitorización de las magnitudes físicas medidas en la red, posibilitando a su vez el control de distintos procesos.

De acuerdo con el modelo de arquitectura tomado como referencia, la capa de aplicación se comunica directamente mediante el protocolo TCP/IP con el nodo central del sistema, a partir de la tecnología WebSocket. Por este motivo, una de las principales ventajas que aporta la arquitectura es la posibilidad de gestión remota de la red, e incluso simultánea de un número determinado de usuarios y sistemas externos conectados a ella.

En lo que respecta al desarrollo software, se ha empleado el lenguaje JavaScript con objeto de im-



Fig. 5. Interfaz gráfica de la aplicación de usuario

plementar la funcionalidad del cliente. Asimismo se ha utilizado para el desarrollo de la aplicación Android una plataforma denominada Apache Cordova [21]. Esta plataforma, basándose en HTML5, CSS y JavaScript, permite el desarrollo de aplicaciones nativas compatibles con distintas plataformas, tales como Android, iOS, Windows Mobile, Blackberry, etc. En la Figura 5 se muestran dos de las principales interfaces de aplicación desarrolladas con objeto de proporcionar a un conjunto de usuarios acceso a la gestión de la red.

La aplicación móvil está estructurada de manera que, una vez autenticado, el usuario tiene acceso a cada uno de los sectores definidos en la vivienda. Se ha desarrollado una página distinta para la gestión de cada uno de los sectores, de manera que en cada uno de ellos se posibilita tanto la monitorización como la actuación sobre las magnitudes físicas establecidas.

Debido a que existe cierta simetría entre la monitorización y actuación de los distintos sectores de la red, se ha tomado el sector denominado *Living Room* como referencia para mostrar la funcionalidad que la aplicación móvil, Figura 5. La primera interfaz muestra los sectores disponibles, y la segunda el panel de control y monitorización de un sector en particular. A partir de este panel, es posible la visualización de las distintas magnitudes físicas de la red, las cuales son refrescadas en intervalos de tiempo prefijados. También se ofrece al usuario información acerca del estado de los distintos actuadores de la red, pudiendo ser modificados de acuerdo con sus necesidades.

VI. PRUEBAS REALIZADAS

La Tabla III muestra el despliegue tanto de hardware como de software llevado a cabo a lo largo de este caso de estudio. Una vez desplegada la red en el entorno descrito previamente, se han llevado a cabo una serie de pruebas con objeto de determinar las limitaciones del sistema así como posibles mejoras. Estas pruebas están orientadas a determinar el grado de idoneidad de la tecnología de red BLE, así como considerar un futuro escalado de la red. Las pruebas llevadas a cabo se detallan a continuación:

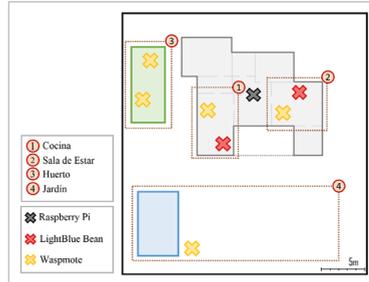


Fig. 6. Despliegue de los nodos en la sala de estar de la vivienda

En la Figura 6 se muestra el despliegue de los nodos en la vivienda unifamiliar descrita.

En la Figura 7 se muestra el modelo final de arquitectura que se ha implementado en escenario propuesto. En ella se pueden observar las tecnologías de redes empleadas para la comunicación entre las distintas capas, junto con el despliegue tanto hardware como software de plataformas utilizadas.

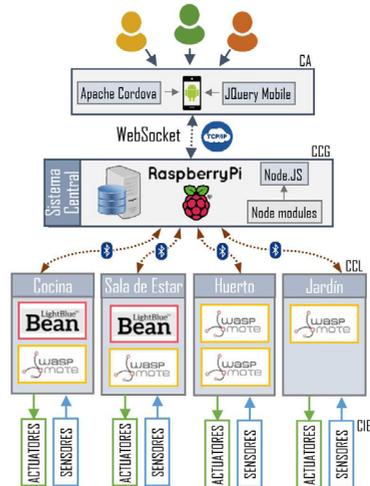


Fig. 7. Arquitectura final del sistema centralizado

1. Puesta en funcionamiento del sistema.

Tras realizar el estudio de cobertura basado en la optimización del nivel de señal RSSI de cada uno de los periféricos percibido por el nodo central, se ha puesto en funcionamiento el sistema. Los nodos periféricos de la red han sido programa-

TABLA III
 DESPLIEGUE DE PLATAFORMAS HARDWARE Y SOFTWARE

Plataforma	Descripción	Versión	Fecha de lanzamiento
Hardware			
LightBlue Bean	Placa controladora con módulo de radio BLE	-	Diciembre, 2014
Wasmote	Placa controladora	Wasmote PRO	Febrero, 2013
Raspberry Pi	Placa computadora de bajo coste	Raspberry Pi 2 B	Febrero, 2015
Software			
Bluetooth	Tecnología de red inalámbrica	BLE (v. 4.0)	Junio, 2010
BlueZ	Pila Bluetooth para Linux	BlueZ 5.35	Septiembre, 2015
Raspbian	Sistema operativo para la placa computadora Raspberry Pi	Raspbian Jessie	Septiembre, 2015
Node.js	Entorno de ejecución JavaScript	Node.js v5.0.0	Octubre, 2015
Cordova	Plataforma de desarrollo de aplicaciones nativas	Apache Cordova 3.0	Abril, 2015
jQuery Mobile	Marco de trabajo para el desarrollo de aplicaciones móviles	jQuery Mobile 1.4.5	Octubre, 2015

dos para enviar información al nodo central en intervalos de 5 minutos, el cual ha sido capaz de gestionar de manera inmediata el flujo de información recibido. Con ello, se verifica una comunicación CCL-CCG directa e inmediata, permitiendo el reposo de los nodos periféricos durante un tiempo razonable con objeto de prolongar la vida útil de sus baterías.

2. **Funcionamiento autónomo del sistema.** Se ha programado un prototipo de medida energética en el sistema central de control. Esta medida consiste en la gestión de la iluminación en función del nivel de luz ambiente percibido en un determinado sector, considerando en todo caso la presencia o no de individuos en la habitación. El correcto funcionamiento de este prototipo orientado a la eficiencia energética nos ha permitido verificar la autonomía del sistema central de control para gestionar los actuadores de la red, permitiéndose un desarrollo futuro que dote de mayor autonomía al núcleo del sistema para la toma de decisiones.

3. **Conexión de múltiples usuarios.** Se ha instalado la aplicación móvil desarrollada en distintos dispositivos, no necesariamente situados en contacto directo con la WSN. Una vez autenticados, todos ellos han sido capaces de monitorizar los parámetros de la red, así como enviar órdenes a los actuadores. Además, la aplicación desarrollada ha considerado la situación en que múltiples usuarios envían un mismo comando de manera simultánea.

VII. CONCLUSIONES Y TRABAJOS FUTUROS

Este artículo ofrece un caso de estudio de una propuesta de arquitectura orientada a la utilización de la tecnología de red BLE para el control y la monitorización de entornos domóticos. Se ha detallado el despliegue real de un conjunto de nodos periféricos en una vivienda unifamiliar, verificando la tolerancia de la red BLE ante la interposición de distintos elementos constructivos. La tecnología de red inalámbrica BLE ha demostrado proporcionar un amplio rango de cobertura a partir de un consumo reducido, lo

que convierte en una tecnología prometedora en el ámbito de aplicaciones orientadas a la cooperación de objetos IoT.

La utilización de un conjunto heterogéneo tanto de sensores como de plataformas controladoras (WSP y LBB) ha permitido validar la escalabilidad de la arquitectura propuesta. Esto se debe a que el sistema de control centralizado es apropiado para la incorporación de nuevos dispositivos independientemente de su implementación física. Asimismo, mediante la programación orientada a eventos del sistema central de control se ha dotado de una mayor versatilidad a la red, ya que la gestión de cada uno de los eventos de conexión y desconexión de los nodos periféricos de la red ha podido ser controlada de manera inmediata, permitiendo que éstos permanezcan un menor intervalo de tiempo activos y, con ello, un menor consumo energético reduciendo la ocupación del espectro electromagnético de la red. Las conclusiones principales que se han alcanzado se detallan a continuación:

- Se ha analizado un conjunto de placas controladoras basadas en BLE (nodos periféricos del sistema), destacando parámetros tales como el alcance, la direccionalidad o la configurabilidad.
- Considerando las características de cada uno de los nodos, se han distribuido en distintos sectores representativos de una vivienda. Para ello, se ha realizado un estudio de campo previo basado en el toma de muestras RSSI, conformando un mapa de cobertura de la vivienda.
- Se ha programado el sistema central para desempeñar dos modos de funcionamiento: basado en el usuario y autónomo. De esta forma, se dota al sistema central de la capacidad de tomar decisiones autónomas, permitiendo en todo momento la gestión de la red por parte del usuario.
- A partir del desarrollo de una aplicación móvil, se ha puesto a prueba el funcionamiento de la red. La comunicación entre los nodos periféricos y el nodo central se ha llevado a cabo correctamente, y distintos usuarios han podido gestionar la WSN de manera simultánea sin necesidad de contacto directo con la misma.

Como trabajo futuro se propone una ampliación de la red, así como la puesta en marcha de nuevos actuadores, que realizarán determinadas acciones demandadas por el servidor o por el propio usuario.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el proyecto POII-2014-010P de la JCCM, por los proyectos TIN2015-66972-C5-2-R (MINECO/FEDER) y TIN2015-65845-C3-2-R.

REFERENCIAS

- [1] I. F. Alyildiz, W. Su, Y. Sankaresubramaniam, and E. Cayirci, "Wireless Sensor Networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, 2002.
- [2] J. Cubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1615–1660, Sept. 2013.
- [3] Bluetooth SIG, "Bluetooth technology basis: Bluetooth Low Energy," <https://www.bluetooth.com/>.
- [4] M. Siekinen, M. Hienkari, J.K. Nurminen, and J. Nieminen, "How low energy is Bluetooth Low Energy? comparative measurements with zigbee 802.15.4," in *Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE*. IEEE, 2012, pp. 232–237.
- [5] M. Collota and G. Pau, "Bluetooth for Internet of Things: A fuzzy approach to improve power management in smart homes," *Computers & Electrical Engineering*, vol. 44, pp. 137–152, 2015.
- [6] A. Demenchev, S. Taylor S. Hodges, and J. Smith, "Power consumption analysis of Bluetooth Low Energy, ZigBee, and Ant sensor nodes in a cyclic sleep scenario," April 2013.
- [7] V. Lopez, C. Garrido, T. Olivares and M.C. Ruiz, "Architecture Proposal for Heterogeneous, BLE-based Sensor and Actuator Networks for Easy Management of Smart Homes," in *Information Processing in Sensor Networks (IPSN)*, April 2016.
- [8] M. Collota and G. Pau "Bluetooth for Internet of Things: A fuzzy approach to improve power management in smart homes," in *Computers and Electrical Engineering*, vol 44 2015.
- [9] W. Narzi, S. Mayerhofer, O. Weichselbaum, S.Haseboeck, N. Hoffer and J. Kepler, "Bo-In/Bo-Out with Bluetooth Low Energy: Implicit Ticketing for Public Transportation Systems," in *Intelligent Transportation Systems*, 2015.
- [10] Punch Through, "The lightblue bean family," <https://punchthrough.com/bean/>.
- [11] Libelium, "Waspmotes" <http://www.libelium.com/products/waspmote/>.
- [12] ATMEL, "Atmega microcontrollers," <http://www.atmel.com/>.
- [13] Punch Through, "4_bea313 technical datasheet document," <https://punchthrough.com/docs/files/LEB313/>.
- [14] Bluegiga BLE112, "Ble112 module," https://www.bluetooth.org/tsg/BleFlores/BLE112_Datasheet1.pdf.
- [15] Raspberry Pi Foundation, "Raspbian jessie operative system," <https://www.raspberrypi.org/downloads/raspbian/>.
- [16] Qualcomm Technologies International, "Csr," <http://www.csr.com/>.
- [17] BlueZ Project, "Official linux bluetooth protocol stacks," <http://www.bluez.org/>.
- [18] Node.js Foundation, "Official node.js homepage," <https://nodejs.org/en/>.
- [19] Sandeermistry, "A node.js ble (bluetooth low energy) central module," <https://github.com/sandeermistry/noble>.
- [20] Kaazing, "Websocket," <http://www.websocket.org/>.
- [21] The Apache Software Foundation, "Official apache cordova homepage," <https://cordova.apache.org/>.
- [22] Raspberry Pi Foundation, "Raspberry pi 2 model b," <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>.

TrustedNet: Protocolo para la creación de redes espontáneas basadas en la confianza

José Vicente Sorribes¹ y Lourdes Peñalver¹

Resumen— En este artículo se presenta un modelo de red inalámbrica y sin infraestructura: TrustedNet. La idea es construir un modelo de red basado en la confianza. Para ello se define un protocolo que mediante el descubrimiento de vecinos y el intercambio y validación de claves permite establecer una red de confianza siguiendo un modelo de las relaciones humanas.

Para el modelado y estudio de este tipo de redes se ha utilizado el simulador Castalia 3.2, una herramienta que funciona sobre OMNET++ y que permite simular redes de sensores inalámbricas.

Se ha validado el modelo a través de varias simulaciones, donde se obtienen el número de vecinos, considerando distintos parámetros, tales como tipos de enlaces, modelos de colisión, BackOFFType, Carrier Sensing. Además, se ha analizado en 1 clicke el tiempo que tarda TrustedNet en crear la red de confianza, y el máximo y promedio del número de vecinos descubiertos para distintos modelos de colisión, obteniendo en todos los casos resultados razonables.

Palabras clave— redes basadas en la confianza, redes ad hoc, descubrimiento de vecinos, simulador Castalia, cifrado.

I. INTRODUCCIÓN

EN las MANETS, redes inalámbricas ad hoc de dispositivos móviles, cada dispositivo puede moverse independientemente en cualquier dirección, pueden aparecer nuevos nodos en la red y/o abandonarla otros.

Los dispositivos (nodos) que la forman, cuentan con un transceptor de radio de alcance limitado, y por lo tanto habrá dispositivos con los cuales puede comunicar directamente (vecinos, en el rango de transmisión del nodo) y otros con los cuales se comunica en varios saltos (multi-hop).

Inicialmente, cada nodo desconoce esos vecinos, por lo tanto en la creación de una red ad hoc, el primer paso tras el despliegue suele ser el descubrimiento de los mismos [1-11], lo que permitirá a cada nodo conocer qué nodos están en su rango de transmisión, útil para fases posteriores, como por ejemplo el encaminamiento.

De esta forma, cada dispositivo puede capturar tráfico destinado a él mismo, así como encaminar mensajes no destinados a su propio uso (destinado a otros) y por tanto debe poder actuar como un router.

Debido a la movilidad de los nodos en las MANETS, cada nodo puede frecuentemente cambiar sus enlaces con otros nodos, esto es, puede dejar de ser vecino de un nodo y serlo de otros o incluso abandonarlo la red.

En este contexto, deseamos desarrollar técnicas que permitan la creación de redes ad hoc basadas en la espontaneidad de las interacciones humanas, como

cuando se junta un grupo de personas que pueden comunicar, intercambiar información o trabajar conjuntamente durante un periodo de tiempo determinado y en una localización delimitada.

Hay muchas áreas de aplicación para las redes espontáneas ad-hoc, tales como la industrial (comunicación entre sensores, robots, y redes digitales), negocio (meetings, control de stock...), militar (entornos duros y hostiles), y enseñanza.

Uno de los principales problemas relacionados con este tipo de redes es la seguridad. Al ser una red ad hoc y que dura un espacio de tiempo delimitado no hay garantías de que exista una Autoridad Certificadora (AC).

Las características de las redes espontáneas ad hoc hacen que los requerimientos sean mucho más complejos que en las redes tradicionales: topología dinámica, ancho de banda restringido, diferentes capacidades de los enlaces y tasas de error altas, limitaciones de energía y capacidad de procesamiento, ausencia de un servidor central y a menudo no hay información a priori de los nodos que formarán la red [30].

Para caracterizar las redes espontáneas, pasamos a enumerar un conjunto de propiedades bien conocidas en la literatura [31] y que afectan a la organización, operación y gestión de ellas:

- No hay topología fija: los dispositivos (nodos) son móviles y pueden moverse libremente en el área que ocupa la red, dentro o fuera del rango de transmisión de otros nodos.
- Cada nodo es un router, y tiene un rango de transmisión limitado, pudiendo encaminar mensajes que no están destinados a ellos mismos.
- Recursos limitados en cuanto a CPU, memoria y energía. Los dispositivos móviles generalmente operan con baterías, que se pueden agotar, y la cantidad de energía disponible para cada dispositivo puede variar. Para ahorrar energía muchos nodos pueden decidir no estar disponibles todo el tiempo (y permanecer en el modo sleep).
- Medio de transmisión físico compartido: El medio de transmisión está accesible para cualquiera en el rango de transmisión con el equipo adecuado.
- Las identidades de cada nodo vienen dadas por direcciones IP: cada dirección es obtenida dinámicamente, y por tanto es difícil asociar una identidad fija con una dirección IP, aunque en nuestro caso usaremos identificadores únicos para cada nodo (por ejemplo, dirección MAC). El protocolo desarrollado debe poder detectar la existencia de direcciones IP duplicadas.
- Vulnerabilidad física: el tamaño reducido de los dispositivos inalámbricos móviles provoca que

¹Dpto. de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia, email: jvsorribes@hotmail.com

puedan ser fácilmente robados y/o posiblemente modificados. Por tanto, la relación con el propietario del nodo no es estable, y el nodo puede ser alterado.

- Falta de administración central: Los dispositivos que componen la red pueden venir de cualquier sitio, siendo nuevos nodos que pasarán a formar parte de la red, y no hay administración central.

Si queremos dotar de seguridad a este tipo de redes en su creación, podemos usar un modelo basado en la confianza.

Las propiedades que hemos enumerado previamente presentan dos importantes restricciones: Los nodos no deberían ser de confianza sin autenticación adecuada de nodo y usuario. Los retos de seguridad en redes inalámbricas espontáneas derivan de las propiedades siguientes: Los servicios tales como confidencialidad, integridad y disponibilidad deben ser proporcionados bajo estas condiciones y estos servicios pueden ser establecidos basados en relaciones humanas de confianza.

En este artículo, se presenta un modelo para la creación de una red ad hoc utilizando un modelo de confianza. En una red de este tipo a la hora de delimitar la seguridad deberemos dotarla de las mismas propiedades que una red tradicional, esto es: confidencialidad, integridad, disponibilidad y control de acceso con autenticación, todos basados en mecanismos de cifrado que deben ser ofrecidos sin administración central y con restricciones de energía.

Si queremos crear redes inalámbricas espontáneas basadas en la confianza, necesitamos establecimiento de confianza, y gestión de claves. También necesitamos disponibilidad de red y seguridad del encaminamiento.

Así mismo, se expone un protocolo para la creación de redes basadas en la confianza, que llamamos TrustedNet, y que utiliza el descubrimiento de vecinos en una de las fases iniciales de formación de la red. A continuación, se evaluarán sus prestaciones mediante simulación con Castalia 3.2 [12].

El resto del artículo se organiza de la siguiente forma: Se presenta un breve estado del arte en la sección II, la herramienta de simulación Castalia 3.2 en la sección III, en la sección IV se describe el modelo de red TrustedNet, la adaptación de la implementación del modelo de Castalia 1.3 a Castalia 3.2 en la sección V, comentarios acerca del código fuente interesante de nuestra implementación en el simulador en la sección VI, exposición de resultados de simulación en la sección VII, y conclusiones y trabajo futuro en la sección VIII.

II. ESTADO DEL ARTE

Existen muchos trabajos relacionados con las Redes Espontáneas. A continuación, exponemos algunos de ellos.

Feeney et al. [13] explican la diferencia entre redes ad-hoc y redes espontáneas. Identifican 5 retos clave introducidos por el entorno de redes espontáneas. Uno de los principales puntos que marca una diferencia entre la red espontánea y redes fijas o móviles es que facilitan la integración de servicios y dispositivos, fijando nuevos

servicios y parámetros de configuración de dispositivos. Tienen que ser realizados sin la intervención del usuario o interferencia en la operación de la red. El mal funcionamiento o fallo de uno de los dispositivos o servicios no compromete la viabilidad de la comunidad. Cualquier recurso usado por la comunidad que funciona mal es automáticamente eliminado y el servicio desregistrado.

Latvakoski et al. [14] proponen una arquitectura de comunicaciones para redes espontáneas que integran comunicación de grupo espontánea a nivel de aplicación y redes ad-hoc al mismo tiempo.

Mani et al. [15] propusieron SCOPE: Un Prototipo para redes sociales espontáneas P2P. Proporciona aplicaciones de red social hechas a medida para casos de uso local. Por debajo de la capa de red, SCOPE sigue el modo ad-hoc 802.11 y no necesita infraestructura. SCOPE sigue el modelo jerárquico P2P. Algunos nodos con mayor capacidad de cómputo se convierten en super-nodos. Los super-nodos forman un overlay y proporcionan el sistema de gestión de datos distribuido para la red social P2P. Los nodos cliente se conectan con super-nodos y confían en ellos para compartir sus contenidos o acceder a la información compartida.

Un método para la unión de redes espontáneas se propone en [16]. Los autores presentan una propuesta para la unión implícita de redes espontáneas siguiendo un modo de movilidad de grupo. El protocolo de encaminamiento entre celdas evita cuellos de botella en los enlaces, y se atribuyen nodos relay a nodos que quieran comunicarse con un nodo en una celda diferente. Algunos protocolos de encaminamiento jerárquicos están basados en la elección de un cluster-head de celda (o nodo punto de referencia).

An Awareness Framework for Collaborative Spontaneous Networks: AWISPA, un estudio del conocimiento en entornos de aprendizaje colaborativos basado en redes inalámbricas espontáneas es llevado a cabo en [17]. Una red espontánea se crea cuando un grupo de estudiantes se aproximan y usan dispositivos de cómputo inalámbricos para llevar a cabo una actividad colaborativa. Para el proceso de evaluación, primero se comprobó si AWISPA cubre los requerimientos para soportar conocimiento en una aplicación síncrona colaborativa, y si una aplicación desarrollada basada en AWISPA cubre el conocimiento en una sesión real con dos grupos de gente colaborando entre ellos. En ambos grupos, hicieron un cuestionario de análisis y descubrieron que la aplicación se ajustaba a AWISPA y por tanto proporciona conocimiento.

Podemos encontrar algunos ejemplos actuales sobre Internet of Things (IoT) en el proyecto CeNSE de HP Labs, centrados en el despliegue de una red de sensores de área mundial para crear un "sistema nervioso central para la Tierra", o el proyecto "A Smarter Planet", una estrategia desarrollada por IBM que considera los sensores como pilares fundamentales en sistemas inteligentes de gestión del agua y ciudades inteligentes [27].

El estándar IPv6 sobre Redes Inalámbricas de Área Personal de Bajo consumo (6Low-PAN), definido por el Internet Engineering Task Force (IETF)[28], permite la

transmisión de paquetes IPv6 a través de redes computacionalmente restringidas.

En [29], los autores se centran en un reto específico: el modelo de conectividad actual entre la WSN e Internet, intentando responder si los nodos sensores deberían delegar todas las comunicaciones de Internet a un conjunto de sistemas de gestión central, o si deberían de convertirse en ciudadanos de primera clase de Internet implementando la pila entera de TCP/IP además de otros estándares como servicios web.

En [39], se utiliza J2ME [35][36] para desarrollar un prototipo de redes espontáneas basadas en la confianza.

Muchos protocolos de encaminamiento para MANETs tales como Destination-Sequenced Distance Vector (DSDV)[18], Dynamic Source Routing (DSR)[19], Ad-hoc On Demand Distance Vector (AODV)[20], y Temporally-Ordered Routing Algorithm (TORA)[21] podrían ser usados en redes espontáneas. Estos protocolos funcionan bajo el concepto de descubrimiento de ruta para localizar el receptor del paquete.

III. SIMULADOR CASTALIA 3.2

La herramienta utilizada para simular TrustedNet y obtener resultados es Castalia 3.2 [12]. Está basada en OMNET++.

Se ha elegido esta herramienta debido a que está orientada a la simulación de redes inalámbricas como la que queremos evaluar. Además, es útil para validar protocolos en condiciones reales de canal, antes de pasar a implementar una plataforma de sensores. Está diseñada para adaptar y extender protocolos, pudiendo implementar y validar los mismos con ella.

En principio, Castalia está orientada a la simulación de redes de sensores inalámbricos (WSN), redes de área corporal (BAN) y, en general, a redes de dispositivos con restricciones de energía (por ejemplo, nodos con batería que se agotará en un tiempo determinado).

Entre sus características destacan las siguientes: modelos de canal avanzados, radio (módulo para canales reales de radio de bajo consumo), sensado, derivas de reloj, MAC y encaminamiento.

Castalia tiene una estructura modular; en concreto, dispone de los módulos de radio, MAC, encaminamiento, etc, y es sencillo de configurar.

OMNET++, la herramienta en la que está basada, utiliza módulos y mensajes como base. Un módulo simple es la unidad básica de ejecución, y acepta mensajes de otros módulos o de él mismo, y de acuerdo con el mensaje ejecuta un fragmento de código. Ese código puede mantener un estado que es alterado cuando mensajes llegan y puede enviar nuevos mensajes. También hay módulos compuestos, que se componen de simples y/o otros módulos compuestos.

En Castalia, hay varios módulos, entre los cuales destacan el `node`, `application` (es el más comúnmente cambiado por el usuario, normalmente creando un nuevo módulo para implementar el nuevo algoritmo/protocolo), `MAC`, `routing`, `mobility manager`. Todos son accesibles para ser modificados por el usuario.

Se ha elegido la versión de Castalia 3.2, dado que es la más reciente, y es más completa que las anteriores.

Algunos bugs y errores han sido corregidos desde la versión 3.1 anterior, y se han añadido mejoras.

- Mejoras respecto a la versión 3.1: Cambios en `TunableMAC`, mejor estructura y limpiado de la `Virtual app/routing/MAC classes` y paquetes genéricos asociados, ejecutable para `gnuplot`, `--title` y `--scale flags` para `CastaliaPlot`, las imágenes (`png`, `gif`, `jpg`) producidas por `CastaliaPlot` (usado para crear gráficas automáticamente basándose en los resultados de `CastaliaResults`) contienen en su cabecera metadata el comando completo que las produce.
- Mejoras respecto a la versión 3.0: Introducción de `CastaliaPlot` (que como hemos visto, sirve para crear gráficas automáticamente basándose en los resultados de `CastaliaResults`), algunas características añadidas a `CastaliaResults` (usado para presentar automáticamente resultados de simulación), mejorado el `TunableMAC` con `CSMA persistent`, y corregidos algunos bugs.
- En Castalia 3.0: ejecutable con OMNET 4.x, y ofrece una forma completamente nueva de ejecutar simulaciones y visualizar resultados, un nuevo módulo de radio, implementa el `Baseline MAC` para BANs, propuesto por el IEEE 802.15 Task Group 6.

Castalia 3.2 ofrece soporte para construir protocolos propios, o aplicaciones definiendo clases abstractas adecuadas.

Todos sus módulos son altamente configurables a través de multitud de parámetros. Estos módulos son implementados usando el lenguaje NED de OMNET++. Con este lenguaje podemos, fácilmente, definir módulos, el nombre del módulo, parámetros e interfaz, y posibles estructuras de submódulo. Los archivos con extensión `.ned` contienen código en lenguaje NED.

La estructura de Castalia 3.2 también se refleja en la jerarquía de directorios del código fuente. Cada módulo se corresponde con un directorio que siempre contiene un archivo `.ned` que define el módulo. Si el módulo es compuesto, hay subdirectorios para definir esos submódulos. Si es un módulo simple, habrá además código C++ (archivos `.cc`, y `.h`). Normalmente, el usuario no cambiará estos archivos, a excepción de tratarse de nuevos módulos simples con nueva funcionalidad.

Será en los archivos `.cc` y `.h` donde principalmente se implementarán los nuevos protocolos.

Cada directorio de módulo contiene también un archivo `omnetpp.ini`, que incluye los parámetros de configuración que definen nuestro escenario de simulación. Tras cada simulación, la herramienta Castalia genera un archivo con los resultados, y otro con la traza `Castalia-Trace.txt` (que nos permite depurar el código).

IV. MODELO TRUSTEDNET

El modelo TrustedNet, que aquí presentamos, se ha desarrollado basándose en la idea de “confianza”. Esta idea fue presentada ya en PGP como una “red de confianza” [37][38]. La administración de este modelo de confianza estará basado en el criterio de “validez” y “confianza” [32][33][34]. Podemos establecer un servicio de administración de clave distribuida a través del uso de una “red de confianza”. Para formar la red, las claves públicas sólo necesitan obtenerse cuando sea necesario.

El modelo de red de TrustedNet se basa en uno previo presentado en [39], donde se incluye un ejemplo intuitivo que explica el concepto en el que se basa el modelo y cómo se intercambiarán datos entre nodos: “Cuando tenemos 3 personas y 2 de ellas conocen un secreto entonces sólo hay una forma de que la tercera sepa ese secreto que es cuando una de las dos personas confía en la tercera persona.”

Antes de definir este modelo, tendremos que tener en cuenta las propiedades de las redes espontáneas que enumeramos previamente en la sección I del presente artículo.

Para la formación de una red espontánea, necesitamos tener un protocolo en 2 fases, que llamamos TrustedNet, que permita el intercambio de una clave pública (PUBLICKEY) entre nodos. Cada nodo hará un handshake inicial con los nodos que están en su rango de transmisión, dado que, como ya sabemos, los nodos están equipados con transceptores de radio de alcance limitado.

En otras palabras, cada dispositivo tiene que descubrir sus vecinos, intercambiar su clave pública con sus vecinos, y contará con una tabla de vecinos local al dispositivo, la cual contendrá los nodos que están en su rango de transmisión (vecinos).

Si la red ad hoc cubre un área grande, la conectividad posterior puede ser obtenida mediante encaminamiento ad hoc.

Además, cada nodo debe tomar el rol de Autoridad Certificadora (AC). Así, todos los nodos formarán una MANET sin ninguna AC4 centralizada.

El protocolo para la creación de redes espontáneas que usa este modelo funciona de la siguiente forma:

Sean dos nodos, que llamamos nodo 1 y nodo 2. En el inicio, cada nodo difunde mensajes BROADCAST hacia los otros nodos. Un nodo 1 acepta otro nodo 2 como su vecino dependiendo de cuántos mensajes recibe el nodo 2 del nodo 1 (en nuestro caso fijamos un umbral del 95% de mensajes recibidos para aceptar un nodo como vecino). Cuando un nodo ha descubierto todos sus vecinos, envía su propia clave pública (PUBLICKEY) hacia todos ellos.

Cuando el mensaje PUBLICKEY llega a un vecino, éste debe responder con un mensaje PUBLICKEYRETURN.

Si, tras la recepción del PUBLICKEY, un vecino en concreto envía un mensaje de respuesta (PUBLICKEYRETURN) indicando que recibió bien la clave pública y que es correcta tras comprobar el hash, y el nodo 1 recibe esta respuesta entonces decimos que el nodo 1 confía en su vecino (trusted). Tras haber concluido este protocolo, ya se puede proceder al

encaminamiento ad hoc cuando sea necesario. En este caso, si sabemos que un nodo confía en otro nodo, puede enviar mensajes directamente hacia él. En el caso de que un nodo quiera enviar un mensaje hacia un nodo no-confiable tiene que hacerlo a través de un nodo de confianza.

A modo de resumen, el modelo sigue los siguientes pasos, como también se puede observar en la Figura 1:

1. Difusión de mensajes BROADCAST para descubrir los nodos vecinos.
2. Envío de la clave pública PUBLICKEY a todos los vecinos.
3. Recepción de mensajes PUBLICKEYRETURN originados por los vecinos.
4. Declaración de nodo vecino como confiable (trusted) o no-confiable (not trusted).

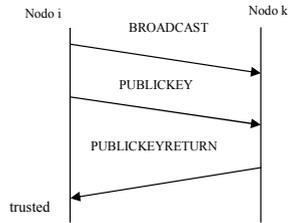


Fig. 1. Modelo TrustedNet

Cuando el vecino ha obtenido la clave pública, será considerada “válida” por el mismo si está seguro de que pertenece a quien la originó. Será considerada como “no válida” cuando no podemos saber a ciencia cierta si esa clave pertenece a quien dice pertenecer. Si confiamos en la clave (tras recibir el PUBLICKEYRETURN), la firmaremos con nuestra propia clave privada y consideramos el nodo como un “vecino de confianza”. Este tipo de vecinos crea, así, una red de confianza.

Dado que estamos considerando MANETS, en el caso de que un nuevo dispositivo sea presentado en la red y no tenga un par de claves, debe generarlas para llevar a cabo autenticación y para comunicarse con otros nodos. Si un nodo deja la red, la red mantiene los datos por si quiere volver a la misma en un futuro, pero en el momento en el que vuelva a ella, tiene que ser autenticado de nuevo.

Un nuevo nodo no tiene que obtener la clave pública de todos los otros nodos, en otras palabras, un nodo no tiene que difundir su información de autenticación a todos los otros nodos de la red. Los nuevos nodos pueden obtener esta información de una forma descentralizada y distribuida, a través de la red de confianza [37][38].

Para explicar más detenidamente todo este proceso, se pasa a exponer un ejemplo simple clarificador del funcionamiento de TrustedNet.

Suponemos que la red está formada por 3 nodos, y que inicialmente los nodos 1 y 2 conocen y confían el uno en el otro. Entonces, el nodo 2 confía en un tercer nodo, el nodo 3. Si el segundo nodo recibe una clave pública del

tercer nodo, y la firma con su clave privada consideramos que el nodo 3 al que pertenece esta clave es de confianza. Más tarde, si el primer nodo quiere obtener esa clave, puede ser obtenida del segundo nodo, y ya que el primer nodo confía en el segundo, 'válida' esta nueva clave firmándola con su clave privada. Si el tercer nodo no es de confianza, cualquier clave firmada por el tercer nodo no será considerada como clave de confianza. Además, el primer nodo nunca firmará la clave del tercer nodo aunque podría pasarla.

V. ADAPTACION A CASTALIA 3.2

El modelo TrustedNet fue implementado inicialmente en Castalia 1.3 para su simulación, pero posteriormente hemos realizado una implementación en Castalia 3.2 (la última versión de Castalia). El modelo implementado con ambas versiones de Castalia es el mismo.

Fundamentalmente, la parte de código implementado más importante se encuentra en los archivos `TrustedNet.h` y `TrustedNet.cc` donde está la mayor parte y lo principal de la implementación del modelo, `trustednet_DataPacket.msg` donde se define la estructura de los paquetes, `trustednet_Graph.h` que contiene la estructura del grafo que representará la red de confianza, `trustednet_Ident.h` donde se define la estructura del identificador del nodo, `trustednet_Key.h` conteniendo el código para manejar las claves pública y privada, `trustednet_Node.h` es la estructura del nodo, y `trustednet_SHA1.h` lo necesario para el uso del hash, además del `omnetpp.ini` donde figuran los parámetros de configuración, es decir, el escenario de simulación de la red de confianza.

VI. ACERCA DEL CODIGO FUENTE

En esta sección se procede a comentar las partes del código fuente más interesantes de la implementación de TrustedNet en Castalia 3.2. En concreto, el código que se pasa a comentar está contenido íntegramente en el archivo `TrustedNet.cc`.

- void TrustedNet::startup()**
 Es el método que se llama al iniciar la simulación para cada nodo de la red. En él, se inicializan los parámetros necesarios para la simulación de nuestro modelo, y de entre ellos, los más destacados son: `packetHeaderOverhead` (overhead de la cabecera del paquete), `constantDataPayload` (el payload del paquete), `random_startup_delay` (tiempo aleatorio de inicio para ese nodo, y salto al inicio del programa al expirar ese tiempo), `neighborTable.clear()` (inicializa la tabla de vecinos y la deja vacía), `totalSimTime` (tiempo total de simulación), `totalSNnodes` (número total de nodos de la red), `txInterval_perNode` (intervalo entre transmisiones del mensaje BROADCAST de los nodos), estructura `node` (el nodo) y su posición, `network.addNode(node)` (añade el nodo a la red).

- void TrustedNet::fromNetworkLayer(ApplicationPacket * rcvPacket, const char * source, double rssi, double lqi)**
 Este método incluye las acciones a realizar por el programa con la llegada de un nuevo paquete de la capa de red, donde `rcvPacket` es el paquete recibido y `source` el origen. En este método de TrustedNet, se extrae el número de secuencia del paquete, el destino, y se comprueba si el paquete va destinado a este nodo o si es broadcast. En caso afirmativo, se extrae el nodo origen del paquete, su destino, los datos, `rssi` (indicación de potencia de la señal recibida), tipo de mensaje, `ident` (identificador), y se llama al método `onReceiveMessage()` el cual realizará distintas acciones según el tipo de paquete recibido, como veremos más adelante.
- void TrustedNet::timerFiredCallback(int index)**
 Incluye las acciones a realizar para cada estado:
 - APP_NODE_STARTUP:** estado inicial que habilita el nodo, fija el `currentPowerLevel=0` (nivel de potencia 0), planifica el estado `APP_TIMER_1` para llamarlo después de haber transcurrido un tiempo `INTER_PACKET_SPACING` (tiempo entre paquetes, ajustable por programa, que en nuestro caso lo hemos ajustado a 0.07 que es aproximadamente el tiempo que un paquete necesita para llegar de un nodo a otro en su rango de transmisión).
 - APP_TIMER_1:** Acciones iniciales del nodo, captura el tiempo de simulación actual (`now`), y averigua qué nodo va a enviar el paquete BROADCAST en ese momento (`int currentNodeTx = ((int)floor(now.dbl() / txInterval_perNode))%totalSNnodes;`). Si este nodo es el que en este momento ha de transmitir ese mensaje BROADCAST (planificado para este instante), y el número de paquetes BROADCAST enviados por ese nodo es aún menor que el `SENT_PACKETS_PER_NODE_PER_LEVEL` (número máximo de paquetes BROADCAST a enviar por cada nodo, ajustable por programa, que en nuestro caso lo hemos ajustado a 5 paquetes), entonces ejecuta `send2NetworkDataPacket(BROADCAST_NETWORK_ADDRESS, serialNumber, BROADCASTING)` que envía el paquete de BROADCAST para descubrir sus vecinos, y `packetsSent++` que incrementa el número de paquetes enviados. Si no, y si el número de paquetes BROADCAST enviados por ese nodo es igual a `SENT_PACKETS_PER_NODE_PER_LEVEL`, lo que significa que el nodo ha enviado todos los paquetes BROADCAST que tenía que enviar, entonces incrementa en 1 el número de nodos que han enviado todos sus paquetes de BROADCAST (`nodesDoneSending++`), y pasa al estado `APP_TIMER_2`, de nuevo en un tiempo `INTER_PACKET_SPACING`.
 - En otro caso, sigue en `APP_TIMER_1`.

APP_TIMER_2: Si todos los nodos han enviado todos sus mensajes BROADCAST, se pasará al estado APP_TIMER_3 para fijar los vecinos que haya encontrado. Si no, sigue en este estado APP_TIMER_2.

APP_TIMER_3: fija el número de vecinos descubiertos, añade los vecinos a la tabla de vecinos del nodo, y pasa al estado APP_TIMER_4.

APP_TIMER_4: Planifica el envío del próximo PUBLICKEY y respuesta de los vecinos (PUBLICKEYRETURN uno detrás de otro), y salta al APP_TIMER_5.

APP_TIMER_5: Envío del PUBLICKEY a cada uno de los vecinos.

APP_TIMER_6: Envío del PUBLICKEYRETURN de cada vecino al nodo origen del PUBLICKEY.

NETWORK_2_APP_FULL_BUFFER: Si el buffer de red está lleno, emite un warning a la traza, comunicando esta circunstancia.

- void TrustedNet::send2NetworkDataPacket(trustednet_Ident destinationIdent, trustednet_Ident sourceIdent, Type type):

Envío de mensaje a la capa de red con parámetros ident (identificador) de origen y destino y tipo de paquete.

- void TrustedNet::send2NetworkDataPacket(const char *destID, double data, Type type):

Otra versión del mismo método con distintos parámetros. Envío de mensaje a la capa de red con parámetros id de destino, datos y tipo de paquete.

- void TrustedNet::onReceiveMessage(int sensorType, trustednet_Ident ident, string msgSender, double theData):

Si se recibió un paquete de tipo PUBLICKEY, captura el ident del origen de ese paquete, y planifica el envío del PUBLICKEYRETURN de forma que lo envíe un vecino detrás de otro y pasa a APP_TIMER_6 para que haga el envío.

Si se recibió un paquete de tipo PUBLICKEYRETURN, fija el vecino origen del paquete como de confianza (trust = true), y captura el tiempo final de simulación para que al recibir el último PUBLICKEYRETURN en respuesta al último PUBLICKEY se registre ese tiempo final.

Si se recibió un paquete de tipo BROADCASTING, actualiza la tabla de vecinos (con el identificador del emisor del BROADCAST).

- void TrustedNet::updateNeighborTable(int nodeID, int theSN): Actualiza la tabla de vecinos.

- void TrustedNet::finishSpecific(): Finaliza la simulación y transfiere los resultados al archivo de salida correspondiente; en concreto, los resultados que nos interesan son: el identificador del nodo con su posición, trusted (identificadores de todos sus nodos vecinos de confianza), not trusted (identificadores de

todos sus nodos vecinos no confiables), número de vecinos del nodo, número de nodos alcanzables, y los nodos alcanzables. Además, a modo de resumen final, se mostrará la media del número de vecinos, el máximo número de vecinos, la media del número de nodos alcanzables, el máximo número de nodos alcanzables y el tiempo final de TrustedNet.

VII. RESULTADOS

A continuación, se exponen los resultados obtenidos tras simulación con Castalia. En todas las simulaciones, variamos el número de nodos, y además determinados parámetros (distintos en cada simulación) de las distintas capas según el escenario que simulamos, y obtendremos como resultado el número de vecinos descubiertos y el tiempo que tarda TrustedNet.

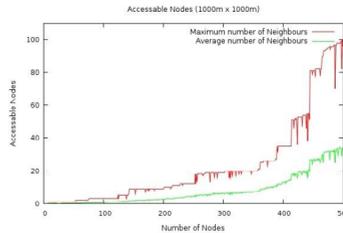


Fig. 2. Número de nodos accesibles, (máximo número de vecinos y número de vecinos promedio).

Como se comprueba en la Figura 2, teniendo en cuenta una red ideal con un umbral de descubrimiento de vecinos del 95% y un área de despliegue de 1000mx1000m, el número de vecinos descubiertos aumenta cuando el número de nodos se incrementa, lo que teóricamente ya se suponía, dado que manteniendo constante el área de despliegue, al incrementar el número de nodos hay más nodos cerca el uno del otro, esto es, hay más vecinos. Se comprueba también que ningún nodo llega a descubrir todos sus vecinos, dado el enorme tamaño del área y la pérdida de paquetes debido a colisiones. Además, el máximo número de vecinos es mayor que el número de vecinos promedio, y existe una sustancial diferencia entre ambos.

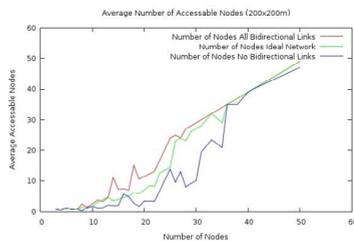


Fig. 3. Promedio del número de nodos accesibles, con parámetros red ideal, sin enlaces bidireccionales, y con todos los enlaces bidireccionales.

En este experimento se ha contemplado un área de 200mx200m, donde se despliegan todos los nodos.

La Figura 3 muestra la diferencia en número de vecinos descubiertos cuando usamos diferentes tipos de enlaces. Los enlaces bidireccionales fuerzan a los links a tener la misma cualidad en ambas direcciones. En el caso de links unidireccionales no hay correlación entre las dos direcciones y el fading es elegido independientemente. En realidad, hay siempre correlación entre links en una dirección y la opuesta. En una red ideal tenemos también enlaces bidireccionales (y la sigma del canal wireless está fijada a 0, lo que lleva a que todos los nodos a la misma distancia del emisor tengan la misma potencia de señal). Los enlaces bidireccionales presentan mejores prestaciones en cuanto a cantidad de vecinos descubiertos.

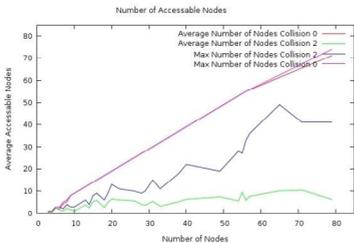


Fig. 4. Máximo y promedio del número de nodos accesibles, para modelos de colisión 0 y 2.

Con un área de 100mx100m, se observa en la Figura 4 que hay mucha diferencia en la cantidad de vecinos descubiertos entre los dos modelos de colisión 0 y 2, obteniendo mejores prestaciones cuando el modelo es el 0 (sin colisiones). Si hay presencia de colisiones (modelo 2), muchos mensajes enviados se pierden y ese es el motivo por el que hay menos vecinos descubiertos que cuando no hay colisiones.

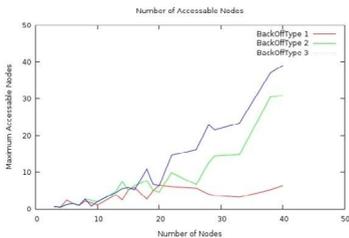


Fig. 5. Máximo número de nodos accesibles, con parámetro BackOffType 1, 2 y 3.

En cuanto a la variación del BackOffType, la Figura 5 muestra que con BackOffType 3 se obtienen mejores prestaciones que con BackOffType 2 y que con BackOffType 1. Para simular este comportamiento se ha elegido el modelo de colisión 2 (modelo de interferencia aditiva).

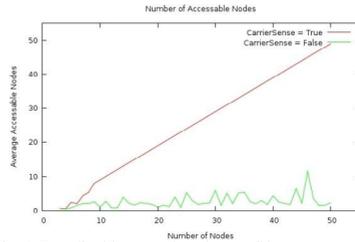


Fig. 6. Promedio del número de nodos accesibles, con parámetro CarrierSense true y false.

Como se observa en la Figura 6, cuando hay Carrier Sensing el número de vecinos descubiertos es mayor que cuando no hay Carrier Sensing.

Para la obtención de las 2 gráficas siguientes se ha utilizado Castalia 3.2, con el modelo de radio CC2420 (ZigBee). Además, se ha fijado un área de 10mx10m, desplegando los nodos en 1 clique (todos los nodos en el rango de transmisión de todos los demás).

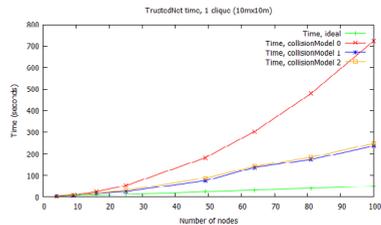


Fig. 7. Tiempo que tarda TrustedNet, para modelo ideal y real (con modelos de colisión 0, 1 y 2).

Con condiciones de radio ideales, se distribuyen los nodos en 1 clique.

En la Figura 7 se representan los tiempos que tarda TrustedNet con el modelo ideal (en el cual todos los vecinos emiten su PUBLICKEYRETURN a la vez) y el modelo real (en el cual solo los vecinos que en realidad hay responden uno tras otro) variando el modelo de colisión a 0, 1 y 2.

Destacar que el resultado para el modelo ideal es idéntico para los 3 modelos de colisión en cuanto al tiempo que tarda el modelo TrustedNet en finalizar, y es el que se observa en la Figura 7, obteniendo un tiempo que aumenta linealmente cuando el número de nodos se incrementa. Así, podemos concluir que el tiempo en finalizar TrustedNet en el caso ideal es de $O(N)$, donde N es el número de nodos que compone la red. Un resultado interesante es que para 100 nodos este tiempo es de 50 segundos.

En cuanto al modelo real, considerando modelos de colisión 0, 1 y 2, el resultado es siempre peor que en el modelo ideal, obteniendo un tiempo más alto con el

modelo de colisión 0, mientras que para los otros dos modos el resultado es similar en ambos.

El hecho de que sea más costoso el modelo de colisiones 0 es debido a que se encuentran más vecinos (dado que no hay colisiones) que en los otros dos modos, en los cuales el número de vecinos descubiertos es menor y por tanto el algoritmo finalizará antes. El tiempo para el modelo real y collisionModel 0 es aproximadamente de $O(N^2)$, mientras que para los otros dos modelos de colisión depende del número de vecinos descubiertos para cada N.

Además, existe otra ventaja al simular con Castalia 3.2, ya que el modelo de colisiones 1 (modelo simplista para colisiones) no se incluía en Castalia 1.3.

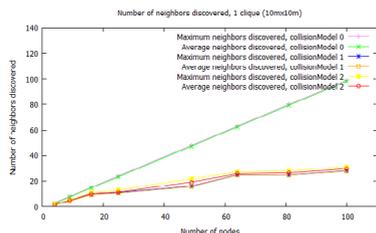


Fig. 8. Máximo y promedio del número de vecinos descubiertos por la fase de descubrimiento de vecinos de TrustedNet, para collisionModel 0, 1 y 2.

Bajo las mismas condiciones que en la simulación para obtener la Figura 7, esto es, con un área de $10m \times 10m$, y condiciones de radio ideales, se distribuyen los nodos en 1 clique.

En la Figura 8 se representan el máximo y promedio del número de vecinos descubiertos por el algoritmo de descubrimiento de vecinos que utiliza TrustedNet, variando el número de nodos de la red y el modelo de colisiones 0, 1 y 2.

Se observa que hay poca variación entre máximo y promedio para cada modelo de colisión, obteniendo mejores prestaciones cuando el modelo de colisión es el 0 (no hay colisiones), mientras que para los modelos de colisiones 1 (modelo simplista para colisiones) y 2 (interferencia aditiva) el resultado es similar en ambos. En los tres casos, el número de vecinos descubiertos aumenta cuando el número de nodos de la red se incrementa, aunque en los modelos 1 y 2 esta tendencia es más suave. Además, en 1 clique y con el modelo de colisiones 0, se puede observar que todos los nodos descubren casi todos sus vecinos, ya que el promedio del número de vecinos descubiertos se aproxima mucho al número real de vecinos de cada nodo, y que es $N-1$ (teóricamente se sabe que en 1 clique con N nodos, cada nodo tendrá $N-1$ vecinos).

VIII. CONCLUSIONES

En este artículo se ha expuesto el modelo TrustedNet para redes de confianza, algunas características importantes de Castalia 3.2, así como la implementación

del modelo con la versión más reciente del simulador Castalia 3.2, y comentarios acerca del código fuente de TrustedNet implementado en esa versión del simulador.

En este trabajo se ha demostrado también la posibilidad de implementar el modelo en la última versión de Castalia.

De los resultados obtenidos tras simulación podemos constatar que cuando el tamaño del área permanece constante el número de vecinos descubiertos aumenta cuando el número de nodos se incrementa, ningún nodo llega a descubrir todos sus vecinos, los enlaces bidireccionales presentan mejores prestaciones en cuanto a cantidad de vecinos descubiertos que los unidireccionales, se obtienen mejores prestaciones cuando el modelo de colisiones es el 0 (sin colisiones), mientras que con el modelo de colisiones 2 hay menos vecinos descubiertos debido a la mayor pérdida de paquetes por colisiones, un BackOffType 3 presenta mejores resultados que el 1 y 2, y cuando hay Carrier Sensing el número de vecinos descubiertos es mayor que cuando no hay. Considerando 1 clique, el resultado en cuanto al tiempo que tarda TrustedNet en finalizar en el modelo ideal es idéntico para los modelos de colisión 0, 1 y 2, obteniendo en todos ellos un tiempo $O(N)$. Este tiempo resulta ser de 50 segundos para una red de 100 nodos, el cual es un resultado razonable. Además, para el modelo real (cada vecino responde uno detrás de otro) el tiempo es mayor que en el ideal, siendo peor el resultado para el modelo de colisiones 0 (aproximadamente de $O(N^2)$) ya que hay mayor número de vecinos descubiertos que en los otros dos modelos de colisión, los cuales presentan resultados similares. Bajo las mismas condiciones de simulación, analizamos el algoritmo de descubrimiento de vecinos que utiliza TrustedNet, observando que hay poca variación entre máximo y promedio para cada modelo de colisión, y que se obtienen mejores prestaciones con el modelo de colisión 0, mientras que para los modelos 1 y 2 el resultado es similar en ambos; en los 3 casos el número de vecinos descubiertos aumenta cuando el número de nodos se incrementa aunque en los modelos 1 y 2 esa tendencia es más suave. Además, en 1 clique y con el modelo de colisiones 0, se puede observar que todos los nodos descubren casi todos sus vecinos, ya que el promedio del número de vecinos descubiertos se aproxima mucho al número real de vecinos ($N-1$) de cada nodo.

Como opción de trabajo futuro, hemos pensado en paralelizar ciertas partes del código del simulador con MPI para dotar de mayor escalabilidad al sistema, además de proponer y evaluar varios protocolos para el descubrimiento de vecinos en redes inalámbricas ad hoc, así como compararlos mediante simulación con Castalia 3.2 con otros protocolos interesantes de la literatura.

AGRADECIMIENTOS

El presente trabajo ha sido financiado parcialmente mediante el "Programa Estatal de Investigación, Desarrollo e Innovación Orientada a Retos de la Sociedad, Proyecto I+D+I TEC2014-52690-R".

REFERENCIAS

- [1] Jennifer L. Welch Alejandro Cornejo, Saira Viqar. Reliable neighbor discovery for mobile ad hoc network. In ACM, editor, DIALM-POMC'10, Cambridge, MA, USA., September 2010.
- [2] Anthony Bussan Eric Fleury Elyes Ben Hamida, Guillaume Chelius. Neighbor discovery in multi-hop wireless networks: evaluation and dimensioning with interference considerations. *Discrete Mathematics and Theoretical Computer Science*, 10(2):87-114, May 2008.
- [3] Xiaofeng Gao Guihai Chen Wei Wang Guobao Sun, Fan Wu. Time-efficient protocol for neighbor discovery in wireless ad hoc networks. *IEEE*, 2013.
- [4] Dongning Guo Jun Luo. Neighbor discovery in wireless ad hoc networks based on group testing. 2008.
- [5] Carla-Fabiana Chiasserini Panagiotis Papadimitratos Marco Fiore, Claudio Casetti. Discovery and verification of neighbor positions in mobile ad hoc networks. 19
- [6] Steven A. Borbosh Michael J. McGlynn. Birthday protocol for low energy deployment and flexible neighbor discovery in ad hoc wireless networks.
- [7] Jose Alex Mathew Muhammed Irfan S, Shameer Ali. Protocol design for neighbor discovery in ad-hoc network. *International Journal of Electronic and Electrical Engineering*, 7(9):915-922, 2014.
- [8] Patrick Schaller Pascal Lafoucade David Basin Srdjan Capkun Jean-Pierre Hubaux Panos Papadimitratos, Marcin Poturlski. Secure neighborhood discovery: A fundamental element for mobile ad hoc networking.
- [9] H. Chenji R. Stoleru, H. Wu. Secure neighbor discovery in mobile ad-hoc networks. pages 35-42, 2011.
- [10] Dennis Goeckel Don Towsley Sudarshan Vasudevan, Michal Adler. Efficient algorithms for neighbor discovery in wireless networks. *IEEE/ACM Transactions on Networking*, 2012.
- [11] David Simplot-Ryl Xu Li, Nathalie Mitton. Mobility prediction based neighborhood discovery in mobile ad hoc networks. *NETWORKING 2011*, 1(6640):241-253, 2011.
- [12] Castalia. Wireless Sensor Network Simulator. <https://castalia.forge.nicta.com.au/index.php/en/>.
- [13] Feeney LM, Ahlgren B, Westerlund A. Spontaneous networking: an application-oriented approach to ad hoc networking. *IEEE Commun Mag* 2001, 39(6):176-181.
- [14] Latvakoski J, Pakkala D, Pääkkönen P: A communication architecture for spontaneous systems. *IEEE Wirel Commun* 2004, 11(3):36-42.
- [15] Mani M, Nguyen A-M, Crespi N: SCOPE: a prototype for spontaneous P2P social networking. In Proceedings of 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops). 2010:220-225.
- [16] Legendre F, de Amorim MD, Fdida S: Implicit merging of overlapping spontaneous networks. In Proceedings of Vehicular Technology Conference, Volume 4. 2004:3050-3054.
- [17] Zarate Silva VH, De Cruz Salgado EI, Ramos Quintana F: AWISPA: an awareness framework for collaborative spontaneous networks. In 36th Annual Frontiers in Education Conference. 2006:27-31.
- [18] Perkins CE, Bhagwat P: Highly dynamic destination sequenced distancevector routing (DSDV) for mobile computers. In Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM'94. 1994:234-244.
- [19] Johnson DB, Maltz DA, Broch J: DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. Boston, MA: Ad Hoc Networking (Addison-Wesley Longman Publishing; 2001.
- [20] Perkins C, Belding-Royer E, Das S: Ad hoc on-demand distance vector (AODV) routing. RFC 3561. 2003.
- [21] Park V, Corson MS: IETF MANET Internet Draft "draft-ietf-MANET-tora-spec03.txt", November 2000. 2012. <http://tools.ietf.org/html/draft-ietf-manet-tora-spec-03> Accessed March.
- [22] Viana AC, De Amorim MD, Fdida S, de Rezende JF: Self-organization in spontaneous networks: the approach of DHT-based routing protocols. *Ad Hoc Networks* 2005, 3(5):589-606.
- [23] Lacuesta R, Peñalver L: IP addresses configuration in spontaneous networks. In Proceedings of the 9th WSEAS International Conference on Computers. Athens, Greece: 2005:1-6.
- [24] Alvarez-Hamelin JI, Viana AC, de Amorim MDias: Architectural considerations for a self-configuring routing scheme for spontaneous networks. *Technical Report 1*. 2005.
- [25] Lacuesta R, Peñalver L: Automatic configuration of ad-hoc networks: establishing unique IP link-local addresses. In Proceedings of the International Conference on Emerging Security Information, Systems and Technologies (SECURWARE'07). Valencia, Spain: 2007:157-162.
- [26] Foulkes EF: Social network therapies and society: an overview. *Contemp Fam Therapy* 1985, 3(4):316-320.
- [27] IBM: A Smarter Planet. 2012. <http://www.ibm.com/smarterplanet>.
- [28] Montenegro G, Kushalnagar N, Hui J, Culler D: RFC 4944: Transmission of IPv6 Packets over IEEE 802.15.4 Networks. 2007.
- [29] Alcaraz C, Najera P, Lopez J, Roman R: Wireless Sensor Networks and the Internet of Things: Do We Need a Complete Integration? 1st International workshop on the security of The internet of Things (SecIoT). tokyo (Japan); 2010. http://www.nics.uma.es/seciot10/files/pdf/alcaraz_seciot10_paper.pdf. Accessed January 2012 1er International Workshop on the Security of The Internet of Things (SecIoT 2010).
- [30] S. Corson and J. Macker: Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations. Technical report, RFC2501, 1999
- [31] B. Smith, "An approach to graphs of linear forms (Unpublished work style)."
- [32] Frank Stajano and Ross Anderson. The resurrecting duckling security issues for ad-hoc wireless networks. In Security Protocols, 7th International Workshop Proceedings, Lecture notes in Computer Science. Springer-Verlag Berlin Heidelberg, 1999.
- [33] Dirk Balfanz, D. K. Smetters, Paul Stewart and H. ChiWong. "Talking to strangers: Authentication in ad-hoc/wireless networks." In n Symposium on Network and Distributed Systems Security (NDSS '02), San Diego, California, 3333 Coyote Hill Road Palo Alto, CA 94304, February 2002.
- [34] Srdjan Capkun, Jean-Pierre Hubaux and Levente Buttyán. Mobility Helps Security in Ad-hoc Networks. Junio 2003
- [35] Alonso Álvarez García José Ángel Morales Grela. Anaya Multimedia. J2ME. Guía Práctica para usuarios. J2ME. 2002.
- [36] Sergio Gálvez Rojas. Lucas Ortega Diaz. Java a tope: J2ME (Java 2 Micro Edition). Universidad de Málaga.
- [37] <http://www.pgpi.org/>
- [38] Jean Pierre Hubaux, Levente Buttyán, Srdan Capkun, "The Quest for Security in Mobile Ad-hoc Networks" Proceedings of the ACM Symposium on Mobile Ad-hoc Networking and Computing (MobiHOC 2001)
- [39] Raquel Lacuesta, Lourdes Peñalver, "Trust on a World of equals" Proceedings of the International Conference on Networking and Services (ICNS 2006), IEEE Computer Society ISBN-0-7695-2622-5, 2006
- [40] Raquel Lacuesta, Guillermo Palacios-Navarro, Carlos Cetina, Lourdes Peñalver, Jaime Lloret, "Internet of Things: where to be is to trust" EURASIP Journal on Wireless Communications and Networking 2012, 2012:203

Evaluación del Rendimiento de la Difusión de Mensajes Utilizando Parada Forzada en Redes Oportunistas

Jorge Herrera-Tapia, Andrés Tomás, Enrique Hernández-Orallo, Pietro Manzoni, Carlos Tavares Calafate, Juan Carlos Cano ¹

Resumen—El rendimiento de las redes móviles oportunistas depende en gran medida de la duración del contacto. Si el contacto tiene una duración inferior a los tiempos de transmisión requeridos, algunos mensajes no serán entregados y el esquema de difusión general se verá seriamente afectado. En este trabajo se propone un nuevo método de difusión, llamado parada forzada (Forced-Stop), basado en el control de la movilidad del nodo para garantizar la transferencia de mensajes completos. Utilizando el simulador ONE y trazas de movilidad reales, comparamos nuestra propuesta con la difusión epidémica clásica. Mostramos que la parada forzada permite mejorar el rendimiento de entrega de mensajes, aumentar la probabilidad de entrega hasta un 30 %, y reducir la latencia de entrega hasta un 40 %, con un impacto mínimo la utilización del buffer y la retransmisión de mensajes. Estos resultados pueden ser relevantes para el diseño de aplicaciones para redes oportunistas, en las que paradas de corta duración pueden favorecer la correcta entrega de datos.

I. INTRODUCCIÓN

Las redes inalámbricas oportunistas punto-a-punto (también conocidas como redes ad-hoc) [1, 2] han sido estudiadas por algunos autores [3] como una subclase de la Redes Tolerantes a Retardo (DTN) [4]. Este modelo es una alternativa a considerar en entornos donde la infraestructura inalámbrica se ha vuelto ineficaz debido a la saturación de peticiones de conexión, o cuando no hay infraestructura de transmisión de datos disponible. Vahdat et al. [5] se refiere a este tipo de redes como redes parcialmente conectadas (debido a la intermitencia de los contactos) y realiza un profundo análisis sobre el encaminamiento de mensajes en este tipo de redes.

Los protocolos de encaminamiento para entornos de comunicación oportunística deben ser capaces de almacenar, transportar y reenviar la información entre dispositivos móviles. Podemos encontrar una amplia caracterización de estos protocolos en diferentes trabajos de investigación [6–8]. El protocolo epidémico [5] pertenece a esta familia de protocolos y su funcionamiento está basado en la difusión indiscriminada de los mensajes.

En el modelo epidémico, el tiempo de contacto entre los nodos es un factor clave en la difusión de mensajes. Si el tiempo de contacto es demasiado corto, no habrá tiempo suficiente para recuperar todos los mensajes pendientes. Sin embargo, si el tiempo de

contacto entre los nodos es lo suficientemente grande, el mensaje será transferido de un nodo a otro, contribuyendo al éxito de la difusión de información entre la mayoría de los nodos en la red.

Con el objetivo de evaluar el rendimiento de los protocolos para la propagación de mensajes en redes oportunistas, utilizamos el simulador de redes inalámbricas ONE (Opportunistic Network Environment) [9]. Este simulador fue diseñado y construido específicamente para evaluar protocolos y aplicaciones DTN; el cual se centra en la capa de red, y no tiene en cuenta las capas inferiores, tales como acceso al medio o física. En nuestro análisis hemos utilizado una traza de movimientos realista obtenida durante quince días mediante los teléfonos inteligentes de los alumnos de la Universidad Nacional Chengchi (NC-CU) [10]. Los patrones de generación de mensajes (frecuencia y tamaño) están basados en estadísticas proporcionadas por las aplicaciones de redes sociales [11].

El contenido de este artículo es el siguiente: primero se presenta en la sección II una visión general de los artículos que abordan las redes oportunistas y difusión de mensajes. La descripción de nuestra propuesta de difusión, experimentos y datos de evaluación se presentan en las secciones III y IV, respectivamente. Por último, en la Sección V, se presentan nuestras conclusiones y posibles ampliaciones de este trabajo.

II. TRABAJOS RELACIONADOS

En las redes ad-hoc oportunistas es muy común que los contactos entre pares de dispositivos móviles sean intermitentes y de corta duración. Este modelo es aplicable a entornos en los que no hay una topología definida o infraestructura inalámbrica para la transmisión de datos. Hay varios trabajos de investigación sobre diseminación de mensajes en redes oportunistas sociales. Los autores de [12] examinan un sistema cooperativo de difusión, en base a la utilidad de los datos, que se define en función de las relaciones sociales entre los usuarios. Los autores analizaron y validaron el rendimiento de este sistema a través de un modelo matemático, caracterizando el proceso de difusión de mensajes. Además, analizaron el comportamiento del sistema con respecto a parámetros clave tales como la definición de la función de utilidad de datos, la carga inicial de datos en los nodos y el número de usuarios en el sistema.

En [13] se propone un modelo analítico para estu-

¹Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, e-mail: jorherta@doctor.upv.es, antodo@upv.es, {pmanzoni, ehernandez, calafate, jucano}@disca.upv.es

diar la difusión epidémica en redes sociales móviles. Este modelo está basado en reglas relativas al comportamiento del usuario, especialmente cuando sus intereses cambian según el tipo de información, pudiendo tener un impacto considerable en el proceso de difusión. A continuación, el modelo se desarrolla a través de ecuaciones diferenciales ordinarias, teniendo en cuenta que la información sólo puede ser enviada desde un nodo a otro cuando se encuentran y están socialmente conectados. Los autores demuestran la exactitud de este modelo se demuestra mediante numerosas simulaciones.

A diferencia de las propuestas antes mencionadas, otros autores analizan la difusión de mensajes desde la perspectiva del tiempo de contacto mediante diferentes modelos matemáticos. Los autores de [14, 15] presentan un estudio analítico que describe el funcionamiento del protocolo epidémico, con el argumento de que las redes móviles conectadas de manera intermitente pueden ser modeladas como cadenas de Markov. Proponen nuevos modelos para la propagación epidémica calculando la probabilidad de distribución a partir de los parámetros típicos de las redes oportunistas, tales como tamaño del mensaje, retardo máximo tolerado y duración de los enlaces. Estos trabajos muestran que, dados un retardo máximo y una determinada movilidad, la probabilidad de entrega de los mensajes esta determinado por el tamaño de los mensajes.

Finalmente, hay otras propuestas como [16–19], que evalúan la difusión epidémica de un mensaje centrado en la movilidad de los nodos. En estos trabajos, los autores estudian la relación entre factores tales como la velocidad, modelo de movilidad y la densidad de nodos.

Los estudios antes mencionados analizan el rendimiento del protocolo epidémico. Sin embargo, estos trabajos no consideran la relación entre el tiempo de duración de contacto y el tamaño del mensaje. En el presente artículo nos centramos en este aspecto evaluando el impacto de las paradas, tiempo de permanencia y tamaño de buffer sobre el rendimiento de la difusión de mensajes.

III. ANÁLISIS E IMPLEMENTACIÓN DE LA DIFUSIÓN DE MENSAJES

En esta sección se detalla el enfoque propuesto para la difusión de los mensajes en redes oportunistas y se describe las modificaciones necesarias para su implementación en el simulador ONE.

A. Esquema de difusión de mensajes

El fundamento de la mensajería basada en contactos es establecer una comunicación de corto alcance directamente entre dispositivos móviles y almacenar los mensajes en estos dispositivos con el fin de lograr su amplia difusión, los mensajes no son enviados o almacenados a través de una estructura de servidores. La distribución de mensajes se basa en la difusión epidémica, un concepto similar a la propagación de enfermedades infecciosas. Básicamente,

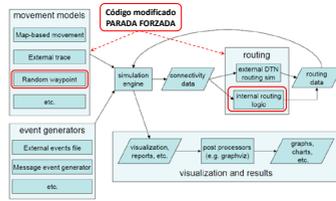


Fig. 1: Modificaciones realizadas al simulador ONE. (La figura está basada en la original de [9]).

cuando un nodo infectado (es decir, un nodo que tiene un mensaje) contacta otro nodo, este es infectado mediante la transmisión del mensaje. El encaminamiento epidémico obtiene un mínimo retardo en la entrega de mensajes a expensas de una mayor utilización de almacenamiento local e incremento del número de transmisiones.

El escenario de difusión funciona de la siguiente manera: los diferentes dispositivos móviles tienen instalada una aplicación de mensajería que notifica y muestra al usuario mensajes de los grupos suscritos. La aplicación debe ser cooperativa, es decir, debe almacenar los mensajes recibidos y difundirlos a otros nodos cercanos. Cada nodo tiene una memoria intermedia limitada (conocida también como buffer) para almacenar los mensajes recibidos de los otros nodos. Cuando dos nodos establecen una conexión intercambian los mensajes almacenados en sus memorias intermedias y comprueban si alguno de los nuevos mensajes es adecuado para el usuario.

El comportamiento de la aplicación cuando los nodos entran contacto entre sí puede tener dos enfoques distintos:

- Los dispositivos continúan en movimiento. En este caso, la transmisión del mensaje depende de la duración del contacto. Si la duración de contacto es menor que el tiempo de transmisión del mensaje, la transmisión falla. Por lo tanto, debemos asumir que algunos contactos no completarán la transmisión del mensaje.
- Los dispositivos dejan de moverse cuando necesitan intercambiar información. En este caso, los propietarios de los dispositivos móviles controlan este intercambio al detener y esperarse hasta que se complete la transmisión del mensaje. Esta modificación del protocolo epidémico garantiza la entrega del mensaje y la denominamos *parada forzada* (Forced-Stop).

B. Modificaciones al Simulador ONE

El simulador ONE está diseñado específicamente para evaluar las prestaciones de los protocolos de difusión basada en contactos. Entre sus principales características están seis principales protocolos de encaminamiento DTN y algunas de sus variantes: Epi-

démico, Spray and Wait, Prophet, First Contact, Direct Delivery, y Maxprop. También proporciona algunos modelos de movilidad relevantes, tales como Random Walk, Random Way Point, Lineal, y de Malla. Además, permite el uso de trazas de movimiento externas obtenidas en condiciones reales.

La figura 1 muestra los módulos de ONE modificados para implementar el protocolo epidémico con parada forzada. Las modificaciones principales están en las clases Java *ActiveRouter*, *DTNHost* y *Connection*. La velocidad de los nodos se configura a $0m/s$ mientras tienen alguna transmisión activa. En cuanto finaliza la transmisión se reanuda el movimiento con misma trayectoria y velocidad.

El generador de un mensajes en la versión original de ONE (clase *MessageEventGenerator*) inyecta un nuevo mensaje usando un intervalo de tiempo aleatorio. Este tiempo aleatorio se distribuye de manera uniforme a partir de un rango configurado en los parámetros de simulación. Con el fin de conseguir una simulación más cercana al comportamiento real, hemos modificado este módulo para generar mensajes utilizando un modelo de Poisson, con una distribución aleatoria exponencial.

Por último, aunque el simulador ONE genera una gran variedad de informes, no existía un mecanismo para obtener la ocupación de la memoria intermedia (buffer). Hemos añadido una nueva clase de informe que emite la ocupación media y máxima del buffer de todos los nodos para cada paso de la simulación. También calcula la ocupación máxima de memoria intermedia durante toda la simulación.

IV. EVALUACIÓN DE PRESTACIONES

El objetivo de esta sección es evaluar la difusión mediante la probabilidad y el retardo de entrega de los mensajes variando los métodos de difusión, tamaño del buffer, tamaño de los mensajes y tiempo de vida (TTL).

A. Descripción de los experimentos

Los experimentos se realizaron con el simulador ONE con las modificaciones descritas en la sección III-B, usando trazas reales pertenecientes a un experimento en el campus de la Universidad NCCU [10]. Las trazas fueron tomadas usando una aplicación Android, instalada en los teléfonos inteligentes de los estudiantes de Universidad Nacional de Chengchi. Participaron un total de 115 estudiantes en el experimento. Los datos pertenecientes al uso de GPS, aplicaciones, conexiones WiFi y Bluetooth fueron grabadas por un periodo de dos semanas, con una resolución de un segundo, y la información de posicionamiento está expresada en metros. La Figura 2 muestra una imagen del simulador ONE ejecutándose con la respectiva información geográfica.

Un aspecto importante de estos experimentos es la generación de carga de trabajo. El patrón de generación de mensajes elegido está relacionado con aplicaciones de redes sociales tipo WhatsApp o Facebook. Consideramos una aplicación de mensajería

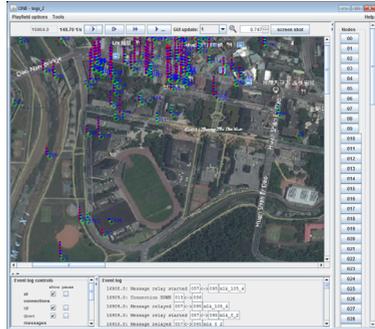


Fig. 2: Simulador ONE ejecutándose con las trazas NCCU.

TABLA I: Principales parámetros de simulación.

Parámetro	Valores
Tamaño del buffer	50 MB, 100 MB, 200 MB, 1 GB
Enrutamiento	epidémico, parada forzada
Tiempo máximo (TTL)	12 , 24 horas

multimedia típica donde cada usuario genera mensajes de diferentes tamaños y los mensajes cortos son mucho más comunes que los grandes. Definimos tres tamaños para los mensajes: un mensaje de texto corto (1 kB) cada hora, una imagen o fotografía (1 MB) cada 18 horas, y un video de corta duración (10 MB) cada 96 horas. Estas frecuencias están basadas en [11], mientras que los tamaños son aproximaciones del contenido típico producido por los teléfonos móviles actuales. Con el fin de obtener un modelo realista del comportamiento del usuario, el intervalo entre los mensajes se genera utilizando una función de Poisson independientemente para cada usuario y tipo de mensaje. Hay que tener en cuenta que esta carga de trabajo no es la misma utilizada en [10], por lo que los resultados presentados aquí pueden diferir de los originales.

El rango (r) de comunicación se estableció en 7.5 m con un ancho de banda de $Bw = 2.1Mb/s$ y una resolución de simulación de $T_s = 0.1s$. Estos valores fueron seleccionados en base a las especificaciones de Bluetooth 2.0, Class 2. Aunque el máximo rango de cobertura de Bluetooth es de 10 m, hemos asumimos una cierta interferencia y por lo tanto reducimos el rango de transmisión. Finalmente, el tiempo de vida de los mensajes (TTL Time To Live) fue configurado a 12 y 24 horas y el tamaño del buffer entre 50 MB y 1 GB. Los parámetros de las diferentes simulaciones están resumidos en la Tabla I.

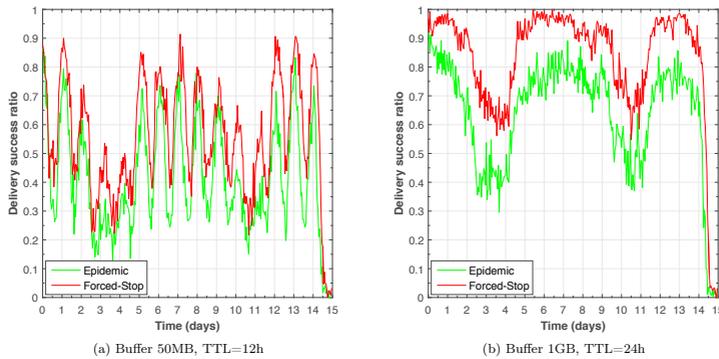


Fig. 3: Probabilidad de entrega por hora.

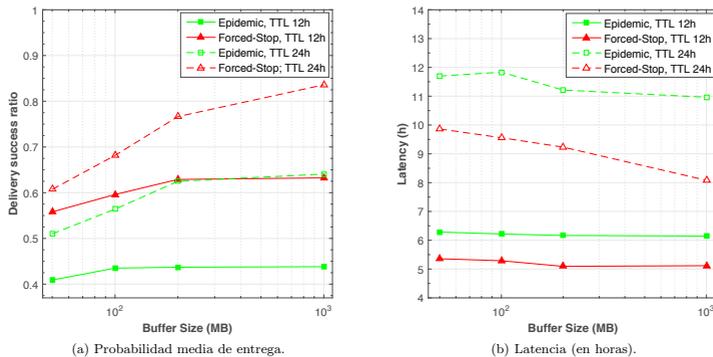


Fig. 4: Probabilidad de entrega y latencia en función del tamaño del buffer (eje x en escala logarítmica)

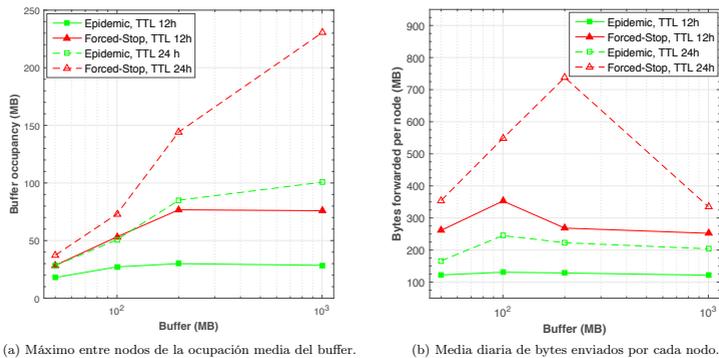
B. Evaluación

En esta sección se estudia el rendimiento de la difusión principalmente mediante dos medidas: el *ratio de éxito en la entrega de mensajes*, que es el número de mensajes que llegan a su destino dividido el número total de mensajes generados, y la *latencia*, que es el tiempo medio que tarda un mensaje en alcanzar su destino desde su creación. También son analizados otros factores importantes como la cantidad de almacenamiento usado y la cantidad de información reenviada.

Aunque hay varios protocolos alternativos para la difusión de mensajes aparte del epidémico (como por ejemplo PROPHET, Spray and Wait, etc.), comparamos el método propuesto solamente contra el enruta-

miento epidémico, porque esta demostrado que este protocolo obtiene la mínima latencia y el mayor ratio de éxito en la entrega de mensajes. Usando las mismas trazas y simulador, los autores de [10] también evaluaron los protocolos PROPHET y Spray and Wait, demostrando que la difusión epidémica ofrece los mejores resultados, a cambio de un mayor consumo de recursos en la red.

La figura 3 muestra el porcentaje de entrega de mensajes en intervalos de 1 hora durante las simulaciones. Este ratio fue obtenido calculando el número de mensajes generados en una determinada hora h : $msj_s[h]$, y el número de esos mensajes que llegaron a su destino $msj_r[h]$, por lo tanto el ratio de entrega por hora es $msj_r[h]/msj_s[h]$. Por limitaciones de



(a) Máximo entre nodos de la ocupación media del buffer. (b) Media diaria de bytes enviados por cada nodo.

Fig. 5: Sobrecarga de los protocolos en función del tamaño del buffer (eje x en escala logarítmica).

espacio, solamente incluimos dos gráficos representativos de los valores extremos, uno para el buffer de tamaño más pequeño (50 MB) con un TTL de 12 horas y otro para el tamaño más grande (1 GB) con un TTL de 24 horas.

En estas gráficas se observa que el ratio de éxito de entrega está relacionado con las actividades de los usuarios; así, ejemplo durante la noche la movilidad es reducida y consecuentemente la probabilidad de entrega también se reduce. En las horas del día, la movilidad se recupera y la probabilidad de entrega sube a los niveles anteriores. Cuando incrementamos el tamaño del buffer y el TTL a 24 horas, los resultados son diferentes y las actividades diarias no son evidentes. Pero podemos diferenciar claramente las actividades semanales, por ejemplo los días 3-4 y 10-11 son fines de semana donde la difusión de mensajes se reduce notablemente.

La figura 4a muestra la probabilidad media de entrega de los mensajes en función del tamaño de buffer para las cuatro combinaciones de TTLs y esquemas de difusión simulados. En este gráfico se observa claramente que a mayor buffer o TTL, más mensajes son almacenados en cada nodo mejorando la probabilidad de entrega. El método con parada forzada mejora la probabilidad de entrega en aproximadamente en un 30% con respecto del protocolo epidémico e incluso más con un TTL de 24 horas. Un aspecto interesante es que el tamaño del buffer no es determinante después de cierto valor suficientemente grande como para almacenar casi todos los mensajes generados.

Con respecto a la latencia, la figura 4b muestra el tiempo promedio de entrega de los mensajes en base al tamaño del buffer para los métodos de disseminación y TTLs. En general, la técnica de parada forzada se reduce el tiempo de latencia en comparación con el protocolo epidémico, disminuyendo alrededor de

20% o 30% para TTLs de 12 y 24 horas. El impacto del tamaño del buffer aumenta al utilizar parada forzada y un TTL de 24 horas haciendo que muchos más mensajes estén circulando por la red. Al aumentar el TTL se incrementa la probabilidad de entrega pero también aumenta la latencia, debido a que la capacidad de entrega de mensajes está limitada por la movilidad de los nodos.

Para determinar la sobrecarga de los protocolos medimos el máximo de la ocupación media del buffer para cada nodo. La figura 5a muestra este máximo para los diferentes valores de TTL y métodos de disseminación utilizados. Como la ocupación del buffer es mayor con parada forzada porque más mensajes permanecen vivos en la red, haciendo que la probabilidad de entrega sea mayor.

Otra métrica relacionada con la sobrecarga introducida por el protocolo es la cantidad de información transmitida por cada nodo. Al igual que en el análisis anterior, los resultados en la figura 5b están agrupados por protocolo y TTL. Este valor se ha obtenido dividiendo la suma de la longitud de todas las transmisiones entre el número de nodos y días de simulación. Al aumentar el TTL y el tamaño del buffer la cantidad de mensajes retransmitidos aumenta considerablemente, pero para el tamaño más grande de buffer baja a los valores iniciales. Este efecto es debido a que los mensajes se retransmiten solamente si no están en el buffer de destino. Si el buffer es pequeño, no hay muchos mensajes vivos en la red y no hay muchas retransmisiones. Y si el buffer es suficientemente grande los nodos conservan casi todos los mensajes anteriores por lo que no hace falta transmitirlos otra vez.

V. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo presentamos una variación de la difusión epidémica, llamada parada forzada (Forced-

Stop), basada en el control de la movilidad de los nodos para lograr la transferencia completa de mensajes en redes oportunistas. Utilizando el simulador ONE y un modelo de movimiento basado en trazas reales, comparamos nuestra propuesta con la difusión epidémica clásica.

Las simulaciones muestran que la estrategia propuesta mejora el rendimiento de todo el proceso de difusión, aumentando la probabilidad y reduciendo los tiempos de entrega a expensas de una mayor ocupación del buffer y retransmisión de datos. Sin embargo, esta retransmisión se reduce con un buffer suficientemente grande como para almacenar muchos de los mensajes activos en la red. Como posibles continuaciones de este trabajo sería interesante investigar en profundidad diferentes estrategias para gestionar el buffer, y comprobar cuanto tiempo es necesario detener el movimiento para obtener mejoras significativas en la difusión.

Estos resultados pueden ser una indicación relevante para los diseñadores de aplicaciones de red oportunistas que podrían integrar en sus productos notificaciones al usuario sobre la necesidad de esperar a finalizar las transmisiones con el fin de aumentar la entrega de datos en general. Además, dada la capacidad del hardware actual, es factible dimensionar el buffer de almacenamiento local para evitar retransmisiones y reducir el consumo de ancho de banda y energía.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por el *Ministerio de Economía y Competitividad, Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, Proyectos I+D+I 2014*, España, bajo el proyecto TEC2014-52690-R, la *Generalitat Valenciana*, España, proyecto AICO/2015/108 y la Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación del Ecuador SENESCYT, y la Universidad Laica Eloy Alfaro de Manabí, Ecuador.

REFERENCIAS

- [1] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic Networking: Data Forwarding in Disconnected Mobile Ad Hoc Networks," *IEEE Communications Magazine*, pp. 134–141, 2006.
- [2] S. Ferretti, "Shaping opportunistic networks," *Computer Communications*, vol. 36, pp. 481–503, 2013.
- [3] J. Niu, J. Guo, Q. Cai, N. Sadeh, and S. Guo, "Predict and Spread: an Efficient Routing Algorithm for Opportunistic Networking," *Wireless Communications and Networking Conference (WCNC), 2011 IEEE*, pp. 498–503, 2011.
- [4] G. S. Thakur, U. Kumar, A. Helmy, and W.-J. Hsu, "On the efficacy of mobility modeling for DTN evaluation: Analysis of encounter statistics and spatio-temporal preferences," *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pp. 510–515, 2011.
- [5] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," *Technical report number CS-200006, Duke University*, pp. 1–14, 2000.
- [6] S. Tornell, C. Calafate, J.-C. Cano, and P. Manzoni, "DTN Protocols for Vehicular Networks: an Application Oriented Overview," *IEEE Communications Surveys & Tutorials*, pp. 868–887, 2015.
- [7] C.-M. Huang, K.-c. Lan, and C.-Z. Tsai, "A Survey of Opportunistic Networks," *22nd International Conference on Advanced Information Networking and Applications*, pp. 1672–1677, 2008.
- [8] L. Dong, "Opportunistic media access control and routing for delay-tolerant mobile ad hoc networks," *Wireless Networks*, vol. 18, pp. 949–965, 2012.
- [9] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE simulator for DTN protocol evaluation," *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques*, 2009.
- [10] T.-C. Tsai and H.-H. Chan, "NCCU Trace: social-network-aware mobility trace," *Communications Magazine, IEEE*, vol. 53, pp. 144–149, 2015.
- [11] <http://www.statista.com/chart/1938/monthly-whatsapp-usage-per-user>, "An Average WhatsApp User Sends Messages per Month," 15/09/2015.
- [12] C. Boldrini, M. Conti, and A. Passarella, "Modelling Data Dissemination in Opportunistic Networks," *Proceedings of the third ACM workshop on Challenged networks*, pp. 89–96, 2008.
- [13] Q. Xu, Z. Su, K. Zhang, P. Ren, and X. Shen, "Epidemic Information Dissemination in Mobile Social Networks with Opportunistic Links," *IEEE Transactions on Emerging Topics in Computing*, vol. 6750, pp. 1–1, 2015.
- [14] J. Whitbeck, V. Conan, and M. D. de Amorim, "Performance of Opportunistic Epidemic Routing on Edge-Markovian Dynamic Graphs," *IEEE Transactions on Communications*, vol. 59, pp. 1259–1263, 2011.
- [15] E. Hernández-orallo, J. Herrera-tapia, J.-c. Cano, C. T. Calafate, and P. Manzoni, "Evaluating the Impact of Data Transfer Time in Contact-Based Messaging Applications," *IEEE Communications Letters*, vol. 19, pp. 1814–1817, 2015.
- [16] V. V. Neena and V. M. A. Rajam, "Performance analysis of epidemic routing protocol for opportunistic networks in different mobility patterns," *2013 International Conference on Computer Communication and Informatics*, pp. 1–5, 2013.
- [17] N. Mehta and M. Shah, "Performance Evaluation of Efficient Routing Protocols in Delay Tolerant Network under Different Human Mobility Models," vol. 8, pp. 169–178, 2015.
- [18] J. Su, A. Chin, A. Popivanova, A. Goel, and E. D. Lara, "User Mobility for Opportunistic Ad-Hoc Networking," *Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2004)*, 2004.
- [19] Z. Feng and K.-W. Chin, "A Unified Study of Epidemic Routing Protocols and their Enhancements," *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pp. 1484–1493, 2012.

Arquitectura para integrar los vehículos eléctricos en las Smart Cities del futuro

V. Torres-Sanz, Julio A. Sangüesa, Piedad Garrido, Francisco J. Martínez¹

Resumen— El continuo desarrollo de las tecnologías de información y de la comunicación está facilitando que cada vez estemos más conectados. Estos avances, junto con el desarrollo de otras disciplinas como la Inteligencia Artificial permiten que nos dirijamos más rápidamente hacia las ciudades del futuro, las Smart Cities.

Las disciplinas en las que más se está investigando dentro de las ciudades inteligentes son los Sistemas de Transporte y las Redes Vehiculares. Sin embargo, uno de los actores comúnmente olvidados y que tendrá un papel importante en los próximos años es el vehículo eléctrico. En este artículo, proponemos una arquitectura que, mediante la utilización de la comunicación entre vehículos (V2V) y la infraestructura (V2I), facilitará la integración del vehículo eléctrico en las Smart Cities, añadiendo funcionalidades que beneficiarán tanto a usuarios como a empresas.

Palabras clave— Vehículo eléctrico, redes vehiculares, V2V, V2I, recarga de vehículos eléctricos, Smart Cities.

I. INTRODUCCIÓN

MÁS de la mitad de la población mundial vive en áreas urbanas, y el movimiento de personas hacia estas áreas se espera que continúe durante las próximas dos décadas [1]. En España, en el año 2009, un 77,2% de la población vivía en ciudades y en el año 2050 se espera que este porcentaje ronde el 86,5% [2]. Esta acumulación de personas en áreas metropolitanas indica que nos encaminamos hacia mega-ciudades, en las cuales se van a generar nuevos problemas, como el aumento del tráfico, aglomeraciones y un mayor desarrollo de infraestructuras. Por otro lado, el desarrollo tecnológico y el aumento de dispositivos que se conectan a Internet, unido al desarrollo de la Inteligencia Artificial (IA), está acelerando la llegada de las Smart Cities o ciudades inteligentes [3].

Las Smart Cities buscan mejorar la calidad de vida, el desarrollo económico, la eficiencia, la sostenibilidad y la operatividad, tanto de instituciones y empresas como de sus habitantes, mediante el uso, coordinación e integración de manera inteligente de las tecnologías de la información y los sistemas de comunicación. Giffinger y Pichler-Milanović [4] describen las características que deben poseer las Smart cities y las agrupan en 6 criterios:

- **Economía:** espíritu innovador, imagen económica y marcas comerciales, productividad, flexibilidad, capacidad para transformar.
- **Movilidad:** Conducción autónoma, Sistemas Inteligentes de Transporte (ITS), transporte sostenible, innovador y seguro.

¹Dpto. de Informática e Ingeniería de Computadores, Universidad de Zaragoza, e-mail: vtorres, jsanguesa, piedad, f.martinez@unizar.es

- **Medioambiente:** sostenibilidad de los recursos naturales, reducción de la contaminación y turismo ecológico.
- **Habitantes:** cualificación, formación continua, flexibilidad, conciliación familiar, creatividad y mente abierta.
- **Forma de vida:** viviendas inteligentes, mayor calidad de vida y mejores condiciones de salud.
- **Administración:** participación en la toma de decisiones, gobierno transparente, Gobierno 3.0.

Uno de los aspectos clave en las Smart Cities son la movilidad y los Sistemas Inteligentes de Transporte [5], que van a permitir mejorar varios aspectos, sobre todo a nivel de seguridad vial y confort de los pasajeros [6–8]. Sin embargo, su aplicación se está realizando a un ritmo más lento del deseado, debido a diferentes factores como la situación económica actual y el alto coste de algunos dispositivos.

Los Sistemas Inteligentes de Transporte pretenden mejorar la operación y seguridad del transporte terrestre mediante el uso de soluciones tecnológicas basadas en la informática y las telecomunicaciones. En 2010, la Directiva 2010/40/UE definió estos sistemas como aplicaciones avanzadas que, sin incluir la inteligencia como tal, proporcionan nuevas aplicaciones y servicios para la gestión del transporte.

Las redes vehiculares o redes inalámbricas de comunicación entre vehículos (VANET, Vehicular Ad-hoc NETWORKS) [9] están siendo utilizadas en la actualidad como una tecnología prometedora para mejorar la seguridad en las carreteras. Su objetivo prioritario es la formación de redes de comunicación entre vehículos (comunicación V2V) [10], así como entre los vehículos y la infraestructura de soporte (comunicación V2I) [11]. Existen multitud de aplicaciones para este tipo de redes, entre las que destacamos:

- Mejora de la ruta a seguir en un desplazamiento.
- Control de la congestión del tráfico.
- Mejora de la seguridad vial.
- Alerta de Colisión en Cruces.
- Aviso a los Vehículos de Emergencia.

Los Sistemas Inteligentes de Transporte y las redes vehiculares son un campo en el cual se están realizando un gran número de investigaciones orientadas, por ejemplo, a conocer la densidad del tráfico [13] o mejorar la seguridad vial. Sin embargo, estos trabajos en su mayoría no suelen tener en cuenta a los vehículos eléctricos y sus características especiales, así como las nuevas oportunidades que éstos ofrecen.

Para una correcta implantación del vehículo eléctrico se deben contemplar sus singularidades, sobre todo a la hora de desplegar los futuros ITS. El uso

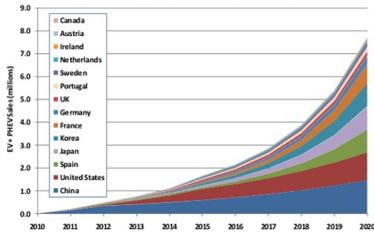


Fig. 1. Previsión de ventas de vehículos eléctricos [12].

de las comunicaciones vehiculares permitirá mejorar no solo la seguridad vial o la gestión del tráfico a nivel global, sino que además podrá ayudar en el proceso de recarga de las baterías de los vehículos eléctricos (reduciendo el tiempo de conexión a la red y el coste asociado a las recargas). Además, permitirá facilitar a las empresas el proceso de generación de electricidad para satisfacer la gran demanda eléctrica que se producirá una vez implantado totalmente este tipo de vehículos.

Las ventas de vehículos eléctricos están aumentando considerablemente, haciéndolo de manera exponencial en todo el mundo. A la cabeza del sector encontramos países como Hong Kong, China, Estados Unidos, Holanda o Noruega. En este último país, 1 de cada 5 vehículos que se vendió en el año 2015 fue eléctrico. Esta tendencia se espera que continúe los próximos años. La Figura 1 muestra la evolución de las ventas de este tipo de vehículos a nivel mundial y la proyección esperada para los próximos años según la International Energy Agency [12], que prevé en torno a 8 millones de vehículos eléctricos en 2026.

Por todo esto, en este artículo se propone una arquitectura de comunicación que, sin despreciar los vehículos propulsados por combustibles fósiles, permita una mejor integración de los vehículos eléctricos en las Smart Cities del futuro mediante el uso de las redes vehiculares. El artículo está organizado de la siguiente manera: en la Sección II se presenta la arquitectura y sus componentes, tanto desde el punto de vista del vehículo (Sección II-A), como la infraestructura necesaria para el procesamiento de los datos (Sección II-B). En la Sección III se muestra el funcionamiento de la arquitectura junto con su protocolo de comunicación. La Sección IV incluye las funcionalidades de la arquitectura y sus beneficios, tanto desde el punto de vista de los usuarios como de las empresas. En la Sección V se comenta el entorno de simulación implementado, así como los modelos matemáticos utilizados. En la Sección VI se presentan los resultados obtenidos en las simulaciones. Por último, en la Sección VII se exponen las conclusiones más importantes.

II. ARQUITECTURA PROPUESTA

La arquitectura propuesta queda reflejada en la Figura 2. Como se observa, consideramos dos partes bien diferenciadas: la primera, denominada On-Board-Unit (OBU), hace referencia a los dispositivos instalados en cada vehículo que captan la información relevante, y por otro lado, la Unidad de Control (UC), que contiene los componentes necesarios para la recepción y tratamiento de dicha información.

Un aspecto que puede llamar la atención respecto a la visión tradicional de las redes vehiculares es la ausencia de RSUs (Road-Side Units). Esto es debido a que la implantación de este tipo de infraestructura supone una importante inversión económica y requiere un tiempo de despliegue relativamente mayor. Por ello, nuestra propuesta se basa en el uso de Internet móvil, un sistema ya implantado.

Estas tecnologías están muy extendidas en la actualidad. De hecho, según el Ministerio de Industria, Energía y Turismo, España dispone de cobertura con tecnología UMTS que alcanza la totalidad de la península [14].

A. Unidad de a bordo (OBU)

El sistema requiere la instalación de una OBU en cada vehículo, que permita recopilar un conjunto de datos y realizar el proceso de comunicación. Los componentes de este sistema son los siguientes:

- **GNSS (Global Navigation Satellite System)**, que es utilizado para obtener las coordenadas del vehículo. Este dato es necesario para estimar cuándo llegará el vehículo a su destino, y con qué nivel de batería.
- **Dispositivo OBD-II**, que se utiliza para obtener información interna del vehículo, como por ejemplo el modelo de vehículo, el nivel de batería, código de avería, etc.
- **Dispositivo de comunicación**, necesario para poder enviar los datos a la UC mediante el uso de tecnologías UMTS/HSDPA. En caso de no disponer de cobertura, los mensajes se enviarán utilizando el estándar 802.11p o WAVE (Wireless Access in Vehicular Environments), para enviar la información a otros vehículos. Cuando alguno de estos vehículos disponga de conexión a Internet, éste enviará la información a la UC.

B. Unidad de Control (UC)

Los datos enviados por los vehículos son recogidos, administrados y procesados por la Unidad de Control, que está compuesta por los siguientes componentes:

- **Servidor**, que es el encargado de obtener, analizar y procesar los mensajes de los vehículos.
- **Módulo DataWarehousing (DW)**. Para poder administrar toda la cantidad de datos que se generará, se hace necesario la utilización de sistemas distintos a las Bases de Datos Relacionales. Será necesario cambiar el sistema tradicional On-Line Transaction Processing (OLTP)

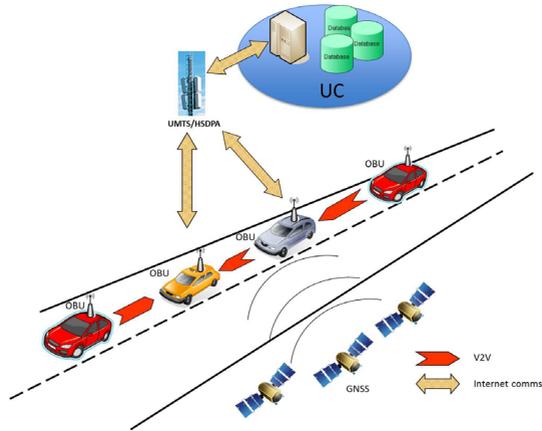


Fig. 2. Arquitectura propuesta.

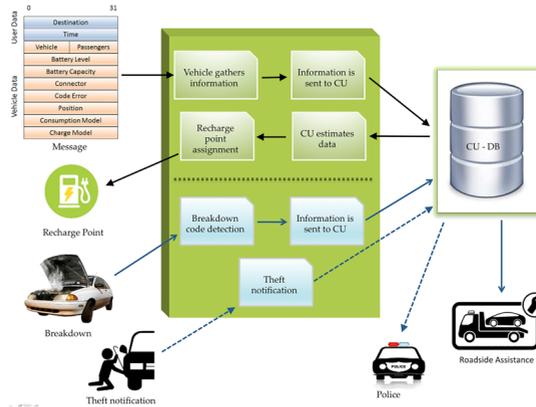


Fig. 3. Diagrama de flujo del protocolo.

por un sistema basado en DW.

III. FUNCIONAMIENTO Y PROTOCOLO

Para el correcto funcionamiento de la aplicación se ha diseñado un protocolo de comunicación. La Figura 3 muestra el protocolo y su funcionamiento. En concreto, el funcionamiento del sistema sería el siguiente:

1. **Definir destino:** una vez que el usuario ha introducido el destino, éste se envía a la UC, junto con un conjunto de datos, como el nivel de batería, modelo, ocupantes y las coordenadas re-

lativas a la posición actual. Los datos serán enviados mediante comunicación V2I o, en caso de no disponer de cobertura, mediante la comunicación V2V. La UC obtendrá los datos y estimará la hora de llegada a su destino, el nivel de batería, y realizará las notificaciones pertinentes.

2. **Reserva de punto de recarga:** en caso de ser requerido, un punto de recarga podrá ser reservado para el vehículo cuando llegue a su destino.
3. **Asistencia en carretera:** en caso de avería, el usuario podrá lanzar un aviso a la UC. En este caso, se enviará el código de error de la avería, y la UC enviará el dato al servicio de asistencia

más cercano al usuario.

4. **Asistencia en accidente:** en caso de producirse un accidente, éste es detectado mediante un conjunto de sensores, enviando la gravedad del accidente y el número de ocupantes del vehículo.
5. **Localización del vehículo:** en caso de robo, una vez notificado a la UC, el vehículo comenzará a notificar periódicamente las coordenadas en las que se encuentra.

IV. APLICACIONES

El objetivo de esta arquitectura es facilitar la integración del vehículo eléctrico en las Smart Cities del futuro, y por lo tanto se basa en la utilización de este tipo de vehículo como actor principal. En esta sección, presentamos las diferentes aplicaciones que la arquitectura ofrece en una ciudad conectada e inteligente, basada en el Internet de las cosas.

Para entender mejor los beneficios que supone el uso de la arquitectura, vamos a suponer por ejemplo, un gran centro comercial, donde además de las plazas de aparcamiento para vehículos tradicionales, existe una zona delimitada para los vehículos eléctricos. Desde el punto de vista de los usuarios supondrá los siguientes beneficios:

- **Reserva de puntos de recarga.** Dado que será posible conocer el momento en el que el usuario va a llegar al centro comercial, se podría reservar un punto de recarga para su uso.
- **Descuentos en el precio de la electricidad.** Al avisar con antelación, y disponer de información sobre el nivel de batería de cada vehículo, las empresas eléctricas pueden conocer en cada momento cuándo y cuánta energía van a tener que producir, lo que puede suponer un descuento en el precio de la electricidad.
- **Reducción del coste.** Dado que el sistema puede saber la necesidad de puntos de recarga, y el tiempo que un vehículo va a estar conectado, se puede planificar la recarga para realizarla en el momento en que la electricidad es más barata.
- **Asistencia en carretera.** Debido a que la OBU tiene acceso al OBD-II, en caso de avería, obtendrá el código interno del fallo, y enviará la información a la UC junto a la posición del vehículo. De esta manera el servicio de asistencia en carretera será más rápido, eficiente y personalizado, pudiendo acelerar la reparación.
- **Asistencia en caso de accidente:** con la ayuda de sensores del vehículo la OBU obtendrá tanto los parámetros que permitirán la estimación de la gravedad de accidente como de la posición del vehículo. De esta forma los servicios médicos podrán conocer de antemano a lo que se enfrentan e ir mejor preparados para asistir a los heridos. Asistir a un herido en la primera hora del accidente (hora de oro), en la cual se produce el 75% de las muertes por accidente de tráfico [15] puede salvar vidas.
- **Seguimiento en caso de robo.** Tal y como se ha comentado en la sección anterior, en caso

de que un vehículo sea robado, además de poder realizar de forma automática la denuncia, el vehículo notificará, ya sea mediante comunicación V2V o mediante V2I, sus coordenadas facilitando su localización en tiempo real.

Desde el punto de vista de las empresas, nuestra arquitectura aportará los siguientes beneficios:

- **Conocer el número de vehículos.** Con nuestro sistema se podría conocer el número de vehículos que va a asistir a un evento. Además, con los sensores que lleva incorporados el vehículo se podría incluso conocer exactamente el número de personas que van a asistir, permitiendo informar a las empresas y que adecuen sus servicios a la demanda esperada.
- **Estimación de la demanda eléctrica.** En el caso de los vehículos eléctricos, conociendo desde donde inician el viaje, su destino y el nivel de batería del vehículo, se podría estimar el nivel de batería con que llegarán a un punto de recarga. Por lo tanto, se conocería de antemano la energía necesaria para las recargas.

V. ENTORNO DE SIMULACIÓN

Para probar el modelo se ha implementado un simulador en Java que permite comprobar y evaluar el funcionamiento del sistema. Sobre todo se ha hecho hincapié en el aspecto de los vehículos eléctricos. El objetivo es calcular con cuánto tiempo de antelación se puede conocer la demanda eléctrica que se necesita para este tipo de vehículos.

Para poder realizar esta prueba se ha creado un escenario de simulación basado en el aparcamiento de un gran centro comercial, en el cual existen plazas específicas para recargar este tipo de vehículos. Se simula un horario comercial desde las 10 hasta las 22 horas. Se ha supuesto que los puntos de recarga solo funcionan en el horario comercial.

La herramienta informática utilizada se divide en dos componentes:

- **Creación de traza de vehículos:** se crean los vehículos, se simula la notificación de los avisos, cuándo llegarán al punto de recarga y se estima con qué nivel de batería llegarán basándose en la distancia a la que se encuentran. Para poder utilizar estos datos se han utilizado estudios sobre los hábitos de los conductores [16–20]. También se han utilizado modelos matemáticos para la parametrización de algunas variables (ver Tabla I), basadas en las investigaciones de Quian et al. [19], Hao Bai et al. [20], Zeng et al. [21], y Cheng et al. [22].
- **Simulador de recargas:** en esta parte de la aplicación se simula el proceso de recarga de baterías. Para simularlo se utiliza el conteo de Coulomb Modificado (Modified Coulomb counting). En concreto, se mide la intensidad que carga integrándola en el tiempo, partiendo del nivel de batería con el que llega el vehículo. Este método se considera preciso a la hora de realizar

estimaciones de carga [23].

El diseño del escenario de simulación ha sido creado basado en Centro Comercial de Puerto Venecia, que es el mayor Centro Comercial de Europa. Contiene más de 10.000 plazas de aparcamiento, algunas de ellas especiales para vehículos eléctricos. En un sábado normal, Puerto Venecia llega a tener picos de entrada de 1.500 vehículos a la hora [24]. En este caso hemos planteado un escenario en el que a lo largo de un día, acuden al centro 3000 vehículos eléctricos.

Los datos que se han utilizado para la simulación son:

- **Modelo de llegada de vehículos:** para modelar este proceso se utiliza un modelo Gaussiano con una media (μ) de 300 minutos y una Desviación Típica (σ) de 130. Esto permite que la llegada de vehículos no sea constante, tal y como sucede en la realidad. Se utiliza este modelo basándonos en la investigación de Subbiah et al. [25], en donde se realiza un estudio de la gente que asiste a un centro comercial en función de su consumo eléctrico.
- **Modelo de duración del viaje:** se ha utilizado el modelo de Weibull con los parámetros ($\alpha=40$, $\beta=1.5$, $\gamma=0$) para modelar la distancia que recorrerán los usuarios antes de llegar al centro comercial.
- **Tiempo de recarga:** se ha utilizado un modelo Gaussiano, con una $\mu = 120$ minutos y $\sigma = 80$. Este modelo determinará el tiempo que el vehículo estará conectado al punto de recarga.
- **Nivel de Batería:** para simular el nivel de batería que tienen los vehículos se ha utilizado un modelo Gaussiano con $\mu = 40\%$ de carga y $\sigma = 20$. Con estos parámetros se plantea un escenario bastante estricto con un nivel de batería similar al de 2 días de uso sin recargar [19].
- **Capacidad de las baterías:** para obtener la capacidad de las baterías se ha utilizado un modelo Gaussiano con una $\mu = 19.000$ (W/h) y $\sigma = 4.000$. De esta manera se consigue un valor medio en base a las ventas de vehículos eléctricos en España, teniendo en cuenta la existencia de los vehículos híbridos enchufables o plug-in hybrid electric vehicle (PHEV). Con la desviación típica se consigue la amplitud suficiente para tener en cuenta tanto los vehículos eléctricos medios, como a los PHEV.
- **Punto de recarga:** el punto de recarga utilizado equivaldría a un modo 2, que correspondería a 32 Amperios con 230 voltios, que según el IEC-62196 es el modo estipulado para los lugares públicos como los centros comerciales.

VI. RESULTADOS DE SIMULACIÓN

Los resultados obtenidos con la simulación del escenario planteado en la sección anterior pueden verse en la Figura 4. En ella queda reflejado la energía que consumen en la recarga los 3000 vehículos simulados. El periodo en el que más energía se consume es

TABLA I
 PARÁMETROS DE ENTRADA DEL SIMULADOR

Parámetro	Valor
Número de vehículos	3000
Modelo de llegada	Gaussian: $\mu = 300$ (min) $\sigma = 130$
Modelo de t. de viaje	Weibull: $\alpha = 40$ (min) $\beta = 1.5$ $\gamma = 0$
Modelo de t. de recarga	Gaussian: $\mu = 120$ (min) $\sigma = 80$
Modelo de Nivel de batería	Gaussian: $\mu = 40$ (%) $\sigma = 20$
Capacidad de batería	Gaussian: $\mu = 19.000$ (W) $\sigma = 4000$
Potencia de recarga	8 (kW)

el comprendido entre las 16 y 17 horas, con un consumo de 7,46 MW, la energía equivalente al consumo de una ciudad de 25.000 habitantes (en kW/hora).

La distribución del consumo eléctrico obtenido en la simulación tiene forma Gaussiana, lo cual concuerda con los modelos utilizados, y también con los estudios realizados por Subbiah et al. [25]. El sistema predictivo de consumo eléctrico, mostrado con una línea en el Figura 4 muestra la antelación en minutos con la que el sistema sería capaz de predecir la demanda de electricidad esperada para la recarga de los vehículos eléctricos.

Los resultados de la simulación muestran que el comportamiento del sistema predictivo mejora conforme avanza la simulación. Esto es debido a que además de conocer el tiempo de recarga de los vehículos que aún están de camino, conoce el tiempo de aquellos que ya están recargando. Tal y como se puede observar, a las 15 horas el sistema es capaz de estimar con precisión la demanda eléctrica necesaria para la recarga de vehículos eléctricos con 62 minutos de antelación, mientras que a las 19 horas, seríamos capaces de estimar la demanda eléctrica con una antelación de 117 minutos.

El parque de turismos en España en el año 2014 fue de 22 millones de vehículos. Supongamos un futuro en el que un 10 % de estos vehículos sean eléctricos. Extrapolando los resultados de la simulación a este escenario, el consumo eléctrico producido por este 10 % de vehículos sería de 5.470 MW. La recarga de un número alto de vehículos eléctricos de forma descontrolada puede producir que el sistema colapse [26]. Por ello, es interesante conocer con antelación la demanda eléctrica que se producirá por este tipo de vehículos para que el sistema energético pueda generarla con antelación.

VII. CONCLUSIONES

El vehículo eléctrico se posiciona como una alternativa al vehículo propulsado por combustibles fósiles. Al mismo tiempo, nos encaminamos cada vez más rápido hacia las ciudades inteligentes, o Smart Cities. En la mayoría de las investigaciones en las que se trabaja con redes vehiculares no se tiene en cuenta al vehículo eléctrico, ni los beneficios que éstos podrían obtener del uso de redes vehiculares.

La implantación del sistema que proponemos sería relativamente económica, y favorecerá la integración del vehículo eléctrico en las Smart Cities del futuro.

Las empresas que deseen utilizar el sistema conocerán el número de vehículos y de personas que van a

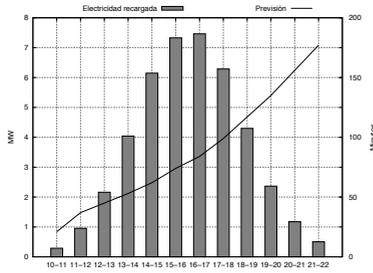


Fig. 4. Resultados obtenidos.

asistir a un evento, se podrá realizar una estimación de la demanda eléctrica que requerirán los vehículos eléctricos, por lo que, además de tener un tiempo de reacción para producirla, pueden desplegar puestos supletorios de recarga en caso de necesitarse.

En el caso de los eléctricos favorecemos el que dispongan de un punto de recarga al asistir a un evento, además de posibilitar posibles descuentos en la tarifa de la luz. Finalmente, el sistema también facilita la asistencia en carretera en caso de avería o accidente.

REFERENCIAS

[1] Hafeedh Chourabi, Taewoo Nam, Shawn Walker, J Ramon Gil-Garcia, Sehi Mellouli, Karine Nahand, Theresa A Pardo, and Hans Jochen Scholl, "Understanding smart cities: An integrative framework," in *45th Hawaii International Conference on System Science (HICSS)*, 2012, pp. 2289-2297.

[2] Johannes Von Stritzky and Casilda Cabrerizo, *Ideas para las ciudades inteligentes del futuro*, Fundación Ideas, 2011.

[3] Michael Batty, Kay W. Axhausen, Fosca Giannotti, Alexei Pozdnoukhov, Armando Bazzani, Monica Wachowicz, Georgios Ouzounis, and Yuval Portugali, "Smart cities of the future," *The European Physical Journal Special Topics*, vol. 214, no. 1, pp. 481-518, 2012.

[4] Rudolf Giffinger and Nataša Pichler-Milanović, *Smart cities: Ranking of European medium-sized cities*, Centre of Regional Science, Vienna University of Technology, 2007.

[5] Chai-Keong Toh, "Future application scenarios for MANET-based Intelligent Transportation Systems," in *Future generation communication and networking (FGCN)*, 2007, vol. 2, pp. 414-417.

[6] Wen Qiang Wang, Xiaoming Zhang, Jiangwei Zhang, and Hock Beng Lim, "Smart traffic cloud: An infrastructure for traffic applications," in *IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)*, 2012, pp. 822-827.

[7] Pravin Varaiya, "Smart cars on smart roads: problems of control," *IEEE Transactions on Automatic Control*, vol. 38, no. 2, pp. 195-207, 1993.

[8] Francisco J. Martinez, Chai-Keong Toh, Juan-Carlos Cano, Carlos T Calafate, and Pietro Manzoni, "Emergency services in future intelligent transportation systems based on vehicular communication networks," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 2, pp. 6-20, 2010.

[9] Francisco J. Martinez, Chai-Keong Toh, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni, "Determining the representative factors affecting warning message dissemination in VANETs," *Wireless Personal Communications*, vol. 67, no. 2, pp. 295-314, November 2012.

[10] Francisco J. Martinez, Chai-Keong Toh, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni, "A street broadcast reduction scheme (SBR) to mitigate the broad-

cast storm problem in VANETs," *Wireless Personal Communications*, vol. 56, no. 3, pp. 559-572, 2011.

[11] Javier Barrachina, Piedad Garrido, Manuel Fogue, Francisco J. Martinez, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni, "D-RSU: A Density-Based Approach for Road Side Unit Deployment in Urban Scenarios," in *International Workshop on IPo6-based Vehicular Networks (Vehi6)*, collocated with the 2012 IEEE Intelligent Vehicles Symposium, Jun. 2012, pp. 1-6.

[12] International Energy Agency, "Projected electric and plug-in hybrid vehicle sales through 2020," Available at <http://image.slidesharecdn.com/mrfrancoisnguyen-12990572907474-phppap01/95/mr-francois-nguyen-5-728.jpg?cb=1299035827> (accessed on 20-05-2016).

[13] Julio A. Sanguesa, Manuel Fogue, Piedad Garrido, Francisco J. Martinez, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni, "Real-time density estimation in urban environments by using vehicular communications," in *Wireless Days (WD)*, 2012 IFIP, Nov 2012, pp. 1-6.

[14] Ministerio de Industria, Energía y Turismo, "Consulta de cobertura," Available at <http://www.minetur.gob.es/telecomunicaciones/banda-ancha/cobertura/consulta/Paginas/consulta-cobertura-banda-ancha.aspx> (accessed on 17-05-2016).

[15] Manuel Fogue, Piedad Garrido, Francisco J. Martinez, Juan-Carlos Cano, Carlos T Calafate, and Pietro Manzoni, "Automatic accident detection: Assistance through communication technologies and vehicles," *IEEE Vehicular Technology Magazine*, vol. 7, no. 3, pp. 90-100, 2012.

[16] Ville Tikka, Henri Makkonen, Jukka Lassila, and Jarmo Partanen, "Case study: Smart charging plug-in hybrid vehicle test environment with vehicle-to-grid ability," in *16th European Conference on Power Electronics and Applications (EPE'14-ECCE Europe)*, 2014, pp. 1-10.

[17] Petter Haugeland and Hans Håvard Kvisle, "Norwegian electric car user experiences," *International Journal of Automotive Technology and Management*, vol. 15, no. 2, pp. 194-221, 2015.

[18] Silvia Cestau Cubero, *Sostenibilidad técnica, económica y ambiental de flotas comerciales de vehículos eléctricos*, Ph.D. thesis, 2015.

[19] Kejun Qian, Chengke Zhou, Malcolm Allan, and Yue Yuan, "Modeling of load demand due to EV battery charging in distribution systems," *IEEE Transactions on Power Systems*, vol. 26, no. 2, pp. 802-810, 2011.

[20] Hao Bai, Shihong Miao, Pipei Zhang, and Zhan Bai, "Reliability evaluation of a distribution network with microgrid based on a combined power generation system," *Energies*, vol. 8, no. 2, pp. 1216-1241, 2015.

[21] Shuang Zeng, Jinguo Zhang, Zhongjun Chi, Xianglong Li, and Yanxia Chen, "The research of temporal and spatial uncertainty of electric vehicle charging load impact on power system," in *International Conference on Power System Technology (POWERCON)*, 2014, pp. 3265-3270.

[22] Lidan Chen, CY Chung, Yongquan Nie, and Rongrong Yu, "Modeling and optimization of electric vehicle charging load in a parking lot," in *IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, 2013, pp. 1-5.

[23] Kong Soon Ng, Chin-Sien Moo, Yi-Ping Chen, and Yao-Ching Hsieh, "Enhanced coulomb counting method for estimating state-of-charge and state-of-health of lithium-ion batteries," *Applied Energy*, vol. 86, no. 9, pp. 1506-1511, 2009.

[24] Herald de Aragón, "Luz verde al nuevo ramal que dará acceso a Puerto Venecia," 2015.

[25] Rajesh Subbiah, Kristian Lum, Achla Marathe, and Madhav Marathe, "A high resolution energy demand model for commercial buildings," in *ETG-Fachbericht-Internationaler ETG-Kongress 2013-Energieversorgung auf dem Weg nach 2050*. VDE VERLAG GmbH, 2013.

[26] Fernando Herrero Frutos, José Villar Collado, Cristian Andrés Diaz, and Francisco Alberto Campos, "Análisis de la influencia del vehículo eléctrico en la demanda eléctrica," *Universidad Pontificia Comillas*, 2011.

Un nuevo enfoque para la adaptación de caudal en transmisiones multicast en SDWN

Estefanía Coronado¹, Roberto Riggio², José Villalón¹, Antonio Garrido¹

Resumen— En los últimos años, la distribución de contenido multimedia en redes inalámbricas está sufriendo un gran auge, siendo el estándar IEEE 802.11 uno de los más utilizados. Debido a la naturaleza del canal, todos los paquetes transmitidos deben ser confirmados, a excepción de las comunicaciones multicast. Bajo este estándar las transmisiones en modo multicast se realizan mediante un procedimiento de broadcast en el que el emisor no espera la confirmación de los paquetes. Esta falta de *feedback* hace que la información deba ser enviada a una velocidad básica y que los paquetes erróneos no puedan ser recuperados. Para solucionar estas limitaciones, la enmienda IEEE 802.11aa incorpora un conjunto de propuestas para el tráfico multicast. A pesar de mejorar las prestaciones del estándar original, se ha demostrado que estos esquemas sufren numerosas limitaciones. En este trabajo se muestra una arquitectura basada en SDN (*Software Defined Networks*) para la mejora de las comunicaciones en modo multicast en redes IEEE 802.11. En dicha arquitectura se pretende dar soporte para adaptar la velocidad de envío del tráfico multicast a las condiciones del canal, optimizando así la utilización del medio inalámbrico y mejorando el rendimiento de la red.

Palabras clave— IEEE 802.11, SDWN, WLANs, multicast, multimedia

I. INTRODUCCIÓN

ACTUALMENTE, las redes inalámbricas se encuentran en un periodo de gran expansión, debido a su bajo coste, su facilidad en el despliegue y a la libertad de movimiento que proporcionan. Otro factor muy importante ha sido la aparición del estándar IEEE 802.11 [1], sus continuas revisiones y el desarrollo de sus enmiendas de mejora. Debido a la naturaleza del canal de transmisión, en este estándar todos los paquetes enviados deben ser confirmados por el receptor. Si el emisor no recibe la confirmación del paquete, éste deberá retransmitirlo. De este forma, se aporta fiabilidad a las transmisiones en un entorno con cambios constantes en la calidad del canal y con alta probabilidad de colisiones. Además, esta información es utilizada por el emisor para adaptar la velocidad de transmisión a las condiciones de canal.

Para evitar una avalancha de colisiones, en las transmisiones en modo multicast el emisor no espera la confirmación de los paquetes. Esta falta de confirmación hace que estos paquetes no puedan ser retransmitidos y que una gran cantidad de ellos sean descartados. Además, el desconocimiento del estado del canal impide adaptar la velocidad de en-

vío. Como consecuencia, el estándar recomienda usar una velocidad básica de transmisión para este tipo de tráfico, reduciendo considerablemente la cantidad de datos que se envían por la red.

Por otro lado, la distribución de contenidos multimedia a través de Internet ha crecido exponencialmente en los últimos años. Entre los posibles modos de acceso a estas aplicaciones, el uso de WLANs (*Wireless Local Area Networks*) es uno de los más habituales. Este tipo de comunicaciones no solo se caracterizan por las altas necesidades de ancho de banda, sino que además imponen restricciones severas en cuanto a retardos, *jitter* y tasas de descarte. Es decir, las aplicaciones multimedia necesitan soporte de QoS (*Quality of Service*). Garantizar estos requisitos de QoS en redes IEEE 802.11 es una tarea compleja debido al modo de operación de su capa MAC (*Media Access Control*), al riesgo de colisión de los paquetes y a las dificultades de transmisión debidas a la propagación de la señal.

Para mejorar el soporte de QoS, la enmienda IEEE 802.11e [2] pretende otorgar una mayor prioridad a las aplicaciones en tiempo real. Con ello, las prestaciones de las aplicaciones multimedia mejoran considerablemente. Sin embargo, en dicha enmienda no se presta atención a las comunicaciones multicast. La enmienda IEEE 802.11aa [3] trata de aportar mayor robustez a las comunicaciones multicast a través de distintos esquemas para la retransmisión de paquetes. Sin embargo, esta enmienda continúa sin incluir mecanismo alguno para la adaptación de caudal en transmisiones multicast.

El aumento del tamaño de las redes y la integración de dispositivos de distintos fabricantes suponen cada vez más un problema para su gestión. Por ello, las redes basadas en SDN (*Software Defined Networks*) han experimentado un reciente auge ya que proporcionan una abstracción del hardware de la red. Así, la inteligencia se recoge en un controlador central que toma las decisiones sobre el control de la misma. Este enfoque permite separar el funcionamiento de la red en un plano de control y un plano de datos, simplificando así su gestión. Sin embargo, resulta complicado encontrar esquemas dirigidos a redes SDN inalámbricas, ya que los estándares publicados, como es el caso de OpenFlow [4], tan sólo se centran en redes cableadas.

El objetivo de este trabajo es adaptar la velocidad de transmisión en modo multicast sobre una arquitectura basada en SDN. Para ello, se establecen periodos de tiempo, donde se alternan el envío de datos sin confirmación y el uso de políticas de retransmisión incluidas en IEEE 802.11aa. Además, el

¹Instituto de Investigación en Informática de Albacete (I3A), Universidad de Castilla-La Mancha, e-mail: {Estefania.Coronado, JoseMiguel.Villalon, Antonio.Garrido}@uclm.es

²CREATE-NET, Trento (Italia), e-mail: rriggio@create-net.org

esquema define dos modos de funcionamiento distintos para conocer el estado del canal. El primer modo toma esta información a partir de las estadísticas recogidas durante el envío de paquetes unicast. Por el contrario, el segundo enfoque reduce el *overhead* con respecto al primero, ya que los paquetes en modo unicast solo son enviados a aquellos receptores con peores condiciones. De este modo, se reduce el ancho de banda utilizado, mejorando así el rendimiento de las comunicaciones en la red.

Este artículo se organiza como sigue. La Sección II describe la enmienda IEEE 802.11aa y una serie de esquemas de adaptación del caudal en transmisiones multicast. La Sección III recoge los principios básicos del paradigma SDN. El esquema propuesto se muestra en la Sección IV, mientras que los resultados de su evaluación se presentan en la Sección V. En la Sección VI se exponen las conclusiones extraídas. Por último, algunas propuestas de trabajo futuro se incluyen en la Sección VII.

II. REDES IEEE 802.11

El IEEE 802.11 es el estándar de referencia en WLANs, debido principalmente a la sencillez de funcionamiento. Sin embargo, precisamente su sencillez le ha ocasionado ciertas limitaciones que se han ido solucionando a través de numerosas enmiendas de mejora. En la primera versión de este estándar se definen dos funciones de acceso al medio a nivel MAC, como son DCF (*Distributed Coordination Function*) y PCF (*Point Coordination Function*). DCF usa un mecanismo de acceso al medio distribuido, donde todas las estaciones utilizan los mismos tiempos de espera para acceder al canal, por lo que no se proporciona soporte de QoS. PCF utiliza la anterior como base para su funcionamiento y, dado que su implementación es opcional, su uso no está extendido en tarjetas comerciales. Por ello, la enmienda IEEE 802.11e define HCF (*Hybrid Coordination Function*) como una nueva función de acceso al canal capaz de otorgar una mayor prioridad a las aplicaciones de voz y vídeo. Sin embargo, esta enmienda no tuvo en cuenta las características especiales que presentan las aplicaciones multimedia en modo multicast.

Las transmisiones multicast suponen una forma eficiente de envío de datos de un emisor a un conjunto de receptores ya que se reduce el coste para enviar un mismo paquete en modo unicast a todos los receptores del grupo. Sin embargo, dado que estas transmisiones se llevan a cabo mediante un procedimiento de broadcast, el emisor no espera la confirmación de los paquetes. Con ello se pretende eliminar las colisiones de los paquetes de respuesta, ya que de lo contrario los ACKs enviados colisionarían, provocando que el emisor retransmitiera la información. No obstante, como consecuencia se produce una falta de fiabilidad en la transmisión. Además, resulta imposible adaptar la velocidad de transmisión a las condiciones del medio, por lo que la velocidad de envío se fija a la tasa básica.

En vista de lo anterior, existen numerosos traba-

jos que tratan de solventar los problemas descritos. Además, el IEEE formó un grupo de trabajo encargado de diseñar una enmienda de mejora (IEEE 802.11aa) que, manteniendo la compatibilidad con los dispositivos existentes, fuera capaz de mejorar las comunicaciones multicast en este estándar.

A. IEEE 802.11aa

La enmienda IEEE 802.11aa presenta el servicio GATS (*Group Addressed Transmission Service*), cuyo objetivo es adaptar la transmisión a las necesidades de las aplicaciones y de las estaciones, aportando así una mayor fiabilidad. Esta fiabilidad se introduce primeramente en la enmienda IEEE 802.11v [5], donde se describe el funcionamiento del WNM (*Wireless Network Management*). Como parte del servicio GATS, se definen dos nuevos mecanismos que pretenden aumentar la fiabilidad: DMS (*Directed Multicast Service*) y GCR (*GroupCast with Retries*).

El servicio DMS consiste en la conversión a nivel TID (*Traffic Identifier*) del tráfico multicast a unicast. Este envío consigue solventar los problemas de fiabilidad y del uso de una tasa fija de transmisión. Sin embargo, este servicio tiene graves problemas de escalabilidad ya que el tráfico de la red depende directamente del tamaño del grupo multicast, pudiéndose utilizar únicamente para grupos muy pequeños.

Por otro lado, en GCR pueden distinguirse tres servicios distintos:

- *Legacy multicast*. Este mecanismo se trata del método de envío en modo multicast definido en el estándar IEEE 802.11 original.
- *GCR-Unsolicited retries*. Mediante esta política, se especifica un número de reintentos, N , de modo que el emisor transmite $N+1$ veces un mismo paquete multicast. De este modo, se aumenta la probabilidad de que la transmisión llegue correcta a su destino. No obstante, el número de retransmisiones innecesarias puede comprometer el rendimiento de la red.
- *Block-ACK*. Este método es el más sofisticado y permite aunar la transmisión en modo multicast con la recepción de información de *feedback*. El mecanismo Block-ACK consiste en el reconocimiento de varios paquetes de datos haciendo uso de un simple paquete ACK. De este modo se permite elevar la fiabilidad en la transmisión sin penalizar en exceso la escalabilidad.

B. Esquemas de adaptación del caudal de envío

La adaptación del caudal de transmisión en multicast pasa por la definición de mecanismos de *feedback* que permitan al emisor conocer el estado del canal. Además, estos mecanismos son utilizados para mejorar la fiabilidad del envío. En este sentido, surgen distintos enfoques que tratan de abordar el problema.

Los esquemas basados en líder son los más habituales, entre los que destaca LBP (*Leader Based Protocol*) [6]. En esta propuesta, la estación con

peor calidad de canal se convierte en líder y se encarga de confirmar los paquetes multicast. El resto de estaciones se limitan al envío de confirmaciones negativas para solicitar la retransmisión de información. Sin embargo, no se define un mecanismo para la selección del líder. ARSM (*Auto Rate Selection Multicast*) [7] define un esquema dividido en dos fases. En la primera de ella, se selecciona o actualiza el líder del grupo, mientras que durante la segunda, se adapta la velocidad de envío usando el SNR (*Signal Noise Ratio*) de las confirmaciones del líder. H-ARSM (*Hierarchical ARSM*) [8] es una evolución de ARSM diseñado para la transmisión de video jerárquico multicast en WLANs. Así, se asegura que todas las estaciones reciben una calidad mínima de la secuencia, y que aquellos con mejor calidad de canal, sean capaces de recibir capas de mejora.

La adaptación del caudal de transmisión basada en SNR es igualmente utilizada en SARM (*SNR - Base Auto Rate for Multicast*) [9]. Este esquema se basa en el envío de paquetes de *beacon* para identificar el peor receptor. En este paquete, el AP incluye el SNR percibido por el peor receptor multicast, de modo que, si alguna estación cuenta con una peor calidad, deberá responder a esta petición. No obstante, la implementación de este esquema conlleva la modificación de los paquetes de *beacon*.

Otro enfoque utilizado es el que busca mejorar el nivel de QoE (*Quality of Experience*) percibido en las aplicaciones multimedia. En [10] se diseña una red neuronal para construir un modelo capaz de mapear mediciones de QoE con tasas de transmisión. Para ello, se emplea una métrica híbrida de QoE que simula la evaluación real de los usuarios.

A pesar de las mejoras, la mayoría de los esquemas mostrados son diseñados y evaluados mediante herramientas de simulación, o introducen mecanismos incompatibles con el estándar IEEE 802.11.

III. SOFTWARE DEFINED WIRELESS NETWORKS

El crecimiento de los servicios de comunicaciones y la explosión de contenidos multimedia ha hecho necesario un replanteamiento en las arquitecturas de red tradicionales. Las tecnologías de red actuales poseen limitaciones como la dependencia de los fabricantes, el aumento de la complejidad a medida que lo hace el tamaño y la necesidad de un personal humano centrado en su continua administración.

En este panorama las redes definidas por software, o SDN, proponen una nueva arquitectura de red totalmente programable. De este modo, la funcionalidad de los dispositivos se divide en los planos de control y de datos, permitiendo así abstraer la infraestructura de la red a servicios y aplicaciones, tal y como se muestra en la Figura 1. Como consecuencia, la inteligencia de la red, hasta ahora distribuida sobre varios elementos de la misma, se concentra sobre un controlador central. Dicho controlador posee una visión completa de la red, lo que le permite tomar las decisiones de gestión y control más apropiadas.

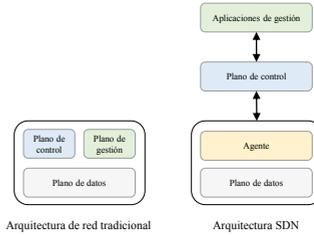


Fig. 1. Evolución de la arquitectura tradicional a la de SDN

La arquitectura SDN se divide en tres grandes capas, conocidas como capas de infraestructura, de control y de datos. La capa de infraestructura, o plano de datos, comprende los dispositivos de red encargados del encaminamiento de paquetes. Estos dispositivos se encuentran en constante comunicación con la capa de control a través de la interfaz *southbound*. Por su parte, las aplicaciones SDN permiten simplificar las tareas de gestión y configuración de nuevos servicios de red. Dichas aplicaciones se sitúan en la capa de aplicación, comunicándose con el controlador a través de la interfaz *northbound*.

Dado que la inteligencia de los dispositivos se simplifica sustancialmente, la interfaz *southbound* se convierte para el controlador en un elemento clave para obtener información del estado de la red. Uno de los estándares más utilizados en las comunicaciones de la interfaz *southbound* es OpenFlow. Sin embargo, se trata de un protocolo dirigido a redes cableadas, por lo que las funcionalidades que proporciona se encuentran relativamente alejadas de las necesidades de las redes inalámbricas.

Todos estos factores han llevado al paradigma SDN a convertirse en uno de los temas de investigación más punteros en WLANs. Recientemente, varios autores han propuesto arquitecturas para redes IEEE 802.11 basadas en el paradigma SDN. Algunas de las propuestas tratan de solventar las deficiencias de OpenFlow en entornos inalámbricos mediante la definición de abstracciones de programación destinadas al control de grandes redes [11] o a la simplificación en la gestión de clientes [12].

Los trabajos anteriores se centran en transmisiones unicast y apenas existe investigación relacionada con comunicaciones multicast. En esta línea, MultiFlow [13] trata de mejorar las comunicaciones multicast a través de la réplica de los paquetes para cada destinatario. De este modo, el controlador de la red se encarga de llevar a cabo esta tarea en función del ancho de banda disponible, mejorando con ello la calidad de las comunicaciones. No obstante, los resultados sólo son presentados como un análisis numérico. Además, este esquema cuenta con la desventaja de que el uso del medio inalámbrico puede superar el de un envío típico en modo multicast cuando el grupo excede un

determinado tamaño.

IV. ARQUITECTURA BASADA EN SDN PARA LA ADAPTACIÓN DEL CAUDAL EN MULTICAST EN REDES IEEE 802.11

El crecimiento del volumen de datos transmitido y del tamaño de las redes, hacen que los APs sean incapaces de encargarse del encaminamiento de paquetes y ejecutar algoritmos de adaptación de caudal en tiempo real.

Las aplicaciones multimedia requieren de rápidos ajustes en la gestión de la red debido a que la pérdida de información puede perjudicar gravemente la calidad percibida por el usuario. Para solventar estos problemas, nuestra propuesta se basa en la implementación de un algoritmo situado sobre un controlador SDN capaz de ajustar de forma inteligente el caudal de las transmisiones multicast. En este sentido, se pretende optimizar el uso del canal inalámbrico y mejorar el rendimiento de las comunicaciones multimedia destinadas a un grupo de receptores.

A. Arquitectura EmPOWER

Dado que el protocolo OpenFlow no es capaz de cubrir las necesidades de las redes inalámbricas, el esquema propuesto toma como referencia la arquitectura de red basada en SDN presentada en [14]. Esta arquitectura es conocida como *EmPOWER* y proporciona un protocolo muy eficiente en la comunicación entre el plano de control y de datos, conocido como Protocolo EmPOWER. Así, se introduce un nuevo paradigma capaz de proporcionar una serie de abstracciones de programación en SDWNs.

EmPOWER define una pila software donde se distinguen, como es habitual en SDN, las capas de infraestructura, control y aplicación (véase la Figura 2). La capa de infraestructura consiste en un camino de datos programable basado en 802.11 diseñado para los APs de la red, conocidos en esta arquitectura como WTPs (*Wireless Termination Points*). Igualmente, la implementación del controlador SD-RAN se sitúa en la capa de control. Por último, en la capa de aplicación se presenta un SDK basado en Python con el objetivo de proporcionar un entorno de programación para aplicaciones de red dedicadas a la gestión de la misma.

La lógica de los APs se divide en dos componentes claramente diferenciados. El router modular Click [15] se emplea para la implementación del camino de datos 802.11, mientras que OpenvSwitch [16] se encarga del manejo de la comunicación con el controlador a través de una conexión TCP persistente. El sistema almacena además información de estado de cada cliente, designada como LVAP (*Light Virtual Access Point*), donde pueden encontrarse detalles relacionados con los procesos de asociación, autenticación, movilidad de la estación o planificación de los recursos utilizados. De este modo, se consigue simplificar el manejo y gestión de la red.

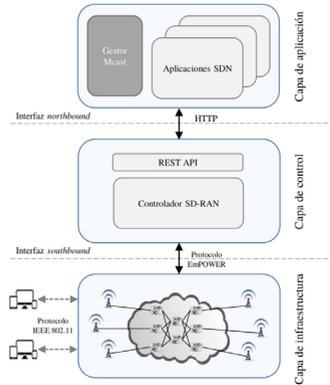


Fig. 2. Arquitectura de *EmPOWER*

B. MCast

El algoritmo *MCast* de adaptación de caudal se sitúa sobre la capa de control, tal y como se muestra en la Figura 2. Como resultado, éste puede beneficiarse de la inteligencia del controlador y de la información contenida en él. Esta propuesta pretende utilizar la información estadística derivada del algoritmo Minstrel [17] para adaptar la velocidad de transmisión, ya que éste actualiza continuamente la probabilidad de éxito de una futura transmisión en función de la velocidad de envío. Sin embargo, esta actualización se realiza a través de las confirmaciones de los paquetes, por lo que este algoritmo no puede aplicarse directamente a comunicaciones multicast. Relacionado con ello, *EmPOWER* actualmente no proporciona soporte para comunicaciones multicast básicas y, por tanto, tampoco se encuentran implementadas ninguna de sus políticas de retransmisión. Por este motivo, añadir estas funcionalidades resulta esencial para el desarrollo posterior del esquema propuesto.

El modo de operación de *MCast* se divide en dos fases que se repiten durante el transcurso de una transmisión (véase la Figura 3). Sin embargo, estas fases no poseen la misma duración: la primera de ellas se extiende durante una décima parte del tiempo, mientras que el resto se dedica a la segunda fase. Esta división se realiza de acuerdo a las políticas de retransmisión utilizadas y busca introducir el mínimo tráfico adicional en la red, a la vez que hace posible obtener información sobre el estado del medio.

La primera fase comienza al inicio de una nueva transmisión, donde se utiliza la política de retransmisión DMS. Con ello, se pretende permitir a Minstrel recoger información sobre los receptores del grupo multicast. Tras finalizar este periodo, el con-

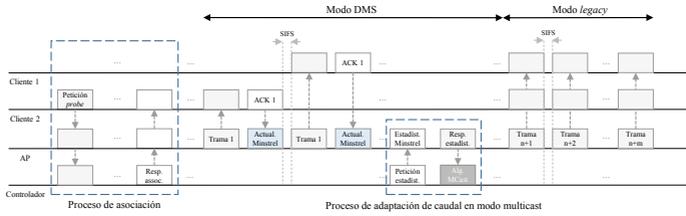


Fig. 3. Funcionamiento de *MCast*

trador solicita esta información a los APs de la red, de donde *MCast* toma los datos necesarios para ajustar el caudal de envío. Este ajuste sigue el enfoque que se describe a continuación. En primer lugar, el algoritmo selecciona para cada estación la velocidad de transmisión con la probabilidad de éxito más alta. A partir de estos datos, se establece un enfoque basado en la técnica del peor receptor. De este modo, se asegura que todas las estaciones reciben la información de forma correcta sin incrementar el ratio de paquetes perdidos.

Tras seleccionar la nueva velocidad de envío, la información se transmite en la segunda fase basándose en el enfoque clásico de multicast, donde el envío de conformaciones está deshabilitado. Sin embargo, en lugar de emplear una velocidad de transmisión básica, el controlador establece el valor calculado en la primera fase como el caudal de las comunicaciones en modo multicast. Una vez que esta segunda fase finalice, el proceso completo vuelve a iniciarse con el objetivo de actualizar la información estadística y permitir a *MCast* recalcular la nueva tasa de envío.

No obstante, en comunicaciones dirigidas a grupos multicast muy amplios, aun tratándose de un corto intervalo de tiempo, el envío de tramas en modo unicast a todas las estaciones podría conllevar la congestión de la red. Por este motivo, se propone un modo alternativo de funcionamiento de *MCast*, donde el envío de tramas unicast tan sólo se realiza para aquellos receptores con peor calidad de canal. No obstante, la identificación de estas estaciones no resulta una tarea trivial.

La enmienda IEEE 802.11k [18] fue definida para facilitar la recuperación de información estadística y adaptar así los parámetros de la red. Para ello se establece un mecanismo de petición y respuesta entre las estaciones y los APs, que permite obtener datos acerca de la potencia de la señal o del nivel de SNR, entre otros. A pesar de las mejoras introducidas, esta enmienda no ha sido aún incorporada por muchos dispositivos comerciales. De hecho, tan sólo los dispositivos Apple ofrecen esta funcionalidad [19]. Esta limitación hace que sea necesario explorar otras opciones. Scapy [20] es una potente herramienta de

manipulación de paquetes cuyas funcionalidades incluyen la generación y captura de paquetes o la decodificación de un amplio número de protocolos de red. Por ello, Scapy se considera una alternativa real para obtener información similar a la ofrecida por la enmienda IEEE 802.11k.

En nuestro entorno, Scapy se ejecuta continuamente en cada estación con el objetivo de conocer detalles como la cantidad de paquetes recibidos o la potencia de la señal. Así pues, estos datos resultan más que suficientes para conocer los peores receptores del grupo. Esta información es enviada periódicamente al controlador utilizando para ello el *REST-server* del controlador. De este modo, *MCast* es capaz de identificar las estaciones a las que debe enviar los paquetes en modo unicast durante la primera fase del algoritmo. Para ello, se asume que la velocidad de transmisión seleccionada será igualmente apropiada para el resto de destinatarios dado que poseen unas mejores condiciones de canal.

Como resultado de este enfoque, el tiempo en el que el canal se encuentra ocupado se reduce considerablemente. Este periodo es aún menor en el segundo modo de funcionamiento debido a la reducción de transmisiones unicast. Sin embargo, Scapy requiere del uso de una interfaz monitora que permita recoger los datos. Por ello, la adaptación de este segundo enfoque no resulta totalmente estándar, algo que no ocurre con el primer modo de funcionamiento y que es totalmente compatible con los dispositivos de red del mercado. En cualquiera de los casos, el resto de transmisiones de la red pueden aprovechar mejor el ancho de banda y mejorar como consecuencia el rendimiento global de la misma.

V. EVALUACIÓN PRELIMINAR

Dada la novedad del trabajo presentado, éste se encuentra en pleno desarrollo. Sin embargo, se ha llevado a cabo una evaluación preliminar que permita comprobar sus prestaciones e identificar posibles deficiencias que deban solventarse. En ella se recogen tan sólo los resultados del esquema compatible con el estándar IEEE 802.11 en el que todas las estaciones reciben las tramas unicast. El objetivo de estas pruebas es evaluar el rendimiento que *MCast* ofrece en comparación con la transmisión multicast donde no se realizan confirmaciones de paquetes y

con aquella donde se emplea DMS.

La evaluación de prestaciones se ha llevado a cabo en un entorno real de trabajo. Para ello, la experimentación se ha realizado sobre el *testbed 5G-EmPOWER* situado en Create-Net (Trento, Italia), donde se ha implantado el esquema de adaptación de caudal multicast desarrollado. Para esta evaluación, este *testbed* está formado por 3 puntos de acceso, los cuales se componen de placas PCEngines ALIX y una pareja de tarjetas de red inalámbricas. De este modo, se ofrece la posibilidad de trabajar tanto en la banda de frecuencia de 2.4 como en la de 5.2 GHz. El controlador de la red se sitúa sobre un portátil Dell E64100 equipado con un procesador Intel i7 y 8 GB de memoria RAM. Está máquina utiliza como sistema operativo Ubuntu 16.04 LTS.

Para llevar a cabo las pruebas se definen 5 grupos multicast, donde el tamaño del grupo oscila entre 1 y 5 receptores. Como receptores se han empleado 5 PCs convencionales: 2 equipos Dell E6540 y 3 equipos Dell E5450, ambos equipados con un procesador Intel Core i7 y 8GB de RAM. Estas máquinas emplean como sistema operativo Ubuntu 16.04 LTS y una versión actualizada de Windows 10. Además, para evitar la variabilidad de los resultados, para cada esquema se han realizado 5 repeticiones de cada prueba, manteniendo las condiciones de evaluación.

MCast define dos periodos de tiempo en su funcionamiento. Dicho periodo se ha establecido en 5 segundos, donde 0.5 segundos se dedican a la transmisión multicast empleando DMS y en los 4.5 segundos restantes la información es enviada sin esperar confirmación alguna. Todos los escenarios han sido evaluados mediante transmisiones de vídeo de un minuto, ya que se trata de un contenido más sensible a retrasos y pérdidas de paquetes. Para ello se han empleado secuencias de evaluación del estándar HEVC (*High Efficiency Video Coding*) como *Race Horses* y *Blowing Bubbles* y la herramienta FFmpeg [21], un potente entorno capaz de codificar, transmitir y recibir secuencias multimedia en tiempo real. Mediante esta herramienta se pretende obtener un comportamiento lo más real posible, ya que la información es transmitida en paquetes de distintos tamaños. Dado que se trata de comunicaciones multicast, la información se transmite mediante UDP, por lo que la única opción de recuperación de paquetes se encuentra a nivel de capa MAC.

La adaptación del caudal en las transmisiones permite igualmente reducir el tiempo de ocupación del canal durante las mismas. Dado que *MCast* transmite la información en el modo multicast habitual en el 90% de los casos, la cantidad de información extra transmitida durante la fase unicast resulta mínima. Esta evolución se presenta en la Figura 4. En ella puede observarse cómo DMS resulta totalmente inviable ya que la cantidad de datos enviados crece incesantemente a medida que lo hace el tamaño del grupo. Por contra, el enfoque clásico multicast mantiene constante el nivel de datos enviado. En este sentido, nuestra propuesta introduce una min-

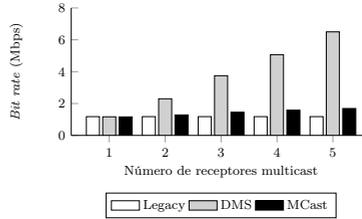


Fig. 4. *Bit rate* generado en función del número de receptores

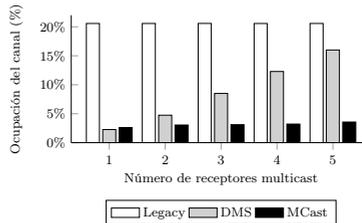


Fig. 5. Ocupación del canal según el número de receptores

ima cantidad adicional de datos con respecto a la transmisión multicast, que a penas se ve incrementado a pesar de aumentar el número de receptores y que resulta muy inferior al generado por DMS.

Como consecuencia de lo anterior, *MCast* es capaz de reducir la ocupación del canal drásticamente, tal y como se observa en la Figura 5. Es importante anotar que en ella tan sólo se muestra el tiempo necesario para transmitir los paquetes de datos, dejando fuera de la comparativa los tiempos de espera entre ellos. En las transmisiones multicast la información es enviada una sola vez independientemente del número de estaciones, por lo que la ocupación del canal permanece constante. Sin embargo, dada la baja velocidad de transmisión empleada, el tiempo que el canal permanece ocupado es demasiado alto. Por otro lado, debido a la réplica de paquetes, DMS prácticamente duplica este tiempo a medida que aparece un nuevo receptor en la red. En este caso, dada la escasa introducción de tráfico adicional, nuestro esquema mantiene prácticamente constante el tiempo de ocupación del canal, manteniéndose en todos los casos por debajo del resto de opciones.

A pesar de emplear una velocidad de transmisión más alta, *MCast* alcanza un rendimiento en las comunicaciones mayor de lo que lo hace el enfoque clásico. Esta situación puede observarse en la Figura 6. Este rendimiento resulta ligeramente superior cuando se utiliza la política DMS, ya que el envío en modo unicast es totalmente adaptado a las condiciones de cada receptor. No obstante, este descenso resulta prácticamente despreciable si se tiene

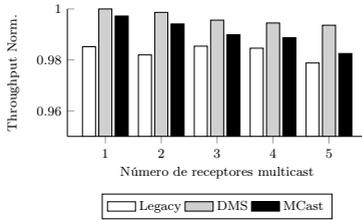


Fig. 6. Throughput multicast normalizado

en cuenta el nivel de rendimiento ofrecido y la optimización lograda en el uso del medio.

VI. CONCLUSIONES

En este artículo se propone un nuevo enfoque para la adaptación de la velocidad de las transmisiones multicast en redes IEEE 802.11, empleando para ello una arquitectura basada en SDN. Así, se lleva a cabo la implementación de un algoritmo, que es ejecutado sobre el controlador central de la red. Mediante la alternancia de políticas de retransmisión, el controlador es capaz de recuperar información sobre la calidad de canal percibida por cada receptor. De esta forma, el algoritmo cuenta con los datos necesarios para calcular la velocidad de envío más adecuada para todas las estaciones del grupo.

Los resultados muestran que nuestra propuesta cumple el objetivo inicial de adaptar la velocidad de transmisión en multicast. Además, se ha demostrado que, manteniendo un nivel de rendimiento similar al ofrecido por el mecanismo multicast clásico y por DMS, se consigue reducir drásticamente el tiempo de ocupación del canal y la cantidad de datos que se transmiten por la red.

VII. TRABAJO FUTURO

Esta nueva línea de investigación trae consigo una amplia cantidad de trabajo. Entre todo ello, resulta importante realizar una batería de pruebas más amplia que permita determinar si existen intervalos de trabajo más óptimos para *MCast*. Igualmente, se plantea un análisis más profundo de los resultados con el objetivo de optimizar el algoritmo y alcanzar o superar así el rendimiento ofrecido por DMS.

En relación con lo anterior, se incluye como un trabajo próximo la evaluación del enfoque de *MCast* dirigido a reducir aún más si cabe el *overhead* introducido en la red. Por último, se plantean propuestas que permitan la movilidad y migración de las estaciones dentro del entorno mostrado.

AGRADECIMIENTOS.

El presente trabajo ha sido financiado conjuntamente por el Ministerio de Economía y Competitividad y la Comisión Europea bajo el proyecto TIN2015-66972-C5-2-R (MINECO/FEDER) y la

beca BES-2013-065457.

REFERENCES

- [1] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," ANSI/IEEE Std 802.11, 2012.
- [2] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 7: Medium Access Control (MAC) Quality of Service (QoS)," ANSI/IEEE Std 802.11e, 2005.
- [3] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 3: MAC Enhancements for Robust Audio Video Streaming," ANSI/IEEE Std 802.11aa, 2011.
- [4] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.
- [5] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 8: Local and Metropolitan Area Networks," ANSI/IEEE Std 802.11v, 2011.
- [6] Joy Kuri and Sneha Kumar Kasera, "Reliable Multicast in Multi-Access Wireless LANs," *Wireless Networks*, pp. 359–369, 2001.
- [7] J. Villalón, P. Cuenca, Y. Seok L. Orozco-Barbosa, and T. Turetli, "ARSM: a cross-layer auto rate selection multicast mechanism for multi-rate wireless LANs," *IET Communications*, vol. 1, no. 5, pp. 893–902, 2007.
- [8] J. Villalón, P. Cuenca, Y. Seok L. Orozco-Barbosa, and T. Turetli, "Crosslayer Architecture for Adaptive Video Multicast Streaming over Multirate Wireless LANs," *IEEE Journal on Selected Areas in Communications*, pp. 699–711, 2007.
- [9] Y. Park, Y. Seok, N. Choi, Y. Choi, and J.-M. Bonnin, "Rate-adaptive multimedia multicasting over IEEE 802.11 wireless LAN," pp. 178–182, 2006.
- [10] Kandaraj Piamrat, Adlen Ksentini, Jean-Marie Bonnin, and Cesar Vihó, "Q-DRAM: QoS-Based Dynamic Rate Adaptation Mechanism for Multicast in Wireless Networks," pp. 1–6, 2009.
- [11] H. Moura, G. V. C. Bessa, M. A. M. Vieira, and D. F. Macedo, "Ethanol: Software defined networking for 802.11 Wireless Networks," in *2015 IFIP/IEEE International Symposium on Integrated Network Management*, 2015, pp. 388–396.
- [12] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazao, "Towards Programmable Enterprise WLANs with Odin," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 115–120.
- [13] Shahin Tajik and Ahmad Rostami, "MultiFlow: Enhancing IP Multicast over IEEE 802.11 WLAN," in *IFIP Wireless Days, WD 2013*, 2013, pp. 1–8.
- [14] R. Riggio, M. K. Marina, J. Schulz-Zander, S. Kulkinski, and T. Rasheed, "Programming abstractions for software-defined wireless networks," *IEEE Transactions on Network and Service Management*, pp. 146–162, 2015.
- [15] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek, "The Click Modular Router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.
- [16] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending Networking into the Virtualization Layer," in *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.
- [17] Dong Xia, Jonathan Hart, and Qiang Fu, "Evaluation of the Minstrel rate adaptation algorithm in IEEE 802.11g WLANs," in *IEEE International Conference on Communications*, 2013, pp. 2223–2228.
- [18] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Radio Resource Measurement of Wireless LANs," ANSI/IEEE Std 802.11k, 2008.
- [19] M. I. Sanchez and A. Boukerche, "On IEEE 802.11K/R/V amendments: Do they have a real impact?," *IEEE Wireless Communications*, vol. 23, no. 1, pp. 48–55, 2016.
- [20] "Scapy packet sniffer project," 2016.
- [21] "FFmpeg project," 2016.

Un Esquema de Colas Eficiente para Reducir HoL Blocking en Topologías Dragonfly con Encaminamiento Determinista Mínimo

Pedro Yébenes¹, Jesús Escudero-Sahuquillo¹, Pedro J. García¹, Francisco J. Quiles¹

Resumen— El número de nodos en los sistemas de computación de altas prestaciones ha crecido en los últimos años, haciendo que la red sea un componente fundamental en su diseño. Para poder interconectar sistemas con miles de nodos, la topología *Dragonfly* se ha hecho muy popular en los últimos años ya que ofrece una alta escalabilidad. Sin embargo, el rendimiento de la topología *Dragonfly* puede reducirse debido a los efectos negativos de la congestión, como por ejemplo el *Head-of-Line (HoL) blocking*. Aunque se han propuesto numerosas técnicas para tratar de evitar este efecto tan dañino, las más efectivas son aquellas diseñadas específicamente para una topología y un algoritmo de encaminamiento concretos. En este artículo presentamos un esquema de colas llamado *Hierarchical Two-Level Queuing (H2LQ)*, diseñado para reducir el *HoL blocking* en topologías *Dragonfly* totalmente conexas, que usen encaminamiento determinista de ruta mínima. Esta propuesta aumenta el rendimiento de la red y requiere menos recursos que otras técnicas. Además, las estructuras que empena H2LQ podrían explicarse también para mejorar otros esquemas de colas existentes.

Palabras clave— Redes de Altas Prestaciones, Topologías *Dragonfly*, *Head-of-Line blocking*, Control de Congestión

I. MOTIVACIÓN

LOS sistemas de computación de altas prestaciones han aumentado su capacidad de computación en los últimos años [1] y cada vez se está más cerca de alcanzar el reto Exascale [2]. Estos sistemas pueden incluir miles de nodos de cómputo o almacenamiento, interconectados mediante una red de altas prestaciones. Este tipo de redes tienen que cumplir unos requisitos exigentes respecto a la latencia y la productividad de cara a satisfacer las necesidades de comunicación de las aplicaciones que se ejecutan en estos sistemas. Por ello, el diseño de la red de interconexión es una cuestión fundamental en el diseño de los sistemas de computación de altas prestaciones. Sino es así, la red se puede convertir en el cuello de botella del sistema y provocar que este no tenga el comportamiento esperado. Este efecto se puede agravar aún más cuando el número de elementos interconectados crece.

Uno de los aspectos más importantes en el diseño de la red de interconexión es la topología [3], que se define como el patrón utilizado para interconectar los componentes de la red, como nodos, conmutadores y cables. Otros aspectos importantes en el diseño de la red, tales como el encaminamiento, la libertad de bloqueos o el consumo energético, están condiciona-

dos por la topología. En cuanto a la clasificación de las topologías de redes de interconexión, tradicionalmente se han utilizado topologías directas e indirectas. Las topologías directas son escalables en coste, pero no en prestaciones, mientras que las indirectas mantienen las prestaciones pero su coste crece exponencialmente con el tamaño de la red. Con el fin de ofrecer una topología que escale tanto en coste como en prestaciones, se han propuesto topologías para sistemas grandes, de tipo jerárquico y/o híbrido, como las topologías *Flattened butterfly* [4], *Dragonfly* [5] o *KNS* [6], que se benefician de la disponibilidad de conmutadores de alto grado del mercado. La topología *Dragonfly* ha ganado popularidad ya que es muy fácil de construir con el hardware actual, ofrece un alto ancho banda de bisección, diámetro bajo y una buena relación rendimiento/coste.

Aunque estas topologías son capaces de obtener un alto rendimiento, este puede verse mermado por a los efectos negativos derivados de las situaciones de congestión. El peor de estos efectos, y el que provoca una mayor degradación de prestaciones, se conoce como *Head-of-Line (HoL) blocking* [7]. Este efecto aparece en los conmutadores o en las interfaces de red cuando un paquete en la cabeza de una *buffer* bloquea al resto de paquetes de ese *buffer*, incluso si piden acceso a puertos o recursos que están disponibles.

En la literatura, se han propuesto muchas soluciones para combatir tanto la congestión en general como el *HoL blocking* particular. Realmente, cualquier solución orientada a en limitar la carga de tráfico en la red o evitar o eliminar situaciones de congestión puede disminuir la probabilidad de que aparezca el *HoL blocking*. Sin embargo, existen muchas técnicas diseñadas específicamente para reducir este efecto. La mayoría de ellas dividen los *buffers* de los puertos de los conmutadores en varias colas (por ejemplo, para soportar distintos canales virtuales [8]). Aprovechando esta organización, los distintos flujos de paquetes se asignan a diferentes colas para eliminar el *HoL blocking* producido entre estos flujos. Algunas de estas soluciones son capaces de prevenir completamente el *HoL blocking*, pero requieren recursos adicionales que no están disponibles en los conmutadores actuales [9], [10], [11]. También hay otros esquemas de colas factibles que reducen parcialmente el *HoL blocking* [12], [13], [14]. Entre estos esquemas de colas factibles, los más eficientes son aquellos diseñados específicamente para una topología de red y un algoritmo de encaminamiento concretos. En [15], [16], [17] se proponen varios ejemplos

¹Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha. E-mail: {pedro.yebenes, pedrojavier.garcia, fco.alfaro, francisco.quiles}@uclm.es

de estas técnicas que tienen en cuenta las propiedades de la topología y del algoritmo de encaminamiento para reducir el *Hol blocking* más eficientemente y/o requiriendo menos colas por puerto.

Siguiendo este enfoque, en este artículo analizamos cómo se comportan los flujos de tráfico en las topologías *Dragonfly* que utilizan un algoritmo de determinista de ruta mínima (conocido como MIN) [5] de cara a proponer un esquema de colas que reduzca eficazmente el *Hol blocking* en esta topología. En primer lugar, hemos estudiado la naturaleza y las causas específicas de la aparición de la congestión en las topologías de tipo *Dragonfly*. También hemos identificado cómo se origina y qué efectos tiene el *Hol blocking* en esta topología. Este análisis previo es clave para diseñar un buen esquema de colas que sea eficiente y escalable. Una vez que ya hemos estudiado las dinámicas de los flujos congestionados e identificado cómo se origina el *Hol blocking*, hemos diseñado un sencillo esquema de colas para reducir el *Hol blocking* y, por tanto, mejorar el rendimiento de la red. Téngase en cuenta que las topologías *Dragonfly* que utilizan el encaminamiento MIN no son libres de bloqueos por lo que se necesita que el sistema de colas prevenga los posibles interbloqueos que el algoritmo de encaminamiento podría generar en la red. Para ello, hemos agrupado las colas en dos redes virtuales: una red virtual principal donde se asignan los flujos de paquetes a colas de forma general y una red virtual «de escape» (ver Sección II) que resuelve las dependencias cíclicas (es decir, los interbloqueos). Cabe destacar que el esquema de colas que hemos diseñado, no solo utiliza colas separadas dentro de la red virtual principal, sino que también las usa en la red virtual de escape, ofreciendo una segunda etapa de prevención del *Hol blocking*, a la vez que se evitan los interbloqueos. Por este motivo, la hemos llamado *Hierarchical Two-Level Queueing (H2LQ)*.

El resto del artículo se organiza de la siguiente manera. La sección II describe la topología *Dragonfly* y el encaminamiento MIN. La sección III explica la congestión en la red y ofrece una visión global de las técnicas que intentan solucionar sus efectos adversos. Además, se analiza el comportamiento de los flujos de tráfico en redes *Dragonfly* con encaminamiento MIN. Nuestra propuesta para reducir el *Hol blocking* en topologías *Dragonfly* se describe en detalle en la sección IV y se evalúa en la sección V. Finalmente, en la sección VI se indican algunas conclusiones.

II. LA TOPOLOGÍA DRAGONFLY

La topología *Dragonfly* es una topología jerárquica que consiste en un conjunto de grupos, cada uno de ellos compuesto de varios conmutadores donde se conectan los nodos finales de cómputo o almacenamiento [5]. Los conmutadores que pertenecen al mismo grupo se interconectan mediante canales locales que forman la subred intra-grupo. A su vez, los grupos se conectan entre sí mediante la subred inter-grupo, a través de los canales globales de los conmutadores. Se puede utilizar cualquier topología para las subredes

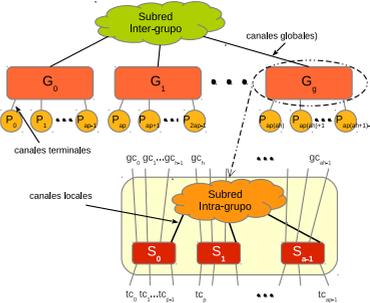


Fig. 1: Diagrama de la topología *Dragonfly*.

intra- o inter- grupo. Sin embargo, en [5] se sugiere el uso de topologías totalmente conexas en ambos niveles para mejorar su relación efectividad/coste. La Figura 1 muestra un diagrama genérico de esta topología. Con más detalle, la topología *Dragonfly* se puede definir mediante tres parámetros.

- a Número de conmutadores en cada grupo.
- p Número de nodos conectados a cada conmutador.
- h Número de canales por los que cada conmutador se conecta a otros grupos (canales globales).

Es decir, cada conmutador tiene p canales terminales para conectar con los nodos, $a - 1$ canales locales que componen la subred intra-grupo y h canales globales que forman la subred inter-grupo. Por tanto, en una red *Dragonfly* totalmente conexas hay $ap(a + 1)$ nodos, $ah + 1$ grupos y $a(ah + 1)$ conmutadores, cada uno con $p + a + h - 1$ puertos. Aunque los parámetros a , p y h pueden tener cualquier valor, se recomienda utilizar $a = 2p = 2h$ para balancear la carga del canal.

Hay varios tipos de algoritmos de encaminamiento propuestos para esta topología [5]. El más sencillo y efectivo es el algoritmo de camino mínimo (MIN). Este define tres pasos para encaminar un paquete desde un origen s conectado al conmutador S_s en el grupo G_s hasta un nodo destino d conectado a un conmutador S_d en el grupo G_d :

1. Si G_s es diferente de G_d y S_s no tiene una conexión global con G_d , los paquetes se encaminan dentro de G_s hacia un conmutador intermedio S_1 conectado con G_d .
2. Si G_s es diferente de G_d , los paquetes se envían desde el conmutador intermedio S_1 a través del canal global para llegar a otro conmutador S_2 en G_d .
3. Si el conmutador de llegada S_2 no es S_d , los paquetes se encaminan dentro de G_d de S_2 hacia S_d .

Sin embargo, el encaminamiento MIN no es libre de bloqueos. Para resolver este problema, también

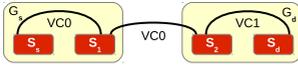


Fig. 2: Canales virtuales evitando interbloqueos.

en [5] se propone utilizar dos conjuntos disjuntos de canales virtuales (VCs), soportados por dos colas en cada puerto (ver la Figura 2). Básicamente, cada conjunto disjunto de VCs (como por ejemplo, el conjunto de todos canales VC0), forman una red virtual (VN), con distintos propósitos. Primero, los paquetes utilizan la VN principal (VCO) mientras están en su grupo origen o cuando viajan hacia otro grupo. Después, los paquetes moviéndose dentro del grupo destino (es decir, están realizando el paso #3 del proceso de encajamiento) se asignan a VN de escape (VC1) para romper los ciclos que aparecen en el grafo de dependencias entre canales. Téngase en cuenta que el VC de escape solo se utiliza en puertos de entrada de la red intra-grupo.

III. CONGESTIÓN: PROBLEMAS Y SOLUCIONES

A. El problema de la congestión y sus posibles soluciones

La congestión consiste en un colapso del tráfico en varios caminos dentro de la red, lo que degrada el rendimiento de la misma. El origen de la congestión es la contención, que ocurre cuando varios flujos de paquetes piden simultáneamente el acceso al mismo puerto de la salida dentro de un conmutador. La congestión también puede aparecer cuando un nodo destino, por cualquier motivo, no puede absorber los paquetes que le llegan desde la red al ritmo con el que los recibe. En estos casos, y asumiendo redes sin pérdidas¹, los paquetes a los que no se les concede el acceso permanecen bloqueados hasta que su petición de cruce es aceptada. Si esta situación continúa en el tiempo, los paquetes bloqueados retrasan el avance de otros paquetes en el mismo *buffer*. Si los *buffers* se llenan, la presión hacia atrás provocada por el control de flujo propaga la congestión a otros conmutadores. Finalmente, la congestión puede llegar a los nodos, lo que incrementa la latencia de los paquetes y degrada el rendimiento general de la red. Cabe destacar que en una situación de congestión, no solo los «flujos calientes» (flujos que contribuyen a la congestión) se ven afectados por el atasco del tráfico. De hecho, los flujos que no contribuyen a la congestión (llamados «flujos fríos») acaban avanzando a la misma velocidad que los flujos calientes porque ambos comparten los mismos *buffers*. Esto es un caso particular del efecto llamado *Head-of-Line (HoL) blocking*, que ocurre cuando un paquete que está pidiendo el acceso a un puerto de salida se bloquea y no deja avanzar a otros paquetes almacenados detrás suya en el mismo

¹Téngase en cuenta que las redes sin pérdidas, es decir, que no descartan paquete en caso de contención intensa, son la opción habitual en los sistemas de altas prestaciones, siendo *InfiniBand* el ejemplo más significativo de tecnología de red para estos sistemas.

buffer, incluso si estos paquetes van dirigidos a otros puertos de salida libres [18].

El *HoL blocking* derivado de situaciones de congestión se puede clasificar en dos tipos, dependiendo del lugar en que se origina la congestión. Por un lado, si el *HoL blocking* afecta a un flujo frío en el conmutador donde se origina la congestión, se llama *HoL blocking* de bajo nivel [7]. Por otro lado, si un flujo frío se ve afectado por la congestión propagada desde otros conmutadores, se conoce como *HoL blocking* de alto nivel [19]. Independientemente del tipo de *HoL blocking*, este efecto se puede considerar la principal causa de la degradación del rendimiento de la red tras la congestión. Realmente, si los flujos calientes no interactúan con los flujos fríos, la congestión no degradaría el rendimiento ya que los caminos de la red que utilizan los flujos calientes funcionan a la máxima velocidad [20].

Hay varios tipos de estrategias para tratar con la congestión. Por un lado, la limitación de inyección [21] utiliza la información de los conmutadores para detectar la congestión e informar a los nodos origen que contribuyen a la congestión de que deben reducir su tasa de inyección. Una vez eliminada la congestión, sus problemas derivados también desaparecen. Esta estrategia la utiliza la especificación *InfiniBand*. Sin embargo, presenta varios inconvenientes, ya que no escala con el tamaño de la red y las notificaciones pueden ser tan lentas que, una vez advertidos a los nodos para que reduzcan su inyección, la información sobre la congestión puede estar obsoleta.

Por otro lado, también se han propuesto otras estrategias basadas en la separación de flujos de datos en colas (o canales virtuales [8]) diferentes, y prevenir el *HoL blocking* causado entre ellos. Hay dos tipos de técnicas. Las primeras consisten en una asignación dinámica de flujos calientes a colas donde se identifican y se separan a estos flujos en colas diferentes. Algunas propuestas son la descrita para ATLAS [22], el mecanismo *Regional Explicit Congestion Notification (RECN)* [10] y *Efficient and cost-effective Congestion-Control (EcoCC)* [11]. Sin embargo, estas técnicas necesitan recursos adicionales que no están disponibles en dispositivos de red del mercado como mecanismos para detectar y separar flujos calientes, *Content-Addressable Memories (CAM)* para registrar los puntos congestionados en cada puerto. En el segundo tipo de técnicas, los paquetes se almacenan en diferentes colas según un criterio de asignación estático, independientemente de las condiciones del tráfico, intentando evitar el *HoL blocking* producido por esos flujos de paquetes. Algunas de estas soluciones tienen una implementación factible pero otras no. Por ejemplo, *Virtual Output Queues at network level (VOQ-net)* [9] asigna a los paquetes dirigidos a cada destino de red a una cola separada, necesitando tantas colas como nodos en la red. Esta técnica previene totalmente el *HoL blocking*, tanto de bajo como de alto nivel, aunque no es escalable. Entre las técnicas de implementación factible, *Virtual Output Queues at*

switch level (VOQsw) [12] asigna paquetes a colas en función del puerto de salida del conmutador, requiriendo tantas colas como puertos en el conmutador, así como *look-ahead routing* [23]. De esta manera, VOQsw previene totalmente el *Hol blocking* de bajo nivel, pero no el de alto nivel. Otras técnicas similares que reducen el *Hol blocking* son *Dynamically Allocated Multi-Queues* (DAMQs) [13], *Destination-Based Buffer Management* (DBBM) [14] y *Dynamic Switch Buffer Management* (DSBM) [24].

En general, las aproximaciones descritas no contemplan las propiedades ni de la topología ni del algoritmo de encaminamiento. Por el contrario, otras sí lo hacen como los esquemas *Output-Based Queue Assignment* (OBQA) [25] y *vTree* [16] diseñadas para topologías *fat-tree* que utilizan los algoritmos de encaminamiento deterministas [26] y [27], respectivamente. De una manera similar *Band-Based Queueing* (BBQ) [17] se adapta a las topologías *KNS* [6] con el algoritmo de encaminamiento *Hybrid-DOR*. En general, todos estos esquemas aprovechan al máximo las colas disponibles para separar los flujos de tráfico tanto como es posible, teniendo en cuenta la distribución de tráfico determinada por la topología y el algoritmo de encaminamiento. Siguiendo este enfoque, en este artículo proponemos un esquema de colas apropiado para las topologías *Dragonfly*.

B. Análisis de las situaciones de congestión en redes *Dragonfly*

La utilización del algoritmo de encaminamiento MIN implica que un paquete cruza como máximo cuatro conmutadores (cuatro saltos) desde su origen hasta su destino. La Figura 3 muestra los distintos tipos de caminos ofrecidos por el encaminamiento MIN. Hemos utilizado acrónimos para referirnos a las conexiones desde los nodos (EN), los canales intra-grupo (IG) y los canales inter-grupo (EG). Hemos denotado el camino desde A a B utilizando en número «2», por lo que se expresa como «A2B». A continuación se describen en profundidad los distintos caminos:

- **Desde los nodos** (EN) un paquete puede seguir uno de estos caminos (ver la Figura 3a):
- Hacia un nodo (EN2EN) conectado al mismo conmutador (flecha azul en la Figura 3a). El paquete realiza un único salto.
- Hacia la red intra-grupo (EN2IG). El nodo destino está en el mismo grupo pero conectado a otro conmutador o se encuentra en otro grupo sin conexión directa con el conmutador origen (flecha roja en la Figura 3a).
- Hacia la red inter-grupo (EN2EG). El nodo destino está en otro grupo conectado con el conmutador al que pertenece el nodo origen (flecha verde la Figura 3a).
- **Desde la subred intra-grupo** (IG) un paquete puede seguir uno de estos caminos (ver la Figura 3b):
- Hacia un nodo de conmutador (IG2EN). Los paquetes generados en grupos distintos se al-

macenan en el puerto de entrada en el canal virtual de escape (flecha azul en la Figura 3b).

- Hacia otro grupo externo conectado al conmutador (IG2EN). Es el segundo salto de un paquete ya que solo puede cambiar de grupo una vez (flecha verde en la Figura 3b).
- **Desde la subred inter-grupo** (EG) un paquete puede seguir los siguientes caminos (Figura 3c):
- Hacia un nodo del conmutador (EG2EN). En este caso la VN de escape no se utiliza (flecha azul en la Figura 3c).
- Hacia otro conmutador del mismo grupo (EG2IG). Los paquetes llegan de otro grupo pero sus destinos están conectados al conmutador por el que entran, por lo que efectúan su penúltimo salto. Una vez que llegan al siguiente conmutador, los paquetes se almacenan en el VC de escape (flecha roja en la Figura 3c).

En situaciones de congestión, es probable que este tipo de caminos causen *Hol blocking*, especialmente cuando aparece un *hotspot*. Como se describe en la Sección II, los paquetes se almacenan en el VC de escape (a partir de ahora, EVC) en los puertos de entrada cuando llegan a otros grupos. Si el EVC en un puerto de entrada se llena, los paquetes generados en otros grupos pero dirigidos a ese puerto de entrada desde los conmutadores del mismo grupo (flecha roja en la Figura 3c) se bloquearán debido al control de flujo. Después, la congestión se propaga hacia atrás ya que un grupo solo tiene una conexión con otro grupo. De hecho, si la cola de un puerto de entrada directamente conectada con la subred inter-grupo (EG) contiene un paquete dirigido a un conmutador cuyo EVC está bloqueado, todos los paquetes que llegan desde el grupo conectado a este puerto de entrada (los flujos EG2IG y EG2EN) no pueden avanzar. Esto puede ocurrir incluso si estos paquetes no están dirigidos al conmutador con el EVC bloqueado, por lo que sufrirán *Hol blocking* de alto nivel.

La Figura 4 muestra un ejemplo de la situación anterior. En ella hay un grupo de una red *Dragonfly* con nueve grupos ($a = 4, h = 2, p = 2$) con dos flujos de paquetes entrantes en cada puerto de entrada de la subred inter-grupo del conmutador 3 (S_3). Esos flujos EG2IG se dirigen a los nodos 0 y 1, así que el paquete en la cabeza de cada VC en esos puertos de entrada se dirige a un nodo conectado al conmutador 0 (S_0), compartiendo la misma conexión IG. Sin embargo, todos los paquetes mostrados en los puertos de entrada del S_3 no pueden avanzar incluso aquellos dirigidos a otros conmutadores que no son S_0 (todos se ven afectados por el *Hol blocking* de alto nivel). Además, los dos grupos conectados S_3 tampoco serán capaces de enviar paquetes hacia este grupo, ya que cada uno de los solo tiene una conexión con el grupo de la Figura 4.

Por otro lado, esta red puede sufrir *Hol blocking* de bajo nivel cuando los paquetes entrantes que llegan de grupos diferentes hacen su último salto y se

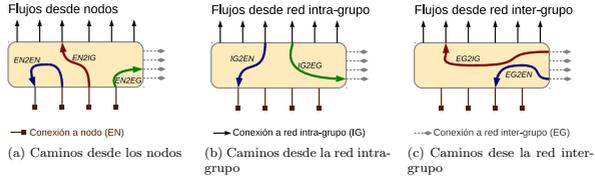


Fig. 3: Tipos de caminos en la topología *Dragonfly*.

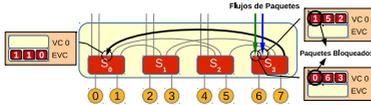


Fig. 4: Puertos de entrada bloqueados debido a la contención en un canal virtual de escape, produciendo *HoL blocking* de alto nivel.

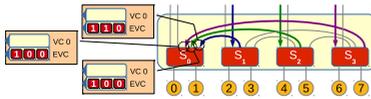


Fig. 5: *HoL blocking* de bajo nivel producido en el último salto de los paquetes con distintos destinos.

almacenen en el EVC, esperando para ser enviados al nodo destino. Como solo hay un único VC para estos paquetes, si varios puertos de entrada tienen en la cabeza de sus VCs paquetes dirigidos al mismo nodo, solo uno de ellos puede cruzar el conmutador. Un ejemplo de este comportamiento se muestra en la Figura 5. Los VCs tienen paquetes dirigidos al nodo 0, pero no pueden enviarse porque están bloqueados por los paquetes dirigidos al nodo 1.

IV. HIERARCHICAL TWO-LEVEL QUEUEING (H2LQ)

Una vez que se ha analizado la congestión en las redes *Dragonfly*, proponemos un sencillo esquema de colas que reduce el efecto *HoL blocking* derivado de la congestión y previene los interbloqueos. Asumimos que los conmutadores utilizados solo tienen *buffers* en los puertos de entrada (*input-queued (IQ) switches*) divididos en varias colas, cada una de ellas asignada un canal virtual (VC). Además, los VCs en los puertos se agrupan en dos redes virtuales (VNs): la VN principal y la VN de escape (EVN). Téngase en cuenta que el *HoL blocking* debe ser reducido tanto en la VN principal como en la EVN. Nuestro enfoque consiste en expandir ambas VNs para que tengan varios VCs. Este es el motivo por el que hemos llamado (en inglés) a esta técnica *Hierarchical Two-Level Queueing* (H2LQ). H2LQ ha sido específicamente diseñada para topologías *Dragonfly* total-

mente conexas que utilizan un algoritmo de enrutamiento MIN (Sección II).

A. Asignación de paquetes a VCs en H2LQ

H2LQ separa los flujos de tráfico en dos VNs: la VN principal y la EVN. El número de VCs que se deben configurar en cada VN está directamente relacionado con los parámetros *Dragonfly* a y p . Idealmente, cada *buffer* debería implementar a VCs para la VN principal y p VCs para la EVN. Sin embargo, como puede que no haya este número de VCs disponibles en los conmutadores, el número de VCs configurados para cada VN debe ser divisor de a o p , respectivamente, para asignar el mismo número de flujos a cada VC.

H2LQ asigna VCs a los paquetes en la VN principal según el conmutador donde se conecta su nodo destino, como indica la Ecuación 1. En la EVN, los paquetes se asignan a VCs basándose en su nodo destino como hace la Ecuación 2. De este modo, los flujos de paquetes dirigidos a conmutadores distintos se almacenan de manera separada siempre y cuando permanezcan en la VN principal. Mientras que en la EVN, los flujos de paquetes enviados a diferentes destinos no comparten VC.

$$VC_{Pr-VN} = \frac{\text{Destino} \% (a \times p)}{p} \% \#VC_{sPr-VN} \quad (1)$$

$$VC_{EVN} = \text{Destino} \% \#VC_{sEVN} \quad (2)$$

En la Ecuación 1, $\#VC_{Pr-VN}$ es el VC en la VN principal donde el paquete se almacenará en el siguiente conmutador utilizando su «Destino» como parámetro, a y p son los parámetros *Dragonfly* y $\#VC_{sPr-VN}$ es el número de VCs configurados en la VN principal. En la Ecuación 2, $\#VC_{EVN}$ es el EVC donde el paquete será almacenado en el siguiente conmutador utilizando su «Destino» como parámetro y $\#VC_{sEVN}$ es el número de VCs dentro de la EVN.

Por ejemplo, en una red *Dragonfly* configurada con $a = 8$, $h = 4$ y $p = 4$ se puede implementar 8 VCs en la VN principal y 4 VCs en la EVN (8+4). Si no hay 12 VCs en los puertos de entrada de los conmutadores, cualquier combinación de 8, 4 o 2 VCs puede ser utilizada en la VN principal y 4, 2 o 1 en la EVN, como por ejemplo 8+2, 4+1, 2+2, etc.

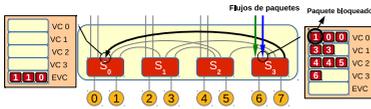


Fig. 6: Varios VCs previniendo el *HoL blocking* de alto nivel en un puerto de entrada.

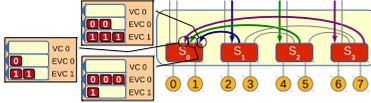


Fig. 7: Varios EVCs evitando el *HoL blocking* de bajo nivel en el salto final de los paquetes provenientes de otros grupos.

B. Ventajas de la asignación de H2LQ

Utilizar varios VCs en la VN principal reduce el *HoL blocking* de alto nivel que aparece en los puertos de entrada de los conmutadores conectados a otros grupos como se explica en la Sección III-B (y mostrado en la Figura 4).

La Figura 6 muestra el mismo escenario que la Figura 4, pero utilizando H2LQ. Cuando utilizamos H2LQ con 4 VCs en la VN principal, los paquetes dirigidos a diferentes conmutadores no comparten VC. Consecuentemente, aunque el *buffer* de entrada de S_0 se llene, los paquetes dirigidos a ese conmutador se almacenan en VC0, por lo que los paquetes almacenados en los VCs restantes (dirigidos a otros conmutadores) pueden fluir libremente. Si el VC0 en S_3 se llena, la congestión podría propagarse a otros grupos, pero solo los paquetes almacenados en VC0 se verían afectados.

Por otra parte, como se explica en la Sección III-B, el *HoL blocking* de bajo nivel puede aparecer en la EVN. La Figura 7 muestra el mismo escenario que la Figura 5, pero utilizando H2LQ. En este caso los paquetes dirigidos al nodo 0 no puede bloquearse por los paquetes que se dirigen al puerto que se conecta con el nodo 1, ya que se almacenan en un EVC diferente, y no se produce *HoL blocking* de bajo nivel.

En general, reducir el *HoL blocking* en ambas VNs mejora significativamente el rendimiento de la red, especialmente en situaciones en las que aparece un *hotspot*, como se muestra en evaluación de la siguiente sección. Además, la idea de utilizar varios VCs en la EVN y asignar los paquetes a VCs según la Ecuación 2 puede ser exportada a otros esquemas de colas para mejorarlos.

V. EVALUACIÓN

En esta sección se muestran los beneficios obtenidos cuando se utiliza H2LQ a través de resultados de varios experimentos. Todos los experimentos se han realizado con el simulador descrito en [28] basado en OMNeT++ [29] que se ha ampliado para incluir to-

pologías *Dragonfly* y el algoritmo de encaminamiento determinista MIN.

Para evaluar la escalabilidad de la propuesta, hemos considerado tres configuraciones diferentes de *Dragonfly* detalladas en la Tabla I. Todas las subredes, tanto inter-grupo como intra-grupo, de todas las configuraciones de la Tabla I son totalmente conexas. Además, en todas las configuraciones de red, asumimos enlaces serie segmentados *full-duplex* con un ancho de banda de 5 GB/s (40 Gbps), un retardo de la propagación en el enlace de 6 nanosegundos (es decir, una longitud de 1.2 metros y un retardo de 5 ns/m), tanto como para los enlaces entre conmutadores como entre nodos y conmutadores. En todos los casos la técnica de conmutación es *Virtual Cut-Through*, la política de control de flujo se basa en créditos y el tamaño de los paquetes es de 4096 B. Además, la arquitectura de conmutador modelada tiene los *buffers* en los puertos de entrada (*input-queued (IQ) scheme*). Los *buffers* pueden almacenar 64 paquetes, siendo su tamaño 256 KB.

TABLA I: Configuraciones *Dragonfly* Evaluadas

#	Parámetros			Nodos	Conmutadores	Puertos por Conmutador
	a	h	p			
1	6	3	3	342	114	11
2	8	4	4	1056	264	15
3	10	5	5	2550	510	19

Nuestra propuesta ha sido evaluada junto a otros esquemas de colas del estado del arte para ver las diferencias de prestaciones que hay entre ellos. Téngase en cuenta que estos esquemas no evitan los interbloques, excepto VOQnet que normalmente ofrece prevención de interbloques. Por tanto, hemos combinado estos esquemas con una EVN que consiste de varios EVCs para una comparación justa con nuestra propuesta. El nombre utilizado para referirnos a las técnicas tiene el patrón: *técnica-X.Y*, donde X es el número de VCs utilizados en la VN principal e Y el número de EVC utilizados en la EVN. Si Y es 1, no hay reducción de *HoL blocking* en la EVN, solo eliminación de interbloques, pero si Y es mayor que 1, los EVCs se asignan de acuerdo a la Ecuación 2. X e Y son normalmente parámetros de la red *Dragonfly*, como a y p , por tanto varían dependiendo de la configuración de la red. Los esquemas de colas modelados son los siguientes:

- **Deadlock Avoidance (DLA).** Se han evaluado las siguientes configuraciones: DLA-1.1 y DLA-1.p. La primera es el esquema básico que no reduce el *HoL blocking* y que utiliza dos colas, una de ella utilizada como EVC. La segunda utiliza p EVCs y muestra lo que ocurre cuando no hay prevención de *HoL blocking* en la VN principal pero sí en la EVN.
- **Virtual Output Queues at network level (VOQnet)** [9]. Como se describe en la Sección III, VOQnet utiliza en cada puerto de entrada una cola por cada nodo en la red necesitando una gran cantidad de memoria RAM. Asumi-

mos un mínimo de dos paquetes por cola, por tanto el tamaño del *buffer* es mayor que en otras técnicas (≈ 2.7 MB para la configuración #1, ≈ 8.3 MB para la configuración #2 y ≈ 20 MB para la configuración #3 de la Tabla 1). Cabe destacar que este esquema no tiene una implementación factible pero es la referencia básica de la máxima prevención del *Hol blocking*. Esta técnica no necesita prevenir los interbloqueos ya que no hay ciclos en su grafo de dependencia de canales.

- **Virtual Output Queues at switch level (VOQsw)** [12]. Se ha evaluado una configuración VOQsw-r.p. La VN principal configura en cada puerto de entrada tantas colas como puertos tiene el conmutador: r (columna «Puertos por Conmutador» en la Tabla 1), así pues los paquetes en la red principal se asignan a esas colas de acuerdo a su puerto de salida del conmutador. Además, esta técnica necesita de *look-ahead routing* como se describe en la Sección III.
- **Destination-Based Buffer Management (DBBM)** [14]. Este esquema asigna un VC en la VN principal a un paquete según la fórmula: $\text{Destino}\% \#VCs$, siendo «Destino» el destino del paquete y «#VCs», el número de VCs dentro de la VN principal. Se han evaluado dos configuraciones: DBBM-a.p y DBBM-a.1.
- **Hierarchical Two-Levels Queuing (H2LQ)**. Es nuestra propuesta, como se describe en la Sección IV. Se han evaluado las configuraciones H2LQ-a.p y H2LQ-a.1.

En las siguientes subsecciones, se han probado estos esquemas en cada configuración de red utilizando los patrones de tráfico uniforme y *hotspot*. En ambos casos, para evaluar el impacto de la carga de tráfico en la red, el ratio de generación de los nodos se incrementa un 10% desde el 10% hasta el 100% de carga, con respecto al ancho de banda del enlace.

Las métricas utilizadas en la evaluación son la productividad normalizada con respecto al ancho de banda del enlace y la latencia de paquete. Las medidas se comienzan a tomar una vez que la red ha llegado a un estado estable después de un período de calentamiento en el que se ha introducido en la red tráfico suficiente para rellenar los *buffers*. Después, se generan y marcan algunos paquetes para que sean los únicos utilicen para medir la latencia. Una vez que todos estos paquetes llegan a sus destinos, la simulación acaba. Esto se hace para incorporar las medidas de los paquetes con latencias altas.

A. Resultados de tráfico uniforme

Los resultados para el patrón de tráfico uniforme se muestran en la Figura 8. Este patrón de tráfico consiste en generar paquetes a destinos aleatorios de la red. La métrica utilizada para evaluar este escenario es la productividad con respecto a la latencia de paquete.

Como se observa a través de la Figura 8a, la Figura 8b y la Figura 8c, los resultados para el tráfico

uniforme son muy similares, independientemente del tamaño de la red. Como era de esperar, VOQnet obtiene el mejor rendimiento. El peor lo obtiene DLA-1.p, siendo incluso peor que el de DLA-1.1. Aunque DLA-1.p utiliza un esquema de colas en la EVN, no es capaz de explotarlo eficientemente. Además, utilizar más colas en el mismo *buffer* hace más pequeñas las colas de la VN principal. Por tanto, reducir el *Hol blocking* en la EVN no es útil sino se reduce en el VN principal.

Cuando no se utilizan varios EVCs en la EVN, DBBM-a.1 y H2LQ-a.1 obtienen un resultado similar, pero nuestra propuesta obtiene un mejor rendimiento utilizando los mismos recursos. Si se utilizan varios EVCs en la EVN, H2LQ-a.p obtiene un rendimiento similar a VOQsw-r.p pero necesitando la mitad de colas. Además nuestra propuesta obtiene un mejor rendimiento que DBBM-a.p en todas las configuraciones de red consideradas. Como conclusión de estos resultados, H2LQ obtiene el mayor ratio rendimiento/VCs. Si solo se utilizan varios VCs en la VN principal con H2LQ, el rendimiento se incrementa un 15%, pero si además se utilizan varios EVCs, el rendimiento se incrementa un 15% adicional. Además, este incremento de rendimiento también se aplica a DBBM.

B. Resultados de tráfico hotspot

El patrón de tráfico *hotspot* es muy común en las bibliotecas de comunicación paralelas como MPI cuando un proceso invoca la función *gather*. Básicamente, consiste en varios nodos enviando paquetes a un único nodo, llamado *hotspot*. En nuestros experimentos el 25% de los nodos de la red envían paquetes a uno de los cuatro posibles destinos *hotspot*. Por tanto, cada *hotspot* recibe tráfico del 6.25% de los nodos de la red. Los nodos restantes (75%) generan tráfico uniforme. Como la Figura 9a, la Figura 9b y la Figura 9c muestran para estos escenarios, los resultados obtenidos son muy similares para todos los tamaños de red.

Al igual que antes, VOQnet obtiene el mejor resultado ya que elimina completamente el *Hol blocking*. Por un lado ambas variantes de DLA consiguen un rendimiento muy bajo porque no son capaces de tratar este efecto tan dañino. VOQsw-r.p, DBBM-a.p y H2LQ-a.p obtienen resultados muy similares, requiriendo las dos últimas técnicas la mitad de recursos. DBBM-a.1 siempre consigue un rendimiento entre un 10% y un 15% menor, dependiendo de la configuración de red, que las técnicas anteriores. Por el contrario, H2LQ-a.1 es capaz de mantener el rendimiento más cercano a los esquemas de colas evaluados que utilizan EVCs en la EVN. Esto muestra que la asignación de flujos a VCs de la Ecuación 1 es más eficiente para reducir el *Hol blocking* que la asignación de DBBM.

En resumen, nuestra propuesta ha sido diseñada teniendo en cuenta el comportamiento de los flujos de la topología *Dragonfly*, por lo que es muy eficiente para tratar el *Hol blocking* en situaciones de conges-

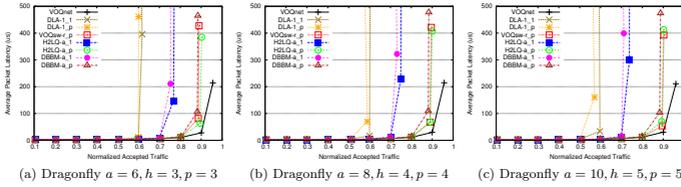


Fig. 8: Resultados para tráfico uniforme. Latencia media de paquete versus tráfico aceptado normalizado.

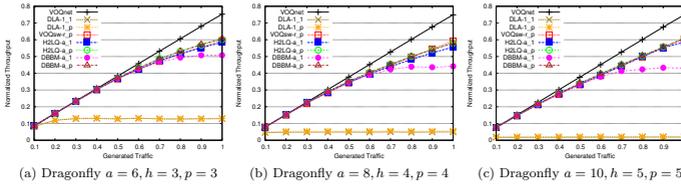


Fig. 9: Resultados para el tráfico hotspot. Rendimiento normalizado versus tráfico generado.

tión. De hecho, nuestra propuesta consigue un rendimiento similar a otros esquemas de colas, necesitando una menor cantidad de recursos. Además, otros esquemas de colas se pueden beneficiar del uso de EVCs en la EVN, como hace DBBM.

VI. CONCLUSIÓN

Para prevenir el *HoL blocking* en topologías *Dragonfly* totalmente conexas de una manera eficiente, en este artículo presentamos el esquema de colas *Hierarchical Two-Level Queuing (H2LQ)*. Primero, analizamos y describimos el comportamiento de los flujos de tráfico en este tipo de redes cuando se utiliza el algoritmo de encajamiento de camino mínimo (MIN). Este análisis nos permite entender cómo se genera el *HoL blocking* en esta topología para encontrar la mejor manera de combatirlo.

Después describimos la propuesta que soluciona este problema dividiendo el tráfico en varias colas o canales virtuales en la red virtual principal, y también en la escape.

Mediante los resultados de simulación presentados en este artículo, podemos concluir que la técnica propuesta es capaz de reducir el *HoL blocking* mientras mantiene el rendimiento de la red durante situaciones de congestión, requiriendo menos recursos que otras técnicas. También se muestra cómo otros esquemas de colas pueden ser optimizados para mejorar su rendimiento utilizando varias colas en la red virtual de escape, tal y como hace nuestra propuesta.

AGRADECIMIENTOS

Este trabajo ha sido conjuntamente financiado por MINECO y la Comisión Europea (fondos FEDER) con los proyectos TIN2012-38341-C04 y TIN2015-

66972-C5-2-R y la beca FPI BES-2013-063681; por la Junta de Comunidades de Castilla-La Mancha con el proyecto PEI1-2014-028-P; y por la Universidad de Castilla-La Mancha y la Comisión Europea (fondos FEDER) con un contrato de acceso al Sistema Español de Ciencia, Tecnología e Innovación (SECTI).

REFERENCIAS

- [1] "Top 500 List." www.top500.org.
- [2] U.S. Department of Energy, "Top Ten Exascale Research Challenges. DOE ASCAC Subcommittee Report," 2014.
- [3] J. Duato, Sudhakar Yamalanchili, and Lionel Ni, *Interconnection Networks: An Engineering Approach*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
- [4] John Kim, William J. Dally, and Dennis Abts, "Flat-tened butterfly: a cost-efficient topology for high-radix networks," in *Proceedings of the 34th annual international symposium on Computer architecture*, New York, NY, USA, 2007. ISCA '07, pp. 126-137, ACM.
- [5] John Kim, William J. Dally, Steve Scott, and Dennis Abts, "Technology-Driven, Highly-Scalable Dragonfly Topology," *SIGARCH Comput. Archit. News*, vol. 36, no. 3, pp. 77-88, June 2008.
- [6] R. Penaranda, C. Gomez, M.E. Gomez, P. Lopez, and J. Duato, "A New Family of Hybrid Topologies for Large-Scale Interconnection Networks," in *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, August 2012, pp. 220-227.
- [7] M. J. Karol, M. C. Hucjaj, and S. P. Morgan, "Input versus output queuing on a space-division packet switch," *IEEE Transactions on Communications*, vol. COM-35, pp. 1347-1356, 1987.
- [8] William Dally, "Virtual-Channel Flow Control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-205, 1992.
- [9] William Dally, P. Carvey, and L. Dennison, "Architecture of the Avici terabit switch/router," in *6th Hot Interconnects*, 1998, pp. 41-50.
- [10] P.-J. Garcia, F. J. Quiles, J. Flich, J. Duato, I. Johnson, and P. Navem, "Efficient, Scalable Congestion Management for Interconnection Networks," *Micro, IEEE*, vol. 26, no. 5, pp. 52-66, 2006.
- [11] J. Escudero-Sahuquillo, E.G. Gran, P.-J. Garcia-Garcia, J. Flich, T. Skeie, O. Lysne, F.J. Quiles, and J. Duato,

- "Efficient and Cost-Effective Hybrid Congestion Control for HPC Interconnection Networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.
- [12] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-Speed Switch Scheduling for Local-Area Networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–352, November 1993.
- [13] Y. Tamir and G.L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Trans. on Computers*, 1992.
- [14] T. Nachiondo, J. Flich, and J. Duato, "Buffer Management Strategies to Reduce HoL Blocking," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 6, pp. 739–753, June 2010.
- [15] J. Escudero-Sahuquillo, P. J. García, Francisco J. Quiles, J. Flich, and J. Duato, "OBQA: Smart and cost-efficient queue scheme for Head-of-Line blocking elimination in fat-trees," *J. Parallel Distrib. Comput.*, vol. 71, no. 11, pp. 1460–1472, November 2011.
- [16] Wei Lin Guay, Bartosz Bogdanski, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie, "vFtree - A Fat-Tree Routing Algorithm Using Virtual Lanes to Alleviate Congestion," in *Proc. of IPDPS*, 2011, pp. 197–208.
- [17] Pedro Yebenes Segura, Jesus Escudero-Sahuquillo, Cristian Gomez Requena, Pedro Javier Garcia, Francisco J. Quiles, and Jose Duato, "BBQ: A Straightforward Queuing Scheme to Reduce HoL-Blocking in High-Performance Hybrid Networks," in *Euro-Par 2013 Parallel Processing*, vol. 8097, pp. 699–712, 2013.
- [18] William James Dally and Brian Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
- [19] M. Jurczyk and T. Schwederski, "Phenomenon of Higher Order Head-of-Line Blocking in Multistage Interconnection Networks under Nonuniform Traffic Patterns," *IEICE Transactions on Information and Systems*, vol. E79-D, no. 8, pp. 1124–1129, Aug. 1996.
- [20] Timothy Mark Pinkston and José Duato, "Appendix E," in *Computer Architecture: A Quantitative Approach*, Elsevier, Ed. Morgan Kaufmann Publishers, 2006.
- [21] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control," in *Proc. of Int. Workshop HPI-DC*, 2005.
- [22] M. Katevenis, D. Serpanos, and E. Spyridakis, "Credit-flow-controlled ATM for MP interconnection: The ATLAS I single-chip ATM switch," in *High-Performance Computer Architecture, 1998. Proceedings., 1998 Fourth International Symposium on*, Feb 1998, pp. 47–56.
- [23] Jih-Kwon Peir and Yann-Hang Lee, "Look-ahead routing switches for multistage interconnection networks," *Journal of Parallel and Distributed Computing*, vol. 19, no. 1, pp. 1–10, sep 1993.
- [24] Wladek Olesinski, Hans Eberle, and Nils Gura, "Scalable alternatives to virtual output queuing," in *Proc. IEEE ICC*, 2009, pp. 1–6.
- [25] Jesus Escudero-Sahuquillo, Pedro J. Garcia, Francisco J. Quiles, Sven-Arne Reinemo, Tor Skeie, Olav Lysne, and Jose Duato, "A New Proposal to Deal with Congestion in InfiniBand-based Fat-trees," *J. Parallel Distrib. Comput.*, vol. 74, no. 1, pp. 1802–1819, Jan. 2014.
- [26] C. Gomez, F. Gilabert, M.E. Gomez, P. Lopez, and J. Duato, "Deterministic versus Adaptive Routing in Fat-Trees," in *Workshop CAC in conjunction with the IPDPS*, March 2007, p. 235.
- [27] Eitan Zahavi, Greg Johnson, Darren J. Kerbyson, and Michael Lang, "Optimized InfiniBand™ fat-tree routing for shift all-to-all communication patterns," *Journal of CCPE*, vol. 22, no. 2, pp. 217–231, 2010.
- [28] P. Yebenes, J. Escudero-Sahuquillo, P.J. Garcia, and F.J. Quiles, "Towards Modeling Interconnection Networks of Exascale Systems with OMNet++," in *Parallel, Distributed and Network-Based Processing*, 2013.
- [29] Andrés Varga, *OMNeT++ User Manual*, OpenSim Ltd.

Integración de Herramientas de Simulación con OpenFabrics Software para el Modelado de Redes de Interconexión InfiniBand

German Maglione-Mathey, Pedro Yebenes, Jesus Escudero-Sahuquillo, Pedro J. Garcia y Francisco J. Quiles¹

Resumen— El diseño de redes de interconexión es uno de los aspectos fundamentales para los sistemas de altas prestaciones (HPC) en la era del Exascale. Las decisiones de diseño que involucran la selección de la topología de red, los algoritmos de encaminamiento, los mecanismos de tolerancia a fallos y el control de congestión son cruciales para obtener el mayor rendimiento posible de la red. En este sentido, desde el punto de vista del software de control de las redes de interconexión, se invierten grandes cantidades de recursos en crear abstracciones software (*middleware*) compatibles con diferentes tecnologías de interconexión lo que permite la interoperabilidad entre las distintas tecnologías. Como por ejemplo el OpenFabrics Software (OFS) utilizado en redes de altas prestaciones donde se requieren bajas latencias de comunicación y alto ancho de banda. OFS es compatible con InfiniBand, iWarp y RoCE. Uno de los desafíos en el diseño de nuevas características que mejoren el rendimiento de las redes de interconexión es la necesidad de utilizar herramientas de modelado y simulación para representar la latencia introducida al tráfico de la red por cada una de las partes que componen OFS. Los diseñadores de redes de interconexión utilizan herramientas de simulación que modelan nuevas características de la red, con el objetivo de representar fielmente su comportamiento y diseñar nuevas arquitecturas de red. Sería muy interesante si algunos de los módulos de OFS se comunicarán con herramientas de simulación, ya que esto haría posible integrar la capa *middleware* con herramientas de simulación para propósitos de testeo y análisis de prestaciones de los nuevos modelos de red en conjunto con la capa *middleware*. En este trabajo presentamos una capa software para integrar algunos módulos de OFS con herramientas de simulación basadas en OMNeT+++. Además, proponemos un conjunto de herramientas para analizar las propiedades de diferentes topologías de red, que servirán de ayuda a los diseñadores de red en la toma de decisiones de diseño y configuración.

Palabras clave— Redes de Interconexión, InfiniBand, Simulación, Modelado, Integración, Herramientas.

I. INTRODUCCIÓN

EL procesamiento y análisis de datos generados por aplicaciones científicas y de Big Data requieren una gran potencia de cómputo, que generalmente se consigue mediante el uso de estructuras de computación de alto rendimiento (High Performance Computing, HPC), como los supercomputadores. Tianhe-2 (MilkyWay-2), el supercomputador desarrollado por la Universidad Nacional de Tecnología de Defensa de China, es actualmente el supercomputa-

dor más potente del mundo, con un rendimiento de 33,86 PETAFL0P/s (1 PETAFL0P/s = mil billones de operaciones de punto flotante por segundo) según el benchmark Linpack [1]. Cabe destacar que el objetivo principal para esta década en el campo del HPC es alcanzar el nivel de computación Exascale [2], lo que significa un rendimiento de exaflop/s (10¹⁸ operaciones de punto flotante por segundo) para el año 2023.

Los sistemas HPC generalmente requieren redes de interconexión de baja latencia y alto ancho de banda. Estas redes permiten altas velocidades de comunicación entre procesos ejecutándose en diferentes nodos de cómputo. Sin embargo, mientras que los sistemas HPC aumentan de tamaño y se reduce el coste de los nodos de cómputo, la red de interconexión debe también optimizar su diseño para abaratar su coste, y optimizar para obtener las mejores prestaciones con el mínimo de recursos necesarios. En efecto, la red de interconexión debe cumplir con los altos requisitos de comunicación de las aplicaciones actuales o de otra manera se convertirá en el cuello de botella del sistema. En consecuencia, cada aspecto de diseño de la red de interconexión (topología, encaminamiento, etc.) debe ser analizado cuidadosamente con el fin de obtener comunicaciones de gran ancho de banda, baja latencia, escalabilidad, y bajo consumo eléctrico.

Con respecto a las tecnologías de redes de interconexión de altas prestaciones, actualmente la arquitectura InfiniBand (IBA) [3] es la más utilizada en clusters HPC. En Noviembre del 2015 el 47,4% de los supercomputadores más potentes del mundo utilizan InfiniBand como tecnología de red de interconexión. Entre otras características IBA ofrece flexibilidad para implementar cualquier topología de red y mecanismos para configurar los algoritmos de encaminamiento necesarios. Además, pueden utilizarse canales virtuales, o Virtual Lanes (VL) en terminología InfiniBand, para garantizar que no existen inter-bloques, para proporcionar calidad de servicio (Quality-of-Service, QoS) o para ofrecer control de la congestión.

Por otra parte, los modelos de simulación se utilizan ampliamente en varios campos de investigación, con el objetivo de probar y verificar nuevas técnicas sin implementarlas en sistemas reales lo que supone, por ejemplo, un ahorro en infraestructura, entre otras múltiples ventajas. En particular, en el caso de las redes de interconexión de altas prestaciones,

¹Depto. de Sistemas Informáticos, Universidad de Castilla-La Mancha, Campus Universitario s/n. 02071, Albacete, España. {german.maglione@dsi.uclm.es, {pedro.yebenes, jesus.escudero, pedrojavier.garcia, francisco.quiles}@uclm.es

los modelos de simulación se utilizan para la evaluación del rendimiento, lo que permite comparar directamente diferentes técnicas y diseños con diferentes cargas de trabajo.

Una de las alternativas más conocidas para desarrollar simuladores de redes de interconexión es OMNeT++ [4], un *framework* que ofrece un conjunto de componentes para crear modelos de simulación (además de un conjunto de bibliotecas y un entorno de desarrollo). Además, en el caso específico de la tecnología InfiniBand puede emplearse un simulador llamado *ibsim*. Este simulador es parte de las herramientas disponibles en el Open Fabric Software [5] (básicamente el software de control del hardware IBA) y modela en detalle cada componente de una red InfiniBand. Sin embargo, y aunque un gran nivel de detalle a la hora de modelar el hardware real es una gran ventaja, tiene el inconveniente de encontrarse limitado a redes de tamaños reducidos. Además, no cuenta con la infraestructura necesaria para generar distintos patrones de tráfico o modelar las cargas de trabajo necesarias para las evaluaciones de rendimiento. Por contra, existen modelos de InfiniBand basados en OMNeT++, como los descritos en [6] y [7], que son capaces de modelar redes de tamaño superior pero no pueden utilizar directamente la información que proporciona el hardware real, teniendo que programar *scripts ad hoc* para cada caso [6]. Otro de los inconvenientes principales de estos modelos es la necesidad de definir el patrón de interconexión para cada red de forma manual.

Esta situación lleva a la duplicación de tareas cuya consecuencia es la proliferación de *scripts ad hoc* que aumentan la probabilidad de cometer errores. La duplicación de código y depuración de errores consumen una parte significativa del tiempo dedicado a la investigación.

En este trabajo presentamos el diseño y la implementación de una nueva capa de software para integrar, por un lado, el software de control del hardware InfiniBand (OpenFabrics Software) y, por otro lado, los simuladores y bibliotecas disponibles para el modelado y simulación de redes de interconexión. El nombre que hemos dado a esta nueva capa software es RAAP HPC Tools. Las RAAP HPC Tools servirán de nexo de unión entre el OpenFabrics Software, las Third-Party Libs (conjunto de bibliotecas disponibles para la generación de topologías y algoritmos de encaminamiento, generación de patrones de tráfico, etc.), los simuladores y el hardware InfiniBand.

Este artículo está organizado de la siguiente manera: En la Sección II se describe la arquitectura InfiniBand; en la Sección III se introduce el *framework* OMNeT++ para el desarrollo de herramientas de simulación; en la Sección IV se describen las RAAP HPC Tools. Y finalmente, las conclusiones de este trabajo pueden verse en la Sección V.

II. INFINIBAND

La arquitectura InfiniBand (IBA) [3], especifica una tecnología de interconexión de alto rendimiento para clusters HPC, que provee baja latencia y un alto ancho de banda con una baja sobrecarga de procesamiento. InfiniBand define una red de interconexión de nodos (de procesamiento y/o almacenamiento), mediante el uso de enlaces, conmutadores y routers. Esta red es independiente del sistema operativo y arquitectura del procesador de los nodos finales. InfiniBand soporta anchos de banda en los enlaces, superiores a otras tecnologías como Ethernet, además de ofrecer mejor latencia y rendimiento. En este aspecto cabe destacar que la hoja de ruta de la IBA Trade Association [8], prevé que los enlaces InfiniBand lleguen a 600 Gbit/s, 12x HDR (high-data rate) en 2017.

Cada nodo en una red InfiniBand necesita una tarjeta Host Channel Adapter (HCA). Los conmutadores tienen más de un puerto InfiniBand y reenvían paquetes desde un puerto a otro para continuar con la transmisión del paquete dentro de la subred¹. Un router se utiliza para reenviar paquetes entre subredes si es necesario. La gestión de la red se realiza utilizando conceptos de las redes definidas por software, utilizando interfaces abiertas y estándares de la industria, controlando los elementos físicos de la red e introduciendo características de ingeniería de tráfico. Específicamente, el Subnet Manager (SM) es una entidad software que configura y mantiene operativa su red local. El SM descubre la topología subyacente e inicializa la red, asigna identificadores únicos a todos y cada uno de los elementos de la red, determina las unidades de transferencia máximas (MTUs) y rellena las tablas de encaminamiento de acuerdo con el algoritmo de encaminamiento elegido.

Cada dispositivo en la subred tiene un identificador local (LID), definido como una dirección de 16 bits asignada por el SM. Todos los paquetes enviados dentro de una subred utilizan, a nivel de enlace, el LID como dirección de destino para el reenvío y la conmutación de paquetes. El tamaño del LID permite subredes de hasta 48.000 nodos finales. Se asignan nuevos LIDS cada vez que una subred se reconfigura. InfiniBand utiliza encaminamiento distribuido basado en destino. Cada conmutador tiene una tabla de búsqueda (Linear Forwarding Table o LFT) que relaciona puerto de salida con LID de destino.

InfiniBand soporta calidad de servicio (QoS) creando canales virtuales (VLs). Los canales virtuales (Virtual Lanes en terminología InfiniBand) son enlaces lógicos separados que comparten un mismo enlace físico. Cada enlace puede soportar hasta 15 VLs estándares (desde VL0 hasta VL14) y un VL de control (VL15), la prioridad de los VL se asigna de forma decreciente, el VL15 tiene la mayor prioridad y el VL0 la menor. A cada paquete de datos se le asigna un nivel de servicio (SL) para asegurar la calidad de servicio (QoS) requerida. Cada router, conmutador

¹En InfiniBand una subred es el término para delimitar un conjunto de nodos interconectados únicamente por conmutadores, a nivel de capa de red.

y HCA tienen una tabla por puerto que relaciona los SLs con los VLs (SL2VL), que rellena el SM, se usa con la finalidad de mapear los SLs asignados a los paquetes a los VLs soportados en ese enlace.

III. EL FRAMEWORK OMNeT++

Una simulación es la imitación del funcionamiento de los procesos del mundo real o del un sistema a lo largo del tiempo. La simulación implica la generación de una historia artificial del sistema y su observación para generar conclusiones sobre las características del funcionamiento del sistema real [9].

El comportamiento de un sistema a medida que evoluciona a lo largo del tiempo se ha estudiado mediante el desarrollo de una *modelo* de simulación. Normalmente, este modelo tiene un conjunto de supuestos sobre el funcionamiento del sistema, que están expresados usando relaciones matemáticas, lógicas y simbólicas entre las *entidades* del sistema. Una vez desarrollado y validado, el modelo se utiliza para investigar cuestiones de tipo "¿qué pasaría si?" sobre el sistema real. Los cambios potenciales del sistema pueden ser simulados para predecir su impacto en el rendimiento del sistema. Además, la simulación se utiliza para estudiar sistemas en la fase de diseño, antes de que dichos sistemas se creen. Así pues, la simulación se puede utilizar como una herramienta de análisis para predecir el efecto de los cambios en los sistemas reales, y como una herramienta de diseño para predecir el rendimiento de los nuevos sistemas.

Una de las alternativas más utilizadas para desarrollar modelos de simulación es OMNeT++ [4]. OMNeT++ es un simulador de eventos discretos basado en C++ que se usa para modelar de redes de comunicación, multiprocesadores y otros sistemas paralelos o distribuidos. Su objetivo principal es el modelado de redes de gran escala.

Un modelo de OMNeT++ consiste en una serie de módulos que se comunican mediante el paso de mensajes. Los módulos activos se llaman módulos simples (es decir, aquellos donde se implementa directamente la funcionalidad). Están escritos en C++ utilizando la biblioteca de clases de simulación. Los módulos simples se pueden agrupar con otros módulos simples formando módulos compuestos que también se pueden combinar con otros módulos simples o compuestos, y así sucesivamente, ya que el número de niveles de herencia es ilimitado. En OMNeT++, al modelo en su conjunto se le denomina *red*, formado por unidades simples y/o compuestas. Esta jerarquía de componentes puede verse en la Fig. 1.

Los *módulos simples* suelen enviar los mensajes a través de *puertas*, pero también los pueden enviar directamente al módulo destino. Las *puertas* son las interfaces de entrada y de salida de los módulos: los mensajes se envían a través de puertas de salida y llegan a las puertas de entrada. Una puerta de entrada y una de salida se unen mediante una conexión. Las *conexiones* se crean en un solo nivel dentro de la jer-

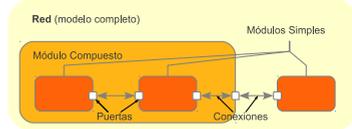


Fig. 1. Arquitectura de un modelo basado en OMNeT++.

arquía del módulo: dentro de un módulo compuesto, las correspondientes puertas de dos submódulos, o la puerta de un submódulo y la puerta de un módulo compuesto pueden conectarse. Debido a la estructura jerárquica del modelo, los mensajes viajan a través de una cadena de conexiones, comenzando y terminando en módulos simples.

Los *módulos compuestos* actúan como "cajas negras" en el modelo, haciendo transparente la transmisión de los mensajes entre su interior y el mundo exterior. Además, pueden definirse tipos de conexiones con propiedades específicas (*canales*) y reusarlos en diferentes sitios. Los módulos pueden tener parámetros que se usan principalmente para trasladar los datos de parámetros de configuración a los módulos simples, que son necesarios para definir la topología del modelo. Igualmente, los módulos compuestos pueden pasar los parámetros a sus submódulos.

En OMNeT++ el usuario describe la estructura del modelo a simular en el lenguaje NED (Network Description). NED permite al usuario declarar módulos simples, conectarlos y ensamblarlos en módulos compuestos. El usuario puede etiquetar algunos módulos compuestos como *redes*, que son los modelos a simular. Los canales son otro tipo de componente, cuyas instancias pueden ser utilizadas en los módulos compuestos.

Finalmente, la configuración y los datos de entrada para la simulación se encuentran en un archivo de configuración que normalmente llama *omnetpp.ini*. El archivo INI es un archivo de texto que consiste en entradas agrupadas en diferentes secciones. Los parámetros de los módulos pueden ser asignados en los archivos NED o en el archivo *omnetpp.ini*. En este archivo, los parámetros de los módulos son referidos por sus caminos completos o los nombres de la jerarquía. Este nombre consiste en una lista separada por puntos de nombres de módulos (desde el nivel jerárquico más alto al más bajo), más el nombre del parámetro. Además, dentro de este archivo INI es posible asignar una lista de valores a un parámetro (o varios) en concreto, con lo cual se podrá ejecutar una simulación por cada uno de los valores definidos y obtener los resultados en ficheros separados para su procesamiento.

OMNeT++ tiene incorporado un soporte para registrar los resultados de las simulaciones mediante ficheros de resultados que pueden contener *vectores de salida* o *escalares de salida*. Los vectores de salida son datos de series a lo largo del tiempo, grabados

desde los módulos simples o los canales. Los escalares de salida son resúmenes de resultados, calculados durante la simulación y generados cuando la simulación acaba. Una vez generados estos ficheros de salida de resultados, pueden analizarse utilizando el IDE de OMNeT++ o *scavetool*, que es su versión de línea de comandos o pueden procesarse con herramientas y lenguajes de programación como Matlab, R, etc. Los datos contenidos en los archivos, pueden graficarse utilizando el propio IDE de la plataforma o utilizando herramientas como Gnuplot.

IV. DESCRIPCIÓN DE LAS RAAP HPC TOOLS

En esta sección se describe la capa software propuesta para integrar los diferentes módulos presentes en OpenFabrics Software (OFS) con los simuladores basados en el framework OMNeT++ [10]. El primer paso consiste en generar un fichero de definición de la topología a simular. Este fichero contiene información acerca de los dispositivos de la red (nodos, conmutadores y enlaces), y debe ser compatible con el formato utilizado por *ibsim*, OpenSM y las herramientas provistas por OFS. En el siguiente paso se utiliza *ibsim* en combinación con OpenSM para simular la construcción de la red, mediante el intercambio de mensajes de control y generar las tablas de encaminamiento y el resto de estructuras definidas dentro de cada elemento de red. Hemos implementado una capa software, a la que hemos bautizado como RAAP HPC Tools (RHT). Este software utiliza las herramientas y módulos de OFS para recabar y procesar la información generada por *ibsim*. El diseño e implementación de la capa RHT se ha realizado en grupo de Redes y Arquitecturas de Altas Prestaciones (RAAP) [11] de la Universidad de Castilla-La Mancha. La generación de los ficheros de entrada para los diferentes módulos de OFS es responsabilidad de la capa RHT. La Figura 2 muestra un esquema de la arquitectura propuesta para la integración de OFS con OMNeT++.

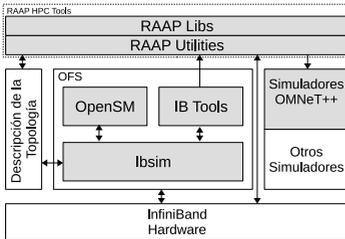


Fig. 2. Integración de los módulos OFS (OpenSM, *ibsim* y IB tools) con las HPC RAAP Tools y simuladores basados en OMNeT++.

Estas herramientas son una evolución de las herramientas *ad hoc* utilizadas en [6] y [12]. Incorporando varias mejoras y utilizando un diseño general para poder ser reutilizadas en diferentes escenarios

de pruebas. En resumen, las aportaciones de este trabajo son:

- Los ficheros de definición de topologías pueden ser generados por herramientas externas, o utilizando *libtopgen* y sus herramientas asociadas. El formato de este fichero es compatible con la salida generada por el comando *ibnetdiscover*. Cabe destacar que es posible utilizar otras herramientas para generar los ficheros de definición de topologías, o utilizar *libtopgen* directamente.
- Hemos implementado la capa RHT, un conjunto de bibliotecas y herramientas asociadas capaces de recabar información acerca de la red y traducir esa información para ser consumida por los modelos de simulación basados en OMNeT++, como los descritos en [7], [13] y [6].
- Se complementan las utilidades de OFS como *ibdniget*, con la capacidad de realizar análisis estáticos de la topología y de los algoritmos de encaminamiento. Es capaz de detectar si un algoritmo de encaminamiento es libre de bloqueos para una topología en particular.

El requisito principal es obtener el fichero de definición de la topología que se quiere estudiar, en un formato compatible con *ibsim*. Este fichero es un archivo de texto con un formato similar a la salida del comando *ibnetdiscover* y puede ser extraído de un cluster real. O utilizando las herramientas de *libtopgen*. Sin embargo, es posible desarrollar herramientas auxiliares que generen dichos ficheros de definición de los patrones de conexión y los algoritmos de encaminamiento, en un formato similar. Una vez obtenidos los ficheros necesarios, los utilizamos como entradas para *ibsim* y OpenSM. Primero *ibsim* crea y conecta los elementos de red (ej: nodos, conmutadores y enlaces) para la topología definida en el fichero de descripción. Como segundo paso OpenSM, en la fase de reconocimiento, *descubre* la topología construida por *ibsim* utilizando los datagramas de administración de InfiniBand (MADs). Después, se aplica un algoritmo de encaminamiento y rellena las tablas de rutas de cada conmutador. Sin embargo *ibsim* no tiene las características necesarias para escenarios de simulación que requieran modelar diferentes patrones de tráfico en la red y carece de las funcionalidades para recabar datos con fines de análisis estadísticos. Una vez que la red está completamente creada y con las estructuras de datos de los conmutadores completas, podemos utilizar las herramientas disponibles en OFS para obtener la información necesaria sobre la red, la velocidad de los enlaces, el contenido de las tablas de encaminamiento, los LIDs, etc. Finalmente, los resultados obtenidos con las herramientas InfiniBand se utilizan como entrada para RHT de cara a generar la descripción de la topología (o los ficheros necesarios) que utilizarán como ficheros de entrada los modelos de simulación. Aunque la estructura de la capa RHT puede ser aplicada a otros simuladores, en este trabajo hemos im-

plementado esta funcionalidad para los modelos de OMNeT++ descritos anteriormente.

Por otro lado, las herramientas que componen la capa RHT extraen datos (utilizando IB Tools) y realizan un análisis sintáctico de los resultados, traduciendo esta información a un formato compatible para los simuladores basados en OMNeT++. Específicamente, la capa RHT implementa ciertas funcionalidades para cumplir con dos objetivos. Primero, traducir la información extraída de *ibsim* en un formato compatible con los simuladores, como por ejemplo ficheros NED e INI necesarios para los modelos basados en OMNeT++. Segundo, proporcionar información sobre las características de la topología y el algoritmo de encaminamiento. Cabe destacar que una vez obtenida la representación lógica de la red, utilizando *ibsim* y OpenSM, puede ser deseable analizar las propiedades de la topología con el fin de proporcionar a los diseñadores de la red con información útil sobre los posibles bloqueos (*deadlocks*), rutas alternativas que mejoran la eficiencia de la red y otro tipo de información que pueda ser de utilidad para el desarrollo de técnicas de control de congestión. Para cumplir con estos objetivos la capa RHT se compone de dos partes diferenciadas: las *RHT Libs* que son un conjunto de bibliotecas que implementan las funcionalidades de generación de topologías, y análisis sintáctico de los diferentes resultados obtenidos de las IB Tools, así como de los ficheros NED e INI utilizados por el framework OMNeT++. Y, las *RHT Utilities* que complementan las bibliotecas descritas antes con utilidades de conversión y análisis. Y además, sirven como base para implementar otras herramientas.

Dentro de las bibliotecas que constituyen *RHT Libs*, se encuentra *libofsparse*. Esta biblioteca aglutina un conjunto de analizadores sintácticos capaces de procesar la salida de todas las IB Tools, como por ejemplo *ibnetdiscover*, *ibdiagnet*, *OpenSM*, etc., y crear el grafo que representa de forma lógica la topología bajo estudio. Los análisis posteriores utilizarán este grafo para extraer información topológica y de encaminamiento. Otra de las bibliotecas importantes es *libtopgen*. Esta biblioteca es un compendio de topologías y algoritmos de encaminamiento, que puede ser utilizada para generar las redes a simular o utilizar las utilidades complementarias para generar los ficheros necesarios para los simuladores. Ambas bibliotecas, *libofsparse* y *libtopgen* ofrecen una interfaz de aplicación (API) consistente y única, con el fin de reutilizar la mayor cantidad de código posible y facilitar, a los desarrolladores de simuladores, la integración de la capa RHT con sus modelos de simulación, estén o no basados en OMNeT++.

Las utilidades que componen *RHT Utilities* operan sobre los grafos generados tanto por *libofsparse* como por *libtopgen*. Dentro de estas utilidades se incluye *ib2om* cuyo objetivo es generar los ficheros de entrada para los simuladores basados en OMNeT++ (ficheros NED e INI). Actualmente están soporta-

dos los simuladores descritos en [6], [13] y [7]. Otra utilidad de la capa RHT es la herramienta *tan* [12], que permite el análisis de varios aspectos de la red. Además de varias métricas como la utilización de *buffers*, VLS, diámetro, el número medio de saltos, etc. Finalmente, *chkcyc* es una utilidad que utiliza la biblioteca *libofsparse* y realiza un análisis de dependencia cíclica de *buffers*, que sirve para detectar si la topología modelada puede generar interbloqueos..

V. CONCLUSIONES

Un diseño meticuloso de la red de interconexión es necesario para asegurar el rendimiento óptimo de los sistemas HPC. En la actualidad la arquitectura InfiniBand (IBA) se presenta como una alternativa popular dentro de la comunidad HPC con respecto a otras tecnologías de interconexión. Esta arquitectura se beneficia de una capa software intermedia (*middleware*) que sirve de nexo entre el hardware InfiniBand y el sistema operativo. Esta capa se denomina OpenFabrics Software (OFS) y se utiliza en aplicaciones que requieren gran eficiencia, bajas latencias y accesos de E/S rápidos para dispositivos de almacenamiento y sistemas de ficheros distribuidos. Además de InfiniBand, OFS es compatible con otras tecnologías de red como: iWarp, RoCE, OmniPath, etc. Por otra parte, los modelos de simulación se utilizan ampliamente en el diseño de redes de interconexión, para modelar y evaluar el impacto que tienen en la red las nuevas técnicas desarrolladas, antes de ser implementadas en sistemas reales. Uno de los mayores desafíos al que se deben enfrentar los desarrolladores de simuladores es modelar de forma precisa las interacciones entre las aplicaciones HPC y las redes simuladas. Por lo que, las latencias introducidas por las transacciones entre las aplicaciones y el *middleware* deben ser modeladas con una aproximación suficiente, para poder reproducir este comportamiento dentro del modelo de simulación.

En este trabajo, describimos el desarrollo de una capa software que integra parte de los módulos de OFS (actualmente *ibsim*, OpenSM e IB Tools) con modelos de simulación basados en el framework OMNeT++. Esta capa software, llamada RAAP HPC Tools (RHT), es capaz de generar los ficheros de entrada requeridos por herramientas de simulación basadas en OMNeT++. Además, utilizando los ficheros de definición de topología, estas herramientas son capaces de analizar las propiedades de la red, como las rutas disponibles, el diámetro de la red, posibles bloqueos en el algoritmos de encaminamiento, etc. La información proporcionada gracias al uso de la capa RHT permite tomar decisiones de diseño en la red de interconexión. Como trabajo futuro, planeamos integrar la capa RHT con otros módulos y tecnologías compatibles con OFS como: iWarp, RoCE, OmniPath, etc. Además, de mejorar la integración a nivel de componentes con el framework OMNeT++ y soportar otros frameworks para el desarrollo de simuladores.

AGRADECIMIENTOS

El presente trabajo ha sido financiado conjuntamente por el MINECO del Gobierno de España y fondos FEDER de la Comisión Europea, a través del proyecto TIN2015-66972-C5-2-R (MINECO/FEDER), la Junta de Comunidades de Castilla-La Mancha a través del proyecto PEII-2014-028-P, la beca FPI BES-2013-063681 y un contrato de acceso al sistema español de ciencia, tecnología e innovación (SECTI).

REFERENCIAS

- [1] TOP500.org, "Top 500 the list," November 2015.
- [2] Advanced Scientific Computing Advisory Committee, "Report on exascale computing," in *The Opportunities and Challenges of Exascale Computing*, Washington, DC, USA, 2010, U.S. Department of Energy, Office of Science.
- [3] Tsuyoshi Hamada and Naohito Nakasato, "Infiniband trade association, infiniband architecture specification, volume 1, release 1.0, <http://www.infinibandta.com>," in *International Conference on Field Programmable Logic and Applications, 2005*, 2005, pp. 366–373.
- [4] OMNeT++, "Omnet++ user manual," .
- [5] OFA, "The open fabrics alliance," 2014.
- [6] Jens Domke, Torsten Hoefler, and Satoshi Matsuoka, "Fail-in-place network design: Interaction between topology, routing algorithm and failures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Piscataway, NJ, USA, 2014, SC '14, pp. 597–608, IEEE Press.
- [7] Mellanox, "Omnet++ infiniband flit level simulation model," 2008.
- [8] IBTA, "The infiniband trade association," 2014.
- [9] B. L. Nelson y D. Nicol J. Banks, J. Carson, *Discrete-Event System Simulation (4th Edition)*, Prentice Hall, 2004.
- [10] G. Maglione-Mathey, P. Yebenes, J. Escudero-Sahuquillo, P. J. Garcia, and F. J. Quiles, "Combining openfabrics software and simulation tools for modeling infiniband-based interconnection networks," in *2016 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, March 2016, pp. 55–58.
- [11] RAAP, "Grupo de redes y arquitecturas de altas prestaciones," .
- [12] German Maglione-Mathey, Pedro Yebenes, Pedro Javier García, Francisco J. Quiles, and Jesús Escudero-Sahuquillo, "Analyzing available routing engines for infiniband-based clusters with dragonfly topology," in *2015 International Conference on High Performance Computing & Simulation, HPCS 2015, Amsterdam, Netherlands, July 20-24, 2015*, 2015, pp. 168–171.
- [13] P. Yebenes, J. Escudero-Sahuquillo, P.J. Garcia, and F.J. Quiles, "Towards modeling interconnection networks of exascale systems with omnet++," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, Feb 2013, pp. 203–207.

Modelado realista de una NoC fotónica en un entorno de simulación CMP detallado

José Puche, Sergio Lechago, Salvador Petit, María E. Gómez y Julio Sahuquillo¹

Resumen—La tecnología fotónica constituye una solución prometedora para el conocido problema de las comunicaciones en los multiprocesadores actuales. Esta tecnología presenta características deseables como latencias reducidas y consumo prácticamente independiente de la distancia. Sin embargo, dado que la tecnología nanofotónica se encuentra todavía en desarrollo, los simuladores de sistemas detallados actuales pueden carecer de modelos precisos de los componentes fotónicos. De este modo, cuando se asumen modelos fotónicos imprecisos, los resultados obtenidos no son representativos.

Este paper resume todos los componentes de una NoC (*Network on Chip*) fotónica completamente operativa y presenta el estado del arte actual de esta tecnología. Además, se evalúa una red fotónica realista formada por dos anillos fotónicos y un mecanismo de arbitraje basado en tokens. Posteriormente, esta propuesta es contrastada con un enfoque no realista. Ambos modelos son estudiados bajo diferentes configuraciones en las que el número de *wavelengths* empleado en cada *waveguide* fotónico es modificado. Los resultados experimentales muestran que el modelo de NoC no-realista presenta una desviación de hasta un factor de 6 en latencia de red respecto al modelo realista. Esto se traduce en una variación del rendimiento del sistema superior al 10% en varias de las aplicaciones estudiadas, lo que demuestra la importancia de modelos precisos cuando se simulan tecnologías en desarrollo como la nanofotónica.

Palabras clave—Fotónica, NoCs, arquitecturas *manycore*, simulación, modelado, CMP.

I. INTRODUCCIÓN

PARA mantener el ritmo marcado por la ley de Moore, los microprocesadores han utilizado las arquitecturas *manycore* durante los últimos 10 años. Los avances tecnológicos han permitido integrar hasta cientos de núcleos en un único chip, incrementando así la capacidad computacional de los mismos. Estos chips multiprocesador (CMPs), sin embargo, necesitan redes de interconexión *on-chip* eficientes que les proporcionen comunicaciones rápidas entre núcleos, cachés y controladores de memoria. De otra manera, la NoC puede limitar seriamente el rendimiento potencial asociado a estas arquitecturas. Además, el número creciente de núcleos y los costes de comunicación inherentes deben tenerse en cuenta dentro de un presupuesto energético limitado.

Las generaciones futuras de CMPs deben afrontar retos como comunicaciones globales eficientes y bajo consumo de energía. En este contexto, las NoCs actuales tienen dificultades para enfrentar estos problemas de diseño debido a las restricciones de la tecnología, especialmente cuando el número de núcleos crece y los mensajes deben recorrer largas distancias hasta alcanzar sus destinos. Las redes eléctri-

cas utilizadas en las arquitecturas CMPs usualmente presentan una topología de malla donde los controladores de memoria (CM) se ubican normalmente en las esquinas o lados del chip. Dado que la NoC debe conectar todos los *tiles*, cuanto más crece el número de núcleos mayor es la distancia entre los nodos y los CM. En una configuración así, cuando un nodo pide un bloque de caché, la latencia y consumo de energía asociados a esta transmisión pueden resultar inaceptables dependiendo de la distancia que separe al nodo del CM. Los multiprocesadores actuales incorporan varios controladores para aliviar este problema pero, desgraciadamente, el número de CM no escala linealmente con el número de nodos.

La necesidad de una tecnología de comunicación *on-chip* más rápida ha llevado a considerar el uso de interconexiones fotónicas compatibles con CMOS como una posible solución. La tecnología nanofotónica ha experimentado un elevado desarrollo durante la última década y se espera que esta tendencia continúe durante los próximos años. Gracias a su alto ancho de banda y a su reducido coste de energía, las interconexiones fotónicas compatibles con CMOS son la más prometedora tecnología para satisfacer las necesidades de ancho de banda de los futuros CMPs.

Las arquitecturas *manycore* y *manycore* pueden utilizar la tecnología nanofotónica para reducir su latencia de red y, con ello, los ciclos de acceso a memoria. Sin embargo, la tecnología fotónica se encuentra todavía en desarrollo lo que dificulta encontrar modelos actualizados de la misma en entornos de simulación. Esta situación supone el empleo de modelos imprecisos cuyos resultados pueden presentar desviaciones importantes. Por ello, entornos de simulación precisos y detallados necesitan contar con simuladores de CMPs que consideren la interconexión nanofotónica.

La tecnología nanofotónica presenta una gran variedad de nuevos componentes que deben ser modelados e incorporados a los entornos de simulación de CMPs existentes. En este paper se presenta un modelo de red fotónica que introduce todos los componentes empleados en una red fotónica realista completamente operativa. También se explica cómo cada uno de estos componentes afecta al rendimiento de la red y cómo el consumo de energía de todo el sistema se puede ver comprometido por ello.

El resto de este paper se organiza de la siguiente forma. La sección II presenta los antecedentes sobre la tecnología fotónica. La sección III introduce el estado del arte de la tecnología fotónica en la actualidad. La sección IV expone el modelado de los componentes fotónicos llevado a cabo. Las secciones V y

¹Dpto. de Ingeniería de Sistemas y Computadores, Centro de Tecnología Nanofotónica, Univ. Politécnica de Valencia, e-mail: jopucia@posgrado.upv.es

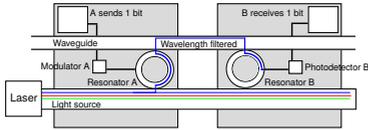


Fig. 1. Comunicación fotónica entre dos nodos A y B.

VI presentan el entorno y los resultados experimentales. Finalmente, la sección VII resume las principales conclusiones de este trabajo.

II. ANTECEDENTES

Los avances en tecnología fotónica sobre silicio permiten actualmente el desarrollo de una red óptica funcional completa en un único chip [1]. La correcta operación de este tipo de redes requiere de la utilización de varios componentes fotónicos, cuya función es explicada brevemente en esta sección.

Una fuente de energía láser se necesita para inyectar la luz dentro del chip. Esta luz se transporta mediante *waveguides* al resto de componentes de la red óptica. Además, la luz del láser puede ser multiplexada en diferentes *wavelengths*. Para ello se utiliza la técnica *Dense Wavelength Division Multiplexing* (DWDM). DWDM permite a varios nodos de la red comunicarse simultáneamente mediante múltiples *wavelengths* en un único *waveguide* [2].

La técnica DWDM requiere anillos resonadores para identificar los diferentes *wavelengths* que componen la señal luminosa. Un resonador es un pequeño anillo que filtra los *wavelengths* correspondiente a un determinado *wavelength*. Por defecto, el *wavelength* filtrado por el resonador depende del diámetro del mismo, que habitualmente se encuentra entre 3 y 4 μm [3]. Un resonador puede ser excitado para filtrar un *wavelength* diferente mediante la aplicación de carga eléctrica o el aumento de su temperatura. Los resonadores deben ser excitados cada vez que van a ser utilizados para realizar una transmisión en un *wavelength* diferente al establecido por defecto, por lo que el consumo de energía dependerá del número de resonadores y transmisores.

Los resonadores se utilizan tanto en los nodos fuente como destino. En los nodos fuente, un modulador eléctrico-óptico traslada la señal eléctrica al *wavelength* filtrado por el resonador. En el nodo destino, el resonador filtra la señal óptica y la dirige hacia el fotodetector, que finalmente transforma la señal óptica en señal eléctrica. Obsérvese que los tiempos asociados a la conversión eléctrico-óptica y viceversa deben ser tenidos en cuenta a la hora de modelar una red fotónica con precisión.

La figura 1 muestra un ejemplo de una transmisión óptica entre dos nodos A y B. Para transmitir un flujo de bits entre ambos nodos, en primer lugar, el resonador en el nodo A debe absorber la luz correspondiente al *wavelength* λ_i del *waveguide* que transporta la luz del láser. Posteriormente, este flujo de bits se modula en la señal óptica filtrada que a su

vez es dirigida al *waveguide* donde será transportada hasta el nodo B. En el nodo B, el resonador filtra el mismo *wavelength* y permite al receptor B absorber la señal óptica y convertirla mediante el fotodetector, obteniendo finalmente la señal eléctrica original.

III. ESTADO DEL ARTE DE LA TECNOLOGÍA FOTÓNICA

Actualmente, la investigación en PICs (*Photonic Integrated Circuits*) se encuentra concentrada en la obtención de láseres híbridos en silicio fiables y moduladores y receptores eléctrico-ópticos. Los resultados prometedores de estas investigaciones marcan el camino hasta conseguir dispositivos ópticos completamente integrados en el chip que permitan sobrepasar las limitaciones inherentes al rendimiento de la tecnología eléctrica.

Las fuentes láser son los dispositivos más difíciles de integrar en el chip debido a potencia, área y a las restricciones por atenuación de la señal óptica. Duan et al. han desarrollado láseres híbridos silicio/III-V que presentan nuevas características y un consumo de energía menor que el alcanzado en trabajos anteriores [4], [5]. Sin embargo, estos avances no consiguen todavía alcanzar el ultrareducido consumo de energía requerido para la integración on-chip. Además, los láseres integrados únicamente son capaces de proporcionar potencias de salida de decenas de mW, lo que implica problemas de atenuación. Las previsiones apuntan a que en los próximos años estas cifras mejorarán, permitiendo la utilización de láseres dentro del chip. De hecho, varios trabajos en redes fotónicas asumen la integración de los láseres on-chip dado que son mucho más eficientes energéticamente [6].

La capacidad de conmutación de los moduladores eléctrico-ópticos representa la característica clave para establecer el ancho de banda operacional en cualquier PIC. El alto ancho de banda de modulación en silicio (tasas de transmisión de datos de hasta 30-50 Gb/s y tiempos de conversión de varios GHz [7], [8]) pueden ser alcanzados mediante cambios de índice inducidos *free-carrier* y utilizando estructuras *biased pn*. [9].

Los receptores ópticos, que convierten la amplitud, fase y polarización de una señal de campo óptica a dominio eléctrico ya han conseguido ser integrados presentando un rendimiento similar al mostrado por los ya empleados en dispositivos ópticos comerciales. Además, presentan tasas de conversión elevadas, de hasta 224 Gb/s con señales PDM-16-QAM [10].

En lo que respecta al estado del arte en la investigación sobre redes híbridas fotoelectrónicas e implementaciones de NoC fotónicas, Vantrease et al. exploran los futuros requerimientos de ancho de banda en muchos y proponen Corona [11], un arquitectura 3D manycore que emplea tecnología fotónica para comunicaciones tanto dentro como fuera del chip. Kurian et. al presentan ATAC [12], un sistema de un millar de núcleos que se comunican mediante una

NoC fotónica. En [13], los autores proponen Firefly, una NoC híbrida que utiliza clustering de nodos y emplea interconexión fotónica para comunicar estos clústers. Finalmente, FlexiShare [14] utiliza un anillo fotónico combinado con DWDM para interconectar un CMP de 64 núcleos.

La investigación en NoC fotónicas se encuentra estrechamente relacionada con la investigación en técnicas de arbitraje con DWDM. Debido a que la mayoría de los esquemas de comunicación basados en DWDM requieren de compartición de wavelengths, varios trabajos consideran el arbitraje como una parte importante de la eficiencia de estas comunicaciones [11], [14]. Atendiendo a esto, Vantrease et. al [15] utilizan también la tecnología fotónica para realizar tareas relacionadas con el arbitraje. De este modo, identifican latencia, utilización media de red y *fairness* como las características básicas que un adecuado mecanismo de arbitraje debe mantener.

IV. MODELADO DE COMPONENTES FOTÓNICOS

Todos los componentes mencionados en la sección II han sido modelados en nuestro entorno de simulación. Para cada componente se identifican las propiedades que puedan afectar al rendimiento del sistema y al consumo de energía.

Asumiendo las restricciones tecnológicas explicadas anteriormente, el láser debe ser ubicado off-chip. El consumo energético de un láser off-chip puede variar en el rango desde 1W hasta más de 5W. El consumo exacto necesario para el láser dependerá además de la atenuación que experimentará la señal luminosa, que en definitiva dependerá de ciertas características del waveguide como los giros, divisiones, uniones, etc. a los que se vea expuesta.

Respecto a los waveguides, presentan dos propiedades principales que pueden afectar a la latencia de comunicación: i) los índices de refracción y ii) su longitud. Los waveguides habitualmente utilizados en prototipos están hechos de silicio de cristal y óxido de silicio, que presentan unos índices de refracción de 3.4401 y 1.4298 respectivamente. Esto supone un índice medio de 2.439. Por tanto, la velocidad de propagación de la luz en el silicio se aproxima en $12,3\text{mm}/100\text{ps}$. En lo que respecta a la longitud, esta depende del número de nodos interconectados y de las dimensiones del chip. En nuestro sistema base (véase sección V), se utiliza un CMP de 576mm^2 [15] formado por 16 tiles y un controlador de memoria, lo que supone una longitud del anillo de interconexión de 116 mm.

Los moduladores y receptores también pueden afectar a la latencia de comunicación. En nuestro entorno de simulación, ambos componentes toman un ciclo a una determinada frecuencia para transmitir 1 bit. Debido a que los moduladores en el estado del arte presentan tiempos de conmutación en torno a los 100 picosegundos, se ha establecido la frecuencia de los moduladores en 10 GHz. En lo que respecta a los receptores, la literatura indica que sus rangos de latencia varían entre decenas y cientos de picosegun-

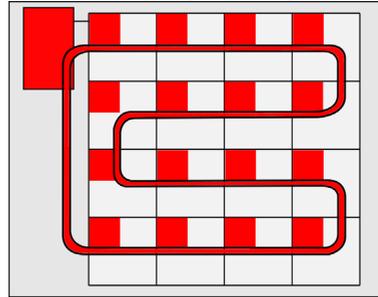


Fig. 2. Waveguide de 116 mm en un 16-tiled CMP.

dos [6]. Por ello, se asume que la latencia de los receptores es igual a la latencia de los moduladores.

La tecnología DWDM permite diferentes esquemas de comunicación para utilizar múltiples wavelengths como un medio de transmisión compartido. En general, existen cuatro esquemas de comunicación DWDM bien conocidos [2]: Single Writer Single Reader (SWSR), Single Writer Multiple Reader (SWMR), Multiple Writer Single Reader (MWSR) y Multiple Writer Multiple Reader (MWMR). Excepto el esquema SWSR, el resto requieren de arbitraje para garantizar acceso libre a los wavelengths [15]. Por tanto, para conseguir un modelo correcto de NoC fotónica, es necesario definir tanto el esquema de comunicación como el mecanismo de arbitraje. Para ello, estrategias de arbitraje basadas en *token* son típicamente utilizadas cuando se evalúan y modelan los esquemas de comunicación y deben tener en cuenta las latencias asociadas a la extracción e inyección de tokens.

V. SISTEMA BASE

La figura 2 muestra un diagrama del sistema base: un CMP de 16 tiles donde los nodos y el CM están conectados mediante un anillo fotónico. Cada tile está formado por un núcleo que permite ejecución fuera de orden y que cuenta además con cachés de nivel L1 y L2 privadas así como con una interfaz de red para acceder al anillo.

El anillo fotónico se compone de 2 waveguides. El primero de ellos es utilizado para las comunicaciones nodo-controlador mientras que el segundo para las comunicaciones en sentido contrario. En el anillo encargado de enviar las peticiones al CM se utiliza el esquema MWSR, lo que implica que dos nodos no pueden realizar una petición al CM simultáneamente. Por lo tanto, se requiere la existencia de un mecanismo de arbitraje entre los núcleos. El arbitraje consiste en pasar un token entre todos los posibles emisores [15]. Obsérvese que, incluso cuando un núcleo se encuentra enviando en solitario, el token debe ser liberado regularmente para comprobar si otros nodos están intentando transmitir nuevamen-

TABLA I
 CONFIGURACIÓN DEL SISTEMA BASE.

Núcleos de cómputo	
Number of cores	16
Frequency	3GHz
Issuing policy	Fuera de orden
Issue/Commit width	4 instrucciones/ciclo
ROB size	256 entradas
Jerarquía de memoria	
L1 Instrucción cache	Privada, 32KB, 8 vías, 64Bytes-line, 2 ciclos
L1 Data cache	Privada, 32KB, 8 vías, 64Bytes-line, 2 ciclos
L2	Privada, 256KB, 16 vías, 64Bytes-line, 11 ciclos
Anillo fotónico	
Topology	Anillo
Waveguides	2
Wavelengths	64 wavelengths por waveguide
Frequency	10 GHz
Modulator lat	1 ciclo
Photodetector lat	100 ps
Arbitration	Token channel
Phit size	64 bits
Roundtrip lat	14 ciclos

te.

Por otro lado, los mensajes enviados desde el CM siguen el esquema de comunicación SWMR. Este esquema evita el retardo debido al arbitraje pero requiere excitar previamente el anillo resonador del nodo destino. Con este objetivo, un wavelength adicional es utilizado para este propósito.

La tabla I resume los principales parámetros del sistema base explicado.

La latencia del anillo fotónico escala con su longitud. Asumiendo un CMP cuadrado de N -núcleos, el área ocupada por cada núcleo tiene una longitud y anchura proporcionales a $1/\sqrt{N}$. Por lo tanto, cuando $N = 16$ y el área del chip es 576 mm^2 se necesita un waveguide de 116 mm de longitud.

Dejando a un lado la latencia de arbitraje, la latencia correspondiente a una transmisión en el anillo completo incluye los tiempos del modulador, del receptor y de cruzar el waveguide. La latencia del waveguide se define mediante la longitud de la transmisión y la velocidad de propagación de la luz en el silicio por lo que, teniendo en cuenta los valores anteriores y empleando una frecuencia de 10 GHz, la latencia asociada a recorrer el waveguide es de 12 ciclos. Esta latencia debe ser sumada a los dos ciclos necesarios para las conversiones eléctrico-óptica y viceversa, por lo que la latencia final se establece en 14 ciclos.

VI. RESULTADOS EXPERIMENTALES

Las diferentes configuraciones han sido modeladas y evaluadas utilizando el entorno Multi2Sim [16], que permite simulaciones detalladas de núcleos con ejecución fuera de orden así como de la jerarquía de memoria. La capa de red de Multi2Sim ha sido ampliamente extendida para modelar la NoC fotónica, incluyendo todos los componentes mencionados anteriormente.

Los experimentos se han llevado a cabo utilizando la suite de benchmarks SPEC2006. Se han eva-

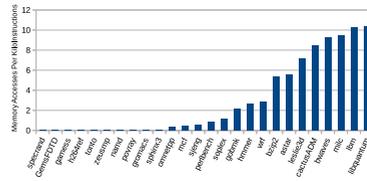


Fig. 3. Memory Accesses Per KiloInstruction (MAPKI).

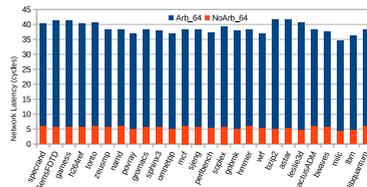


Fig. 4. Latencia media de red en las ejecuciones con y sin retardo por arbitraje y 64 wl/wg.

luado tanto ejecuciones individuales como mezclas multiprogramadas. Cada aplicación ejecuta al menos 100M de instrucciones después de realizar un *fastforward* de los 300M de instrucciones iniciales. En la evaluación de las cargas multiprogramadas, todas las aplicaciones se mantienen en ejecución hasta que el último benchmark termina el número de instrucciones indicado. Esta metodología evita que los benchmarks más rápidos se vean más afectados por la contención que los benchmarks de ejecución más lenta.

A. Caracterización de los benchmarks

En primer lugar se ha analizado y caracterizado la actividad de memoria de cada benchmark ejecutado individualmente. Los benchmarks que presentan un mayor número de fallos en su último nivel de caché acceden a la red de interconexión más frecuentemente. El sistema utiliza paquetes de 8 bytes para las peticiones y acusos de recibo y paquetes de 72 bytes para los bloques de datos.

La figura 3 muestra los *Memory Accesses Per KiloInstruction* (MAPKI) de cada uno de los benchmarks estudiados en orden creciente. Las aplicaciones en el lado izquierdo de la gráfica presentan un MAPKI reducido, por lo que su rendimiento no se ve excesivamente afectado por la NoC.

B. Ejecución individual

A continuación se han evaluado las aplicaciones en solitario en el sistema propuesto. Pese a ello, dado que el arbitraje es una característica clave en la NoC fotónica, los tiempos asociados al mismo por liberación de token son igualmente tenidos en cuenta en estas ejecuciones. Esto significa que, antes de enviar un mensaje al controlador de memoria, el núcleo de-

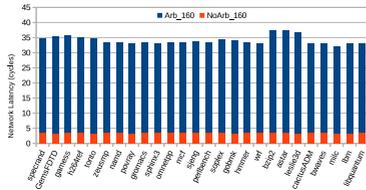


Fig. 5. Latencia media de red en ejecuciones con y sin retardo por arbitraje y 160 wl/wg.

be esperar a que el token recorra el anillo para comprobar que no hay otros núcleos que puedan quedar en inanición.

La figura 4 muestra la latencia media de red de las aplicaciones estudiadas teniendo en cuenta el retardo asociado al arbitraje y asumiendo 64 wavelengths en cada waveguide. En el gráfico, la parte inferior de cada barra hace referencia al esquema NoArb64, que no modela los retardos por arbitraje y en el que, por tanto, los mensajes no esperan la latencia de token mencionada. Como puede observarse, la latencia de red para esta configuración es homogénea y cercana a los 5 ciclos de red para todos los benchmarks evaluados. Este valor se incrementa hasta 35 ciclos aproximadamente cuando se modela el arbitraje, lo que significa que el modelado del mismo introduce una desviación

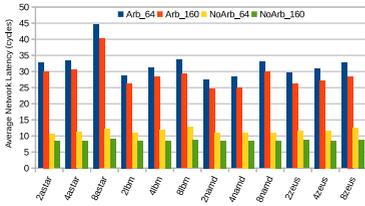


Fig. 7. Latencia media de red en ejecuciones con y sin retardo por arbitraje en configuraciones de 64 wl/wg y 160 wl/wg.

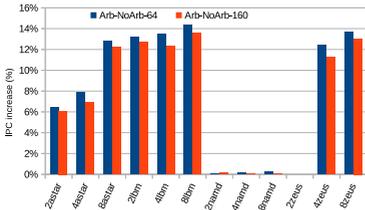


Fig. 8. Incremento de IPC en ejecuciones con y sin retardo por arbitraje en configuraciones de 64 wl/wg y 160 wl/wg.

nentes fotónicos como los waveguides, moduladores y fotodetectores presentan.

Con el objetivo de cuantificar la desviación que un modelo fotónico incorrecto puede presentar en un entorno de simulación detallado, se ha modelado y evaluado una propuesta realista basada en un anillo fotónico compuesto por dos waveguides y una técnica simple de arbitraje, y se la ha comparado con una propuesta no realista sin modelo de arbitraje. Los resultados experimentales muestran que la variación media en la latencia de red entre las dos aproximaciones puede superar hasta un factor de 10, lo que se traduce en una desviación del IPC superior al 10% en algunos casos. Estas desviaciones demuestran que los entornos de simulación actuales han de ser debidamente extendidos con modelos precisos y fiables para poder obtener resultados representativos cuando se usan tecnologías novedosas.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Ministerio Español de Economía y Competitividad (MINECO), por los fondos del Plan E bajo subvención TIN2015-66972-C5-1-R, por el proyecto ExaNest, financiado por la investigación y programa de innovación Horizon 2020 de la Unión Europea bajo el acuerdo No 671553 y por la Generalitat Valenciana mediante subvención del proyecto AICO/2016/059.

REFERENCIAS

[1] S. Abadal, A. Cabellos-Aparicio, Jose A. Lazaro, E. Alarcón, and J. Sole-Pareta, "Graphene-enabled hybrid architectures for multiprocessors: Bridging nanophotonics and

nanoscale wireless communication," in *Transparent Optical Networks (ICTON), 2012 14th International Conference on*, July 2012, pp. 1–4.

[2] A. Chan Bergman K. Carloni, L. P. Bibermani and Hendry G., *Photonic Network-on-Chip Design*, vol. 68, Springer-Verlag New York, 2014.

[3] Qianfan Xu, David Fattal, and Raymond G. Beausoleil, "Silicon microring resonators with 1.5- μ m radius," *Opt. Express*, vol. 16, no. 6, pp. 4309–4315, Mar 2008.

[4] Guang-Hua Duan, Jean-Marc Fedeli, Shahram Keyvania, and Dave Thomson, "10 gb/s integrated tunable hybrid iii-v/si laser and silicon mach-zehnder modulator," in *European Conference and Exhibition on Optical Communication*. 2012, p. Tu.4.E.2, Optical Society of America.

[5] G. H. Duan, C. Jany, A. Le Liepvre, M. Lamponi, A. Accard, F. Poingt, D. Make, F. Lelarge, S. Messaoudene, D. Bordel, J. M. Fedeli, S. Keyvania, G. Roelkens, D. Van Thourhout, D. J. Thomson, F. Y. Gardes, and G. T. Reed, "Integrated hybrid iii x2013;v/si laser and transmitter," in *Indium Phosphide and Related Materials (IPRM), 2012 International Conference on*, Aug 2012, pp. 16–19.

[6] Yu Li, Yu Zhang, Lei Zhang, and Andrew W. Poon, "Silicon and hybrid silicon photonic devices for intradatacenter applications: state of the art and perspectives," *Photon. Res.*, vol. 3, no. 5, pp. B10–B27, Oct 2015.

[7] Ansheng Liu, Ling Liao, Doron Rubin, Hat Nguyen, Berkhan Ciftcioglu, Yoel Chetrit, Nahum Izchaky, and Mario Paniccia, "High-speed optical modulation based on carrier depletion in a silicon waveguide," *Opt. Express*, vol. 15, no. 2, pp. 660–668, Jan 2007.

[8] D. J. Thomson, F. Y. Gardes, Y. Hu, G. Mashanovich, M. Fournier, P. Grosse, J.-M. Fedeli, and G. T. Reed, "High contrast 40gb/s optical modulation in silicon," *Opt. Express*, vol. 19, no. 12, pp. 11507–11516, Jun 2011.

[9] R. Soré and B. Bennett, "Electrooptical effects in silicon," *IEEE Journal of Quantum Electronics*, vol. 23, no. 1, pp. 123–129, Jan 1987.

[10] Po Dong, Sethumadhavan Chandrasekhar, Xiang Liu, Lawrence Lee Buhl, Ricardo Aroca, Yves Baeyens, and Young-Kay Chen, "224-gb/s pdm-16-qam modulator and receiver based on silicon photonic integrated circuits," in *Optical Fiber Communication Conference/National Fiber Optic Engineers Conference 2013*. 2013, Optical Society of America.

[11] D. Vautreasse, R. Schreiber, M. Monchiero, M. McLaren, N.P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R.G. Beausoleil, and J.H. Ahn, "Corona: System implications of emerging nanophotonic technology," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, June 2008, pp. 153–164.

[12] George Kurian, Jason E. Miller, James Psota, Jonathan Eastep, Jifeng Liu, Jurgen Michel, Lionel C. Kimerling, and Anant Agarwal, "Atac: A 1000-core cache-coherent processor with on-chip optical network," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, New York, NY, USA, 2010. PACT '10, pp. 477–488, ACM.

[13] Yan Pan, Prabhat Kumar, John Kim, Gokhan Memik, Yu Zhang, and Alok Choudhary, "Firefly: Illuminating future network-on-chip with nanophotonics," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 429–440, June 2009.

[14] Yan Pan, J. Kim, and G. Memik, "Flexishare: Channel sharing for an energy-efficient nanophotonic crossbar," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, Jan 2010, pp. 1–12.

[15] D. Vautreasse, N. Binkert, R. Schreiber, and M.H. Lipasti, "Light speed arbitration and flow control for nanophotonic interconnects," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, Dec 2009, pp. 304–315.

[16] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli, "Multi2sim: A simulation framework for cpu-gpu computing," in *PACT*. 2012, pp. 335–344, ACM.

Arquitectura NoC Híbrida con Conmutación de Paquete-Circuito Dirigida por el Protocolo de Coherencia

Miguel Gorgues Alonso¹ y José Flich Cardo²

Resumen—

Actualmente los chips multiprocesadores tienen a incrementar el número de núcleos, normalmente en el último nivel de la memoria cache se implementa un sistema distribuido. La red en el chip (NoC) es la encargada de interconectar los núcleos, los controladores de memoria y las caches, teniendo un gran impacto en la latencia de acceso a memoria. En las NoC normalmente se utiliza conmutación de paquete, sin embargo, la conmutación de circuito debería conseguir mejores prestaciones si se consigue evitar el tiempo de establecimiento del circuito (estableciéndolo antes de que sea necesario). En Prosa, el protocolo de coherencia se encarga del establecimiento de estos circuitos con el objetivo de establecer dichos circuitos antes de que sean necesarios y únicamente para el período de tiempo que estos que es requerido. Nuestra propuesta utiliza los controladores de circuito PROSA para controlar un cluster de cuatro conmutadores en la red. Se detallan todos los detalles de implementación de PROSA, incluyendo la lógica de establecimiento de circuito con mensajes ACK, modificaciones del protocolo y las modificaciones del conmutador para establecer los circuitos cuando es necesario. Resultados de aplicaciones reales demuestran la reducción de la latencia de red en un 34 %, lo que se traduce en una reducción de un 21 % en la latencia de fallos de cache (en un sistema de 64 nodos). PROSA requiere un 9.66% más de área, pero recude el consumo de energía en un 3 %.

I. INTRODUCCIÓN

LOS sistemas Chip Multiprocessor (CMPs) se usan en las redes en el chip (NoCs) [1] para conseguir comunicaciones más rápidas entre los recursos, principalmente procesadores, memorias caches y controlador de memoria. Esto produce que la NoC juegue un papel importantísimo en el rendimiento del CMP, principalmente cuando los mensajes se transmiten a lo largo de la ruta crítica de la comunicación lo cual afecta a las latencias de comunicación del proceso y más importante, a la latencia de acceso a memoria.

Durante más de una década, las NoCs (no solo para CPMs) se han investigado con dos objetivos principales, productividad de la red y latencia de la red. Aunque la productividad de la red es un aspecto importante en la NoC, en los CMPs la carga media de los enlaces suele ser baja para las aplicaciones contenidas en los benchmarks (PARSEC, SPLASH, ...). Esto significa que en los sistemas CMP, la latencia de red empieza a ser la métrica más significativa a la que hacer frente en las NoCs. En efecto, una transacción de memoria empieza cuando un procesador realiza una petición de escritura o lectura en su cache privada de datos L1, y finaliza cuando se recibe el dato. La NoC se utiliza para enviar peticiones y datos entre los distintos niveles de la cache y el controlador de memoria. Por lo tanto, la NoC juega un rol vital desde que se realiza la petición de un bloque hasta que se recibe.

¹Department of Computer Architecture at Universitat Politècnica de València, Spain

²Department of Computer Architecture at Universitat Politècnica de València, Spain

En este artículo se enfoca en la reducción de latencia de las transacciones de memoria. Se propone PROSA, Protocol-oriented circuit Switch Architecture, un codiseño de la NoC y el protocolo de coherencia. Nuestro enfoque mejora la NoC mediante el uso del controlador PROSA (PC), que se encarga de gestionar los circuitos y resolver cualquier posible conflicto. El controlador maneja cuatro conmutadores PROSA (PR) y establece sus circuitos locales cuando es necesario.

Los resultados muestran que PROSA mejora los resultados del diseño base elegido, reduciendo la latencia de memoria en un 34 % de media. PROSA también reduce la latencia del fallo de lectura y escritura en las caches L1 un 21 %. Además, PROSA consigue establecer correctamente el 90 % de los circuitos antes de que sean requeridos. La propuesta requiere un 9.66% más de área, pero ahora más de un 3% de consumo de energía.

PROSA, a diferencia de las propuestas anteriores de conmutación de circuito utiliza la información proporcionada por el protocolo de coherencia para mejorar las prestaciones. Aunque algunos trabajos anteriores ya combinan protocolos con el diseño de la NoC [2] [14] [13] [12], nuestra propuesta permite al protocolo de coherencia el establecimiento de circuitos en la red antes de que estos se requieran. De este modo se oculta el retardo del establecimiento de circuito. Por otra parte, PROSA no necesita recursos adicionales en las colas de los puertos de entrada del conmutador, únicamente necesita un multiplexor y un demultiplexor para cada puerto bidireccional del conmutador. En la sección V se describen con más detalles los trabajos relacionados.

El resto del artículo está organizado de la siguiente forma: en la Sección II se describe y analiza el protocolo de coherencia. Esto centrará nuestra propuesta y proporcionará una justificación de las necesidades. En la Sección III se describe la propuesta. En la Sección IV se evalúan y analizan los resultados de la propuesta. Los trabajos previos relacionados se describen en la Sección V y el artículo finaliza con las conclusiones en la Sección VI.

II. ANALISIS DEL PROTOCOLO DE COHERENCIA

Para este análisis se asume que los CMP utilizan caches L1 privadas para cada núcleo y una cache L2 compartida pero distribuida entre todos los tiles del CMP. Los dos controladores de memoria (MC) están situados en las dos esquinas superiores del sistema basado en una malla 4×4 . La NoC utiliza conmutadores que están diseñados con 4 etapas y con grado 7 (cuatro puertos para conectar los conmutadores vecinos y otros tres para conectar L1, L2 y el MC).

El sistema implementa el protocolo MOESI [3] para

las caches L1, mientras que los bloques de datos en las caches L2 pueden estar en estado P (privado), S (compartido), C (cacheado) o I (invalido). En estado C ninguna cache L1 tiene una copia del bloque. Se asume una política de caches inclusivas y con una política de escritura *write back*. Se asume también un mapeado estático de los datos en los bloques de los bancos L2.

La figura 1 muestra una simplificación de la máquina de estados del protocolo para las operaciones de lectura y escritura. En la figura los círculos representan los nodos. El texto en la parte superior del círculo representa el estado actual, mientras que en la parte inferior se representa el estado después de la transacción. Los mensajes enviados entre los nodos se representan mediante flechas.

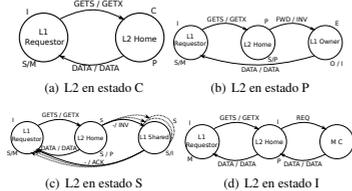


Fig. 1: Transacciones del Protocolo de Coherencia.

Cuando ocurre un fallo de lectura en la L1, se envía un mensaje GETS al banco L2 Home. Según el estado del bloque en la cache L2 se realizan diferentes acciones: Si el bloque está en estado S o C (Figuras 1(a) y 1(c)), la L2 envía el dato a la cache L1. Si el bloque está en estado P (Figura 1(b)), la cache L2 envía un mensaje de forward(FWD) a la cache L1 que tiene el bloque en propiedad y el estado de la cache L2 cambia a S. Cuando el mensaje de FWD llega a la cache L1 propietaria del bloque, ésta envía el dato a la cache L1 demandante y cambia el estado actual del bloque de P a O. Finalmente, cuando la cache L2 está en estado I (ocurre en un fallo en la L2, Figura 1(d)), se envía un petición del bloque al MC. Cuando la L2 recibe el dato desde MC, la L2 envía el dato a la L1 que realizó la petición. El estado del bloque en la L2 se configura en P y en la L1 en E.

Cuando ocurre un fallo de escritura en la L1, se envía un mensaje de GETX al banco L2 Home. Al igual que ocurre con la lectura, si el estado del bloque en la L2 Home es C (Figura 1(a)), la L2 envía el dato a la L1 y cambia el estado del bloque a P. Un caso diferente ocurre cuando el bloque está en estado P, (Figura 1(b)). La L2 envía un mensaje de invalidación (INV) a la L1 propietaria del bloque. Cuando el mensaje de INV llega a la L1, ésta envía el dato a la L1 que ha realizado la petición y el bloque cambia de E a I. Cuando el bloque está en estado S (Figura 1(c)), la L2 envía el dato a la cache L1 y envía un mensaje de INV al resto de caches L1 que tienen el bloque en estado S. Cuando estas caches reciben el mensaje de INV, estas envían un mensaje de ACK a la cache L1 que ha requerido el dato y cambian el estado de S a I. Finalmente, en el caso de fallo en la L2, bloque en estado I (Figura 1(d)), se procede igual que un fallo de lectura.

Como se ha visto, el protocolo tiene cuatro posi-

bles caminos dependiendo del tipo de operación (lectura/escritura), tipo de acceso a la L1 (fallo/acierto) y tipo de acceso a L2 (acierto/fallo). Siempre que ocurre un fallo en la L1 y la L2 involucran a la NoC y por tanto la latencia de transacción de memoria aumenta. Los factores que afectan principalmente a la latencia de memoria son: la distancia entre las caches L1 y L2, la distancia entre la cache L2 Home y el MC y la congestión en la red.

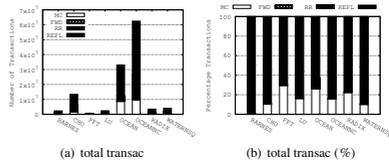


Fig. 2: Tipos de transacciones de memoria para diferentes aplicaciones SPLASH-2.

Para realizar un mejor análisis de este efecto, la Figura 2(a) muestra las transacciones de memoria clasificadas por tipos: Transacciones de MC (debido a fallos en ambas caches L1 y L2), transacciones RR (debido a un fallo en la L1 y un acierto en L2 con un bloque limpio), transacciones FWD (debido a un fallo en la L1 y la L2 manda un mensaje de FWD al propietario) y transacciones de remplazo (REPL, transacciones debidas a que los bancos en las L2 están llenas). Se ejecutan varias aplicaciones del conjunto de aplicaciones SPLASH-2 sobre el sistema descrito en la Sección IV.

Lo primero que se puede apreciar son las diferentes cantidades de transacciones entre cada una de las aplicaciones. Esto depende de la complejidad de cada una de estas. Sin embargo, si se normalizan los números, Figura 2(b), se aprecian similitudes. Existe un alto porcentaje de transacciones de REPL en todas las aplicaciones. Esto ocurre principalmente debido al tamaño de las caches L2 y del dataset de las aplicaciones, pues estas transacciones no pueden ser predichas por el protocolo de coherencia. Se puede observar un pequeño porcentaje de transacciones de FWD. Todas las aplicaciones muestran menos de un 3% de transacciones de este tipo. Ahora nos centraremos en los tipos de transacciones más interesantes para nuestra propuesta. Algunas aplicaciones (FFT y RADIX) lanzan un alto porcentaje de transacciones de MC, mientras que otras (BARNES Y WATERNSQ) tienen un porcentaje marginal. Estas diferencias se deben al número de aciertos en la L2. Finalmente, las transacciones de RR son altamente representativas en algunas aplicaciones (BARNES Y WATERNSQ) y mínimas en otras (FFT). Esto se debe también al número de aciertos en la L2. En ambos casos, las transacciones de MC y de RR, se puede predecir el tráfico entre la L2 y MC, y entre L2 y L1. PROSA aprovechará este hecho estableciendo circuitos para las transacciones MC y RR.

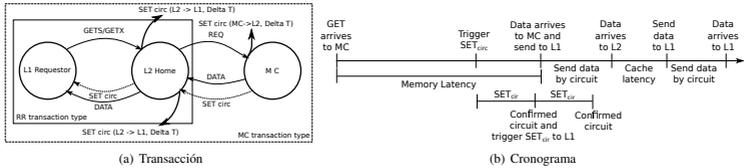


Fig. 3: Nueva acción PROSA lanzada por el protocolo de coherencia y cronograma de establecimiento de circuito para una transacción MC.

III. PROSA

PROSA establece circuitos entre MC y L2 y L2 y L1 antes de que estos sean requeridos. En este sección primeramente, se muestra las modificaciones realizadas en el protocolo de coherencia. Seguidamente, se señalan las modificaciones en la NoC (el controlador PROSA y el conmutador PROSA).

A. Protocolo de coherencia PROSA

Para la programación de los nuevos circuitos, se ha añadido una nueva acción denominada SET_{CIRC} . Esta acción involucra la fuente y el destino del circuito y el periodo de tiempo que será requerido el circuito (Delta T). El protocolo lanza esta acción en las transacciones MC y RR (mirar Figura 3(a)). En las transacciones MC, siempre que se recibe una petición se calcula cuando llegará el dato desde la memoria principal al MC. Basándose en esta predicción, el MC lanza la acción SET_{CIRC} después de un periodo de espera (DP). DP es igual al periodo de llegada del dato menos el periodo de establecimiento del circuito (CSP). La acción SET_{CIRC} establece el circuito entre el origen y el destino. Cuando el dato llegue al MC desde la memoria, principal el circuito habrá sido establecido y se mantendrá activo mientras el dato sea transmitido. Cuando SET_{CIRC} llegue a la cache L2, el circuito se confirma y en paralelo se lanza un nuevo SET_{CIRC} entre las caches L2 y L1. Por tanto, cuando los datos lleguen a la L2, después del acceder a ésta, el bloque se enviará a la L1 usando un nuevo circuito. Estos dos circuitos son predecibles, es decir, serán necesarios independientemente del estado de la red. La Figura 3(b) muestra los tiempos de ambos circuitos.

En las transacciones RR (entre caches L2 y L1), siempre que se recibe un mensaje GETS/GETX, se lanza una acción SET_{CIRC} entre las caches L2 y L1. Igualmente, estos circuitos se establecerán solo en el periodo de tiempo en el que el dato va a estar disponible. Sin embargo, al contrario que en los casos anteriormente descritos, estos circuitos son especulativos, debido a que puede que no sean requeridos si el bloque está en estado P o I. Por lo tanto, para este tipo de circuito se ha implementado de una forma eficiente y automática la eliminación de los circuitos cuando estos han terminado la transmisión o cuando esta claro que no van a ser requeridos.

B. Red de Circuitos PROSA

PROSA sigue un enfoque de cluster donde un controlador PROSA (PC) controla cuatro conmutadores (Figura

4(a)). Cuando una acción SET_{CIRC} se lanza en una L2 o un MC, un mensaje de petición de circuito (RC) se envía al PC local. Si la RC gana los recursos necesarios para el establecimiento del circuito en el cluster, ésta avanza hacia el siguiente PC a través del camino. Si no gana los recursos, se genera un mensaje de respuesta NACK que se envía a la fuente del mensaje RC. Cuando un RC alcanza el PC que controla el nodo destino del circuito se genera un mensaje de respuesta ACK. Entonces, el ACK se envía de vuelta por la red de PCs hacia el nodo origen. Cabe destacar que los conmutadores de la red no se ven afectados por los mensajes de RC, ni ACK, ni NACKs, debido a que todos estos mensajes son transmitidos por la red de PCs.

En la parte superior de la Figura 4(c), se muestra el mensaje RC compuesto por 8 campos. P_x hace referencia al puerto por el cual llega el RC al cluster. src y dst hacen referencia al origen y destino del circuito. $Delta T$ and $Delta T'$ muestran el número de ciclos de espera en los que el circuito debe configurarse y eliminarse respectivamente. $Type$ hace referencia al tipo de mensaje RC (Petición, ACK, NACK). ID identifica el circuito dentro de la red, y finalmente, GT (Golden Token) hace referencia a la distancia entre src y dst . Este campo se utiliza para asignar las prioridades entre las peticiones.

Debajo de la estructura del RC, en la Figura 4(c), se muestra el Arbitro del Recurso (RA) que controla cuando un recurso específico puede ser reservado para un periodo de tiempo. Un PC consiste en varios módulos RA y hay un módulo RA por cada puerto de salida en el cluster controlado por el PC. En total, cada cluster tiene veintiocho módulos RA, todos mostrados en la Figura 4(d) (por ejemplo R_{0c} corresponde al puerto de salida sur en el conmutador nord-este del cluster). Los módulos RA arbitran entre las peticiones, cuyo número de peticiones depende del número de puertos de entrada que tienen dependencias con este recurso (puerto de salida). Estas dependencias vienen dadas por el algoritmo de encaminamiento, en nuestro caso se asume encaminamiento DOR.

Cuando un mensaje RC llega a un módulo RA, se chequea el campo dst para decidir si el puerto de salida controlado por el módulo RA está a lo largo de la ruta entre src y dst . Si es así, el RC pasa al Arbitro Estático (SA) que arbitra entre todas las peticiones entrantes en el módulo. Se da prioridad a las peticiones con un valor más grande en el campo GT (los caminos más largos tienen más prioridad). Entonces, el RC avanza al módulo Comparador de Tiempos (TC), quien comprueba que no existe solapación

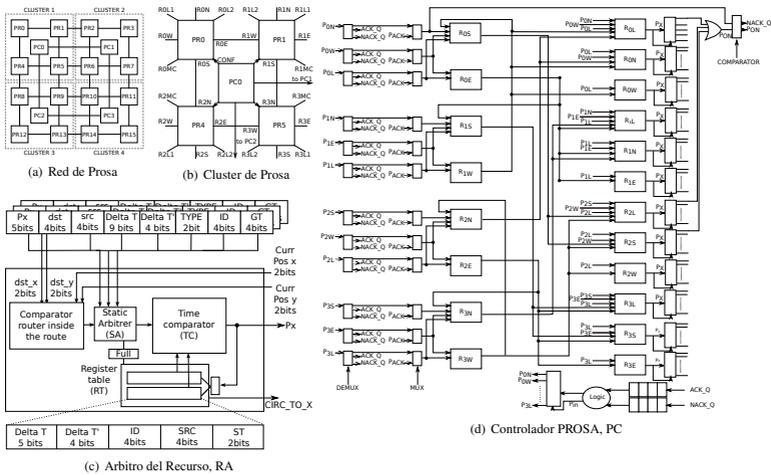


Fig. 4: Controlador PROSA, Componentes detallados.

entre los circuitos establecidos previamente y el circuito solicitado. Finalmente, si no existe conflicto con los circuitos previos, entonces el mensaje RC se almacena en la Tabla de Registros (RT) y se transmite hacia el siguiente PC en la red. Destacar que solo un mensaje RC puede acceder a la tabla por ciclo. Como se demostrará, esto no tiene efecto en los resultados y reduce la complejidad.

La Tabla de Registros (RT) almacena la información del circuito. Para cada circuito se almacenan los siguientes campos: *Delta T*, *Delta T'*, *ID*, *src*, y *ST*. Cuando una petición ordinaria gana el recurso, *Delta T*, *Delta T'*, *ID*, *src* toman el valor del campo equivalente en el mensaje RC. *ST* almacena el estado del circuito, que puede ser *sin confirmar*, *confirmado*, o *vacío*. Cada ciclo se decrementa el valor de *Delta T* de los circuitos reservados. Cuando éste alcanza cero, el módulo RA activa las salidas *Cir.to.P* las cuales controlan el establecimiento del circuito en los conmutadores PROSA (detallado más adelante). A partir de este momento *Delta T'* se decrementa en uno cada ciclo hasta que alcanza cero. Cuando esto sucede se limpia el registro y se desactiva la señal *Cir.to.P* eliminando de esta forma el circuito.

Cuando un mensaje ACK o NACK llega al RA, esta petición sigue el mismo camino que realizaría una petición ordinaria, pero con pequeñas diferencias. Primero, el mensaje RC avanza a SA. Los mensajes ACK/NACK tienen una mayor prioridad que los mensajes ordinarios. Además, por el diseño del controlador PC se garantiza que únicamente acceda un ACK/NACK por ciclo, por lo tanto, los ACK/NACK siempre ganarán el acceso a SA. Entonces, el TC comprueba la información almacenada. Los mensajes ACKs consolida la información almacenada y los mensajes NACKs la eliminan. Finalmente, el mensaje RC se transmite al siguiente RA dentro de la ruta

del cluster o al siguiente controlador PC.

La Figura 4(d) muestra el controlador PC. Éste contiene 28 RAs, 7 por cada conmutador. Para simplificar el dibujo, se han agrupado todas las salidas al conmutador (L1, L2 y MC) en una única RA (R_{iL}), por lo que se muestran solamente veinte RAs. Cada controlador PC implementa dos colas para almacenar los mensajes ACK/NACK. Un demultiplexor situado en los puertos de entrada (a la izquierda) separan las peticiones ordinarias de los ACK/NACK que utiliza el campo *type* como selector. Cuando llega una petición ordinaria continua hacia el PC. Sin embargo, si la petición entrante es de tipo ACK/NACK, ésta se envía a la cola correspondiente. La siguiente etapa en el PC es un multiplexor, que multiplexa entre las peticiones entrantes y las colas de ACK/NACK, dando prioridad a los ACK/NACK. En caso de conflicto, la petición ordinaria se descarta (generando un NACK que se encola en la cola). Finalmente, el mensaje seleccionado entra en el árbol RA.

Los módulos RA están enlazados siguiendo las dependencias impuestas por el algoritmo de encaminamiento (en nuestro caso DOR). Un ejemplo del cableado de los RA es $R_{0E} \rightarrow R_{1S} \rightarrow R_{3S}$ para mensajes que vienen del conmutador PR0 y su destino está debajo del conmutador PR3. Los módulos RA tienen tantas dependencias entre las entradas y las salidas como posibilidades de encaminamiento existentes. A lo largo de este camino, si una petición gana todos los módulos RA requeridos (desde la izquierda a la derecha), la petición será enviada al siguiente PC o generará un ACK si el destino del circuito está controlado por el PC actual. Unos comparadores en los puertos de salida de los módulos PC (a la derecha) comprueban si una petición ha ganado todos los recursos requeridos. En caso contrario, se genera un NACK y se

almacena en la cola de NACKs.

La Figura 4(d) muestra las colas de ACK y NACK y su bloque de lógica de control. Esta lógica garantiza que solo un mensaje de ACK/NACK tenga acceso al controlador PC cada ciclo, asegurando en dicho caso que esos mensajes siempre ganen todos los módulos RAs. Los mensajes NACK tiene mayor prioridad a los mensajes ACK. Además, esta lógica cambia el puerto de entrada de los mensajes ACK, calculando los puertos de entrada que utilizaría una petición ordinaria. De esta forma el mensaje ACK/NACK atraviesa el mismo camino que la petición ordinaria. Nótese que los mensajes NACK generados localmente en el controlador PC necesitan ser reintertados en el árbol RA otra vez para eliminar todos los recursos reservados y después ser enviado al controlador PC previo.

C. Conmutador PROSA

La Figura 5 muestra las modificaciones realizadas en el conmutador base. Únicamente se ha añadido un demultiplexor por cada puerto de entrada, un multiplexor en cada puerto de salida, las conexiones entre estos elementos y un repetidor asíncrono por puerto de salida. El repetidor permite transmitir los flits rápidamente reduciendo el retraso del cable, como se utiliza en SMART [2]. Esta tecnología permite que un flit atraviese toda la red entre el origen y el destino en un ciclo.

Los conmutadores funcionan con normalidad hasta que se activa la señal *Circ.to.X*, donde *X* es el puerto de salida (L1, L2, MC, N, E, W, S). En este caso, los puertos de entrada y salida correspondientes se conectan y el flit entrante es rápidamente transmitido por el circuito. Al mismo tiempo, los árbitros VA y SA asociados a estos puertos se desactivan.

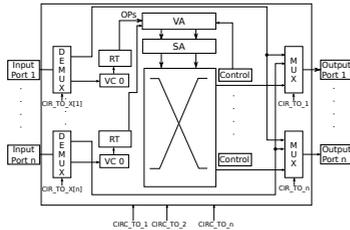


Fig. 5: Conmutador PROSA.

Las señales *Circ.to.X* son generadas por el controlador PC, cada señal está generada por un solo módulo RA (el que maneja el puerto de salida). La señal *Circ.to.X* indica que puerto de salida tiene que ser conmutado hacia el puerto de salida 'X'. Se utiliza un bit para cada uno de los posibles puertos de entrada. Entonces, un demultiplexor selecciona el puerto de entrada mediante una puerta OR a las señales *Circ.to.X* asociadas al puerto de entrada y un multiplexor selecciona el puerto de salida mediante una puerta OR de todos los cables asociados a la señal *Circ.to.X*.

Los circuitos se limpian de forma distribuida y silen-

ciosa. Cuando los valores Delta T y Delta T' alcanzan cero en el RA, la conexión se elimina.

D. Ejemplo de configuración del circuito PROSA

La Figura 6 muestra un ejemplo del proceso de configuración de un circuito en Prosa entre un MC y una L2, seguidamente a que el protocolo haya lanzado la acción *SET_CIRC*. El establecimiento del circuito comienza cuando la petición REQ llega al MC. Se asume que este evento ocurre en t_0 y que todos los recursos están disponibles (no habrá conflictos con otros circuitos). También se asume que el MC conoce que el dato requerido estará disponible en el ciclo 10 (desde memoria principal). El MC procesará la petición de circuito en un ciclo, por tanto, en t_1 , se enviará un mensaje RC a su PC (PC0).

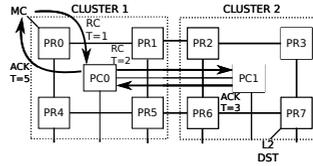


Fig. 6: Ejemplo de establecimiento de circuito en PROSA.

En t_1 , el mensaje RC alcanza PC_0 e intenta obtener todos los recursos necesarios que forman parte del camino entre MC y L2 a través del cluster. RC competirá por estos recursos, $R_{0E} \rightarrow R_{1E}$ (puerto este de PR_0 y PR_1). En cada módulo RA se almacenará la petición RC. Además, se configura Delta T a 9 (valor de Delta T en la petición) en los módulos R_{0E} , R_{1E} . Delta T' se obtendrá el valor del número de ciclos requeridos para la transmisión del bloque (número de flits del mensaje).

En t_2 , el mensaje RC se transmite desde PC_0 a PC_1 . Destacar que los valores de Delta T almacenados se decrecientan en uno. En PC_1 , se aplica el mismo proceso para los recursos $R_{2E} \rightarrow R_{3S} \rightarrow R_{7L}$ en PC_0 . Después de ganar estos recursos, PC_1 genera un mensaje ACK que se almacena en la cola de ACK. El campo Delta T para estos módulos RA se configura en 8.

En t_3 , todos los recursos con un circuito establecido (aunque sea en estado *noconfirmado*) decrecientan el valor del campo Delta T en uno. Además, en t_3 , PC_1 procesa el mensaje ACK. Como se ha comentado anteriormente, todos los mensajes ACK y NACK tienen la prioridad más alta, por lo cual el mensaje ACK ganará todos los módulos RA necesarios, confirmando de esta forma el circuito en PC_1 y enviando de vuelta el mensaje ACK a PC_0 . Finalmente, en t_4 , el mensaje ACK llega a PC_0 , siendo procesado en t_5 y ganando todos los módulos RA necesarios y confirmando el circuito en PC_0 . El mensaje ACK se transmite el nodo MC, confirmando el circuito en el interfaz de la red (NI).

En el caso de que cualquier recurso no este disponible durante el proceso de configuración del circuito, el PC donde no está disponible generará un NACK y lo encolará en la cola de NACK. Los recursos serán liberados en

el siguiente ciclo y el mensaje NACK será transmitido al PC previo.

El tiempo avanza y en t_{11} , el valor del campo Delta T en el módulo RA asociado a R_{0E} alcanza 0, entonces la señal $Circ_to_E$ (en R_{0E}) se activa, conectando el circuito entre el puerto de entrada del MC y el puerto de salida Este en PR_0 . Este circuito estará conectado el número de ciclos requeridos por el mensaje (Delta T'). Esto sucede en todos los recursos que tienen el circuito programado (R_{1E} , R_{2E} , R_{3S} y R_{7L}) realizando el mismo proceso. En este ciclo el nodo MC recibe el dato desde la memoria principal e inyecta el primer flit. El flit atraviesa toda la red alcanzando el puerto de salida de la cache L2 en at $PR7$. Finalmente, cuando Delta T' alcanza cero, el circuito se elimina de forma distribuida en cada conmutador cuando el último flit atraviesa el conmutador.

IV. EVALUACIÓN DE PROSA

Para analizar el rendimiento se utiliza un simulador ciclo a ciclo dirigido por eventos que modela varias topologías de red y arquitecturas del conmutador. Se ha modelado un conmutador de dos etapas (una con Input Buffer (IB), Routing (R), VC allocator and Switch allocator (VASA), que se modelan en una sola etapa, y otra etapa de Crossbar (X), con canales virtuales (VC) y con conmutación en el crossbar a nivel de flit, como se utiliza en Garnet [4]. La tabla I muestra los parámetros de simulación para los conmutadores y la jerarquía de memoria. Las caches L1 son privadas al núcleo mientras que las caches L2 son compartidas y distribuidas entre todos los nodos.

Parámetros	Red	L1	L2	MC
Topología	mallá 8x8			
VCs/Tiempo vuelo	4/1 ciclos			
Tamaño mensaje	8/72 bytes			
Flit/Tamaño cola	8/72 bytes			
sets/vías/Tamaño línea (B)		32/4/64	128/16/64	
Latencia cache/tag		2/1	4/2	
Num. MCs				2
Topen/Tactu/Tread				16/16/16

TABLA I: Parámetros y valores utilizados para los conmutadores y las caches.

Se evalúan tres mecanismos: el conmutador base (BASELINE), la propuesta Dejavu [5] (DEJAVU) y PROSA. En PROSA los datos que se envían a través de circuitos requieren un solo ciclo para cruzar la red (usando SMART). Se analizan aplicaciones de los benchmarks SPLASH [6] y PARSEC [7].

Deja Vu anticipa los circuitos entre nodos para esconder la latencia de configuración dividiendo la NoC en dos planos: plano de control y de datos. El plano de control se encarga de la configuración de los circuitos. Este plano tiene mayor frecuencia y un voltaje más alto, por lo cual es más rápido que el plano de datos. En esta NoC, la petición anticipa la ruta al mismo tiempo que empieza a transmitirse el dato por la red más lenta. Deja Vu configura los circuitos en el orden en que están reservados.

A. Resultados

La Figura 7(a) muestra los resultados del tiempo de ejecución de las aplicaciones. PROSA reduce, de media,

un 33% el tiempo de ejecución respecto al BASELINE. PROSA obtiene unas ganancias menores en aplicaciones con pocos fallos en L2 (BODYTRACK) o con tiempos de ejecución cortos (LU), en los cuales mejora sobre un 5% el tiempo de ejecución. Sin embargo, en aplicaciones balanceadas(OCEANNC) o aplicaciones con un gran número de transacciones MC (FFT, CANNEAL, FMM, BLACKSCHOLES), PROSA consigue una mejora de hasta un 50%. DEJAVU consigue mejoras poco significativas en este apartado, como se puede ver en [5].

La Figura 7(b) muestra los resultados de latencia. DEJAVU mejora las prestaciones en latencia a BASELINE alrededor de un 10% de media. PROSA obtiene una mejora de un 7% en aplicaciones con un bajo número de transacciones MC (BODYTRACK). Sin embargo, PROSA mejora hasta un 39% a BASELINE en aplicaciones con un gran número de fallos en la cache L2 (CANNEAL). De media, PROSA mejora alrededor de un 31.14% al BASELINE en latencia end-to-end. En latencia de red la mejora sube hasta el 34.16%.

Ahora se analizarán los resultados de latencia de fallo en la L1 normalizados al BASELINE (Figura 8). En este caso, al igual que ocurre con el tiempo de ejecución, DEJAVU mejora ligeramente al BASELINE, mientras que PROSA obtiene una mejora de un 23% de media.

La Tabla II muestra estadísticas sobre los circuitos PROSA. La primera columna muestra el porcentaje de ACKs recibidos (número de circuitos establecidos correctamente). En todos los casos es mayor del 85.58% y de media se han establecido un 90.62%. La segunda columna detalla el porcentaje de NACKs recibidos y las siguientes cinco columnas muestran los diferentes tipos de conflictos en el controlador PC. La tercera y la cuarta columnas muestran el número de NACKs generados en los puertos de entrada cuando una petición ordinaria entra en conflicto con una ACK/NACK. Las tres últimas columnas hacen referencia a los conflictos generados en el arbitro RA debido a (1) que la tabla de registros está completa (FB), (2) el periodo de tiempo requerido se solapa con un periodo de tiempo reservado anteriormente, (3) porque dos peticiones colisionan en el mismo módulo RA (ARB). Como se puede observar, la mayoría de los conflictos son generados por conflictos temporales o por que la petición entra en conflicto con un ACK en el mismo recurso. Sin embargo, el tamaño actual de la tabla de registros en cada módulo (4 entradas) parece ser el adecuado debido que el número de conflictos debido a que la tabla de registros esta llena es despreciables.

B. Implementación de PROSA

Se ha implementado toda la infraestructura de PROSA para un sistema CMP de 4×4 . Cada módulo PC se ha implementado y testeado en Verilog. Se ha utilizado el diseño de un conmutador canónico con flits de 64-bit, cuatro canales virtuales con capacidad para 9 flits y siete puertos conectados a la red 2D y las caches L1, L2 y al MC. Se ha utilizado Design Vision tool de Synopsys con 45nm Nangate open cell library [8]. Los resultados de energía se han obtenido mediante Orion-3 power library [9].

La Tabla III muestra los sobrecostes en área de PROSA

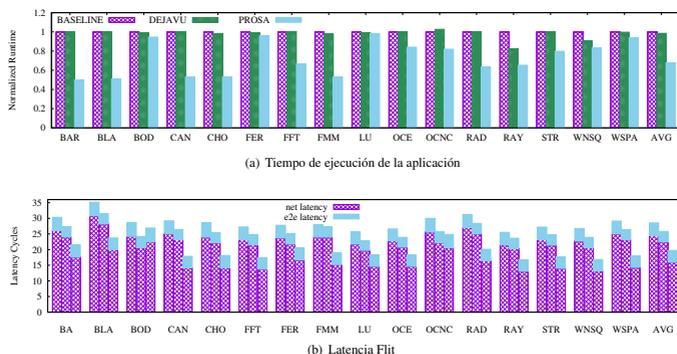


Fig. 7: Resultados de rendimiento para diferentes arquitecturas (BASELINE, DEJAVU, PROSA orden de columna).

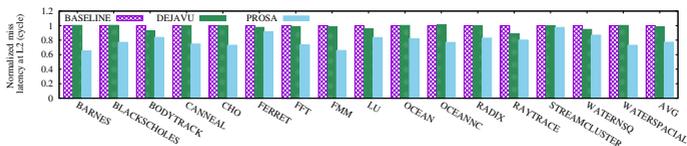


Fig. 8: Resultados de latencia de memoria para diferentes arquitecturas. Normalizadas al caso base.

	CONF. INPUT		CONF. RA				
	REC.ACK	REC.NACK	NACK	ACK	FB	TMP	ARB
MIN	(OCNC) 85.98%	(LU) 4.28%	(LU) 0.01%	(BOD) 0.63%	0%	(WSPA) 1.92%	(WSPA) 0.31%
AVG	90.62%	9.38%	0.04%	3.94%	0.00%	4.61%	0.79%
MAX	(LU) 95.72%	(OCNC) 14.02%	(WSPA) 0.18%	(WSPA) 9.27%	0%	(OCNC) 9.04%	(OCNC) 1.70%

TABLA II: Número de ACK y NACK generados en PROSA y número de conflictos generados en el módulo.

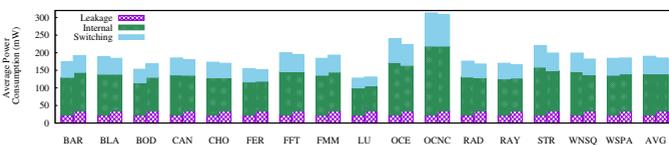


Fig. 9: Resultados consumo de energía, BASELINE Y PROSA.

respecto a diferentes configuraciones. En todos ellos, y en aras de la comparación justa, se han contado los componentes para construir un cluster de cuatro conmutadores. En el caso de PROSA se han incluido todos los componentes (incluyendo el PC y los cuatro PRs).

Como se puede apreciar, los conmutadores PROSA únicamente requieren un 1.8% más de área en comparación con el conmutador base. Los controladores PROSA (PC) utilizan menos área que un conmutador base (el 22%). Sin embargo, este componente es nuevo y tiene que ser considerado como un sobre coste adicional. Para hacer esta comparación justa, la tabla muestra el área

requerida por un cluster. En este caso, el cluster PROSA toma 9,66% más de área. El sobre coste de PROSA puede ser considerado como alto. Sin embargo, se ha de considerar las mejoras en las prestaciones que ofrecen los circuitos de PROSA.

La tabla añade dos configuraciones adicionales dignas de ser analizadas. La primera, *flattenedbutterfly*, muestra el sobre coste para una topología flattened butterfly, la cual tiene una mayor conectividad en cada dimensión y dirección. En este caso el aumento de los puertos de entrada hace que el sobre coste de esta red ascienda hasta el 41%. La segunda se trata de una configuración similar a

Configuración	área (μm^2)	% área adic.
Baseline router	243784	-
PROSA Router (PR)	248200	1.8 %
PROSA Controller (PC)	76547	-
Baseline cluster	975136	-
PROSA cluster	1069347	9.66 %
Flattened Butterfly cluster	1380109	41 %
2xBaseline cluster	1658560	70 %

TABLA III: sobrecostes de área para diferentes conmutadores y topologías NoC.

la del conmutador base doblando el tamaño de las colas. Esto permite transferencias más rápidas entre los nodos, sin embargo, esta topología requiere hasta un 70 % más de área, debido principalmente al tamaño de las colas.

La Figura 9 muestra los resultados de energía. Aunque PROSA aumenta la energía estática un 42 %, el total de la energía consumida se reduce un 3 %. Esta bajada se produce por la reducción de un 10 % de la energía interna y de switching. Como se describe en [5] DEJAVU consigue una reducción de un 30 % de energía (pero sin la mejora en la latencia). Destacar que la reducción de 30 % del tiempo de ejecución se transformará en un mayor ahorro de energía.

V. TRABAJOS RELACIONADOS

La conmutación de circuito [10] se ha usado en un gran número de trabajos previos en arquitecturas NoCs con el objetivo de reducir la latencia de comunicación dentro del chip. Una vez el circuito está establecido, los datos no tienen que atravesar las etapas de encaminamiento y arbitrages en cada conmutador. Sin embargo, el tiempo de establecimiento de circuito normalmente causa una baja utilización de los recursos y degrada las prestaciones. Por otra parte, la conmutación de mensaje mejora la utilización y el rendimiento de la red, dividiendo el mensaje en pequeños bloques y transmitiendo estos a través de la red.

Algunos trabajos intentan obtener el beneficio de ambos mecanismos implementando redes híbridas con conmutación de paquete-circuito. Kumar [11] propone Express Virtual Channels (EVC) que permite a los paquetes sobrepasar conmutadores intermedios a lo largo de la ruta. EVs solo permite conectar nodos en la misma dimensión, no siendo posible hacer cambios de dimensión con los circuitos. PROSA, por otra parte, permite conectar cualquier par de nodos sin importar su localización, por lo que ofrece una mayor flexibilidad.

Jerger [12] propone circuit switched coherence (CSC). Esta propuesta establece circuitos permanentes entre pares de nodos que compartan datos frecuentemente en lugar de desconectar estos enlaces y establecerlos cada vez que se requiere enviar un dato. Esto permite enviar datos entre los mismos nodos rápidamente. Sin embargo, si otro par de nodos requieren el recurso, los datos que estaban siendo enviados por el circuito cambian a conmutación de paquetes hasta que alcanzan su destino. Yin [13] propone una red de conmutación híbrida en la que los paquetes son transmitidos a lo largo de la red de conmutación de paquete mientras el circuito se configura en paralelo, usando TDM. Esta propuesta también requiere

tiempo para establecer el circuito. Mazloui [14] propone otro conmutador híbrido. Este mecanismo configura el circuito a lo largo de la red mientras se está transmitiendo el mensaje de petición del dato entre la cache L1 y la cache L2. Cuando la petición alcanza su destino y el dato está preparado para ser transmitido, el mecanismo envía un mensaje de prueba para activar el circuito reservado y transmitir el mensaje. Todos estos mecanismos requieren un periodo de configuración de la red. Sin embargo, PROSA esconde la latencia de establecimiento del circuito basándose en el protocolo de coherencia. Además cuando la transmisión del dato finaliza, los recursos se liberan permitiendo enviar nuevos paquetes mediante la conmutación de paquete.

Abousamra [5] propone Deja Vu descrito en la Sección IV. Krishna [2] presenta SMART, una red multi salto capaz de atravesar toda la ruta entre el origen y el destino en un solo salto. Esta propuesta requiere el establecimiento del circuito un ciclo antes de que el dato sea transmitido. Pueden establecerse circuitos parciales. La propuesta necesita una red extra para enviar las peticiones de establecimiento (SMART-hop Setup Request, SSR). SMART no está basado en el protocolo de coherencia como nuestras propuesta, además nuestro mecanismo se basa en los circuitos SMART.

Como resumen, PROSA permite establecer circuitos entre cualquier par de nodos ocultando la latencia de establecimiento. PROSA se basa en información del protocolo de coherencia para establecer estos circuitos. Además, PROSA programa los circuitos para el periodo de tiempo en que serán requeridos, sin tener conflictos con otros circuitos que utilicen el mismo recurso (en otro periodo de tiempo). De hecho, los circuitos de PROSA pueden ser dirigidos por los protocolos de coherencia o incluso para otras aplicaciones de nivel superior, donde se detecten ráfagas de tráfico que se puedan predecir. Las propuestas vistas anteriormente no se basan en protocolos de coherencia o se basan en arquitecturas y tecnologías más caras.

VI. CONCLUSIONES

En este artículo se ha presentado PROSA, una novedosa arquitectura que permite al protocolo de coherencia dirigir el establecimiento de circuitos para conexiones futuras, predecibles o especulativas, entre los controladores de memoria (MC) y los bancos de caches L2 y entre las caches L2 y las caches L1. El establecimiento de las conexiones se realiza para cada paquete y establece las conexiones para el periodo de tiempo requerido para enviar el mensaje entre los MC y L2 y las L2 y L1. Para las predicciones de circuito fallidas el circuito se elimina silenciosamente. PROSA diseña una infraestructura separada para el establecimiento de los circuitos. Por lo que la NoC no se ve afectada por los mensajes de control de circuitos adicionales que se introducen en la red.

Los resultados muestran una reducción de la latencia de red de hasta un 35 % por la correcta predicción y utilización de los circuitos con PROSA. Además PROSA mejora el caso base reduciendo el tiempo de ejecución en un 33 %, mejorando además los resultados obtenidos en DEJAVU, obteniendo resultados similares de energía.

PROSA requiere un 9.66% más de área respecto al conmutador base, sin embargo, PROSA ahorra un 3% más de energía en consumo de energía. En referencia a los sobrecostes de PROSA son razonables dados los beneficios óptimos obtenidos en el rendimiento.

REFERENCIAS

- [1] L. Benini et al., "Networks on chips: A new soc paradigm," *Computer*, vol. 35, pp. 70–78, Jan. 2002.
- [2] T. Krishna et al., "Breaking the on-chip latency barrier using smart," in *Proceedings of HPCA*, HPCA '13, (Washington, DC, USA), pp. 378–389, IEEE Computer Society, 2013.
- [3] D. J. Sorin et al., *A Primer on Memory Consistency and Cache Coherence*. Morgan & Claypool Publishers, 1st ed., 2011.
- [4] Q. Shubo et al., "The design and implementation of two-cycle noc router," in *Proceedings of ICSTCT 2010*, pp. 233–235, Nov 2010.
- [5] A. Abousamra et al., "Déjà vu switching for multiplane noCs," in *NOCS*, pp. 11–18, IEEE, 2012.
- [6] S. C. Woo et al., "The splash-2 programs: Characterization and methodological considerations," in *Proceedings of ISCA '95*, (New York, NY, USA), pp. 24–36, ACM, 1995.
- [7] C. Bienia et al., "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of PACT '08*, (New York, NY, USA), pp. 72–81, ACM, 2008.
- [8] "nangate freepdk45 generic open cell library ver 1.0.," <http://www.s12.org/openeda.s12.org/projects/nangatellib/>.
- [9] A. Kahng et al., "Orion3.0: A comprehensive noc router estimation tool," *Embedded Systems Letters, IEEE*, vol. 7, pp. 41–45, June 2015.
- [10] W. J. Dally et al., *Principles and Practices of Interconnection Network*. San Mateo: Morgan Kaufmann, 2nd ed., 2004.
- [11] A. Kumar et al., "Express virtual channels: Towards the ideal interconnection fabric," in *Proceedings of ISCA '07*, (New York, NY, USA), pp. 150–161, ACM, 2007.
- [12] N. D. E. Jeger et al., "Circuit-switched coherence," in *NOCS*, pp. 193–202, IEEE Computer Society, 2008.
- [13] J. Yin et al., "Energy-efficient time-division multiplexed hybrid-switched noc for heterogeneous multicore systems," in *Proceedings of IPDPS '14*, (Washington, DC, USA), pp. 293–303, IEEE Computer Society, 2014.
- [14] A. Mazloumi et al., "A hybrid packet/circuit-switched router to accelerate memory access in noc-based chip multiprocessors," in *Proceedings of DATE 2015, Grenoble, France, March 9-13, 2015*, pp. 908–911, 2015.

Guaranteeing latencies in DVFS-based NoCs under unbalanced traffic loads

José V. Escamilla, José Flich¹ and Mario Roberto Casu²

Resumen— Dynamic Voltage and Frequency Scaling (DVFS) can be a very effective power management strategy not only for on-chip processing elements but also for the network-on-chip (NoC). In this paper we propose a new approach to DVFS in NoC, which combines a congestion management strategy with a feedback-loop controller. The controller sets frequency and voltage to the lowest values that keep the NoC latency below a predetermined threshold. To cope with burstiness and hotspot patterns, which may lead the controller to overdrive the NoC with too high frequencies and voltages, leading to excessive power consumption, the congestion management strategy promptly identifies the flows that caused the abnormal traffic situation and eliminates them from the latency calculation, leading to a significantly higher power saving. Compared to a baseline DVFS strategy without congestion management, our results show that our proposal saves up to 33% more power when bursty or hotspot-based traffic patterns are detected. In addition, since we also apply power-gating to make an efficient use of the network buffers, we achieve an improvement of up to 10% in power savings when no bursts or hotspots are present.

Palabras clave— on-chip networks, congestion, DVFS, power, head-of-line blocking, CMPs, MPSoCs

I. INTRODUCTION

INTEGRATING a large number of processing elements into a single chip (CMPs, MPSoCs) is becoming the standard design choice in industry. This strategy offers good performance/power trade-off while saves costs. These systems must be delivered with built-in networks, known as network-on-chip (NoCs) [1]. Usually, the NoC is designed with strict requirements in terms of throughput and latency so it becomes one of the most important chip components to guarantee the expected chip performance. In addition, the NoC may represent up to 20% of the overall chip power consumption [2].

The current trend is to increase the number of processing units as long as technology shrinks. Examples of this trend are the 72-cores Tile-Gx [3] or the 256-cores Kalray MPPA-256 (Bostan) [4]. With the number of processing elements increasing, it is mandatory to use strategies that reduce overall power consumption without affecting significantly system performance. One of the most successful mechanism to perform this is DVFS [5], which drives voltage and frequency dynamically at runtime to fit the workload requirements.

DVFS-based mechanisms essentially collect metrics from the system to find out how it is performing.

According to this, the system reacts by increasing or decreasing frequency and voltage to meet the system requirements, thus saving power when requirements are low. The application of this mechanism to the NoC is not trivial. One main issue is to find the frequency that fits the whole NoC requirements. For small systems or systems with a very regular workload, it may become trivial. However, in larger or non-balanced workloads, that target may be difficult to achieve since some parts of the network may be overloaded while the rest of the network is completely underutilized.

On the other hand, an emerging trend is, instead of using several identical cores, to manufacture heterogeneous chips [6]. This paradigm is based on the fact that specialized processing units are more efficient at performing specific jobs. However, due to this heterogeneity the network load becomes very unbalanced and unpredictable, characterized by hotspot-based traffic patterns [7]. In addition, some application traffic patterns may naturally generate abrupt traffic bursts [8] and generate congested network regions.

Because of these reasons, it becomes apparent that guaranteeing acceptable performance levels while reducing power consumption is a real challenge. To illustrate this issue, in Figs. 1-3 we can see how a system that uses the NoC latency as metric to drive the DVFS mechanism (DMSD) [9] behaves under a critical traffic pattern. This pattern mixes a *background* traffic generated by regular nodes in a 8x8 2D mesh (e.g. general purpose processing units) with a *hotspot* traffic injected by four nodes towards a single node. This hotspot is representative of traffic generated by device hardware accelerators [10] or irregular traffic generated by some applications [8], both characterized by short and heavy-weight data bursts from/to neighbor nodes. Specifically, the background traffic is generated and received by all nodes not belonging to the hotspot following a uniform distribution pattern. Accordingly, the hotspot traffic is generated by injecting traffic at a high data rate to a given node from all of its neighbors.

The results reported in Figs. 1-3 have been obtained with a cycle-accurate NoC simulator which models 4-stages pipeline routers: IB (input buffer), RT (routing), VA/SA (virtual channel and switch allocation), X (link crossing). In Tab. I the rest of configuration parameters are described. The values in Tab. I are used to obtain our baseline results¹. To obtain our power results we used a modified version

¹Grupo de Arquitecturas Paralelas, DISCA, Universitat Politècnica de València, e-mail: jose1o@upv.es, jflich@disca.upv.es.

²Department of Electronics and Telecommunications, Politecnico di Torino, e-mail: mario.casu@polito.it.

³We also evaluated the robustness of our solution when some of these parameters are varied, as we will see in Sec. III-E.

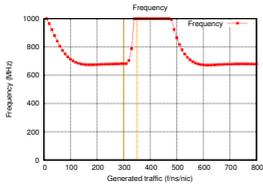


Fig. 1. Frequency for DMSD under hotspot traffic.

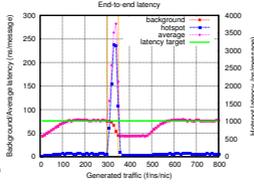


Fig. 2. End-to-end latency per traffic type for DMSD under hotspot traffic.

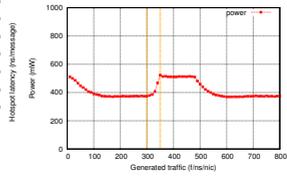


Fig. 3. Power consumption for DMSD under hotspot traffic.

of Orion v3.0 [11].

Fig. 1 shows the frequency increase as the hotspot is activated at $300 \mu s$. Fig. 2 shows how this increase differently affects the latency of two different traffic classes: congested traffic (hotspot traffic) and regular traffic (background traffic). The regular traffic is unnecessarily accelerated and its latency becomes less than a predetermined target (highlighted with a horizontal green dashed line), whereas the latency of the congested traffic increases significantly in spite of the high NoC clock frequency. As shown in Fig. 3, the consequence is a net power waste for an unwanted decrease of latency of the real productive traffic (the background one in our example).

In this paper we tackle such problem. We combine a latency-driven DVFS strategy (DMSD, as proposed in [9]), with a congestion management mechanism (ICARO [12]). Our goal is to use the congestion management strategy to discriminate and separate both traffic types, allowing the network to apply frequency and voltage policies based on the real productive traffic, hence allowing the DVFS strategy to guarantee a given end-to-end latency while optimizing power consumption.

The paper is organized as follows. First, we briefly describe DMSD and the methodology we used to obtain our results. Then, we describe ICARO, the selected congestion management mechanism. Then, we present the combined strategy and its internal arrangement. Next, we show the results and finally we plot the conclusions and the future work plans.

II. ANALYSIS OF THE DMSD DVFS POLICY

The purpose of the *Delay-based Max Slow Down (DMSD)* DVFS policy is to decrease the NoC frequency and voltage as much as possible without compromising the system performance [9]. To achieve this, DMSD uses the average end-to-end latency as a performance metric, and a Proportional-Integral (PI) controller that adapts frequency and voltage so as to keep that metric close to a *latency target*. The higher the latency target, the larger is the power saving. In our experiments, we set it equal to the latency that is obtained with an injection rate 5% less than the saturation point under uniform traffic.

In Fig. 4 an overview of an NoC provided with DMSD is depicted. Each node stores in a register

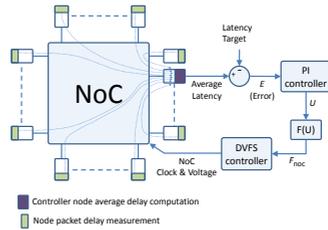


Fig. 4. All nodes in the network send latency measures to the PI controller to set the new frequency.

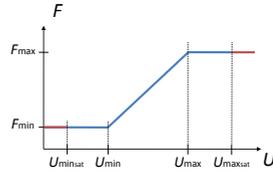


Fig. 5. Conversion from U to frequency.

the average end-to-end latency, updated each time a flit is received. Periodically, all nodes send the average latency to a given node, which computes the overall end-to-end latency for the whole system. In addition, this node contains the PI controller and the voltage and frequency controllers. Upon receiving all latencies, the overall latency at time n , L_n , is computed and the noise filter described by (1) is applied to obtain L'_n . Then, the *error* E_n is computed by subtracting the *latency target* L_t from L'_n as shown in (2). The error is then passed to the PI controller, which generates the signal U_n according to (3).

$$L'_n = \alpha L'_{n-1} + (1 - \alpha)L_n \quad (1)$$

$$E_n = L'_n - L_t \quad (2)$$

$$U_n = U_{n-1} + K_I E_n + K_P (E_n - E_{n-1}) \quad (3)$$

In (3), K_I and K_P are the integral and proportional gains determined empirically and used to ad-

Network configuration	
Topology	8x8 2D regular mesh
Routing policy	XY
Switching technique	Wormhole (bit-level)
Flow control	credits
Flit size	128 bits
Message size	10 flits
Switch queue size	4 flits
Virtual Channels	4 per Virtual Network
DMSD configuration	
Frequency range	333 - 1000 MHz
Voltage range	[0.56, 0.9] V
K_i, K_p	0.025, 0.0125
U saturation range	[-15, 15]
α	0.7

TABLE I
 BASELINE SIMULATION CONFIGURATION.

just the PI controller behavior while guaranteeing stability.

Finally, U is used to determine the frequency. For this, U is bounded within $U_{sat_{min}}$ and $U_{sat_{max}}$ and the range from U_{min} to U_{max} is linearly translated to frequency, as shown in Fig. 5. A voltage-to-frequency mapping is then used to apply the correct voltage for a given clock frequency.

DMSD performs well under stationary traffic patterns [9]. As shown in Figs. 1-3, however, the high intensity of a few data flows (hotspots) which are not representative of the whole system load, disrupts the DVFS strategy, leading to a waste of power by increasing the frequency and voltage unnecessarily.

Notice that this effect could be avoided by implementing Voltage and Frequency Islands (VFI) [13][14]. However, this would imply extra silicon area and power to implement the VFIs separate DVFS controllers, and it would require to either know at design-time where hotspots will be located, or the ability to confine the hotspot traffic in a separate voltage island at run-time. In contrast, our approach consists in implementing a congestion control mechanism (ICARO) to detect hotspots and filter them out, regardless their location and intensities.

Besides adding the support for buffer power-gating, whose importance will be disclosed later in Sec. III-C, by comparing Orion results with post-synthesis results obtained with our RTL version of the router, we found appropriate scaling factors that helped us remap the Orion results into those obtained on an industrial 28-nm CMOS technology. This was the target technology that we used for the implementation of routers and network-interfaces with support for congestion management.

III. IMPLEMENTING CONGESTION MANAGEMENT

Hotspot flows mask the overall system performance, increasing significantly the overall latency while the network resources not used by the hotspot flows may be underutilized. Our approach consists in identifying those hotspot flows, and separating them from the rest of the network traffic (background traffic). For this purpose, we pick ICARO, a congestion-control mechanism that detects, identifies, and isolates congestion within the network.

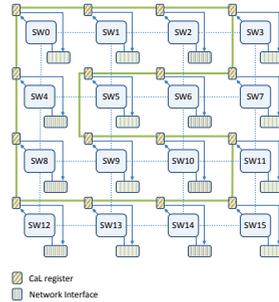


Fig. 6. Congestion Notification Network for a 4x4 mesh.

A. ICARO

ICARO removes the Head-of-Line (HoL) blocking by first identifying congestion, and then by isolating the congested flows involved in it into dedicated Virtual Networks (VNs). As the congested traffic is delivered through separate resources, it does not share buffer resources with the non-congested traffic, so HoL-blocking is removed. ICARO consists of three stages: congestion detection, notification and isolation.

A.1 Congestion Detection

The congestion is detected at the routers level, by keeping track of which input ports are requesting any output port. If more than one input port is requesting a given output port for too much time, that output port is marked as oversubscribed, so congestion at that output port is declared. However, two or more input ports could be requesting a given output port at a low data rate, not leading to congestion. Because of this, only input ports exceeding a given utilization threshold are considered.

A.2 Congestion Notification

Once congestion is detected, it must be notified to the NIs to isolate the traffic that causes the congestion before being injected into the network. To perform this, ICARO uses a dedicated congestion notification network (named *CaL* network²), which consists of a ring of N registers connected by links of $\log_2(N) + p + 1$ width, where N is the number of nodes and p the routers radix. An example of a *CaL* network is shown in Fig. 6. Other mechanisms for fast notification delivery include *express channels* [15] and circuit-switched networks. However, express channels may imply high area overhead while circuit-switched networks suffer from high latency penalties when establishing circuits.

²This network, in [12] is called CNN. However, since we extend its utility to deliver also other sort of messages, we changed its name to *CaL*(Congestion And Latency) network

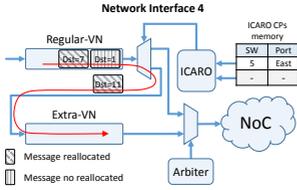


Fig. 7. NI with ICARO for reallocating congested messages.

A.3 Congestion Isolation

In absence of congestion, the NI allocates all messages in *regular-VNs*, as determined by the NI allocator. Once notifications are received, the NI uses the congestion information and calculates a message route to know whether that message will cross any of the *congested points (CPs)*. If so, the message is reallocated into a special VN (*extra-VN*) to be injected and delivered through it along all the path to destination. Otherwise, the message is injected through the current regular-VN. By doing this, flows contributing to the congestion are isolated into the *extra-VN*, keeping the non-congested traffic into the *regular-VNs*. In this way, DMSD will be able to measure the latency of the non-congested traffic (data flowing through the *regular-VNs*). Fig. 7 depicts an example of this process for the NI 4 in a 4x4 2D mesh.

When congestion ends, the conditions leading to detect CPs at routers will not occur anymore. Therefore, all the routers that previously detected CPs will detect this end of congestion and notify this event to the NIs, which will react by removing those CPs from their congestion notification board. Accordingly, since the messages pending to be injected at NIs will not cross any stored CP, those messages will be normally injected through the *regular-VN*.

B. Delivering latency measurements with the CaL network

In the original DMSD formulation, packets containing the measured end-to-end latency are sent to the controller node via piggybacking [9]. Intense congestion situations, however, may delay the delivery of those packets and the reaction of the PI controller, potentially causing the PI controller to oscillate. On the other hand, ICARO implements the CaL dedicated network, which we modified to support the delivery of those latency values in addition to congestion notifications. By doing this, the metrics necessary to set the frequency properly are guaranteed to timely arrive at destination (with low aggregated overheads, as we show in Sec. III-D).

In a DMSD-based system there are two types of nodes: those that send their latency metrics, and one that receives them. Thus, we implement two slightly different logic blocks to connect to the CaL network. Fig. 8 shows the logic associated to a typ-

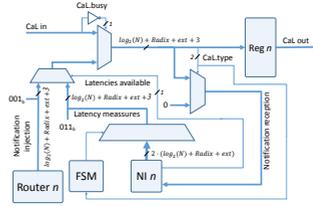


Fig. 8. CaL network register associated logic for regular nodes adapted to DMSD.

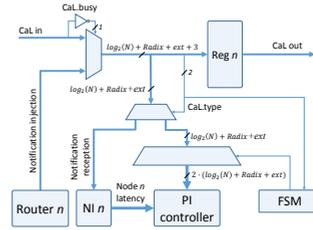


Fig. 9. CaL network register associated logic for the node provided with the PI/DVFS controller adapted to DMSD.

ical router/NI. It implements the necessary logic to send and receive ICARO notifications and to send DMSD latencies. It is worth to note that, despite ICARO notifications are sent in one cycle, the logic has been modified to serialize the transmission of 32-bit latencies through the CaL network by extending its links width by ($ext=8 \cdot Radix$) bits. Congestion notifications can be seamlessly interleaved with DMSD latency notifications because one bit identifies the type of CaL message. DMSD notifications from different nodes are guaranteed to arrive in order since nodes will send them after a fixed (and different for each node) time offset. Similarly to the sender node in Fig. 8, Fig. 9 shows the logic for the receiver node. This node sends and receives ICARO notifications like any other CaL node, adds its own latency measurements to the received ones, and forwards them to the PI controller.

C. Power-Gating Extra-VN Buffers

To avoid wasting power when there is no congestion, we implement a mechanism to power-off the extra-VN buffers via a centralized Power-Gating Controller (PGC), which resides in the same node that implements DMSD. All the buffers of an extra-VN (in all NIs as well as in all routers) are powered on/off simultaneously. Power-on is easy: the PGC node sniffs the CaL network and when it catches the first congestion notification it broadcasts a power-on message through the CaL network. On the contrary, power-off is not trivial. Snooping the CaL network in search of the “end of congestion”

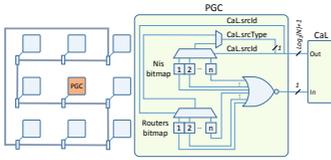


Fig. 10. Power-gating controller.

messages is not a valid strategy because there might still be messages in the extra-VN, either in the NIs pending to be injected, in the routers on their way to destination, or both. Therefore, to safely turn off the extra VNs, the PGC must be informed through the CaL network by both all NIs and routers about their detection of a *congestion-free* situation:

C.1 Network Interfaces detection

To make sure that no congested traffic will be injected into the network from a given NI, two conditions must be satisfied. First, the extra-VN buffers must be empty. In addition, the NI congestion notification board must be clean (no new notifications). If both conditions are met, the NI notifies its *congestion-free* status to the PGC with a special message sent through the CaL network.

C.2 Routers detection

At routers the mechanism is simpler. Each time the *extra-VN* buffer utilization increases from 0 to 1, the router sends a message to the PGC to inform that is storing congested traffic. On the other hand, when the buffer utilization decreases from 1 to 0, the router sends another message to inform that is *congestion-free*.

The PGC is provided with two N -width bitmaps, where N is the number of nodes in the network: one bitmap for the NIs and one for the routers. These bitmaps are updated any time a NI or a router notifies the PGC about its status (0=no congested traffic stored, 1=congested traffic stored). In this way, the PGC has a complete *congestion picture* of the network. When the PGC detects that all NIs and routers are *congestion-free*, it commands to power-off the extra-VN buffers; otherwise, it commands to power-on the buffers. Fig. 10 sketches the PGC bitmaps, the logic to power-on/off the buffers, and its connection to the CaL network. We quantify the advantage of using power-gating in Sec. III-E.

We are aware that turning on/off all the *extra-VN* buffers is suboptimal since several buffers could not be reached by any congested flow. Because of this, as a future work we plan to implement a new policy to turn the buffers on/off selectively.

D. Area Overhead Analysis

The bars in Fig. 11 illustrate the area overhead for a NI and a router with support for ICARO, with respect to a baseline implementation (no DMSD, no

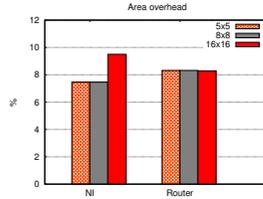


Fig. 11. ICARO+DMSD area overhead of different meshes.

ICARO)³. The results have been obtained after synthesis on our 28-nm technology, in the conditions of Tab. I, except for the mesh size that we let vary. We notice that the overhead is small, less than 10%, even for the case of a large 16×16 mesh.

E. Experimental Results

In this section we first report simulation results obtained in the baseline configuration of Tab. I. These results show that our combined DVFS and congestion management strategies can effectively solve the problem outlined in Sec. I that is at the basis of our work. Then, we report results obtained with a sensitivity analysis in which we varied several configuration parameters to check the robustness of our solution. Note that, for our experiments DMSD as well as ICARO are provided with the same amount of VNs in order to compare both solutions with the same amount of resources, providing each VN with the same amount of VCs. However, since DMSD does not require several VNs to work properly and these additional resources may affect negatively to its power consumption we perform an additional experiment comparing against the baseline with only 1 VN.

Figs. 12-14 compare the *DMSD* and the *DMSD+ICARO* cases in terms of latency, frequency, and power, in the baseline scenario. Notice that to properly compare the two cases, the two systems have the same total buffering resources. Note also that, in the case of ICARO, the *extra-VN* is composed of as many VCs as the *regular-VNs*.

Since ICARO effectively separates the background traffic from the hotspot one, DMSD can effectively measure only the latency of the background traffic. Therefore, thanks to the PI controller, DMSD keeps the latency of the background traffic around the 76-ns *latency target*, as shown in Fig. 12. In fact, as Fig. 13 shows, the NoC clock frequency is not influenced anymore by the activation of the hotspot traffic. This, in addition to the use of power-gating, results in a significant improvement of the power consumption, as shown in Fig. 14. When the hotspot is not active (from time 0 μ s to 300 μ s,

³We obtained overhead results only for ICARO since DMSD and the PG controller overheads are negligible compared to the ICARO's overhead.

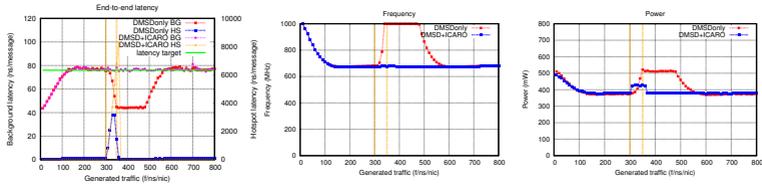


Fig. 12. End-to-end latencies for the background and the hotspot traffic. Fig. 13. Frequencies for DMSD and DMSD+ICARO. Fig. 14. Power consumption for DMSD and DMSD+ICARO.

Scenarios								
Label	Mesh Size (nodes)	Queue Size	VCs	Msg. Length (flts)	Num. HS	HS Dur. (ns)	Lat. target (ns)	
Baseline	8x8	4	4	10	1	50us	76	
5x5	5x5	4	4	10	1	50us	66	
16x16	16x16	4	4	10	1	50us	105	
qs2	8x8	2	4	10	1	50us	79	
qs8	8x8	8	4	10	1	50us	81	
qs16	8x8	16	4	10	1	50us	72	
vcs2	8x8	4	2	10	1	50us	60	
vcs8	8x8	4	8	10	1	50us	97	
ml5	8x8	4	4	5	1	50us	62	
ml20	8x8	4	4	20	1	50us	96	
2HS	8x8	4	4	10	2	50us	76	
3HS	8x8	4	4	10	3	50us	76	
short	8x8	4	4	10	1	25us	76	
large	8x8	4	4	10	1	100us	76	

TABLE II
ROBUSTNESS ANALYSIS SCENARIOS CONFIGURATION.

and then again after around 380 μs), the extra-VN buffers are powered-off, resulting in lower power for the DMSD+ICARO case. When the hotspot is active, the extra-VN buffers are switched on, hence the power increases. Still, since the clock frequency in the DMSD+ICARO case is less than the DMSD case, the power consumption is also significantly reduced.

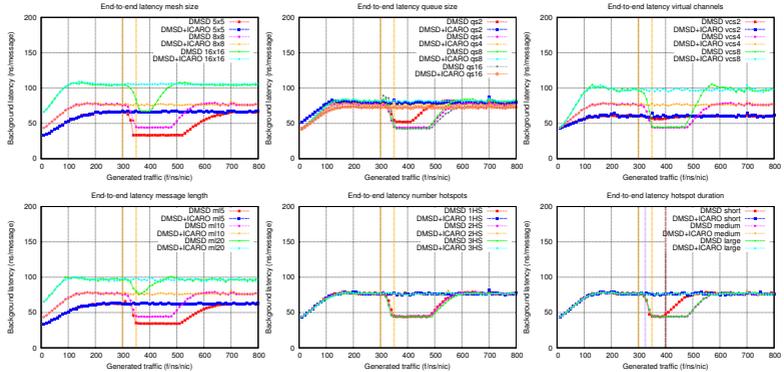
To validate our results under different network configurations, we changed several network parameters: mesh size, router buffers queues size, number of virtual channels, message length, number of hotspots, and hotspot duration. All the cases analyzed are described in Tab. II, in which every case is assigned a label that is used next in the graph keys. As Fig. 15 shows for all the configurations analyzed, in the DMSD+ICARO case the background traffic correctly tracks the prescribed target, hence avoiding the excessive power consumption that characterizes the reference DMSD case. Note that, since the goal of our approach is to keep the background latency around the latency target, for better understanding, hotspot latencies have been omitted in the graphs. Also note that the hotspot start/stop time is highlighted with vertical bars and that in the hotspot duration graph the three different hotspot ending times are highlighted with different colors. Please note that the latency target value for a given scenario depends not only on the saturation point, which is highly correlated with the system configuration, but also on the latency curve gradient. Therefore, in some

system configurations the calculated latency target seems not to follow an intuitive progression like in the VCs analysis graph shown in Fig. 15.

Fig. 16 summarizes the improvement of power consumption of the DMSD+ICARO case, in all the configurations of Tab. II. Two different improvement values are reported. The first one is due to the extra-VN power-gating (no-HS in the graph), measured at time 290 μs (just before the hotspot activation); the second one corresponds to the power-saving during the hotspot duration (HS in the graph) and is calculated by averaging the power spent from time 300 μs to time 600 μs, since this is the time range in which the hotspots affect any of the cases analyzed. Note that the power overhead due to the additional hardware required by our proposal is already included in the power consumption graphs.

As Fig. 16 shows, for all the cases considered, the combination of DMSD and ICARO leads to a significant power improvement compared with DMSD only when hotspot is active. When no hotspot is active, our proposal saves power by switching the extra-VN off, achieving up to 10% power saving. However, for the most part of cases no power-saving is achieved since ICARO uses only 1VN, therefore, the data rate is higher compared with the DMSD case which increases the power consumption significantly. When congested traffic is detected, ICARO manages this sort of traffic and DMSD tunes the frequency properly saving up to 33% power consumption. In the results obtained when the hotspot is present, we observe more variability. This is expected as, for calculating the average, we take values from the same range of time for all cases but the durations of the effects of the hotspots are not the same for those cases, therefore, the weight of those values over the average is not the same.

To finalize our experiments we analyze our proposal against DMSD provided with 1VN. DMSD, as opposed to ICARO, does not require several VNs to perform properly, so we performed the same robustness analysis shown above but providing DMSD with 1VN. In the same way, as ICARO does not require the extra-VN to be provided with several VCs, for this simulations ICARO is configured with the same number of VCs for the regular-VN as the DMSD case and only 1VC for the extra-VN. In Figure 17 power results for this analysis are shown. As can



network to select at each router the output port which messages are forwarded through. Differently from our approach, RCA collects metrics delivered through the regular network. Thus, if some metrics travel along already congested routes, this may slow down the metrics collection, causing the mechanism to make wrong decisions. Besides, adapting the routes to avoid hotspots may result in moving the location of such hotspots from one place to another. Finally, avoiding hotspots may be impossible if all the flows are bound to the same destination (e.g. the memory controller).

The authors of [30] propose HPRA, a hotspot-formation prediction mechanism. HPRA uses an Artificial Neural Network-based (ANN) hardware that gathers buffer utilization data to predict the formation of hotspots. Then, HPRA classifies the traffic into two classes: hotspot-destined traffic (HSD) and non-hotspot-destined traffic (nonHSD). HSD traffic is throttled at source while the nonHSD traffic is routed avoiding paths containing hotspots routers. However, in the cases in which the ANN fails to predict hotspots, it may redirect traffic to an unpredicted hotspot, causing an even worse degradation of the system performance. Besides, HPRA suffers from the same metrics delivering issue of previously described RCA.

In [31], the authors propose a mechanism to monitor the state of the network in order to select the best path to deliver each packet. To select the best next router for a given packet at each hop, each router in the network must know the best path to follow from the current node to the packet's destination. This requires sending back a special message with the route status information for each message sent from a given node to a given destination. This may cause a waste of network bandwidth and, in presence of several or sudden congestion, it may be affected by the same problem of delayed metrics delivery described before. In addition to this, this mechanism requires that each node keeps a table composed of one entry for each node in the network. This means that the total stored aggregated data in the whole network grows quadratically with the number of nodes, which clearly hampers scalability for large mesh sizes. In contrast, in our case every node only stores a 32-bit latency value, which results in a linear growth with the number of nodes.

V. CONCLUSIONS AND FUTURE WORK

In this paper we present an integrated approach for saving power while guaranteeing latencies in NoCs under non-stationary traffic patterns. We demonstrate that by integrating a congestion management strategy and a loop-based DVFS controller to tune the frequency for saving power while guaranteeing a latency target, we obtain a power-effective strategy. As our results show, we save up to 33% power compared with the baseline DVFS system in presence of hotspots. In addition to this, we propose a power-gating mechanism to power-off buffers when

not needed, resulting in up to 10% power saving in absence of congested traffic. To obtain these results, our approach requires a small area overhead, less than 10%.

As future work we plan to compare the approach reported in this paper with another one that separates traffic classes using physically distinct networks with two different DVFS controllers rather than different virtual networks. In addition to this, as already mentioned, we plan to implement a smarter mechanism to power buffers on/off selectively to achieve best power saving results.

ACKNOWLEDGMENTS

This work was supported by the Spanish Ministerio de Economía y Competitividad (MINECO) and by FEDER funds under Grant TIN2015-66972-C05-1-R and by Ayudas para Primeros Proyectos de Investigación from Universitat Politècnica de València under grant ref. 2370. We also want to thank specially to the HIPEAC project which fund the internship during which this work was developed.

REFERENCIAS

- [1] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70-78, Jan 2002.
- [2] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The alpha 21364 network architecture," in *Hot Interconnects 9, 2001.*, 2001, pp. 113-117.
- [3] T. Corp., "Tilera tile multicore processors," Available at <http://www.tilera.com/products/processors/TILE-Gx-Family>.
- [4] Kalray. (2014) Kalray, mppa-256 bostan. [Online]. Available: <http://www.kalrayinc.com/kalray/products/>
- [5] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey, "A voltage reduction technique for digital systems," in *Solid-State Circuits Conference, 1990. Digest of Technical Papers. 37th ISSCC., 1990 IEEE International, Feb 1990*, pp. 238-239.
- [6] R. Kumar, D. Tullsen, N. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," *Computer*, vol. 38, no. 11, pp. 32-38, Nov 2005.
- [7] A. Bakhoda, J. Kim, and T. Amodi, "Throughput-effective on-chip networks for manycore accelerators," in *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on, Dec 2010*, pp. 421-432.
- [8] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and exploiting program phases," *Micro, IEEE*, vol. 23, no. 6, pp. 84-93, Nov 2003.
- [9] M. R. Casu and P. Giacccone, "Rate-based vs delay-based control for dvfs in noc," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, ser. DATE '15*. San Jose, CA, USA: EDA Consortium, 2015, pp. 1096-1101. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2757012.2757067>
- [10] R. Hou, L. Zhang, M. Huang, K. Wang, H. Franke, Y. Ge, and X. Chang, "Efficient data streaming with on-chip accelerators: Opportunities and challenges," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on, Feb 2011*, pp. 312-320.
- [11] A. Kahng, B. Lin, and S. Nath, "Orion3.0: A comprehensive noc router estimation tool," *Embedded Systems Letters, IEEE*, vol. 7, no. 2, pp. 41-45, June 2015.
- [12] J. Escamilla, J. Flich, and P. Garcia, "Icaro: Congestion isolation in networks-on-chip," in *Networks-on-Chip, 2014, NoCS 2014, 8th International Symposium on, Sept 2014*, pp. 159-166.
- [13] D. Lackey, P. Zuchowski, T. Bednar, D. Stout, S. Gould, and J. Cohn, "Managing power and performance for system-on-chip designs using voltage islands," in *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on, Nov 2002*, pp. 195-202.

- [14] U. Ogras, R. Marculescu, D. Marculescu, and E. G. Jung, "Design and management of voltage-frequency island partitioned networks-on-chip," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 3, pp. 330–341, March 2009.
- [15] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: Towards the ideal interconnection fabric," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 150–161, Jun. 2007. [Online]. Available: <http://doi.acm.org/10.1145/1273440.1250681>
- [16] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das, "A case for dynamic frequency tuning in on-chip networks," in *Proc. 42nd Int. Symp. Microarchitecture (MICRO-42)*, Dec. 2009, pp. 292–303.
- [17] L. Guang, E. Nigussie, L. Koskinen, and H. Tenhunen, "Autonomous DVFS on supply islands for energy-constrained NoC communication," in *Arch. Comput. Sys. - ARCS 2009*, ser. Lect. Notes Comput. Sc., 2009, vol. 5455, pp. 183–194.
- [18] L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *Proc. 9th Int. Symp. High-Perf. Comput. Arch. (HPCA)*, 2003, pp. 123–124.
- [19] J. Zhan, N. Stoimenov, J. Ouyang, L. Thiele, V. Narayanan, and Y. Xie, "Optimizing the NoC slack through voltage and frequency scaling in hard real-time embedded systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 11, pp. 1632–1643, Nov. 2014.
- [20] R. Hesse and N. E. Jerger, "Improving DVFS in NoCs with coherence prediction," in *Proceedings of the 9th International Symposium on Networks-on-Chip*, ser. NOCS '15. New York, NY, USA: ACM, 2015, pp. 24:1–24:8.
- [21] X. Wang, T. Wang, T. Mak, M. Yang, Y. Jiang, and M. Daneshdalan, "Fine-grained runtime power budgeting for networks-on-chip," in *The 20th Asia and South Pacific Design Automation Conference*, Jan 2015, pp. 160–165.
- [22] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar, "A 2 Tb/s 6x4 mesh network for a single-chip cloud computer with DVFS in 45 nm CMOS," vol. 46, no. 4, pp. 757–766, Apr. 2011.
- [23] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras, "In-network monitoring and control policy for DVFS of CMP networks-on-chip and last level caches," *ACM Trans. on Design Automation of Electronic Systems*, vol. 18, no. 4, pp. 1–21, Oct. 2013.
- [24] X. Chen, Z. Xu, H. Kim, P. V. Gratz, J. Hu, M. Kishinevsky, U. Ogras, and R. Ayoub, "Dynamic voltage and frequency scaling for shared resources in multi-core processor designs," in *Proc. 50th Design Automation Conference (DAC)*. ACM Press, 2013, pp. 114:1–114:7.
- [25] J.-Y. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou, "Up by their bootstraps: Online learning in artificial neural networks for cmp oncore power management," in *High Performance Computer Architecture (HPCA)*, 2014 IEEE 20th International Symposium on, Feb 2014, pp. 308–319.
- [26] A. Bianco, P. Giaccone, M. R. Casu, and N. Li, "Exploiting space diversity and dynamic voltage frequency scaling in multiplane network-on-chips," in *2012 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2012, pp. 3080–3085.
- [27] J. Henkel, H. Bukhari, S. Garg, M. U. K. Khan, H. Khdr, F. Kriebel, U. Ogras, S. Parameswaran, and M. Shafiqe, "Dark silicon: From computation to communication," in *Proceedings of the 9th International Symposium on Networks-on-Chip*, ser. NOCS '15. New York, NY, USA: ACM, 2015, pp. 23:1–23:8. [Online]. Available: <http://doi.acm.org/10.1145/2786572.2788707>
- [28] M. K. Yadav, M. R. Casu, and M. Zamboni, "LAURA-NoC: Local automatic rate adjustment in network-on-chips with a simple DVFS," vol. 60, no. 10, pp. 647–651, Oct. 2013.
- [29] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *HPCA*. IEEE Computer Society, 2008, pp. 203–214.
- [30] E. Kakoulli, V. Soteriou, and T. Theodorides, "Hpra: A pro-active hotspot-preventive high-performance routing algorithm for networks-on-chips," in *Computer Design (ICCD)*, 2012 IEEE 30th International Conference on, Sept 2012, pp. 249–255.
- [31] F. Farahnakian, M. Ebrahim, M. Daneshdalan, P. Liljeberg, and J. Plosila, "Q-learning based congestion-aware routing algorithm for on-chip network," in *Networked Embedded Systems for Enterprise Applications (NESEA)*, 2011 IEEE End International Conference on, Dec 2011, pp. 1–7.

Evaluando un escenario de pruebas para el IoT entre la emulación y el uso de dispositivos reales

Jorge E. Luzuriaga¹, Marco Zennaro², Juan Carlos Cano¹, Carlos Calafate¹ y Pietro Manzoni¹

Resumen— El manejo de una gran cantidad de dispositivos, el tráfico que estos generan, así como la utilización de estos datos eficientemente son los retos a los que nos enfrentamos desarrolladores, investigadores y entusiastas del Internet de las cosas y de las redes inalámbricas de Sensores.

Para aliviar este problema, en este artículo proveemos una metodología que permite rápidamente recrear escenarios, investigar cómo se comunican diferentes dispositivos, así como ejecutar y observar el comportamiento de aplicaciones y servicios con un nivel de abstracción que nos proporciona la capa de aplicación.

Esta metodología la hemos validado con un caso de estudio, en el cual utilizando el protocolo de mensajería MQTT-SN (*Message Queuing Telemetric Transport for Sensor Networks*) en una arquitectura con múltiples nodos como publicadores y suscriptores. Donde ha sido evaluado su comportamiento y rendimiento bajo diferentes configuraciones utilizando dos plataformas distintas con una misma finalidad, la simulación y el uso de dispositivos reales en un testbed.

Palabras clave— testbed; iot; m2m; mqtt; mqtt-sn; platform evaluation; iot4d.

I. INTRODUCCIÓN

EL internet de las cosas es un sistema adaptable y auto configurable compuesto de redes de sensores, objetos industriales, objetos de nuestro día a día y objetos inteligentes [3]. Con el propósito de interconectar todas estas cosas, para de alguna u otra manera hacerlas inteligentes, programables y capaces de interactuar con los seres humanos.

Para observar el comportamiento de una aplicación destinada al Internet de las cosas normalmente en un laboratorio tenemos dos opciones: la primera que es probar la aplicación usando simulaciones y/o emulaciones, y la otra alternativa es implementar la aplicación en un testbed, compuesto por dispositivos reales.

El coste es una de las principales razones por la que laboratorios con un pequeño presupuesto consideran simuladores/emuladores como primera opción, porque normalmente son gratuitos incluso si estos son de pago es la opción más conveniente si se compara con tener que adquirir un gran número de dispositivos o al menos los suficientes para realizar las pruebas. Específicamente en redes inalámbricas de sensores las motas simuladas tienen menos restricciones que el hardware real por que valores como

memoria y de CPU se pueden configurar.

Los resultados obtenidos nos ayudan a comprender el comportamiento de un sistema/aplicación/componente bajo determinadas situaciones pero con una reducida precisión y fidelidad [7].

Otra desventaja es que normalmente una simulación requiere grandes recursos de cómputo, no importa cuantas mejoras sean hechas en los emuladores porque no se puede conseguir probar aplicaciones de la misma forma en la que se haría con dispositivos reales [9].

La ejecución de pruebas en dispositivos físicos es una parte indispensable del proceso de desarrollo de aplicaciones y servicios que no debe ser omitida ni olvidada. Realizar pruebas con dispositivos físicos permite corregir y resolver muchos otros problemas que no han sido tomados en cuenta por imprecisiones asumidas en una simulación. Pero se debe considerar que pasar una aplicación que ha sido validada en simulación a un test-bed no es trivial. En un testbed tenemos ciclos de temporización exactos pero capacidades limitadas de depuración y continuamente padecemos por los recursos limitados de las motas.

En el presente artículo, evaluamos la integración de las redes inalámbricas de sensores con las redes empresariales de nuestro día a día, mediante el uso de protocolos de comunicación data-céntricos orientado a mensajes (MOM) estructurado como publicador/suscriptor (pub/sub), debido al soporte del mismo frente a sistemas heterogeneos, topologías de red dinámicas y a su escalabilidad respecto a el número de nodos [4].

El resto del artículo está ordenado de la siguiente manera: en la sección II presentamos la metodología que proponemos, junto con un caso de estudio. La sección III describe conceptos claves de las plataformas físicas y simuladas para las motas WSN. En la sección IV definimos la arquitectura y la evaluamos, los resultados de la misma se presentan en la sección V. La sección VI muestra los trabajos relacionados. Finalmente, las conclusiones y el trabajo futuro se presentan en la Sección VII.

II. METODOLOGÍA

En figura 1 resumimos como debemos proceder para crear una aplicación IoT. La primera cosa por hacer es seleccionar la plataforma entre las diversas plataformas de desarrollo. Una vez seleccionada

¹Grupo de Redes de Computadores (GRC), Universitat Politècnica de València, e-mail: jorlu@upv.es, [jucano, calafate, pmanzoni]@disca.upv.es

²International Centre for Theoretical Physics (ICTP) / ICT4D Lab, e-mail: mzennaro@ictp.it

la plataforma vamos a realizar el diseño de todos los componentes que componen nuestra red. Es la parte más crítica porque debemos tener en cuenta que trabajamos con dispositivos con recursos limitados tanto energéticos así como de procesamiento y de espacio en memoria. Debemos definir con nuestra aplicación software que dispositivos funcionarán como sensores, como actuadores, o como gateways. Además de el canal que vamos a usar, que protocolos de comunicación y si haremos procesamiento de datos directamente en el nodo antes de enviar los datos al nodo coordinador. Teniendo siempre en mente que debemos optimizar la duración de la batería de los nodos. En el lado del *backend* la gestión de los datos, si estos se van a almacenar en una base de datos local o serán enviados a un servicio web.

La instalación requiere un análisis previo de interferencia de nuestra red de sensores con otras redes inalámbricas como WiFi o Bluetooth, eligiendo obviamente los canales libres. Y según el escenario posicionar las motas de forma óptima y estratégica.



Fig. 1. Metodología para el desarrollo de aplicaciones IoT.

A. Caso de Estudio

En esta sección presentamos un caso de estudio que sigue la metodología propuesta para las dos plataformas seleccionadas basadas en la arquitectura que se muestra en la figura 2. Ambas plataformas usan el sistema operativo Contiki sobre el cual vamos a evaluar la implementación de MQTT-SN.

Para la plataforma física del testbed hemos seleccionado las motas Z1 de Zolertia. Para la simulación de nuestra arquitectura MQTT-SN hemos seleccionado el simulador Cooja.

Con el objetivo de comparar los resultados que nos ofrece ejecutar una aplicación que recoge datos y los envía utilizando el protocolo MQTT en un entorno real contra uno simulado.

B. Procedimiento

El procedimiento que sigue la metodología varía según las especificaciones del caso de prueba. Consideramos necesario que si el desarrollador no está familiarizado con el software y el hardware que va

a utilizar debe involucrarse de la máxima información posible. Una vez que tenga claro como funcionan las cosas se requiere definir las entradas y salidas que el sistema recibirá y/o ofrecerá. Se escribirá y se compilará el código para la plataforma elegida, especificando si el programa reaccionará a eventos internos o bien el programa se ejecutará hasta alcanzar un tiempo prefijado. Luego se preparará y se ejecutará el test en un ambiente seleccionado.

Finalmente en la verificación de los datos recolectados de las diferentes plataformas, estos se deben validar y comparar en sí. En la plataforma simulada los datos están típicamente disponibles en ficheros *log*, y en las plataformas del testbed esta información se puede recolectar mediante los ficheros *log* de los sistemas involucrados.

III. CARACTERÍSTICAS DE LAS MOTAS DE NUESTRA RED INALÁMBRICA DE SENSORES

En esta sección se realiza una descripción de los componentes software y hardware necesarios para la evaluación de nuestro caso de estudio¹

En este proyecto, cuando nosotros cargamos el firmware a las motas de la red inalámbrica de sensores, lo hacemos utilizando el sistema operativo Contiki, cuando el código de una aplicación está preparado es distribuido y cargado a todas las motas de la plataforma testbed específicamente a las motas Z1, una imagen de esta plataforma la podemos observar en la figura 3. Contiki también nos ofrece la posibilidad de probar nuestras aplicaciones sobre una plataforma de simulación llamada Cooja, una imagen de esta plataforma la podemos ver en la figura 4.

A. El Sistema Operativo Contiki

Es un sistema operativo de código abierto creado por Adam Dunkel en el 2002 desarrollado específicamente para dispositivos inalámbricos de bajo consumo y con limitada cantidad de memoria. Los requerimientos de memoria son de únicamente 10 Kbytes de memoria RAM y 30 Kbytes de ROM [10]. Actualmente es soportado por una amplia y activa comunidad de desarrolladores.

Contiki soporta la pila del protocolo IPv6 para el protocolo de encaminamiento de bajo consumo en redes con pérdidas de sus siglas en inglés RPL, así como para el direccionamiento y la compresión de cabeceras 6LoWPAN para enlaces 802.15.4.

B. Plataforma hardware: La placa Z1 de Zolertia

Es el módulo de bajo consumo que sirve como plataforma de desarrollo de aplicaciones para redes inalámbricas de sensores producida por Zolertia en la ciudad de Barcelona. La placa soporta los sistemas operativos de código abierto más usados por la comunidad de redes inalámbricas de sensores como TinyOS y Contiki [11].

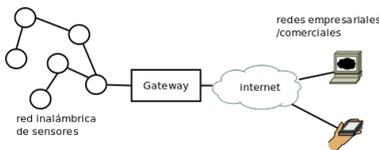


Fig. 2. Arquitectura de las aplicaciones IoT.

¹ Todo el código fuente recopilado, estudiado y adaptado a las necesidades de este proyecto está disponible en el siguiente repositorio: <https://github.com/jluzuria2001/TS-IT/>.



Fig. 3. Imagen con algunos de los componentes de la plataforma testbed.

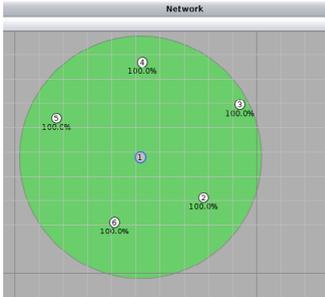


Fig. 4. Captura de pantalla de la plataforma de simulación con 6 motas donde el número uno es el gateway.

La placa ofrece flexibilidad y extensibilidad a cualquier combinación de fuentes de alimentación, sensores y actuadores. Y dispone de un conector USB micro-B por el cual puede ser alimentada y además sirve para cargar el firmware de Contiki.

C. Plataforma de simulación: El simulador Cooja

Cooja es un simulador de red diseñado específicamente para redes inalámbricas de sensores que permite ser usado para probar y depurar aplicaciones.

Cooja permite tener diferentes tipos de motas con diferentes aplicaciones en una misma simulación. Cuando se crea una mota, Contiki debe ser compilado y cargado en Cooja antes de empezar una simulación. Los resultados de la simulación por defecto no se conservan pero se pueden almacenar manualmente en un fichero de texto para su posterior procesamiento.

Una de las principales ventajas es que no existe ninguna diferencia entre el código para una mota simulada con el código de una mota física. Si se emula una mota Z1 el mismo firmware es simulado tal cual podría ser subido y ejecutado en una placa Z1. Si se simula una mota Contiki, las librerías y los módulos principales son idénticos como podrían ejecutarse en un hardware real pero los controladores difieren [7].

Cooja típicamente no interpreta datos de las mo-

tas simuladas. No entiende del protocolo MAC simplemente envía los bytes transmitidos entre diferentes motas, como sucedería con motas reales, con la probabilidad de que los paquetes sean descartados en algún lugar de la pila de Contiki.

El modelo de radio es crucial para la precisión de la simulación, el más usado en Cooja es *Unit Disk Graph* debido a su simplicidad, donde los nodos se comunican e interfieren en círculos de radio fijo. Pero también para calcular la probabilidad de que dos motas se comuniquen se puede utilizar los modelos de radio de grafo directo o el modelo *Free Space*. Además ofrece una multitud de opciones para configurar cómo ruido, ganancias de la antena, objetos atenuantes con refracciones y reflexiones, aunque esto sea probablemente lejano a la realidad comparado a un ambiente real.

D. MQTT y MQTT-SN

MQTT (*Message Queuing Telemetric Transport*) es un protocolo publicador/subscriptor ligero diseñado específicamente para aplicaciones móviles y comunicaciones de máquina a máquina. Es un protocolo abierto optimizado para comunicaciones sobre redes con escaso ancho de banda y conexiones de red intermitentes [5].

Sin embargo MQTT trabaja sobre el protocolo de red subyacente TCP/IP, para garantizar la entrega de mensajes sin pérdidas. Pero esto es demasiado complejo para dispositivos simples, pequeños de bajo coste tales como sensores y actuadores inalámbricos [3]. Tomando en cuenta las peculiaridades de un ambiente de comunicación inalámbrica, donde los enlaces de radio presentan altas tasas de fallos, debido a su susceptibilidad a perturbaciones, desvanecimientos e interferencias [2].

En el 2008, *A. Stanford-Clark* y *H. Linh Truong* ambos de IBM publican las especificaciones del protocolo MQTT-SN que puede ser considerado como una versión de MQTT adaptada para afrontar estos problemas pero utilizando como protocolo de transporte UDP.

IV. DEFINICIÓN DE LA ARQUITECTURA

La arquitectura consta por una parte de la red inalámbrica de sensores, que es la que podremos probar sobre la plataforma física y bajo simulación [6]. La red inalámbrica de sensores está compuesta de clientes publicadores y suscriptores Y la red empresarial que añade dos componentes: un broker de mensajería igual al que tendríamos en un escenario de MQTT y un gateway cuya principal función es la traducción de mensajes de MQTT a MQTT-SN y viceversa. Esta parte de la red es ejecutada siempre sobre la plataforma física que en nuestro escenario utilizamos Raspberry Pis 2B.

A. Evaluación de la Arquitectura

Evaluamos la arquitectura bajo dos diferentes y posibles enfoques incrementando el número de

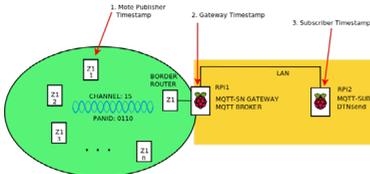


Fig. 5. Diagrama de la arquitectura que interconecta una red inalámbrica de sensores con una red empresarial cableada.

clientes MQTT-SN desde uno hasta cinco para cada uno de los enfoques.

El primer enfoque consiste en la publicación de mensajes desde las motas a un suscriptor MQTT. Mediante la transferencia de mensajes pequeños de tamaño alrededor a 20 Bytes que encajen en un frame 802.15.4 transparentemente sin necesidad de ser fragmentados. Por efectos de evaluación el contenido de cada mensaje producido por una mota contiene: el identificador del nodo, el valor sentido de la temperatura, el número secuencial de los mensajes enviados y la marca de tiempo del instante en que el mensaje fue publicado.

`nodeID : 44, temp : 24, counter : 154, ts1 : 663783`

Cuando el mensaje atraviesa el gateway y es recibido por el broker MQTT este es publicado para todos los suscriptores que están interesados en este tema en específico.

Para el segundo enfoque la publicación de mensajes se realiza desde un publicador MQTT-SN a las motas suscritas, publicamos mensajes pequeños de la misma manera que en el anterior enfoque, los mensajes contienen la marca de tiempo del instante en que fueron publicados junto con el número de secuencia de mensaje con una cadencia de envío constante fijada en un segundo, una vez que el gateway recibe el mensaje lo publica a todas las motas suscritas al tema.

Como entrada establecemos que enviamos 1800 mensajes, cuando las motas operan como publicadoras lo hacen con una periodicidad variable generada por un número pseudo randomico entre 1 y 10 segundos. Y para cuando las motas estan como suscriptoras, el publicador externo lo realiza con una periodicidad fija de un segundo.

V. RESULTADOS

Despues de ejecutar el caso de prueba 5 veces en ambas plataformas con el primer y el segundo enfoque tenemos los siguientes resultados:

A. 5 Motas Publicadoras y 1 Suscriptor

El tiempo que toma ejecutar las pruebas en un ambiente simulado así como en un entorno real debido a las diferencias en el tiempo de ejecución de las plataformas, como se puede ver en la figura 6, toma cerca de 66 minutos en el simulado mientras que en el testbed sobre las 3 horas y media aproximadamente.

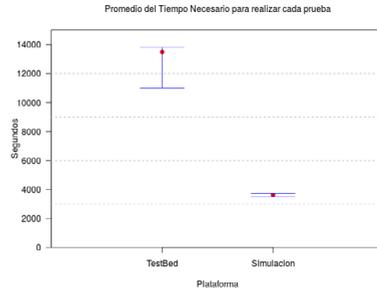


Fig. 6. Diagrama de la arquitectura que interconecta una red inalámbrica de sensores con una red empresarial cableada.

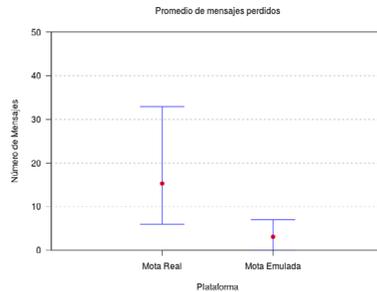


Fig. 7. Diagrama de la arquitectura que interconecta una red inalámbrica de sensores con una red empresarial cableada.

El número de mensajes perdidos en un ambiente simulado con cinco motas publicadoras es en promedio cerca de 6 veces menor al de un testbed. Como lo podemos observar en la figura 7.

B. 5 Motas Suscriptoras y 1 Publicador

El tiempo que toma ejecutar estas pruebas es de 30 minutos, en el cual 1800 mensajes son publicados a una tasa de 1 mensaje por segundo. En estas pruebas nos encontramos algunos problemas el principal es que uno de los cinco suscriptores que pierde la conexión con el broker, donde este cierra las conexiones con los clientes que no reciben la respuesta a una petición *ping*.

Respecto al número de mensajes perdidos si juntamos todas las pruebas son 45000 mensajes publicados de las cuales en la plataforma testbed 41434 mensajes de han sido recibidos por las motas, es decir el 7% del total de los mensajes se han perdido. Mientras que en una plataforma simulada tenemos 43487 mensajes recibidos lo que nos deja un 3,4% de mensajes perdidos.

El número de mensajes perdidos en una plataforma testbed con cinco motas suscritas a un tema en común es en promedio cerca del doble al que podemos encontrar en una plataforma simulada.

VI. TRABAJO RELACIONADO

Debido al apogeo del internet de las cosas, varios protocolos están siendo usados ampliamente por desarrolladores de aplicaciones y por la industria en general, entre ellos el protocolo MQTT por las múltiples ventajas que nos ofrece frente a sus competidores. También podemos encontrar interesantes trabajos académicos como en [1] donde mediante simulación realizada con NS2 se muestra el impacto de varios parámetros como la calidad de servicio para un sistema MQTT-SN y se observa el rendimiento del protocolo en la red. Además los autores realizan un modelado derivando la probabilidad de entrega y el retardo teóricamente como función del protocolo MQTT y otros parámetros. Los autores como trabajo futuro se plantean realizar sus mediciones sobre una implementación real como la que nosotros presentamos en este artículo.

En [8] los autores presentan los resultados de una comparación realizada con tres protocolos distintos para el IoT. Dos de ellos son ZigBee y 6LoWPAN, implementados como soluciones centralizadas. El tercero es una solución de software distribuido llamado SDWN de redes inalámbricas definidas por software. Todos ellos evaluados en la plataforma EuWiN observando métricas de rendimiento como pérdida de paquetes, tiempo de ida y vuelta (RTT) y la sobrecarga generada en la red con diferentes tipos de tráfico. Los resultados muestran que SDWN es la mejor solución en ambientes estáticos o semi estáticos por ejemplo con aplicaciones para hogares y edificios inteligentes, debido a la óptima explotación de recursos combinado con el reducida sobrecarga. Pero SDWN presenta degradaciones de rendimiento y limitaciones en ambientes dinámicos debido a la gran cantidad de tiempo que requiere para refrescar las rutas. Así en presencia de condiciones dinámicas ZigBee y 6LoWPAN continúan siendo las mejores opciones.

VII. CONCLUSIONES Y TRABAJO FUTURO

Las motas simuladas en una red inalámbricas de sensores permiten captar rápidamente como puede funcionar una aplicación/servicio/componente debido a que no se consideran en su totalidad las restricciones impuestas por el hardware real como pueden ser memoria y CPU.

Pero una parte fundamental del proceso de desarrollo de aplicaciones y servicios para las futuras aplicaciones destinadas al Internet de las cosas y a las ciudades inteligentes es la ejecución de pruebas en dispositivos físicos, si se consideran los sensores y actuadores como la base tecnológica facilitadora de la intercomunicación entre los seres humanos y las máquinas.

En este artículo hemos evaluado la arquitectura del protocolo MQTT-SN bajo redes inalámbricas de

sensores en ambas plataformas, la física y la simulada, con respecto al incremento del número tanto de publicadores como de suscriptores, con el objetivo de analizar sus potencialidades y puntos en los que se podría optimizar.

Como trabajo futuro estamos trabajando en la integración del protocolo MQTT-SN con la capa común de protocolos tolerantes a intermitencias en la conexión y a retardos, tanto en la red inalámbrica de sensores como en la red empresarial.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por el Ministerio de Economía y Competitividad, Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, Proyectos I+D+I 2014, España, con la ayuda de referencia TEC2014-52690-R. Agradecemos también al Centro de Física Teórica Abdus Salam ICTP en especial a T/ICT4D por la hospitalidad brindada durante los últimos tres meses correspondientes al periodo de movilidad de la estancia doctoral.

REFERENCIAS

- [1] K. Govindan and A. Azad, *End-to-end service assurance in IoT MQTT-SN* 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), 2015.
- [2] Collina, M. and Bartolucci, M. and Vanelli-Coralli, A. and Corazza, G.E. *Internet of Things application layer protocol analysis over error and delay prone links*, Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC), pp.398,404, 8-10 Sept. 2014
- [3] A. Stanford-Clark and H. Linh Truong, *MQTT for Sensor Networks (MQTT-SN) Protocol Specification*.
- [4] J. Luzuriaga, M. Perez, P. Boronat, J. Cano, C. Calafate and P. Manzoni *Impact of mobility on Message Oriented Middleware (MOM) protocols for collaboration in transportation* IEEE 19th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2015.
- [5] Chen, Whei-Jen and Gupta, Rahul and Lampkin, Valerie and Robertson, Dale M. and Subrahmanyan, Nagesh, *Responsive Mobile User Experience Using MQTT and IBM MessageSight* IBM Corp., 2014.
- [6] M. Wehrle, C. Plessl, J. Beutel and L. Thiele, *Increasing the reliability of wireless sensor networks with a distributed testing frameworks*, Proceedings of the 4th workshop on Embedded networked sensors, EmNets '07, 2007.
- [7] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, *Protothreads: simplifying event-driven programming of memory-constrained embedded systems*, In Proceedings of the 4th international conference on Embedded networked sensor systems, Acm 2006.
- [8] C. Buratti, A. Stajkic, G. Gardasevic, S. Milardo, M. Abrigiani, S. Mijovic, G. Morabito and R. Verdano, *Testing Protocols for the Internet of Things on the EuWiN Platform* IEEE Internet of Things Journal, vol. 3, no. 1, pp. 124-133, 2016.
- [9] Davis, Ernesto García and Calveras, Anna and Demirkol, Ilker, *Improving packet delivery performance of publish/subscribe protocols in wireless sensor networks* Sensors 2013, 13, 648-680; doi:10.3390/s130100648
- [10] A. Liñán Colina, A. Vives, M. Zennaro, A. Bagula and E. Pietrosenoli, *IoT in 5 Days*, 1st ed. 2016.
- [11] Zolertia *Z1 Datasheet v1.1*. Rev.C. Available: <http://zolertia.sourceforge.net>, March, 2010 (Last accessed on May 20, 2016)

Un recorrido por el análisis forense

Juan Manuel Castelo Gómez¹ y José Luis Martínez Martínez¹

Resumen—Este artículo explica los fundamentos del análisis forense, enumerando las herramientas y distribuciones principales usadas y se resuelve un reto forense publicado por la Digital Forensic Research Workshop en 2009.

Palabras clave— Análisis Forense, DFRWS 2009 Challenge, Jornadas Sarteco.

I. INTRODUCCIÓN

HOY día es imposible imaginar una empresa que no tenga las medidas de seguridad adecuadas para que la información que maneja esté lo suficientemente protegida como para que no acceda a ella alguien que no debe, incluso existen leyes como la Ley Orgánica de Protección de Datos [1] que establecen unos requisitos mínimos de seguridad y penalizan gravemente su incumplimiento. Para poder cumplir con estos requisitos, se realizan auditorías y se invierten millones de euros en tener un sistema lo más seguro y avanzado posible, teniendo departamentos específicos que se encargan de que la seguridad sea un proceso continuo y un sistema nunca esté desactualizado [2], teniendo como objetivo impedir que se sufran daños económicos o de imagen que puedan acabar con el negocio. En este punto, el análisis forense [3] tiene una participación muy activa, ya que permite descubrir posibles problemas en un sistema, determinar si cumple con su función y, en el peor de los casos, determinar las causas por las que se ha producido un incidente y evitar que vuelvan a ocurrir. De esta forma, podremos proteger la información que trata la empresa, de manera que solo las personas autorizadas puedan acceder a ella, y nos cercioraremos de que los sistemas son seguros, impidiendo que un dispositivo infectado pueda causar una incidencia en la organización. En este aspecto tiene especial relevancia el análisis forense de la red, que permitirá proteger las comunicaciones que se establecen en la empresa y evitar que se propague un posible malware que afecte a un integrante de la red.

Existen también otros campos en el que aplicamos el análisis forense. Un archivo en un disco duro o un proceso en memoria pueden ser la clave para encontrar a un delincuente [4], incluso puede que un ordenador sea el objetivo de un delito o el medio que se utilice para poder cometerlo. Por ello, el análisis forense se está erigiendo como uno de los medios más utilizados en un proceso legal para poder hallar evidencias y poder determinar los hechos acaecidos en un delito [5].

En este artículo, se mostrarán tanto los conceptos básicos, incluyendo herramientas, que debemos cono-

cer del análisis forense para, posteriormente, aplicarlas en un caso simulado de análisis forense en el cuál podremos ver la gran cantidad de información que podemos obtener de un dispositivo y la utilidad que tiene para determinar cómo y qué ha ocurrido en un suceso.

Está dividido en 5 secciones, en la primera sección se recogerá el fundamento teórico ligado al análisis forense, tras ello, veremos las herramientas más relevantes en los principales tipos de forense que existen y finalizaremos con un caso práctico en el que resolveremos un reto forense publicado en el año 2009 por la Digital Forensic Research Workshop en el que apreciaremos cómo se realiza un análisis forense.

II. FUNDAMENTO TEÓRICO

El análisis forense se define como el conjunto de principios y técnicas que comprenden el proceso de adquisición, conservación, documentación, análisis y presentación de evidencias digitales y que, llegado el caso, puedan ser aceptadas legalmente en un proceso judicial. Las características principales que debe poseer un análisis forense es que éste debe ser verificable (se debe poder comprobar la veracidad de las conclusiones), reproducible (se debe poder realizar de nuevo las pruebas realizadas), documentado (se debe documentar el proceso de manera comprensible y detallada) e independiente (las conclusiones deben ser las mismas independientemente de quién realice el proceso y la metodología que emplee) [6].

Por tanto, un análisis forense puede ser realizado con diversas finalidades: preventiva/auditora, correctiva, de detección o judicial/probatoria. A su vez, podemos clasificar el análisis en función del dispositivo al que se le va a realizar o el sistema operativo que utiliza.

A continuación, se van a describir las fases en las que está constituido un análisis forense.

A. Asegurar la escena

El objetivo de esta fase es evitar que se produzca alguna alteración en la escena que pueda comprometer la integridad de las posibles evidencias. Para ello, primeramente, se garantizará la seguridad de las personas que se puedan encontrar en la escena y, posteriormente, se restringirá el acceso a la escena a las personas que no estén autorizadas, estableciendo un perímetro de seguridad.

Será clave identificar las posibles escenas y crear los perímetros para todas ellas, aunque al final resulte que no tenían relación con la escena principal.

Otro punto clave es impedir que la información que hay en los dispositivos de la escena pueda ser alterada o incluso borrada, por lo que se deberán tomar

¹Instituto de Investigación en Informática. Universidad de Castilla-La Mancha, Campus Universitario, 02071, Albacete, España. e-mail: juanmanuel.castelo@alu.uclm.es, joseluis.martinez@uclm.es

las medidas adecuadas para protegerla, incluso hasta desconectándolos de las redes, inclusive la eléctrica.

B. Identificación de evidencias

En la informática forense, se define evidencia como cualquier información contrastable encontrada en un sistema [7]. Para que una evidencia sea útil, ésta ha de cumplir cinco características: que sea admisible (debe tener valor legal), auténtica (debe ser verídica), completa (debe representar la prueba desde un punto de vista objetivo), fiable (no debe haber dudas sobre su obtención) y creíble (debe ser comprensible) [8].

C. Adquisición y preservación de evidencias

Para realizar la adquisición de las evidencias nos basamos en la norma RFC 3227 [9] que recoge una serie de recomendaciones sobre las pautas que se deben seguir a la hora de adquirir las evidencias del sistema y en qué orden realizar la adquisición. Para cada sistema o dispositivo existe un orden de volatilidad, por lo que habrá que determinar para cada uno qué se debe adquirir primero.

La adquisición se realizará haciendo una copia bit a bit del soporte que contiene la evidencia. Esto supone copiar toda la información del dispositivo, lo que se denomina clonado. Todas las evidencias que se adquieran deben estar perfectamente identificadas. El objetivo es proporcionar la mayor cantidad de información que se pueda sobre la evidencia. Además, toda evidencia tendrá que ir identificada con un código hash MD5 [10] o SHA1 [11].

Por otro lado, se debe preservar la cadena de custodia que tiene como fin garantizar la autenticidad de una evidencia [12].

D. Análisis de evidencias

La forma de proceder con el análisis viene determinada por el momento de realizar el análisis. Nos encontramos con dos posibles situaciones: realizar un análisis en caliente, en el que utilizaremos las pruebas originales, o un análisis en frío, en el que usaremos clones de las pruebas originales, que evitarán que alteremos las muestras iniciales. Durante el análisis debemos determinar cuándo, cómo y qué ha ocurrido, así como quién ha participado [13].

Para la creación de la línea temporal lo más sencillo es fijarnos en las fechas de modificación, creación y acceso que tienen cada archivo. Cualquier indicio o patrón que podamos encontrar puede ser clave para poder encauzar el análisis. Encontraremos muchos tipos de datos a analizar, desde ficheros ocultos hasta información eliminada.

Para determinar el procedimiento llevado a cabo para realizar el ataque es clave el poder acceder a la información volátil que tenía el sistema. En ella, encontraremos los últimos programas ejecutados y mucha información en memoria que puede indicar cómo se actuó [14].

En la identificación de los autores nos podemos basar en los usuarios que tiene sesión iniciada, los contactos de una agenda o las conexiones de red

abiertas que, a partir de la dirección IP, nos determinarán quién estaba conectado al dispositivo.

Por último, para determinar el impacto causado es clave ver qué información se ha visto comprometida. Es importante poder determinar los daños económicos y el alcance que ha tenido un ataque o un problema en el sistema, sin olvidar otros posibles tipos de daños como los de imagen [13].

E. Realización y presentación del informe

El informe es uno de los elementos esenciales de un análisis forense, ya que es lo que releja los resultados, las técnicas y los procedimientos llevados a cabo durante todo el análisis. Debe estar escrito de forma clara, adaptándolo en función de los conocimientos informáticos del receptor.

III. HERRAMIENTAS DISPONIBLES

Dentro del mundo del análisis forense encontramos una gran variedad de herramientas que implementan diferentes técnicas, incluso se han creado distribuciones que tienen como objetivo recopilar las herramientas más importantes, de forma que sea mucho más sencillo realizar todo el proceso de adquisición, preservación, análisis y documentación [15].

A. Análisis de las distribuciones

A.1 Linux

Linux cuenta con una serie de herramientas aplicables en forense. En concreto, los comandos “dd”, “fdisk”, “grep” o “md5sum” nos son muy útiles, sobre todo en los forenses de discos [16].

A.2 CAINE

CAINE, Computer Aided Investigate Environment, es una distribución GNU/Linux que surgió en el 2008 como una tesis de Giancarlo Giustini en la Universidad de Módena y Reggio Emilia [17].

Su característica principal radica en que es una distribución live, es decir, no necesita instalación para ser ejecutada.

En lo que respecta a la versión actual, denominada CAINE 7.0 o DeepSpace, consiste en una distribución Ubuntu 14.04 que utiliza el kernel Linux 3.13 y el entorno de escritorio MATE 18.2, contando únicamente con una versión de 64 bits.

Las herramientas que nos proporciona CAINE debemos decir que son múltiples y variadas y se nos presentan separadas en diferentes categorías según su utilidad, entre las que destacamos:

- Clonado de discos: dd y Guymager.
- Análisis: Autopsy.
- Hasheo: GtkHash y QuickHash.
- Creación de cronología: NbTempo y Log2TimeLine.
- Forense a memoria: Volatility.
- Visionado de bases de datos: SQLiteman y SQLite database browser.
- Forense a red: WireShark o ZenMap [18].

A.3 DEFT

DEFT, Digital Evidence & Forensics Toolkit, es una distribución live basada en GNU/Linux que es ejecutable tanto desde USBs como CDs. Es una creación de Stefano Fratapietro, que sigue en cargo del proyecto actualmente junto con la ayuda de otros compañeros como Sandro Rossetti o Paolo Dal Checco [19].

La versión actual de DEFT, denominada 8.2, está basada en Ubuntu 12.10 con escritorio LXDE. Está disponible en las versiones de 32 y 64 bits y en los idiomas inglés, italiano y español.

Las herramientas destacables de esta distribución son: Autopsy, Foremost, Dash, Guymager, Xplico, Maltego y Mobius Forensic [20].

A.4 SIFT

SIFT, SANS Incident Forensic Toolkit, es una recopilación de herramientas con licencia GNU/Linux en la que encontramos las principales utilidades para poder realizar un análisis forense. Fue creado por un grupo de expertos que pertenecían al instituto estadounidense SANS en 2008, en el que destacaba Rob Lee [21].

Su instalación se realiza mediante línea de comandos partiendo de un sistema operativo Ubuntu 14.04, ya que no es una distribución.

Encontramos las herramientas Volatility, DDRescue, Foremost o WireShark [22].

A.5 Kali Linux

Kali Linux es la distribución GNU/Linux por excelencia de seguridad. Está basada en Debian y contiene más de 600 herramientas que permiten realizar diversas tareas como auditoría, ingeniería inversa, análisis forense o test de penetración. La distribución fue creada en el año 2013 por la empresa Offensive Security, una entidad que proporciona cursos y certificaciones relacionadas con la seguridad y que tienen como apoyo esta distribución [23].

En Kali Linux encontramos infinidad de herramientas, pero de análisis forense destacamos Autopsy, Bulk, DFF, Foremost, Volatility y Volafax.

B. Herramientas más relevantes

En este apartado vamos a destacar las principales herramientas forenses que son necesarias para poder realizar un análisis forense de disco, memoria y red, que serán los que efectuaremos en la siguiente sección. Comenzamos con las aplicaciones para realizar forenses a disco, en el que encontramos cuatro clases de herramientas: de clonado, montaje, recuperación y análisis.

La herramienta por excelencia para realizar clonados de disco es FTK Imager [24]. Esta aplicación, desarrollada por la empresa AccessData, está disponible para ser ejecutada de forma portable, es decir, sin necesidad de instalación, tanto para sistemas Linux como Windows.

Otra herramienta utilizada para el clonado es Guymager [25]. Esta aplicación solo está disponible para

ser ejecutada en sistemas Linux y cuenta con la ventaja de que proporciona una interfaz gráfica. Simplemente debemos buscar el dispositivo origen y elegir la opción de clonarlo o adquirirlo.

Para el montaje encontramos herramientas muy simples como el comando mount [26] de Linux, muy utilizado incluso fuera del ámbito forense para poder trabajar con dispositivos de almacenamiento extraíble. Si preferimos una versión gráfica, podemos optar por la aplicación XMount [27].

Las aplicaciones de análisis y las de clonado suelen incluir como funcionalidad el poder montar las imágenes o discos, por lo que a veces no se suelen utilizar herramientas específicas de montaje.

Para la recuperación de archivos, la herramienta más utilizada es Foremost [28]. Esta aplicación fue desarrollada por la Oficina de Investigaciones Especiales de la Fuerza Aérea de los Estados Unidos, aunque ahora está disponible con licencia pública, siendo también de código libre. Otra herramienta destacable es Photorec [29].

Si buscamos una aplicación para realizar un análisis, la mejor opción es Autopsy [30]. Las funcionalidades que nos permite son: el análisis de archivos, metadatos, obtención de información del sistema y usuarios, visionado del historial web y búsquedas, la recuperación de archivos y la creación de cronologías e informes. También se suele utilizar la herramienta DFF (Digital Forensic Framework) [31], aunque es menos completa.

Para realizar análisis de memoria, lo primero que necesitamos es realizar la adquisición de la información contenida en la memoria. Para ello, podemos utilizar herramientas como FTK Imager [24] o LiMe [32], en caso de que queramos adquirir la memoria de un sistema Linux. Una vez que disponemos del dumper de memoria, pasamos a analizarlo. La aplicación más utilizada es Volatility [33] que contiene una inmensa cantidad de opciones y opera en sistemas Windows y Linux. Podremos determinar información como los procesos que estaban en ejecución, las conexiones que se estaban realizando en el sistema.

En el caso del análisis de red, también contamos con dos tipos de herramientas: de adquisición y de análisis. Para la adquisición es necesaria la utilización de aplicaciones que nos permitan capturar tráfico. La más común de ellas es Tcpdump [34] que funciona bajo línea de comandos en sistemas Linux. La captura de tráfico se almacenará en un fichero .pcap que será el que analizaremos.

Para el análisis destaca WireShark [35], que nos permite ver todos los paquetes que se han intercambiado, viendo la información que contienen todos sus campos y ofreciendo una gran cantidad de filtros. Otra herramienta interesante es NetworkMiner [36] que nos representa de forma más resumida la información más relevante que se puede obtener de la captura como los host que aparecen, las conexiones o las consultas DNS efectuadas. Estas dos herramientas mencionadas permiten también la captura de tráfico

y el análisis in vivo.

También es destacable Xplico [37] que permite extraer de una captura de tráfico información sobre los protocolos más importantes que nos serán representados de forma gráfica lanzando un servicio web en el puerto 9876.

IV. ANÁLISIS FORENSE

A modo de resumen ofrecemos una comparativa que nos ayudará a determinar cuál es la distribución más completa de todas las que hemos analizado, basándonos en las herramientas ofrecidas y su aplicación en los principales tipos de análisis forense.

TABLA I
COMPARATIVA DE LAS DISTRIBUCIONES.

	CAINE	DEFT	SIFT	Kali Linux
Forense a discos	✓	✓	✓	✓
Forense a memoria	✓	✗	✓	✓
Forense a móviles	✗	✓	✗	✗
Forense a red	✗	✓	✗	✗

Podemos observar que no hay ninguna distribución que consiga reunir las herramientas necesarias como para poder realizar los principales tipos de análisis forense. Para realizar nuestro análisis, tendremos como base la distribución CAINE, ya que nos ha resultado la más fácil de utilizar y ofrece un muy buen soporte debido a la gran comunidad con la que cuenta.

El análisis que vamos a llevar a cabo es un reto del DFRWS [38] publicado en 2009 en el que se combina el análisis de disco, memoria y red.

El supuesto en el que nos encontramos es el siguiente:

- Un usuario llamado “nssal” estaba distribuyendo mediante una consola Sony Playstation 3 imágenes con contenido ilegal. Dichas imágenes tenían como temática el famoso festival Mardi Gras. Por ello, se procedió a monitorizar el tráfico generado por el usuario.
- Tiempo después se descubre que este usuario estaba comunicándose con otro individuo que trabajaba en la Universidad Johns Hopkins como administrador de redes y se sospecha que puede haber existido un intercambio de información entre ambos, por lo que se procedió a realizar un análisis forense para esclarecer lo ocurrido.

Las evidencias que se nos han proporcionado para analizar son las que se listan a continuación:

- Una imagen de disco de una consola Sony Playstation 3 que pertenece al usuario “nssal”.
- Una imagen de un dispositivo de almacenamiento extraíble conectado a la consola Sony

Playstation 3 del usuario “nssal” y perteneciente a él mismo.

- Un volcado de memoria realizado a la consola Sony Playstation 3 del usuario “nssal”.
- Una captura de red en la que se monitoriza el tráfico generado por el usuario “nssal”.
- Una captura de red en la que se realiza una presunta comunicación entre el usuario “nssal” y una persona perteneciente a la Universidad Johns Hopkins.
- Una captura de red proporcionada por el administrador de redes de la Universidad Johns Hopkins en la que se comunica con el usuario “nssal”.
- Una imagen de disco de una consola Sony Playstation 3 que pertenece al administrador de redes de la Universidad Johns Hopkins.

El objetivo del análisis es determinar si ha habido un intercambio de información entre ambos usuarios y si hay algún contenido inusual en alguna de las evidencias adquiridas.

El procedimiento a seguir durante este análisis será el siguiente:

- Primeramente se analizará la imagen de la consola Sony Playstation 3 del usuario “nssal”. Para ello, utilizaremos la herramienta Autopsy y ExifTool, esta última con el objetivo de analizar los posibles metadatos que existan.
- Se procederá, posteriormente, a analizar la imagen del dispositivo de almacenamiento extraíble conectado a la consola Sony Playstation 3 del usuario “nssal”. Además de las herramientas mencionadas anteriormente, utilizaremos la aplicación GtKHash para la generación de códigos hash.
- La siguiente evidencia a analizar será la adquisición de memoria realizada a la consola Sony Playstation 3 del usuario “nssal”, empleando la herramienta ramparser.
- Pasaremos a analizar las capturas de tráfico. Comenzaremos analizando el tráfico monitorizado del usuario “nssal”, luego analizaremos la presunta comunicación entre “nssal” y un individuo de la universidad Johns Hopkins y finalizaremos con la captura proporcionada por el administrador de redes de la Universidad Johns Hopkins. Las aplicaciones a utilizar serán NetworkMiner, WireShark y Xplico.
- La última evidencia a analizar será la imagen de disco de la consola Sony Playstation 3 perteneciente al administrador de redes de la Universidad Johns Hopkins. Se utilizarán las mismas herramientas que las empleadas en el análisis de la imagen de disco y usb de “nssal”.

Tras la realización del análisis, se ha recopilado la siguiente información que tiene relación directa con el caso:

- En la carpeta “Images” del usuario “nssal” se encuentran una serie de fotografías tomadas por

él mismo con la temática del festival Mardi Gras. Los metadatos contenidos en esas imágenes indican que las fotografías han sido tomadas por "nssal".

- El usuario "nssal" realiza varias comunicaciones con el administrador de redes de la Universidad Johns Hopkins. Dichas conexiones se realizan mediante el protocolo SSH, lo que hace que se encripte la información que se envía, no pudiendo determinar la información intercambiada.
- De las comunicaciones realizadas entre ambos dispositivos vemos que "nssal" ha enviado más de 4 Gigabytes de información al administrador y ha recibido 300 Megabytes. Está claro que ha habido intercambio de información, pero no podemos determinar si ese contenido es ilícito.
- Se encuentra una carpeta denominada "Pictures" en el usuario "jhui", que contiene las mismas fotografías tomadas por el usuario "nssal", creadas a la misma hora que se realizó la conexión SSH. Esto es lo que nos hace determinar que las imágenes han sido intercambiadas durante la comunicación.

También se ha encontrado la siguiente información que, aunque no tiene relación directa con el objetivo del caso, es inusual encontrarla en un sistema:

- En la carpeta del usuario "jhui", perteneciente a la consola Sony Playstation 3 del usuario "nssal", encontramos un historial de comandos y un archivo llamado "backd00r". Tras analizar el historial, se confirma la ejecución del archivo "backd00r", instalando una puerta trasera en la máquina de "nssal", pudiendo tener acceso a su máquina cuando desee.
- En la carpeta del usuario "nssal", "Recipes", perteneciente a la consola Sony Playstation 3 del usuario "nssal", encontramos cuatro archivos cuyo contenido son recetas extrañas.
- En una de las conexiones realizadas entre los dos dispositivos, el administrador de redes tiene acceso a la máquina del usuario "nssal", de la cual elimina una carpeta denominada "backd00r". Dicha conexión se realiza a través de la puerta trasera instalada.
- Se detectan conexiones activas entre el administrador de redes y "nssal" en el momento que se realiza la adquisición de memoria. Dichas conexiones han sido creadas por un proceso denominado "backd00r", lo que confirma que la puerta trasera seguía activa en el momento de la adquisición de la memoria de la consola Sony Playstation 3 de "nssal".
- En la carpeta del usuario "goatboy", perteneciente a la consola Sony Playstation 3 del administrador de redes, encontramos una carpeta denominada "Recipes", creada por él mismo. El contenido de esta carpeta es el mismo que encontrado en la consola de "nssal", aunque encontramos una carpeta más que contiene un archivo con nombres

y direcciones. En el historial de comandos ejecutados en el terminal, se determina que la carpeta "Recipes" ha sido creada por el usuario "goatboy".

- En la carpeta del usuario "jhui", perteneciente a la consola Sony Playstation 3 del administrador de redes, encontramos el código correspondiente a la puerta trasera instalada en la máquina de "nssal".

A partir de las pruebas obtenidas hemos establecido una cronología que se detalla a continuación:

- "Nssal" y el administrador de redes se ponen en contacto para establecer una sesión SSH. La primera comunicación se realiza a las 17:42 del 11 de Marzo de 2009, finalizando las comunicaciones a las 18:02 del mismo día.
- En esas comunicaciones se realiza un gran intercambio de información entre ambos usuarios, incluyendo ciertas imágenes tomadas por "nssal" del festival Mardi Gras.
- En una de los intercambios realizados, el administrador de redes copia el contenido de una carpeta denominada "Recipes" a la máquina de "nssal" que contiene información de cómo preparar ciertas sustancias.
- El administrador de redes, en uno de los accesos que realiza a la máquina de "nssal", instala una puerta trasera que posteriormente utiliza para borrar los restos dejados de la instalación de dicha puerta trasera, aunque sigue en funcionamiento, como comprobamos en el análisis de memoria, permitiendo al administrador de redes conectarse a la consola de "nssal" cuando desee.

V. CONCLUSIONES

Tras conocer la base teórica del análisis forense, sus herramientas y, sobre todo, su aplicación práctica podemos concluir que es una técnica muy útil para poder recolectar y analizar la información que puede contener un dispositivo, siendo extremadamente eficaz para determinar lo que ha ocurrido en él.

AGRADECIMIENTOS

Este trabajo ha sido cofinanciado por el Ministerio de Economía y Competitividad y la Comisión Europea bajo el proyecto TIN2015-66972-C5-2-R (MINECO/FEDER).

REFERENCIAS

- [1] Noticias Jurídicas, "Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal." http://noticias.juridicas.com/base_datos/Admin/1o15-1999.html.
- [2] ISO/IEC 27001- Information Security Management.
- [3] Miguel López Delgado, *Análisis Forense Digital*, 2007.
- [4] Larry Daniel, *Digital Forensics for Legal Professionals: Understanding Digital Evidence from the Warrant to the Courtroom*, Syngres, 2012.
- [5] Council of Europe, *Convenio sobre la Cibercriminalidad*, 2009.
- [6] Asier Martínez Retenaga, *Guía de Toma de Evidencias en Entornos Windows*, INCIBE - Instituto Nacional de Ciberseguridad.

- [7] Juan Garrido Caballero, *Análisis Forense Digital en Entornos Windows*, Informática64, 2012.
- [8] American Board of Information Security & Computer Forensics and American Board for Certification in Homeland Security, "Digital Forensics: An Introduction."
- [9] D. Brezinski. Internet Engineering Task Force, "Guidelines for evidence collection and archiving," RFC 3772, Feb. 2002.
- [10] Request For Comments, *RFC 132 - The MD5 Message-Digest Algorithm*.
- [11] Request For Comments, *RFC 3174 - US Secure Hash Algorithm 1 (SHA1)*.
- [12] Dr. Santiago Acurio Del Pino, *Manual de Manejo de Evidencias Digitales y Entornos Informáticos*.
- [13] Carles Gervilla Rivas, "Metodología para un análisis forense," M.S. thesis, Universidad Oberta de Catalunya, 2014.
- [14] Juan Luis García Rambla, *Un Forense Llevado a Juicio*.
- [15] Information Security and Forensics Society (ISFS), *Computer Forensics Part 2: Best Practices*, 2009.
- [16] Barry J. Grundy, *The Law Enforcement and Forensic Examiner. Introduction to Linux A Practitioner's Guide to Linux as a Computer Forensic Platform*, 2008.
- [17] Caine-live.net, "History of the Project. CAINE," <http://www.caine-live.net/page4/history.html>.
- [18] Caine-live.net, "List of tools. CAINE," <http://www.caine-live.net/page11/page11.html>.
- [19] Deflinux.net, "About — DEFT Linux - Computer Forensics live CD," <http://www.deflinux.net/about/>.
- [20] Stefano Fratepietro, Alessandro Rossetti, and Paolo Dal Checco, *DEFT 7 Manual. Digital Evidence & Forensic Toolkit*, 2012.
- [21] Networking SANS SysAdmin Audit and Security Institute., "Sans SIFT Kit/Workstation: Investigative Forensic Toolkit," <https://digital-forensics.sans.org/community/downloads>.
- [22] Networking SANS SysAdmin Audit and Security Institute. SANS.org, "Packages - SANS Investigative Forensics Toolkit 3.0 documentation," <http://sift.readthedocs.org/en/latest/user/packages.html>.
- [23] Kali Linux. Kali.org, "About Kali Linux," <https://www.kali.org/about-us/>.
- [24] AccessData Corp. Command Line Imager Support, "Using Command Line Imager," .
- [25] Guy Voncken. Guymager.net, "Guymager Free Forensic Imager," <http://guymager.sourceforge.net/>.
- [26] Tutorialspoint.com, "Mount - Unix Linux Command," http://www.tutorialspoint.com/unix_commands/mount.htm.
- [27] "XMOUNT - www.penguin.lu," <https://penguin.lu/xmount>.
- [28] United States Air Force Office of Special Investigations. Foremost.org, "Foremost - Recovery Tool," <http://foremost.sourceforge.net/>.
- [29] CGSecurity. CGSecurity.org, "PhotoRec ES - CGSecurity," http://www.cgsecurity.org/wiki/PhotoRec_ES.
- [30] Brian Carrier. Sleuthkit.org, "Autopsy - The Sleuth Kit," <http://www.sleuthkit.org/autopsy/>.
- [31] Arxsys. Arxsys.fr, "Digital Forensics Framework (DFF)," <http://www.arxsys.fr/>.
- [32] 504ensicsLabs. Github.com, "LiME - Linux Memory Extractor," <https://github.com/504ensicsLabs/LiME>.
- [33] The Volatility Foundation. Volatilityfoundation.org, "The Volatility Foundation - Open Source Memory Forensics," <http://www.volatilityfoundation.org/>.
- [34] Man2htnl. Tcpcdump.org, "Manpage of TCPDUMP. User Commands," <http://www.tcpcdump.org/tcpcdump-man.html>.
- [35] Wireshark Foundation. Wireshark.org, "Wireshark - Network Protocol Analyzer," <https://www.wireshark.org/>.
- [36] Netresec AB. Netresec.org, "NetworkMiner - The NSM and Network Forensics Analysis Tool," <http://www.netresec.com/?page=NetworkMiner>.
- [37] Gianluca Costa & Andrea De Franceschi. Xplico.org, "Xplico - Open Source Network Forensic Analysis Tool (NFAT)," <http://www.xplico.org/>.
- [38] DFRWS - Digital Forensics Research Conference, "DFRWS 2009 Forensics Challenge Challenge Data and Submission Details," <http://www.dfrws.org/2009/challenge/submission.shtml>.

Arquitecturas del subsistema de memoria y almacenamiento secundario

Particionado dinámico de cachés compartidas para maximizar la equidad entre tareas

Vicent Selfa, Julio Sahuquillo, María E. Gómez y Salvador Petit ¹

Resumen.—Las peticiones de las distintas aplicaciones en ejecución en los procesadores multinúcleo y/o multihilo actuales compiten entre ellas por el acceso y uso de los recursos compartidos, como las cachés o la memoria principal. Esta compartición de recursos afecta de manera diferente a las prestaciones de cada una de las aplicaciones, complicando en gran manera la toma de decisiones en base a su comportamiento. En otras palabras, las aplicaciones, cuando se ejecutan concurrentemente junto a otras aplicaciones, *progresan* de manera distinta a como lo harían si se ejecutasen en solitario, en un entorno sin compartición de recursos. Así, mientras que algunas aplicaciones apenas ven afectado su progreso por el hecho de estar compartiendo recursos, otras aplicaciones, en cambio, pueden ver su progreso drásticamente reducido al ejecutarse junto con otras cargas. Este problema se conoce como falta de equidad, inequidad o *unfairness* y ha sido atacado recientemente a nivel software, utilizando políticas que intentan maximizar la equidad en el planificador del sistema operativo. No obstante, esta aproximación impide que el planificador se centre en maximizar las prestaciones, por lo que una implementación puramente hardware que minimice los problemas de equidad sería muy interesante. En este artículo proponemos FPCP (*Fair Progress Cache Partitioning*), que reduce la inequidad del sistema particionando la caché de último nivel y distribuyendo las vías disponibles entre las aplicaciones en ejecución en base a estimaciones sobre su progreso. Estas estimaciones de progreso se realizan en tiempo de ejecución, sin necesidad de un análisis previo *offline*. Los resultados experimentales muestran que FPCP consigue que el *unfairness* sólo supere el 10% en un 1.5% de las cargas analizadas, 10 veces menos que en un sistema sin particionado de caché, todo ello sin decrementar las prestaciones.

Palabras clave.—Particionado de cache, progreso por hilo, equidad del sistema, multinúcleo.

I. INTRODUCCIÓN

LA compartición de recursos en los procesadores multinúcleo actuales (CMPs) mejora tanto el grado de utilización del hardware como la eficiencia (i.e. mayor productividad), reduce las latencias de comunicación y es compatible con el modelo de memoria compartida. No obstante, puede favorecer la contención cuando peticiones de distintos hilos compiten entre ellas por los recursos compartidos. Por ello, cuando se ejecutan cargas multiprogramadas, las prestaciones de las aplicaciones individuales pueden verse afectadas significativamente en comparación con las que se obtendrían de haberse ejecutado las mismas aplicaciones en solitario. Este comportamiento es el causante de los llamados problemas de equidad en el sistema.

Distintas definiciones de equidad pueden encontrarse en la literatura. En este artículo asumimos que un sistema es equitativo si todas las tareas en

¹Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: viselol@disca.upv.es

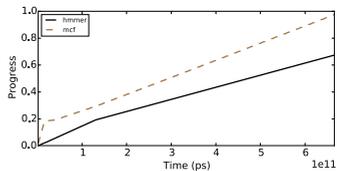


Fig. 1: Progresos de *hmmer* y *mcf*, ejecutándose concurrentemente.

ejecución tienen progresos similares. Es decir, el sistema es equitativo si la degradación de prestaciones que sufren las aplicaciones respecto a su ejecución en solitario es similar. La equidad es un problema de gran importancia en los CMPs actuales, ya que tiene varias consecuencias indeseables: i) hace impredecible el tiempo de ejecución de una aplicación, ya que este pasa a depender parcialmente del resto de aplicaciones en ejecución en el sistema, ii) complica la planificación de tareas basada en prioridades (p.e. puede causar inversión de prioridades) y iii) permite ataques de denegación de servicio.

Para ilustrar el problema de la inequidad, la figura 1 muestra el progreso dinámico de dos aplicaciones *hmmer* y *mcf* ejecutándose concurrentemente en un procesador multinúcleo. Los valores finales de las líneas de la figura, 67% y 97%, indican el porcentaje de instrucciones que *hmmer* y *mcf*, respectivamente, ejecutan en comparación con lo que ejecutarían si se hubieran ejecutado en solitario durante el mismo tiempo. Estos resultados muestran que, mientras las prestaciones de *mcf* apenas se ven afectadas, es decir, progresa a un ritmo similar a cuando se ejecuta en solitario, las de *hmmer* se decrementan muy notablemente, cayendo un 33%.

Debido a que el hardware actual no tiene en cuenta los problemas de equidad, trabajos recientes [1], [2] se han centrado en el planificador de tareas del sistema operativo para intentar que las aplicaciones con una misma prioridad experimenten un progreso similar cuando se ejecutan concurrentemente.

Estos trabajos proponen políticas de planificación orientadas a equilibrar el progreso de las aplicaciones a nivel de *quantum* del sistema operativo. Con este objetivo, estas propuestas dedican varios *quants* cada cierto tiempo a ejecutar en solitario las aplicaciones de las que se quiere estimar el progreso, evitando así las interferencias del resto de aplicaciones. Posteriormente, los datos obtenidos son usados

por el planificador para elegir las aplicaciones a ejecutar, buscando maximizar la equidad y asumiendo que serán válidos para los siguientes quantums. Por lo tanto, hay un cierto sacrificio de prestaciones a cambio de una mejor estimación del progreso, algo que se podría evitar con una estimación de progreso basada en el hardware.

Otros trabajos han propuesto ideas para atacar la inequidad centrándose en las cachés L2 como recurso compartido. Algunos de ellos [3] intentan conseguir una compartición equitativa de la caché, buscando igualar el número de fallos de caché que sufren las aplicaciones en ejecución. No obstante, una política de reparto equitativo de un componente concreto no tiene por que, necesariamente, traducirse en una mayor equidad del sistema.

Este artículo propone FPCP (*Fair Progress Cache Partitioning*), que ataca el problema de la falta de equidad a nivel del hardware, usando estimaciones de progreso que no requieren de un análisis *offline* de la aplicación. Basándose en estas estimaciones, FPCP distribuye las vías de la caché L2 entre las aplicaciones en ejecución para minimizar de esta manera la inequidad.

Este artículo demuestra que la inequidad del sistema debido al uso compartido de la caché puede corregirse en tiempo de ejecución distribuyendo de manera adecuada los recursos de caché. En comparación con *Utility Caché Partitioning* (UCP) [4], otro esquema de particionado de caché, FPCP mantiene la inequidad por debajo del 10% en el 98.5% de las cargas evaluadas, mientras que este porcentaje baja hasta un 67% para UCP.

El resto de este trabajo se organiza como sigue. La sección II resume los trabajos relacionados. La sección III explica cómo se estima el progreso en tiempo de ejecución, mientras la aplicación se ejecuta junto a otras cargas. La sección IV presenta el mecanismo de particionado de la caché. La sección V evalúa la propuesta. Finalmente, la sección VI presenta las conclusiones más importantes.

II. TRABAJOS RELACIONADOS

Diversos trabajos recientes se han centrado en conseguir una utilización equitativa de la cache. En [5] Iyer presenta el *Cache Quality of Service (CQoS) framework*, que otorga distintas prioridades a las peticiones de los hilos que comparten una cache. CQoS requiere soporte por parte del hardware para hacer cumplir las prioridades otorgadas a los hilos. En [6], Chang y Solhi particionan la caché y asignan de manera iterativa estas particiones a los hilos en ejecución durante intervalos de tiempo acotados. De esta manera, al menos durante un tiempo cada aplicación dispone de espacio extra en caché, garantizando un nivel mínimo de calidad de servicio o *QoS*. Además, proponen una métrica para medir la calidad de servicio y regular en base a ella la cantidad de espacio que se le asigna a cada hilo.

Otros trabajos se han centrado en conseguir prestaciones equitativas en las caches compartidas. Fed-

rova et al. [7] proponen una técnica basada en software que regula el tiempo que el sistema operativo otorga a las tareas para evitar que el IPC que obtienen sea más bajo que el estimado. Esta estimación del IPC se realiza comparando el ratio de fallos de caché que tiene la aplicación con el ratio de fallos que tendría si la caché se hubiera distribuido equitativamente entre las tareas en ejecución.

La mayoría de las técnicas que buscan garantizar que el sistema sea equitativo, que es el objetivo de nuestra propuesta, se basan en software, concretamente en modificar las políticas del planificador de tareas del sistema operativo [1], [2]. De hecho, el trabajo de Kim et al. [3] es uno de los pocos que intenta mejorar la equidad del sistema particionando el espacio de la caché compartida sin requerir modificaciones en el sistema operativo. Este trabajo evalúa distintas métricas de prestaciones para encontrar la que proporcione la mejor correlación con el progreso. No obstante, se requiere de un análisis *offline* de las aplicaciones a ejecutar, cosa que es poco práctica.

Una aproximación interesante es la realizada por Qureshi y Patt [4], *Utility Cache Partitioning*, que particiona la caché para minimizar el número de fallos y maximizar la productividad. Otros trabajos orientados a incrementar la productividad de la caché, como [8] y [9] requieren modificaciones en el sistema operativo.

III. ESTIMAR EL PROGRESO

FPCP estima el progreso en tiempo de ejecución para una tarea t de acuerdo con la Ecuación 1, donde $C_{t,multicore}$ es el tiempo de ejecución medido para la tarea t durante su ejecución en paralelo con otras aplicaciones y $C_{t,alone}$ es la estimación del tiempo de ejecución que la aplicación experimentaría si se hubiera ejecutado en solitario. El problema principal es obtener una estimación precisa de las prestaciones en solitario usando únicamente información recopilada durante la ejecución de la aplicación junto con otras aplicaciones.

$$Progress_t = \frac{C_{t,alone}}{C_{t,multicore}} \quad (1)$$

Este trabajo estima $C_{t,alone}$ restándole a los ciclos de ejecución de la aplicación los ciclos en los que la aplicación no ha podido progresar debido a interferencias por parte de otras aplicaciones (ver la Ecuación 2, donde $I_{t,multicore}$ representa los ciclos perdidos por culpa de interferencias).

$$C_{t,alone} = C_{t,multicore} - I_{t,multicore} \quad (2)$$

Nuestra aproximación para estimar las interferencias entre aplicaciones se basa en el modelo presentado por Du Bois et al. [10] para procesadores multinúcleo, que identifica en tiempo de ejecución los fallos de caché que no se habrían producido de haberse ejecutado la aplicación en solitario (i.e. fallos interhilo). Para ello utilizamos en cada núcleo una estructura llamada *Auxiliary Tag Directory* (ATD), que mantiene un registro de cual sería el estado de

la caché si no estuviera siendo accedida por otras aplicaciones (Qureshi et al. [4]). En el sistema estudiado, la interferencia entre aplicaciones afecta a varias métricas de prestaciones: i) el ratio de fallos y el número de conflictos de banco y puerto en la L2 y ii) la latencia de red entre L1 y L2 y entre L2 y memoria principal.

El ratio de fallos de L2 que experimenta un hilo concreto se incrementa porque el resto de hilos pueden reemplazarle bloques de caché que luego intenta acceder. Además, la contención se puede incrementar debido a que al haber más accesos por unidad de tiempo a la caché, el número de conflictos de banco y puerto se incrementa. El modelo de Du Bois et al. identifica los ciclos que se dedican a esperar debido a que el recurso está siendo ocupado por otro hilo así como los que se dedican a traer un bloque reemplazado por otro hilo.

El modelo, no obstante, ha de estimar el impacto real de estos ciclos de espera en las prestaciones, ya que los procesadores modernos disponen de distintos mecanismos para reducir el impacto que los ciclos de espera tienen sobre las prestaciones (e.g. ROB, lanzamiento fuera de orden, paralelismo a nivel de memoria, etc.). Debido a esto, los ciclos extra debido a fallos provocados por otros hilos únicamente se cuentan como ciclos de interferencia cuando el acceso a memoria que los sufre termina bloqueando el ROB.

Téngase también en cuenta que aunque la inequidad puede originarse en todos los recursos compartidos del sistema, este trabajo se centra solamente en la caché L2, ya que es uno de los mayores contribuyentes a la inequidad total. Por lo tanto, para simplificar el modelo, las interconexiones entre L1 y L2 y entre L2 y memoria principal se han modelado como *crossbars*. No obstante, como FPCC mide el tiempo que el ROB pasa detenido debido a instrucciones de memoria con una alta latencia, puede detectar y corregir hasta cierto punto parte de la inequidad originada en otras partes del sistema.

IV. MECANISMO DE PARTICIONADO

Como ya hemos mencionado, un sistema es equitativo si todas las tareas en ejecución progresan al mismo ritmo con respecto a su ejecución en solitario. El mecanismo de particionado propuesto intenta minimizar la inequidad del sistema reduciendo las diferencias entre los progresos de las distintas aplicaciones. FPCC recopila datos sobre las interferencias entre hilos a intervalos regulares; estima el progreso y la inequidad al final de cada intervalo de acuerdo con las ecuaciones 3 y 1, respectivamente, y distribuye las vías de la caché.

Los intervalos son de 150000 reemplazos en L2, pero las simulaciones muestran que los resultados no son demasiado sensibles a este parámetro. Nótese que dependiendo de los patrones de acceso a la caché de las aplicaciones en ejecución, nuestra propuesta puede actuar cientos o incluso miles de veces entre dos llamadas al planificador del sistema operativo.

$$Unfairness = \frac{\max_{i,j \in Tasks} Progress_i}{\min_{i,j \in Tasks} Progress_j} \quad (3)$$

Nuestra propuesta consiste en tres pasos: i) estimación del progreso de las tareas en ejecución, ii) estimación de la inequidad del sistema y iii) redistribución de vías. A continuación se discuten las acciones asociadas a cada uno de los pasos para un procesador con dos núcleos.

Al principio de la ejecución nuestro mecanismo está desactivado y las vías de la caché se distribuyen siguiendo la política LRU. Después de un periodo de calentamiento, el mecanismo se activa y se particiona la caché en partes iguales, dependiendo del número de hilos que accedan a la caché¹. Primero, el mecanismo estima el progreso de cada una de las tareas en ejecución, tal y como se ha explicado en la sección III. A continuación, se calcula la inequidad en base a las estimaciones sobre el progreso individual. Utilizando estas estimaciones se modifica la distribución de las vías de la caché para el siguiente intervalo. En el último paso, para intentar igualar gradualmente los progresos, la tarea con el progreso más alto cede una vía a la tarea con un menor progreso, garantizando un mínimo de una vía por tarea. Este mecanismo es similar a un sistema de control de lazo cerrado, centrado en minimizar las diferencias en los progresos y por lo tanto maximizar la equidad.

Una versión preliminar del algoritmo usaba un umbral mínimo de inequidad para activar el particionado de vías. Se exploraron distintos umbrales de inequidad, pero los resultados experimentales mostraron que los mejores resultados se obtenían sin usar ningún tipo de umbral. Por otra parte, aunque el particionado presentado en este artículo se lleva a cabo modificando la política de reemplazo (que se asume LRU) otras políticas de caché como las presentadas en [11] y [12] también pueden usarse.

FPCC corrige la inequidad en gran manera en comparación con aplicar una política LRU sin particionado de cache. No obstante, bajo ciertas circunstancias, cómo cuando las estimaciones de progreso no son suficientemente precisas, la distribución de vías puede no ser óptima y perjudicar la equidad. FPCC implementa un mecanismo de *feedback* para paliar este problema. Si la inequidad del sistema empeora durante n intervalos consecutivos, el mecanismo de particionado se detiene durante m intervalos (los resultados experimentales presentados en este artículo se han obtenido con $n = 2$ y $m = 5$), durante los cuales se emplea la política de reemplazo LRU sin modificar. Esto permite mantener los resultados de inequidad siempre mejores o iguales a los que se obtienen con la política LRU. Nuestros resultados muestran, no obstante, que este mecanismo no se dispara frecuentemente, ocurriendo únicamente en una de las 406 cargas testadas y deteniendo el mecanismo de particionado durante un 6.6% del tiempo de

¹Aunque el particionado inicial puede ayudar a reducir la inequidad inicial más rápido, este no es determinante, por lo que podrán usarse otros sin afectar a los resultados a largo plazo.

TABLA I: Configuración del sistema.

Núcleos	2 núcleos a 3GHz
Política de issue	Fuera de orden
Ancho de issue/commit	4 instrucciones/ciclo
Tamaño del ROB	128 entradas
Cola de loads/stores	64/48 entradas
IL1 caché (privada)	32KB, 8vías, 64B por bloque, 2cc
DL1 caché (privada)	32KB, 8vías, 64B por bloque, 2cc
L2 (compartida)	2MB, 16vías, 64B por bloque, 11cc, 16 MSHR
Latencia de memoria principal	200 ciclos sin carga

ejecución.

Aunque este ejemplo explica la propuesta para un procesador con dos núcleos, la propuesta puede extenderse fácilmente a un mayor número de núcleos, mediante la técnica de *divide y vencerás*. Así, las distintas tareas pueden agruparse en pares y cuartetos formando un árbol binario e intercambiar vías entre nodos del árbol.

Respecto a la complejidad hardware de la propuesta, nótese que, a excepción de la ATD, los contadores de prestaciones y los operadores necesarios para implementar la propuesta se encuentran disponibles en los procesadores actuales. Por otra parte, el coste computacional del algoritmo de particionado es de orden $O(n)$, donde n es el número de núcleos/hilos/tareas. Este tiempo se invierte en calcular el progreso de las tareas, encontrar los valores mínimos y máximos y realizar los ajustes en el número de vías. Dado que estas operaciones no se encuentran en el camino crítico del procesador, pueden ser retrasadas hasta que, por ejemplo, el procesador esté detenido debido a un evento de memoria de alta latencia, evitando así pérdidas de prestaciones.

V. EVALUACIÓN EXPERIMENTAL

La evaluación se ha realizado mediante simulación, utilizando una versión extendida del simulador ciclo a ciclo Multi2Sim [13], que modela todos los detalles de un procesador multinúcleo y multihilo moderno.

El sistema modelado dispone de 2 núcleos con cachés L1 privadas y una caché L2 de 2MB compartida, con tantos bancos como puertos y 16 vías. Es en este nivel de caché donde se implementa el mecanismo de particionado. Tanto la contención a nivel de banco como a nivel de puerto han sido modeladas. La tabla I resume los parámetros del sistema más relevantes.

Hemos comparado FPCP tanto contra un sistema sin particionado de caché usando la política LRU como contra *Utility-Based Cache Partitioning* (UCP). Con UCP, cada núcleo tiene hardware dedicado a estimar la *utilidad* de cada una de las vías que se le asignan al núcleo, basado en *Dynamic Set Sampling* (UMON-DSS). Este hardware monitoriza 64 sets de la caché y estima la utilidad que cada núcleo da a las vías asignadas. En base a los datos que obtiene UCP hace una redistribución de las vías de la caché, buscando minimizar el número de fallos. Los tamaños

de las particiones se obtienen usando un algoritmo voraz descrito en [4] y se ajustan cada 5 millones de ciclos.

Para evaluar nuestra propuesta usamos cargas multiprogramadas de dos aplicaciones, tomando aplicaciones del conjunto de la colección de aplicaciones SPEC2006 CPU con los valores de entrada de referencia. En total se han estudiado 406 combinaciones de aplicaciones, para considerar un número de escenarios lo más amplio posible. Los resultados se obtuvieron ejecutando las cargas durante 2000 millones de ciclos después de saltar los primeros 500 millones de instrucciones de cada aplicación.

A. Resultados experimentales

El objetivo de esta sección es doble. Por una parte ilustrar como FPCP reduce dinámicamente la inequidad. Por otra parte, comparar la propuesta contra otra propuesta de particionado de caché reciente.

Para estudiar el funcionamiento de FPCP, hemos analizado la evolución del progreso, la inequidad, el MPKI y las vías de caché usadas por cada una de las aplicaciones de las cargas mientras FPCP está activo. La figura 2 muestra los resultados para una carga formada por las aplicaciones *hmmcr* y *mcf*.

Como se puede ver en la figura 1, que muestra el progreso de estas mismas aplicaciones sin ningún tipo de particionado en la caché, esta carga tiene un problema de inequidad importante. Esto puede apreciarse también en la figura 2a, la cual muestra la evolución de la inequidad para un sistema con nuestra propuesta y para otro sin ningún tipo de particionado. En este caso, la inequidad viene principalmente de las notable diferencias entre el MPKI de las dos aplicaciones: durante la mayor parte de la ejecución, el MPKI de *mcf* está por encima de 20, mientras que el de *hmmcr* se mantiene alrededor de 2. La razón de estas diferencias es que esta parte de la ejecución de *hmmcr* es computacionalmente muy intensiva, y por lo tanto muy sensible a los fallos de caché. En cambio, durante esta fase de *mcf*, el cuello de botella está en el acceso a memoria, experimentando un gran número de fallos de caché. Por ello, *mcf* es poco sensible a variaciones pequeñas en el ratio de fallos de caché y además tiende a acaparar una gran parte del espacio de caché. Si no se actúa para evitarlo, esto se traduce en progresos muy desiguales para las dos aplicaciones, como ya se ha visto. FPCP detec-

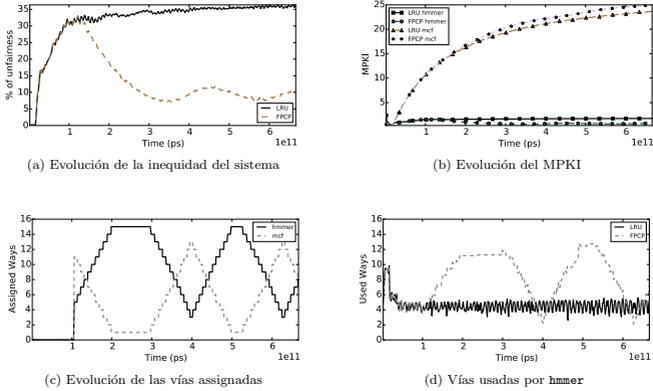


Fig. 2: Resultados de la carga formada por *hmer* y *mcf*

ta estas diferencias de progreso y asigna más vías a *hmer* para reducir la inequidad. Como puede verse en la figura 2, nuestra propuesta incrementa el MPKI de *mcf* y decreta el de *hmer*, reduciendo significativamente la inequidad del sistema, que pasa de un 36% a un 10%.

Hemos caracterizado la relación entre interferencia (cuantificada como la suma de fallos interhilo) entre hilos e inequidad cuando no se usa ningún particionado. Se pueden distinguir tres categorías de aplicaciones: i) pocas interferencias y baja inequidad, ii) interferencia media-alta y baja inequidad y iii) interferencia media-alta y alta inequidad. Nótese que niveles medios o altos de interferencia pueden dar lugar tanto a escenarios con un bajo grado de inequidad como a escenarios con alta inequidad. El primer caso ocurre cuando todas las aplicaciones de la carga sufren un grado similar de interferencia y por lo tanto progresos similares. El segundo caso ocurre cuando algunas aplicaciones sufren más interferencias que otras. Dado que FPCP reduce las diferencias entre los progresos de las aplicaciones en ejecución, la inequidad total del sistema se reduce significativamente. De hecho, la mayoría de las cargas que se sitúan en la tercera categoría cuando no se realiza ningún particionado de la cache pasan a situarse en la primera o la segunda categoría cuando se usa FPCP.

FPCP se ha comparado contra LRU (i.e. un sistema sin particionado) y contra UCP, tanto en términos de prestaciones como de inequidad. La figura 3 presenta la inequidad para las 406 cargas analizadas ordenadas ascendentemente por su grado de inequidad (cuanta más inequidad, peor). Como puede observarse, un 14.8% y un 33.0% de las cargas superan un 10% de inequidad cuando se usa una caché convencional con política LRU y cuando se usa UCP, respectivamente. No obstante, cuando se usa FPCP únicamente un 1.5% de las cargas superan este um-

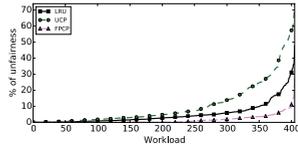


Fig. 3: Inequidad del sistema en las 406 cargas

bral. La causa principal del alto grado de inequidad de UCP es que se centra en minimizar el número total de fallos de caché sin tener en cuenta su impacto *real* en las prestaciones del sistema. Por ejemplo, evitar un fallo de caché que se solaparía con otro debido al paralelismo a nivel de memoria (MLP) puede no tener ningún impacto en las prestaciones finales. No obstante UCP funciona bien si la penalización por fallar en la L2 es muy alta.

Respecto a las prestaciones, el IPC acumulado para las 406 cargas obtenido por FPCP mejora, de media, alrededor de un 1.5% y 3% en comparación con LRU y UCP, respectivamente.

VI. CONCLUSIONES

Proporcionar equidad a nivel de hardware libera al planificador de tareas del sistema operativo de tener que reservar quantums para estimar prestaciones en solitario y le permite centrarse en maximizar las prestaciones. Este artículo ha presentado FPCP, que busca reducir la inequidad del sistema actuando únicamente a nivel hardware. Nuestra propuesta actúa en los niveles compartidos de caché y distribuye las vías disponibles entre las aplicaciones en ejecución.

Como se ha podido ver, la inequidad del sistema puede reducirse en gran manera estimando los pro-

gresos de las aplicaciones en ejecución y usando estos valores para guiar un mecanismo de particionado de la caché. A diferencia de propuestas previas, FPCP tiene la ventaja de no necesitar un análisis previo *offline* de las aplicaciones a ejecutar.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad (MINECO) y por los fondos del Plan E mediante subvenciones a los proyectos TIN2015-66972-C5-1-R y TIN2014-62246-EXP, así como por la Generalitat Valenciana mediante subvención del proyecto AICO/2016/059.

REFERENCIAS

- [1] Josué Feliu, Julio Sahuquillo, Salvador Petit, and José Duato, "Addressing fairness in *smt* multicores with a progress-aware scheduler," in *IPDPS*, 2015, pp. 187–196.
- [2] Chenggang Wu, Jin Li, Di Xu, Pen-Chung Yew, Jianjun Li, and Zhenjiang Wang, "Fps: A fair-progress process scheduling policy on shared-memory multiprocessors," *TPDS*, vol. 26, no. 2, pp. 444–454, Feb 2015.
- [3] Seongbeom Kim, Dhiruba Chandra, and D. Sathin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," in *PACT*, Sept 2004, pp. 111–122.
- [4] M.K. Qureshi and Y.N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *MICRO*, Dec 2006, pp. 423–432.
- [5] Ravi Iyer, "CQoS: a framework for enabling QoS in shared caches of CMP platforms," in *ICS*, 2004, pp. 257–266.
- [6] Jichuan Chang and Gurindar S Sohi, "Cooperative cache partitioning for chip multiprocessors," in *ICS*, 2007, pp. 242–252.
- [7] Alexandra Fedorova, Margo Seltzer, and Michael D Smith, "Improving performance isolation on chip multiprocessors via an operating system scheduler," in *PACT*. IEEE Computer Society, 2007, pp. 25–38.
- [8] M. Awasthi, K. Sudan, R. Balasubramonian, and J. Carter, "Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches," in *HPCA*, Feb 2009, pp. 250–261.
- [9] Sangyeun Cho and Lei Jin, "Managing distributed, shared L2 caches through os-level page allocation," in *MICRO*, Washington, DC, USA, 2006, pp. 455–468, IEEE Computer Society.
- [10] Kristof Du Bois, Stijn Eyerma, and Lieven Eeckhout, "Per-thread cycle accounting in multicore processors," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 29:1–29:22, Jan. 2013.
- [11] Daniel Sanchez and Christos Kozyrakis, "Vantage: Scalable and efficient finegrain cache partitioning," in *ISCA*, 2011.
- [12] Yuejian Xie and Gabriel H. Loh, "Pipp: Promotion/insertion pseudo-partitioning of multi-core shared caches," in *ISCA*, New York, NY, USA, 2009, pp. 174–183, ACM.
- [13] Rafael Ubal et al., "Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors," in *SBAC-PAD*, 2007, pp. 62–68.
- [14] *Standard Performance Evaluation Corporation, available online at <http://www.spec.org/>*.
- [15] Kristof Du Bois, Stijn Eyerma, and Lieven Eeckhout, "Per-thread cycle accounting in multicore processors," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 29:1–29:22, Jan. 2013.
- [16] N. Miralaei, J. Soman, and T. Jones, "PAM: A Processor Ageing Model Based on Critical Path Delays," in *Proceedings of the Designing with Uncertainty - Opportunities & Challenges Workshop*, 2014.
- [17] *Standard Performance Evaluation Corporation, available online at <http://spec.org/cpu2006>*.
- [18] Z. Zhu and X. Zhang, "Access-Mode Predictions for Low-Power Cache Design," *IEEE Micro*, vol. 22, no. 2, pp. 58–71, 2002.

Leveraging OSD+ Devices to Efficiently Implement Data Objects in FPFs

Juan Piernas¹ and Pilar González-Férez²

Resumen—

OSD+ devices are enhanced object-based storage devices (OSDs) able to deal with both data and metadata operations via data and directory objects, respectively. So far, we have focused our efforts on designing and implementing efficient directory objects in OSD+s. This paper, however, presents our recent work on also supporting data objects in these devices. It also describes how the coexistence of data and directory objects in OSD+s can be profited to efficiently implement data objects and to speed up some common file operations, such as create, delete, stat, etc. We have compared our OSD+-based Fusion Parallel File System (FPFS) with OrangeFS through different microbenchmarks and HPCS-IO scenarios. The results show that FPFS can provide a performance up to 75× better than OrangeFS for metadata, and up to 34% for data.

Palabras clave— Data objects, OSD+ devices, exascale, FPFs.

I. INTRODUCTION

FILE SYSTEMS for HPC environment have traditionally used a cluster of data servers for achieving high rates in read and write operations, for providing fault tolerance, and scalability, etc. However, due to a growing number of files, and an increasing use of huge directories with millions or billions of entries accessed by thousands of clients at the same time [1], [2], [3], [4], [5], some of these file systems also utilize a cluster of specialized metadata servers [6], [7], [8], [9] and have recently added support for distributed directories [8], [10].

Unlike those file systems, that have separate data and metadata clusters, in-house Fusion Parallel File System (FPFS) uses a single cluster of *object-based storage device+* (OSD+) [11] to implement those clusters. OSD+s are improved object-based storage devices (OSDs) that, in addition to handle data objects as traditional OSDs do, can also manage directory objects. Directory objects are a new type of object able to store file names and attributes, and support metadata-related operations, like the creation and deletion of regular files and directories. By using these OSD+ devices, an FPFs metadata cluster is as large as its corresponding data cluster, and metadata is effectively distributed among as many nodes as OSD+s comprising the system. OSD+s are implemented through a thin software layer on top of existing mainstream computers, leveraging many features of the underlying file system. Thanks to this approach, OSD+s have a small overhead, and provide a high performance. Indeed, FPFs is able to

create, stat and delete hundreds of thousands of files per second with a few OSD+ devices [11], [12]. FPFs also supports huge directories by dynamically distributing them among several OSD+s [13]. The OSD+s storing a distributed huge directory work independently of each other, thereby improving the performance and scalability of the file system. Throughput in FPFs can be increased even further by means of batch operations, which allow us to send several metadata operations to a server in a single request [14].

So far, we have focused on the development of the metadata part of FPFs. In this paper, however, we describe how we have implemented the support for data objects. We will show that the utilization of a unified data and metadata server (i.e., an OSD+ device) provides FPFs with a *competitive advantage* with respect to other file systems that allows it to speed up some file operations, such as creating and deleting files, getting the status of files, etc.

We have evaluated the performance and scalability of FPFs with data-object support through different microbenchmarks and HPCS-IO scenarios [15], and compared the results with that obtained by OrangeFS [8]. Results show that, for metadata-intensive workloads, FPFs provides a throughput that is, at least, one order of magnitude better than that of OrangeFS, and almost two orders of magnitude in some cases. For workloads with large files and large data transfers, FPFs can obtain a bandwidth up to 34% better than the bandwidth achieved by OrangeFS thanks to the smaller overhead introduced by our file system.

The rest of the paper is organized as follows. Section II describes the FPFs architecture. Section III details how data objects are designed and implemented. Results are provided in Sections IV and V. Related work is described in Section VI. Finally, Section VII concludes the paper.

II. OVERVIEW OF FPFs

Generally, parallel file systems have three main components: clients, data servers and metadata servers. Data servers are usually OSD [16] or OSD-alike devices that export an object interface. Metadata servers, however, frequently implement customized interfaces, and permanently store metadata in private storage devices [10] or in objects allocated in the data servers [9].

Unlike these file systems, FPFs [11] uses a single kind of servers that act as both data and metadata servers (see Fig. 1). This approach consequently enlarges the metadata cluster's capacity that becomes as large as the data cluster's. To merge data and me-

¹Dpto. de Ingeniería y Tecnología de Computadores, Univ. de Murcia, e-mail: piernas@dittec.um.es.

²Dpto. de Ingeniería y Tecnología de Computadores, Univ. de Murcia, e-mail: pilar@dittec.um.es.

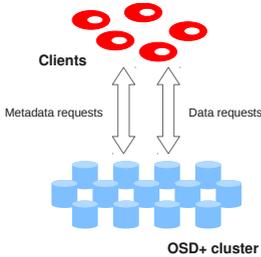


Fig. 1. FPFS's overview. Each OSD+ supports both data and metadata operations.

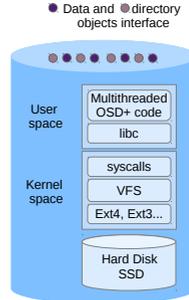


Fig. 2. OSD+ layers.

tadata servers into a single one, FPFS uses OSD+ devices, which are enhanced OSD devices. OSD+s are capable of managing not only data (as common OSDs do), but also metadata. OSD+ devices simplify the complexity of the storage system as well, since no difference between two types of servers is made. Moreover, having a single cluster increases system's performance and scalability, since there will not be underutilized data or metadata servers.

A. OSD+

OSDs implement not only low-level block allocation functions, but also more complex tasks by taking advantage of their intelligence [9]. OSD+s leverage this intelligence too, taking it a step further by delegating metadata management to storage devices.

Traditional OSDs deal with *data objects* that support operations like creating and removing objects, and reading from and writing to a specific position in an object. Our design extends this interface to define *directory objects*, capable of managing directories. As a consequence, OSD+ devices support metadata-related operations like creating and removing directories and files, getting directory entries, etc. In addition to the usual operations on directories, OSD+s also provide functions to internally deal with metadata operations that may involve the collaboration of several OSD+s (e.g., renames, directory permission changes, and links).

Currently, there exist no commodity OSD-based disks, so mainstream computers exporting an OSD-based interface are usually used [17]¹. Internally, a local file system stores the objects; we take advantage of this by *directly mapping* operations in FPFS to operations in the local file system.

Each OSD+ is composed of a user-space multithreaded process and a conventional file system. The process uses the file system as storage backend, and

¹Seagate's Kinetic drives are key/value servers with Ethernet connectivity. They have a limited object-oriented interface that supports a few operations on objects identified by keys. Kinetic drives could be seen as an early implementation of something similar to Gibson's proposal [18], but, due to their limited design, they still need a higher level layer like Swift [19] to carry out basic operations, such as mapping large objects, coordinating race conditions on write operations, etc.

issues Linux syscalls to perform operations on that file system. The local file system must be POSIX-compliant and support extended attributes, since they are used by our implementation. Fig. 2 shows the layers that compose an OSD+.

B. Clients

Clients are the processes accessing FPFS. For fast prototyping and evaluation, the current implementation entirely runs clients in user-space. There exists an FPFS library (`libfpfs`) that clients use to issue requests. This approach is similar to that used by PVFS2/OrangeFS [8] and other file systems [20].

FPFS establishes communications between clients and OSD+s via TCP/IP connections, and request/reply messages. Each OSD+ launches one thread to attend the requests of a client, and to perform operations on the local disk on behalf of the client. When an operation involves several OSD+s, the OSD+ contacted by a client cooperates with other OSD+s to carry out the operation in a transparent way to the client (see Section II-D).

C. Namespace Distribution

FPFS distributes directory objects (and so the file-system namespace) across the metadata cluster to make metadata operations scalable with the number of OSD+s, and to provide a high performance metadata service. For the distribution, FPFS uses the deterministic pseudo-random function CRUSH [9]:

$$oid = CRUSH(hash(dir\ fullpath)). \quad (1)$$

CRUSH receives the hash of a directory's full path-name as input, and returns the ID of the OSD+ containing the corresponding directory object as output. This allows clients to directly access any directory *without performing a path resolution*.

We choose CRUSH because it guarantees a probabilistically balanced distribution of objects through the system. However, FPFS does not depend on a particular distribution function, so other functions are also possible.

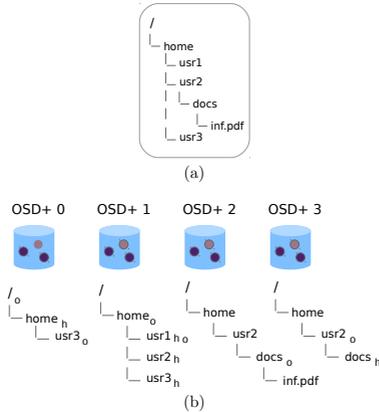


Fig. 3. (a) An FPFS namespace. (b) A possible mapping of the FPFS namespace to a 4-OSD+ cluster.

Hash partition strategies present different scalability problems on cluster resizings, permission changes, and renames. FPFS addresses the first problem through CRUSH, which minimizes migrations and imbalances when adding and removing devices. FPFS manages renames and permission changes via lazy techniques [21]; fortunately, these operations are infrequent for directories [11], [21], so they will not impact the overall performance.

D. Directory Objects

A directory object is implemented as a regular directory in the local file system of its OSD+. In this way, any directory-object operation is directly translated to a regular directory operation. The full pathname of the directory supporting a directory object is the same as that of its corresponding directory in FPFS. Therefore, the directory hierarchy of FPFS is imported within the OSD+s by partially replicating its global namespace.

Internally, an OSD+ uses three types of directories, differentiated through extended attributes. These directory types can be seen in Fig. 3, which shows how an FPFS’s directory hierarchy is mapped to a 4-OSD+ cluster. The first type (attribute *o*) is assigned to directory objects stored in the OSD+, i.e., objects that CRUSH and their full pathnames have assigned to the OSD+. The second type (attribute *h*) refers to empty directories created in a directory object; they represent subdirectories and allow FPFS to preserve the complete filesystem hierarchy to provide standard directory semantics (e.g., scan). The third one (no attribute) is for directories used for supporting the paths of the directories implementing objects.

For each regular file that a directory has, the directory object conceptually stores its attributes, and

the number and location of the data objects that store the content of the file. There are two exceptions: size and modification time attributes of the file, which are stored at its data object(s). In our current implementation, these “embedded i-nodes” [22] are i-nodes of empty files in the local Linux file system; the number and location of the data objects are also stored in those empty files as extended attributes.

Implementing directory objects by means of regular directories in a local file system has, at least, two important advantages. The first one is that the implementation is simpler and its overhead smaller since most part of the functionality is provided by the underlying file system. The second one is that, when a metadata operation is carried out by a single OSD+ (create, unlink, etc.), the backend file system itself ensures its atomicity and POSIX semantics. Only for operations like `rename` or `rmdir`, that usually involve two OSD+s, the participating OSD+s need to deal with concurrency and atomicity by themselves through a *three-phase commit protocol* (3PC) [23], without client involvement.

E. Huge directories

FPFS also implements management for huge directories, or *hugedirs* for short. These are directories with millions or billions of entries accessed by thousands of clients at the same time. Hugedirs are common for some HPC applications, such as those that use a directory as a light-weight database [24].

To manage hugedirs, FPFS proposes a dynamic distribution of their entries among multiple OSD+s [13]. FPFS considers a directory is huge when it stores more than a given number of files. Once this threshold is exceeded, the directory is shared out among several nodes. The threshold can be 0, thereby distributing a directory from the start.

The subset of OSD+s supporting a hugedir is composed of a *routing OSD+* and a group of *storing OSD+s*. The former contains the *routing directory object* and is in charge of providing clients with the hugedir’s distribution information. The latter have the *storing directory objects* that store the directory’s content. The *routing OSD+* can also be part of the *storing* group in case it keeps any directory’s content. Indeed, for small directories, the routing and storing objects are the same.

A client unaware of the distribution of a directory contacts its routing object by using Equation 1, as it does with any other regular directory object. As reply, the client receives the *distribution list* with the *IDs* of the routing and storing OSD+s. Then, the client retries the operation, but changes the previous directory-level function for a file-level counterpart:

$$oid = osd_set[(hash(filename) \% osd_count)], \quad (2)$$

where *osd_set* is the list of storing OSD+s, and *osd_count* is that list’s size. The result is the storing OSD+ having the entry of the given file name.

Clients accessing a hugedir cache its distribution list. FPFS uses timestamps to detect when this cac-

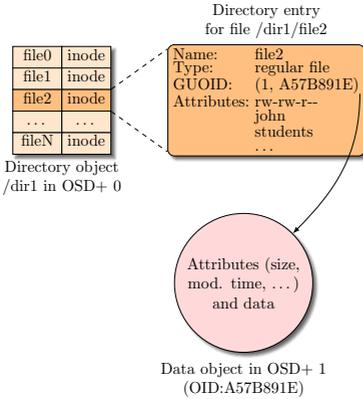


Fig. 4. A regular file in FPFS. The directory entry contains the i-node and also a reference to the data object.

hed information gets out of date due to the rename or deletion of a huredir. In that case, clients clean up the distribution information for the directory and retry the operation following Equation 1. If the directory gets huge again, clients will change to Equation 2 with the new distribution list.

III. DATA OBJECTS

Data objects are storage elements able to store information of any kind. They can also have associated attributes that users can set and get. Data objects support different operations, being the reading and writing of data the most important. Data objects receive an ID, which we call *object ID* (OID), when they are created. These OIDs allow us to unequivocally identify an object inside a given device, although there can be duplicated OIDs among different devices. Therefore, a data object is globally identifiable by means of its OID and the ID of the device holding it. We call this pair (device ID, object ID) a *globally unique object ID* (GUOID).

In this section, we first explain the relationship between regular files and data objects. Then, we describe how data objects are implemented in FPFS. Finally, we show how the implementation can be optimized thanks to the use of OSD+ devices.

A. Regular files and data objects

As we have seen in Section II-D, when a regular file is created in FPFS, three related elements are also created: a directory entry, an i-node and a data object. From a conceptual perspective, the i-node is embedded into the directory entry, so these two elements are stored together in the corresponding directory object, while the data object is stored separately. Fig. 4 depicts this situation.

FPFS can use two different policies for selecting

the OSD+ where the data object of a file is created: *same OSD+* and *random OSD+*. The former, used by default, stores a data object in the OSD+ of the directory object storing its file's entry. This approach reduces the network traffic during file creations because no other OSD+s participate in the operation. The latter chooses a random OSD+ instead. This second approach can potentially achieve a better use of resources in some cases by keeping a more balanced workload, although it increases the network traffic during creations.

Regardless the allocation policy for data objects, the i-node of a regular file stores a reference to its data object by means of its GUOID, that is, the ID of the OSD+ storing the data object, and the OID of the data object in that server.

B. Implementation of data objects

FPFS internally implements data objects as regular files in the underlying local file system used by an OSD+. There exists a data directory where those regular files are stored. This directory is different from that holding directory objects, although they can share the same local file system, and, thus, the same storage device(s).

When a data object is created, its OID is generated as a random integer number. Although collisions are highly unlikely, if another object exists with that number, a different random number is generated. A string with the hexadecimal representation of the random number is used as name of the local regular file supporting the object. All the names have the same length (8 characters), padding with character 0 when necessary. To avoid too large directories, which usually downgrade performance, files are distributed into 256 subdirectories. A file is located in a directory whose name is made up of the first two hexadecimal characters of the file's name.

An `open()` call on an FPFS file always returns a file descriptor in the OSD+ storing its data object to directly operate on the object. Current implementation supports `read()`, `write()`, `fstat()`, `lseek64()`, and `fsync()` operations. All of them operate on the data object whose descriptor is passed as argument.

The `open()` call also returns a key, which we call *secret*, for the data object, so only those clients that have been granted access to the file can use the returned descriptor to operate on the data object. These clients send the secret along with each request, and the target OSD+ verifies the secret before allowing the operation on the data object.

Note that, while a `stat()` call affects a directory and its embedded i-node, a `fstat()` call affects a data object. The former will return all the attributes a traditional `stat()` call returns, except the size and modification time. These attributes are returned by the latter. Therefore, if a process wants to obtain a "traditional" behavior, it should combine both operations. Obviously, a library can hide these details, and exports these differences through appro-

priate functions or flags.

Also note that a regular file always needs one data object at least. However, data objects can exist without associated files; clients can directly issue data-object operations to the OSD+ devices, so they can create, delete, read, write, etc. data objects without using a file. This makes FPFS flexible and adaptable to different environments. This functionality, however, has not been implemented yet.

C. Optimizing the implementation

If the default allocation policy for data objects is active (i.e. a directory entry for a regular file and its corresponding data object are stored in the same OSD+), we can profit the fact that an OSD+ device manages both data and metadata to speed up the creation of files and other operations. When a file is created, the target OSD+ internally creates an empty file in the directory of the local file system supporting the directory object. This empty file usually acts as dentry and embedded i-node, as we have seen in Section II-D. But because, at the end, data objects are also implemented as files internally, it can also act as data object. Consequently, creation is quite fast. The operation is atomic too: the three elements will either exist or not after the operation. File systems with separate data and metadata servers (at least, from a conceptual point of view) incur in a noticeable overhead due to independent operations in different servers, and the network traffic generated to perform those operations and guarantee their atomicity.

The default policy to allocate data objects also has two other effects: (1) distribution of data objects is carried out on a per-directory basis; this increases spatial locality, which can improve the performance in some cases; and (2) if a directory is shared out, their files and data object will be distributed too.

The overlap between a dentry-inode and its data object disappears, however, in a few cases:

- When a directory object is moved.
- When a file has several data objects.
- For hard links.

The first of those cases occurs when a directory object is migrated from an OSD+ to another due to a rename, or when a directory becomes huge and it is distributed (only dentries are moved, data objects remain in their original servers). At those moments, a data object appears with its own identity.

The second case appears when a file has several data objects, each on a different OSD+ device. In this case, those objects will exist by themselves right from the start. Usually, a file uses more than one data object for improving its read and write transfer rates, since those operations can be issued to its data objects in parallel. Note, however, that we have not implemented this kind of files yet.

Finally, the third case happens when there exist hard links. For every file having more than one link, FPFS creates an *i-node object*. Internally, this

TABLE I
CLUSTER NODES' TECHNICAL SPECIFICATIONS.

Platform	Supermicro X7DWT-INF
CPU	Two Intel Xeon E5420 quad-core at 2.50 GHz
RAM	4 GB
System disk	HDD Seagate ST3250310NS (250 GB)
Test disk	SSD Intel 520 Series (240GB)
OS	64-bit CentOS 7.2
Interconnect	Gigabit network
Switch	D-Link DGS-1248T

i-node object is the i-node of a file supporting an empty data object. As data object, it has a GUUID, which we can see as a sort of system-wide unique i-node number. This makes the creation of hard links straightforward: the directory entry for a new hard link simply stores the new file name and the GUUID of the i-node object of the source file. Note that, with this approach, we leverage objects already present in FPFS, and we do not need to introduce additional mechanisms. This way of tackling with hard links is similar to that of Ganger *et al.* [22], although we do not need a file for storing externalized i-nodes.

The i-node object for a hard link stores all the file attributes (except size and modification time, as we have already explained), references to its data objects, and a counter to track of the number of links.

Hard links make opening a file slower because they add an additional step: fetching an i-node object. Fortunately, these links are rare [25], so they will have a very small effect on the overall performance. Nevertheless, once a file is open, `read()` and `write()` operations will proceed directly to its data object(s).

Note that, whether a dentry-inode and a data object are internally supported by the same file or not, an `open()` call always returns a file descriptor to directly operate on the data object.

IV. EXPERIMENTS AND METHODOLOGY

Performance of FPFS has been analyzed by running different benchmarks. It has also been compared with performance of version 2.9.3 of OrangeFS. This section describes the system under test, and the benchmarks run to carry out the analysis.

A. System under test

The testbed system is a cluster made up of 12 compute and 1 frontend nodes. Technical specifications of each compute node are summarized in Table I. The test disk supports the OSD+ device for FPFS, and the storage device for OrangeFS.

`noop` is set for SSDs as I/O scheduler, since it usually achieves the best performance for these devices compared to other available Linux schedulers [26].

We use Ext4 as backend file system for both FPFS and OrangeFS. Formatting and mounting options are important for performance [11]. For FPFS, we format Ext4 file systems with options

```
-J size=400 -i 4096 -I 512
```

`-O dir_index, extents, uninit_groups`

that set the journal size, bytes-per-inode ratio, inode size, and use of hashing in directories, extents and some uninitiated structures, respectively. They follow options used by Lustre when formatting its metadata server [27]. For OrangeFS, we use the same format options, although we set a smaller inode size (`-I 256`) and a larger bytes-per-inode ratio (`-i 16384`). OrangeFS utilizes a Berkeley database for metadata, so we try to benefit large files this way.

Mount options also try to increase the metadata performance. We use `noatime`, `nodiratime` and `data=writeback` for the Ext4 file systems used by both FPFs and OrangeFS.

Finally, we have configured FPFs and OrangeFS to shared out directories among all the available servers right from the start, when applicable. This is because version 2.9.3 of OrangeFS crashes for relatively small values (< 1000) of the configuration parameter `DistrDirSplitSize`, which specifies the default for number of directory entries on a server before splitting. Hence, instead of using a small value that would cause an almost immediate split, we opt for distributing directories when they are created.

Note that we have tried to compare FPFs with version 2.8 of Lustre and 9.2 of Ceph (latest available versions at the time of this writing), but both fail when more than one metadata server is used.

B. Benchmarks

To evaluate the performance, we have used the following scenarios of the HPCS-IO suite [15]:

- Scenario 4: there are 10 directories per process, where files with sizes ranging between 1 kB and 64 kB are created. The test creates as many files as possible in 50 seconds. We use 64 processes for this scenario.
- Scenario 8: in this case, the test uses 128 processes for creating 128 files of 32 MB each.
- Scenario 9: a single process issues a `stat()` operation on empty files in a sequential order.
- Scenario 10: like scenario 9, but `stat()` operations are issued by 10 processes (this small number of processes is imposed by the scenario).
- Scenario 11: like scenario 9, but the process issues `stat()` operations in a random order.
- Scenario 12: like scenario 11, but `stat()` operations are issued by 128 processes.

The last four scenarios operate on 256 directories created with 10000 empty files in each. Hence, they use 2560000 files altogether.

We have discarded scenarios that involve large files or shared files, since we do not support files with multiple data object yet. Version 1.2.0-rc1 of the suite has been used.

Processes in the different tests are shared out among four compute nodes. We have tried to use more than 64/128 processes in some scenarios, but processes start crashing if the file system is OrangeFS. FPFs does not have this problem.

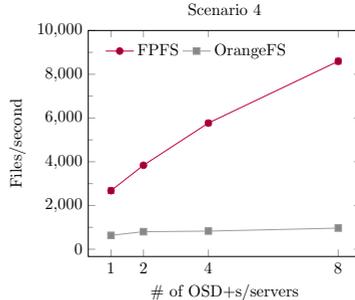


Fig. 5. HPCS-IO scenario 4.

Since some scenarios of HPCS-IO place different synchronization points among the processes, achieved performance is not as high as it could be. Chosen scenarios do not clearly show the benefits of a distributed directory either. For this reason, we have also run the following three benchmarks:

- *Create*: each process creates a subset of empty files in a shared directory. This benchmark basically generates a write-only metadata workload.
- *Stat*: each process gets the status of a subset of files in a shared directory. This is a read-only metadata workload (remember that `noatime` and `nodiratime` mount options are used).
- *Unlink*: each process deletes a subset of files in a shared directory. This is a read-write metadata workload.

Each benchmark uses 256 client processes shared out among four compute nodes. There is no synchronization of any type among the processes. A benchmark finishes when the last process completes.

V. EXPERIMENTAL RESULTS

Here we show different results for the benchmarks described in the previous section. Results shown for every system configuration are the average of at least five runs of each benchmark. Confidence intervals are also shown as error bars, for a 95% confidence level.

We format the test disks before every run of the scenarios 4 and 8, and the preprocess for scenarios 9–12 of HPCS-IO. We also format the test disks before every run of the create test. For the rest of the benchmarks, we unmount/remount the disks between tests.

A. HPCS-IO

Fig. 5 depicts the results obtained by FPFs and OrangeFS for scenario 4 of HPCS-IO. First thing to notice is that the performance provided by FPFs is almost one order of magnitude better than that of OrangeFS when 8 servers are used. Second thing is that OrangeFS hardly improves its performance by adding servers. Considering that this scenario cre-

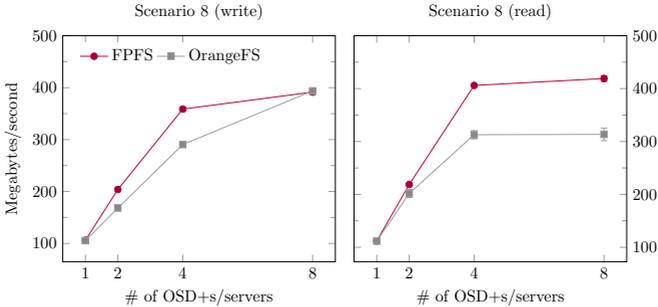


Fig. 6. HPCS-IO scenario 8 (write and read rates).

ates many small files, we can conclude that FPFS deals with metadata operations much better than OrangeFS. This is due to two reasons. First one, as we will show in Section V-B, is that FPFS produces much less network traffic than OrangeFS per metadata operation. Second reason is that overhead introduced by the FPFS's implementation seems to be smaller than that added by OrangeFS.

When there are only a few files and large data transfers, results obtained by FPFS and OrangeFS are more alike. Fig. 6 shows this case through HPCS-IO scenario 8. However, despite being a test dominated by data transfers, FPFS still provides a higher aggregated bandwidth than OrangeFS: up to 23.5% for writes and 4 servers, and up to 34% for reads and 8 servers. Note that rates hardly increase when the number of servers grows from 4 to 8. This is because network interface cards (NICs) in the clients are saturated with eight servers. This is more evident for the read part of this test when the file system is FPFS. Since we have four compute nodes to run clients, and each node has a single Gigabit NIC, the NICs provide a theoretical bandwidth of 500 MB/s altogether, which, in practice, decreases because of the overhead introduced by the network stack.

In order to explain the difference in throughput between FPFS and OrangeFS, we have analyzed the network traffic. This analysis shows that OrangeFS adds more network overhead (around 4.5%) than FPFS (around 2.5–4.1%) in this test. However, these overhead differences are small compared to the differences seen in the graphs, so the better performance of FPFS should be due to other reasons: better metadata management in FPFS, smaller latencies, etc.

Fig. 6 also shows another interesting fact. Since the read phase is performed immediately after the write phase, and taking into account that both write and read rates are smaller than the aggregated bandwidth provided by the network, it is obvious that neither FPFS nor OrangeFS implements a cache of any kind in the clients.

Fig. 7 depicts results for HPCS-IO scenarios 9,

10, 11 and 12, which only perform `stat()` operations on 2560000 empty files. In scenarios 9 and 11, only one process carries out the operations, so their graphs look quite similar. These graphs also show that FPFS achieves around one order of magnitude more operations/s than OrangeFS, and that FPFS provides a steady performance regardless the number of servers, while OrangeFS's performance slightly decreases when there are more servers.

In scenario 10, FPFS's performance is more than 12× better than OrangeFS's. Both file systems provide a quite steady performance, regardless the number of servers, despite the fact there are 10 clients issuing `stat()` operations. The situation changes for FPFS in scenario 12, where it greatly improves its performance that also scales up with the number of servers. OrangeFS, however, does not change its behaviour, and it basically provides the same performance as in scenario 10. Due to this, FPFS gets more than 34× operations/s than OrangeFS. A reason for the FPFS's different behavior between scenarios 10 and 12 is that there are 128 clients issuing `stat()` operations in scenario 12, so servers receive more load and are better profited.

B. Single shared directory

This section analyzes the performance and scalability of FPFS and OrangeFS when hundreds of processes concurrently access a single shared huge directory to create, get the status and delete files. As stated in Section IV-B, different synchronization points used by HPCS-IO scenarios limit the final performance and do not show all the potential that a file system can offer, specially for distributed directories. For this reason, we have run this additional set of microbenchmarks (also described in Section IV-B) that do not synchronize clients in any way.

There are 256 clients spread across four compute nodes that work on equally-sized disjoint subsets of the files. The directory contains $F \times N$ files, where F is either 200000, 400000 or 800000, and N is the number of servers. By changing the number of fi-

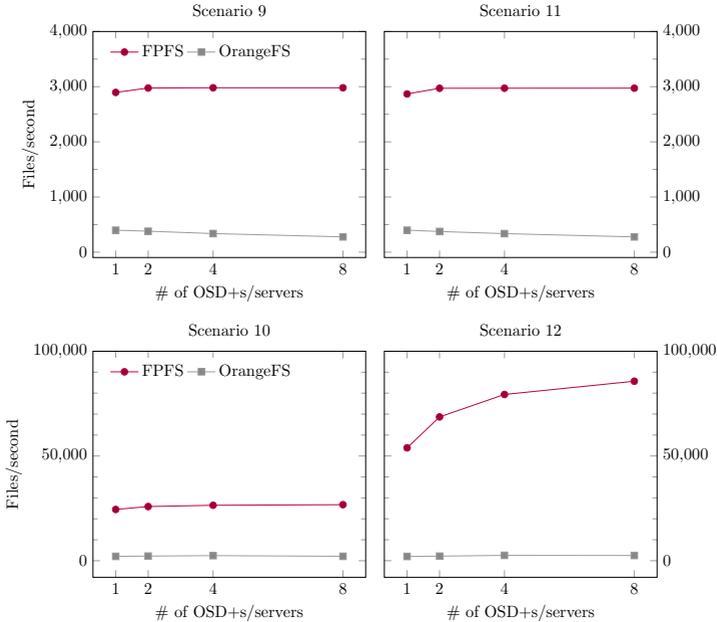


Fig. 7. HPCS-IO scenarios 9, 10, 11 and 12.

les per directory, we can analyze the effect of the directory size on performance and scalability. The directory is distributed right from the start, as we have already explained. Files are uniformly distributed among the storage servers, so all of them roughly receive the same load.

Experimental results can be seen in Fig. 8, where graphs provide operations per second for the create, stat and unlink tests. The first remarkable thing is the huge performance difference between FPFS and OrangeFS. FPFS always gets one order of magnitude more operations/s than OrangeFS at least, but it is usually much better and it achieves up to $75\times$ more operations/s than OrangeFS in some cases of the stat test. Also, it is worth noting that, with just 8 OSD+s and a Gigabit interconnect, our proposal is able to create, stat, and delete more than 205 000, 298 000 and 221 000 files per second, respectively.

These performance differences between FPFS and OrangeFS are explained by the better metadata management carried out by FPFS, and by the network traffic generated by each file system. After analyzing that traffic, we have seen that OrangeFS generates, at least, $5\times$ more network traffic than FPFS, reaching $16\times$ in some cases of the unlink test. Network traffic is a limiting factor of networked file sys-

tems [14] and, in this regard, FPFS is quite efficient.

Another remarkable fact is that the directory size can determine the performance in some cases, specially for FPFS in the unlink test. This is because Ext4's performance usually drops as the number of directory entries grows. Indeed, the problem in unlink is that, if a directory grows, disk writes increase by a factor much greater than the increase in the number of files. Consequently, FPFS gets a better throughput when there are 200 000 files per object than when there are 800 000. The situation is different in OrangeFS. Since OrangeFS stores directory entries in a few files of a BerkeleyDB, it does not produce huge directories, so the results are usually very similar for different directory sizes, and there are only some appreciable differences in the stat test.

Finally, Fig. 8 shows that OrangeFS seems to present some scalability problems for `stat()` operations, since lines tend to be flat with four or more servers. This problem does not appear in FPFS.

VI. RELATED WORK

There exist many file and storage systems nowadays. This section, however, focuses on some of the existing file systems that implement a metadata cluster, support the distribution of directories, and use

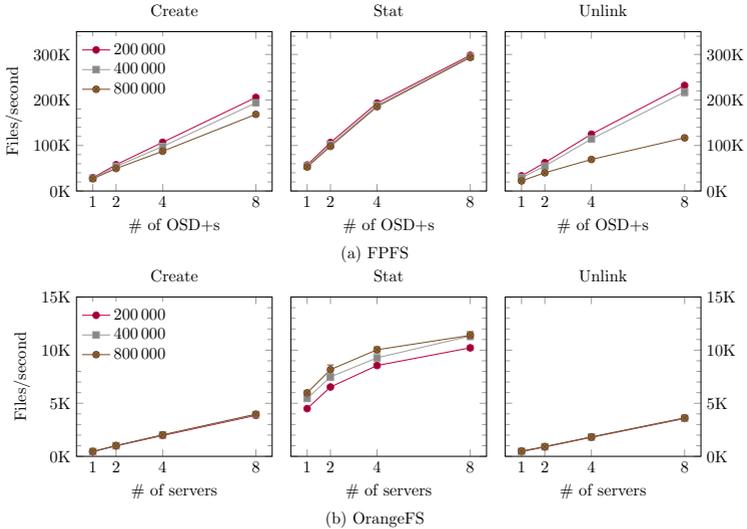


Fig. 8. Shared huge directory. Weak scaling when the number of files per server is 200 000, 400 000 or 800 000. Results for (a) FPFs and (b) OrangeFS. Note the different Y-axis scales between FPFs and OrangeFS.

OSD or similar devices.

Ceph [9] has a cluster of OSD devices where data objects are stored. OSD devices work in an autonomous manner in order to provide data-object redundancy for fault tolerance, among other things. Ceph uses CRUSH and placement groups to distribute objects and their replicas across the cluster. However, it uses dynamic sub-tree partitioning for the namespace. Contents of directories are written to objects in the OSD cluster, although metadata operations are carried out by a small cluster of metadata servers. Each metadata server controls the popularity of metadata within a directory, and adaptively splits a directory when it gets too big or experiences too many accesses. Despite all these features, at the time of this writing, the metadata part of Ceph is still unstable and many operations does not work. Therefore, we have not been able to successfully run Ceph with the proposed workloads and setups.

Like Ceph, OrangeFS [8] provides a cluster of data servers to improve the performance and scalability. These servers are not OSD devices, although they play a similar role. OrangeFS has supported several metadata server for quite a long time, but only recently the distribution of a directory among several servers has been introduced [28]. This distribution is based on ideas from extendible hashing [29] and GIGA+ [2]. When a directory is created, an array of *dirdata objects* (each on a metadata server) is allocated. Directory entries are then spread across the different *dirdata objects*, whose number is configura-

ble per directory.

Finally, Lustre [10] also offers a cluster of data servers through OSD devices. Traditionally, Lustre has provided a single metadata server with failover features, but latest versions also allow to use several MDTs in the same file system. A directory can also be shared out among several MDTs, but this distribution is statical, and it is set up when the directory is created. Like Ceph, Lustre still fails with the proposed setups, so we have not been able to successfully use it either.

FPFS shares some important features with all the above file systems: existence of several data and metadata servers, use of OSD or similar devices, data objects, distributed directories, etc. However, design and implementation aspects determine the performance and scalability of all of them. For instance, all but FPFs utilize separate data and metadata services, which makes it difficult when not impossible to optimize some operations that involve both data and metadata elements. OSD+ devices deployed in FPFs also add a small-overhead thin software layer that leverages the underlying local file system to provide their services. Thanks to these devices, FPFs can provide better performance and scalability than the other parallel file systems in many workloads, specially in those that are metadata-intensive.

VII. CONCLUSIONS

We have described the implementation of data objects in an OSD+ device. We have showed how

OSD+s, which are able to deal with both data and metadata operations, can internally optimize their implementation to speed up some common file operations. This kind of optimizations are not possible in other file systems like Lustre, OrangeFS or Ceph, where data and metadata elements are, from a conceptual point of view, managed independently.

We have added support for data operations to our OSD+-based Fusion Parallel File System and compared its performance with that achieved by OrangeFS. Results show that, for metadata-intensive workloads such as creating, stating and deleting files, PFPS provides a throughput that is, at least, one order of magnitude better than that of OrangeFS. Moreover, PFPS can easily process more than 200 000 metadata operations per second, reaching almost 300 000 in some cases, while OrangeFS hardly achieves a few thousands of operations per second. For workloads with large data transfers, PFPS can obtain up to 34% more aggregated bandwidth than OrangeFS.

As future work, we plan to add resilience support to both data and directory objects.

ACKNOWLEDGEMENTS

Work supported by the Spanish MEC, and European Commission FEDER funds, under grants TIN2012-38341-C04-03 and TIN2015-66972-C5-3-R.

REFERENCIAS

[1] Richard Freitas, Joseph Slember, Wayne Sawdon, and Lawrence Chin. “GPFS scans 10 billion files in 43 minutes.” Tech. Rep. RJ10484, IBM Almaden Research Center, 2011.

[2] Swapnil Patil and Garth Gibson. “Scale and concurrency of GIGA+: File system directories with millions of files,” in *Proceeding of the 9th USENIX Conference on File and Storage Technologies (FAST’11)*, February 2011, pp. 15–30.

[3] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. McLarty. “File system workload analysis for large scale scientific computing applications,” in *Proceedings of the 21st IEEE Conference on Massive Storage Systems and Technologies (MSST’04)*, 2004, pp. 139–152.

[4] Ric Wheeler. “One billion files: Scalability limits in Linux file systems,” in *LinuxCon’10*, August 2010.

[5] John Bent, Garth Gibson, Gary Gridler, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, and Meghan Wingate. “PLFS: A checkpoint filesystem for parallel applications,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC’09)*, 2009, pp. 1–12.

[6] Andreas Dilger. “Lustre metadata scaling,” April 2012, Tutorial at the 28th IEEE Conference on Massive Data Storage (MSST’12).

[7] Shafeeq Simanmohideen, Raja R. Sambasivan, James Hendricks, Likun Liu, and Gregory R. Ganger. “A transparently scalable metadata service for the Ursa Minor storage system,” in *Proceedings of USENIX Annual Technical Conference (ATC’10)*, 2010, pp. 1–14.

[8] The PVFS Community. “The Orange file system,” <http://orangesf.org>, March 2015.

[9] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. “Ceph: A scalable, high-performance distributed file system,” in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI’06)*, 2006, pp. 307–320.

[10] OpenSFS and EOPFS. “The Lustre file system,” <http://www.lustre.org>, March 2015.

[11] Ana Avilés-González, Juan Piernas, and Pilar González-Férez. “A metadata cluster based on OSD+ devices,” in *Proceedings of the 23rd International Symposium on*

Computer Architecture and High Performance Computing (SBAC-PAD), 2011, pp. 64–71.

[12] Ana Avilés-González, Juan Piernas, and Pilar González-Férez. “Scalable metadata management through OSD+ devices,” *International Journal of Parallel Programming*, vol. 42, no. 1, pp. 4–29, February 2014.

[13] Ana Avilés-González, Juan Piernas, and Pilar González-Férez. “Scalable huge directories through OSD+ devices,” in *Proceedings of the 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013, Belfast, United Kingdom*, 2013, pp. 1–8.

[14] Ana Avilés-González, Juan Piernas, and Pilar González-Férez. “Batching operations to improve the performance of a distributed metadata service,” *The Journal of Supercomputing*, vol. 72, no. 2, pp. 654–687, February 2016.

[15] Cray Inc.. “HPCS-IO,” <http://sourceforge.net/projects/hpcs-io>, October 2012.

[16] M. Mesnier, G. R Ganger, and E. Riedel. “Object-based storage,” *IEEE Communications Magazine*, vol. 41, no. 8, pp. 84–90, August 2003.

[17] N. Ali, A. Devulapalli, D. Dalessandro, P. Wyckoff, and P. Sadayappan. “An OSD-based approach to managing directory operations in parallel file systems,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC’08)*, 2008, pp. 175–184.

[18] Garth A. Gibson, David Nagle, Khalil Amiri, Jeff Butler, Fay W. Chang, Howard Gobioff, Charles Hardin, Erik Riedel, David Rochberg, and Jim Zelenka. “A cost-effective, high-bandwidth storage architecture,” in *Proceedings of the international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’98)*, Oct. 1998, pp. 92–103.

[19] SwiftStack Inc.. “Kinetic motion with Seagate and OpenStack Swift,” <https://swiftstack.com/blog/2013/10/22/kinetic-for-openstack-swift-with-seagate/>, October 2013.

[20] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. “The Hadoop distributed file system,” in *Proceedings of the 26th IEEE Conference on Massive Storage Systems and Technologies (MSST’10)*, 2010, pp. 1–10.

[21] S. A. Brandt, E. L. Miller, D. D. E. Long, and L. Xue. “Efficient metadata management in large distributed storage systems,” in *Proceedings of the 20th IEEE Conference on Mass Storage Systems and Technologies (MSST’03)*, 2003, pp. 290–298.

[22] G. R. Ganger and M. F. Kaashoek. “Embedded inodes and explicit groupings: Exploiting disk bandwidth for small files,” in *Proceedings of USENIX Annual Technical Conference (ATC)*, January 1997, pp. 1–17.

[23] D. Skeen and M. Stonebraker. “A formal model of crash recovery in a distributed system,” *IEEE Transactions on Software Engineering*, vol. 9, no. 3, pp. 219–228, May 1983.

[24] Swapnil Patil, Kai Ren, and Garth Gibson. “A case for scaling HPC metadata performance through de-specialization,” in *Proceedings of 7th Petascale Data Storage Workshop Supercomputing (PDSW’12)*, November 2012, pp. 1–6.

[25] S. Weil. *Ceph: reliable, scalable, and high-performance distributed storage*, Ph.D. thesis, University of California, Santa Cruz, (CA), December 2007.

[26] Jaeho Kim, Yongseok Oh, Eunsam Kim, Jongmoo Choi, Donghee Lee, and Sam H. Noh. “Disk Schedulers for Solid State Drivers,” in *Proceedings of the 7th ACM International Conference on Embedded Software*, 2009, pp. 295–304.

[27] Sun-Oracle. “Lustre tuning,” <http://wiki.lustre.org/manual/LustreManual18.HTML/LustreTuning.html>, 2010.

[28] Shuangyang Yang, Walter B. Ligon III, and Elaine C. Quarles. “Scalable distributed directory implementation on Orange File System,” in *Proceedings of 7th international Workshop on Storage Network Architecture and Parallel I/Os (SNAPI’11)*, 2011.

[29] Ronald Fagin, Jurg Nievergelt, Nicholas Pippenger, and H. Raymond Strong. “Extendible hashing - a fast access method for dynamic files,” *ACM Transactions on Database Systems*, vol. 4, pp. 315–344, September 1979.

Mecanismo de clasificación de páginas basado en el paso de tokens entre TLBs

Albert Esteve,¹ Alberto Ros,² Antonio Robles¹ y María E. Gómez¹

Resumen.— Clasificar los accesos a memoria como datos privados o compartidos se ha convertido en un esquema fundamental para lograr eficiencia y escalabilidad en sistemas multi- y many-core. Puesto que la mayor parte de los accesos a memoria tanto en aplicaciones secuenciales como paralelas son considerados datos privados (accedidos por un único núcleo) o de solo lectura (no se escriben), consagrar el coste del mantenimiento de la coherencia en cada acceso a memoria resulta en un rendimiento sub-óptimo, al mismo tiempo que limita la escalabilidad y eficiencia del sistema. Este trabajo propone TokenTLB, un mecanismo de clasificación de páginas basado en el intercambio y la cuenta de tokens (testigos). La observación principal tras nuestra propuesta es que, a diferencia del mantenimiento de la coherencia, usar tokens para clasificar datos obtiene todos los beneficios de un protocolo basado en tokens sin la carga de un sistema de arbitraje complejo, cuya aplicación ha desalentado la inserción de dichos protocolos en los procesadores actuales. Contar tokens en las TLBs supone una forma natural y eficiente para la clasificación de páginas de memoria. Por un lado, evita el uso de solicitudes persistentes o arbitraje, ya que si dos o más TLBs compiten para acceder a una página, los tokens se distribuyen apropiadamente y la clasifican como compartida. Por otro lado, TokenTLB también favorece la compartición de la traducción entre las TLBs del sistema, lo cual mejora el rendimiento del mismo y reduce gran parte del tráfico en comparación con otros mecanismos de clasificación basados en la difusión de mensajes entre TLBs. Esta reducción es debida a que solo las TLBs que poseen tokens adicionales están a cargo de suministrarlos junto con la traducción de la página (una respuesta por fallo de TLB). TokenTLB incrementa de forma efectiva los bloques clasificados como privados hasta un 61.1%, permitiendo la detección de datos de solo lectura (hasta un 24.4% de bloques compartidos de solo lectura). Si aplicamos TokenTLB para optimizar la caché de directorio, la energía dinámica consumida por la jerarquía caché se reduce un 27.3% sobre el sistema base.

Palabras clave.— Clasificación de datos; Protocolo de tokens; Privado-compartido; Datos de solo lectura

I. INTRODUCCIÓN

LOS procesadores multinúcleo (CMPs) están compuestos de una creciente cantidad de núcleos. Esto, a su vez, requiere del uso de jerarquías de memoria multi-nivel por motivos de rendimiento y un modelo de memoria compartida para facilitar la programabilidad. Los modelos de memoria compartida requieren protocolos de coherencia caché para sacar partido a su potencial mejora de rendimiento. Los protocolos basados en directorio son los más adecuados para afrontar los nuevos retos de escalabilidad [1–4], éstos requieren menor ancho de banda

de red comparados con los protocolos *snooping*. Sin embargo, no son capaces de distinguir si los datos accedidos son privados o compartidos. Así, no se saca partido a las potenciales oportunidades para la mejora del rendimiento y la escalabilidad que ofrece esta característica.

Un gran número de propuestas recientes emplean un mecanismo de clasificación de datos y/o accesos en privados o compartidos para así superar las limitaciones en términos de escalabilidad de los CMPs actuales [1, 5–22]. La idea principal en todos estos trabajos es que la naturaleza de los datos privados o compartidos es distinta y, por tanto, las referencias hechas a estos datos se pueden optimizar de acuerdo a dicha naturaleza. La clasificación de datos se ha convertido, por tanto, un mecanismo clave para el diseño de multiprocesadores más escalables y eficientes.

En un escenario ideal, la clasificación debe realizarse con una baja sobrecarga en términos de tráfico, rendimiento o área. Sin embargo, clasificar datos conlleva a menudo grandes requerimientos en términos de almacenamiento para rastrear el estado de compartición. En otras propuestas no se tiene en cuenta las transiciones de compartido a privado, por lo que la precisión en la clasificación se puede ver comprometida, limitando los beneficios potenciales [1, 10, 12]. Por otro lado, algunos mecanismos de clasificación almacenan su estado de clasificación en las memorias de directorio o las cachés [5, 7, 11, 17, 22], limitando su aplicabilidad. En general, cuanto antes se obtenga la clasificación en un acceso a memoria, mejor. Finalmente, la detección de escritura también ha sido explorada, extendiendo el ámbito del esquema de clasificación y mejorando su eficacia [6, 10], ya que los accesos a datos de solo lectura representan una gran fracción de los accesos a memoria.

Además, la latencia al traducir direcciones se añade al camino crítico para los accesos a memoria y se han dedicado numerosos esfuerzos para reducir dicho impacto. Específicamente, las transferencias de TLB a TLB [9, 18] están indicadas para la clasificación adaptativa de datos aplicados a la desactivación de la coherencia [1, 6], un mecanismo eficiente para mejorar la escalabilidad de los directorios en CMPs. Sin embargo, estas transferencias inundan la red con respuestas tras cada fallo de TLB, muchas de ellas traducciones de página replicadas. Incluso aunque los fallos de TLB son poco frecuentes (solo un 2% de los accesos a la TLB son fallos), estos mensajes no escalan adecuadamente con el tamaño del sistema. Por último, la reclasificación de compartido a privado no se produce de forma inmediata, sino que es pospues-

¹Department of Computer Engineering, Universitat Politècnica de València, e-mail: alesgar@gap.upv.es, {megomez,arobles}@disca.upv.es.

²Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: aros@itdec.um.es..

ta hasta que la página es desalojada de la memoria principal y accedida de nuevo, penalizando por ello la precisión en la clasificación.

En este trabajo proponemos TokenTLB, un mecanismo de clasificación novedoso, implementado directamente en la estructura de TLB e inspirado por los protocolos de coherencia token [23–25]. TokenTLB se basa en la observación de que, a diferencia de la coherencia, aplicar los tokens para la clasificación no requiere utilizar peticiones persistentes o mecanismos de arbitraje complejos. Las peticiones persistentes son un tipo especial de solicitud ideada para resolver las carreras de datos que ocurren ocasionalmente cuando muchos núcleos intentan escribir datos al mismo tiempo. Estas peticiones suponen una fuente de complejidad para el protocolo de coherencia, representando a su vez uno de los motivos principales por el cual los protocolos de coherencia basados en tokens no se han implementado ampliamente en los procesadores actuales. Sin embargo, TokenTLB evita estas carreras dedicando los tokens a la clasificación de datos. Si dos o más TLBs compiten al acceder a una página, los tokens se distribuyen entre ellas de forma natural y la página es clasificada como compartida. En otras palabras, la clasificación no requiere un *owner token*. Las principales contribuciones de TokenTLB son:

1. TokenTLB está diseñado como un mecanismo de clasificación a nivel de página en las TLBs, basado en tokens. Permite la compartición a través de TLBs, lo cual acelera las traducciones de página, favoreciendo una mejora del rendimiento del sistema.
2. TokenTLB reduce el consumo de la red en comparación a otras propuestas similares basadas en TLBs. Solo TLBs que son propietarias de tokens están a cargo de proveerlos junto con la traducción de la página, lo que supone alrededor de una respuesta por fallo de TLB.
3. TokenTLB extiende la caracterización de la clasificación a través de la detección de escrituras a páginas, al mismo tiempo que realiza una clasificación basada en la cuenta y el intercambio de tokens. La clasificación basada en tokens hace posible que se identifique de forma natural e inmediata como una entrada de TLB evolucionada de compartida a privada.
4. TokenTLB introduce un predictor capaz de resolver fallos de TLB a través de mensajes punto a punto, incrementando así su escalabilidad. El predictor se basa en un pequeño buffer llamado Token Predictor Buffer (TPB), el cual se encarga del almacenamiento durante un corto espacio de tiempo de potenciales TLBs propietarias.

Por medio de simulaciones ciclo a ciclo de un sistema CMP de 16 núcleos, incluyendo una gran variedad de cargas de trabajo tanto científicas como comerciales, se muestra que TokenTLB aumenta la cantidad de bloques clasificados como privado en el momento del fallo hasta un 61.1%, y los bloques com-

partidos de solo lectura hasta un 24.4%. Además, si la clasificación se aplica a la desactivación de la coherencia, ésta conlleva una reducción en la utilización del directorio, dejándolo en un 28.8% de ocupación. También se reduce el consumo de la jerarquía caché hasta un 27.3% sobre el sistema base. Finalmente, la inclusión de la TPB ha permitido reducir aún más el tráfico de las TLBs prácticamente un 20% sobre el protocolo TokenTLB base. Específicamente, TokenTLB, combinado con TPB, genera una sola respuesta por fallo de TLB. En el caso de que la traducción sea resuelta en la tabla de páginas, ninguna TLB debe responder. Por tanto, TPB permite, junto con TokenTLB, un mecanismo más escalable.

II. MOTIVACIÓN Y TRABAJO RELACIONADO

A. Clasificación de Datos

Los mecanismos de clasificación de datos están ganando interés, ya que permiten un gran número de optimizaciones en el manejo de bloques basándose en su estado de compartición. Hay muchos ejemplos recientes en la literatura para esquemas de clasificación. Específicamente, Kim *et al.* [12] evitan solicitar datos coherentes a través de mensajes de difusión en los protocolos *snooping* al acceder bloques privados, lo cual reduce el tráfico de red generado. Alternativamente, Y. Li *et al.* [15] presentan una pequeña estructura buffer cerca de las TLBs, llamada buffer de compartición parcial (PSB). Cuando una página se hace compartida, estará probablemente alojada en la PSB tras un fallo de TLB. Así, la traducción de la página se puede obtener con baja latencia y menores recursos de almacenamiento. Además, Hardavellas *et al.* [10] y Kim *et al.* [14, 16] mantienen los bloques privados en el banco NUCA local, reduciendo así la latencia de acceso a las caches NUCA. Ros y Kaxiras [26] proponen un protocolo de coherencia simple y eficiente que implementa una política de escritura aplazada para los bloques privados y de escritura inmediata para los compartidos. Finalmente, Cuesta *et al.* [1, 6] proponen evitar el almacenamiento de bloques privados en los directorios, desactivando así el mantenimiento de la coherencia para éstos bloques. La desactivación de la coherencia permite así directorios más pequeños y rápidos.

La mayor parte de las propuestas descritas usan mecanismos de clasificación que se benefician de estructuras del sistema operativo existentes (i.e., las TLBs o la tabla de páginas) para clasificar las páginas y almacenar el estado de las mismas. Así evitan requerir estructuras hardware adicionales. Por otro lado, las propuestas asistidas por el compilador [14, 16] lidian con la dificultad de saber en tiempo de compilación (a) si una variable va o no a ser accedida y (b) en qué núcleos van a ser planificados y replanificados los datos. Además, los mecanismos basados en directorio [5, 7, 11, 17, 22] solo descubren el estado de compartición de los datos tras acceder a la estructura de directorio o caché, limitando por tanto su aplicabilidad a optimizaciones donde el conocimiento a-priori del estado de los datos accedi-

	A-priori	Solo lectura	Adaptativo	Preciso
Directory	✗	✓	✓	✓
TLB	✓	✗	✓	✓
OS	✓	✓	✗	✓
Compiler	✓	✓	✓	✗
TokenTLB	✓	✓	✓	✓

TABLA I: Propiedades de los esquemas de clasificación

dos no sea requerido. Por último, mecanismos basados en las propiedades de los lenguajes de programación [20, 21], a pesar de ser muy precisos, no son aplicables a la mayor parte de los códigos existentes. Por contra, los mecanismos basados en el sistema operativo realizan una clasificación en tiempo de ejecución válido para cualquier código, evitando dichas dificultades.

El principal problema de la clasificación basada en el sistema operativo es que realiza una clasificación no adaptativa. Si una página transita de privada a compartida, permanecerá en ese estado durante el resto del tiempo de ejecución (a no ser que se desaloje la página de la memoria principal). En aplicaciones ejecutándose durante un largo período de tiempo, muchas páginas podrían llegar a ser consideradas como compartidas en algún momento, negando las ventajas que se derivan de la clasificación.

Para realizar una clasificación adaptativa que detecte temporalidad en páginas privadas y la migración de sus hilos, se introdujo la *clasificación basada en TLBs* [9, 18], que se utiliza transferencias entre TLBs para consultar las páginas accedidas desde otros núcleos del sistema y así descubrir de forma natural si los bloques dentro de las mismas están presentes en una caché remota, y por tanto la página es compartida, o si, por el contrario, la página esta actualmente privada. Las transferencias entre TLBs están basadas en la observación de que las comunicaciones entre núcleos del CMPs son mucho más rápidas comparadas a las comunicaciones en procesadores tradicionales. Otros trabajos ya se han beneficiado de dicha observación con diferentes propósitos [27, 28].

Sin embargo, las transferencias entre TLBs generan respuestas replicadas, que a su vez incluyen réplicas de la traducción de cada núcleo del sistema tras cada fallo de TLB, lo cual hace que el consumo de la red se incremente de forma drástica [9]. Por otro lado, enviar difusiones en la red de forma frecuente, generando múltiples respuestas cada vez, no es una propuesta escalable, ya que la cantidad de etapas de paso de mensajes incrementa de forma proporcional al tamaño del sistema. Además, reclasificar páginas a privado requiere que la traducción sea eliminada completamente de las TLBs del sistema para que sea efectiva, limitando por tanto la precisión del mecanismo de clasificación. Finalmente, la detección de escritura no ha sido explorada, limitando así el esquema de clasificación a la clásica dicotomía privado-compartida para un mecanismo adaptativo.

La tabla I resume las propiedades principales de los mecanismos de clasificación en el estado del arte comparados con TokenTLB. En primer lugar, conocer la clasificación antes (*a-priori*) de acceder a la

caché es crítico para la aplicabilidad del mecanismo de clasificación. Algunas propuestas almacenan el estado de compartición en el directorio y, por tanto, no pueden emplearse para técnicas tales como la *desactivación de la coherencia* [1] o *reactive NUCA* [10], entre otros. Además, la clasificación *adaptativa* conlleva una enorme mejora en la precisión del mecanismo, detectando la migración de los hilos y los accesos a datos en diferentes fases privadas de la página. La clasificación de *solo lectura* es también una propiedad de gran calado, ya que los bloques compartidos de solo lectura pueden llegar a suponer hasta un 48.7% de todos los bloques accedidos [6]. Sin embargo, ningún mecanismo de clasificación basado en TLBs ha explorado previamente la detección de escritura. Finalmente, los compiladores deben ser conservadores en la clasificación, ya que no poseen información acerca del estado de compartición en tiempo de ejecución, y por tanto, no están obligados a realizar una clasificación *precisa*, ya que la privacidad no se puede garantizar siempre.

B. Coherencia Token

Martin *et al.* presentó TokenB [24], que captura los mejores aspectos de los protocolos de *snoop* y directorio: fallos de caché con baja latencia y la no dependencia en redes de interconexión totalmente ordenadas. Raghavan *et al.* presentó también Token tenure [29], basándose en una caché de directorio para almacenar los tokens.

Los protocolos token garantizan la seguridad de la coherencia a través de la cuenta de los tokens: un procesador puede escribir si y solo si contiene todos los tokens, y puede leer si y solo si contiene al menos un token para dicho bloque. Sin embargo, puesto que las peticiones son enviadas a todos los procesadores a través de mensajes de difusión, se pueden producir condiciones de carrera en el protocolo cuando compiten por un bloque de memoria, y por tanto no resuelven el fallo caché en ninguno de sus intentos. Para poder evitar la inanición y garantizar la resolución de los fallos, los protocolos token invocan *peticiones persistentes* tras diez tiempo de fallo medios no satisfechos.

Las peticiones persistentes causan grandes problemas, ya que requieren arbitraje, añaden una sobrecarga de latencia poco flexible y requieren estructuras adicionales no escalables en la oblea de silicio. Ésta es una de las principales razones por las que los protocolos token no se han impuesto en los procesadores actuales. Por contra, las TLBs no modifican directamente las traducciones en la estructura TLB. Como consecuencia, usar tokens para clasificar y distribuirlos a través de las TLBs puede evitar éstas condiciones de carrera. En el caso de que varias TLBs disputen por la traducción de una misma página, ésta será sencillamente clasificada como compartida.

III. TOKENTLB

Nuestro objetivo es lograr todas las propiedades deseables para un mecanismo de clasificación: realizar la clasificación previamente al acceso a la jerarquía caché; implementar un mecanismo completamente adaptativo que sea capaz de realizar una reclasificación de forma precisa; mejorar la caracterización de la clasificación al discernir los accesos de escritura, reconociendo así las páginas de solo lectura; y realizar una clasificación precisa en tiempo de ejecución que sea válida para cualquier código.

Para este fin, este trabajo propone TokenTLB, una técnica de clasificación adaptativa basada en el conteo de tokens. TokenTLB acelera los fallos de TLB a través de un eficiente sistema de resolución de traducciones entre TLBs, al mismo tiempo que copa con la sobrecarga de tráfico que habitualmente conlleva su uso.

A. Clasificar contando tokens: Concepto

TokenTLB asocia un número fijo de tokens a cada entrada de traducción. En un sistema con N núcleos, debe haber N tokens por entrada. Los tokens no se pueden crear o destruir. Los tokens se intercambian a través de mensajes entre TLBs junto con la traducción de la dirección de página. Una entrada de TLB se clasifica de acuerdo con su conteo de tokens: privada si contiene todos los tokens (N), compartida mientras contenga un subconjunto de todos los tokens de la página (de 1 a $N - 1$), e inválido si no contiene ningún token. Finalmente, solo entradas de TLB válidas (es decir, que mantengan al menos un token) puede contestar tras una petición de traducción.

Además, para poder discernir si la página ha sido escrita o no, cada entrada de traducción tiene asociado un *flag* de escritura (W , *written*) que es enviado junto con los tokens en transacciones de TLB. El flag de escritura incrementa el alcance de la traducción, añadiendo categorías adicionales: Privada de solo lectura, privada-escrita, compartida de solo lectura y compartida-escrita.

B. Solicitud de tokens tras un fallo de TLB

TokenTLB inicia la resolución de la página en la tabla de páginas tras un fallo de TLB en paralelo con un mensaje de difusión a otras TLBs del sistema. Inicialmente, la tabla de páginas mantiene los N tokens para cada traducción. En consecuencia, tras el primer fallo de TLB para una página de memoria, la tabla de páginas concede todos los tokens a la TLB que lo solicitó. A partir de ahí, los tokens son mantenidos por las TLBs y enviados a través de mensajes como respuesta a otras TLBs solicitándolos, facilitando su intercambio. Cuando una TLB recibe una petición de una traducción de otra TLB, comprueba si posee la entrada de traducción (es decir, si mantiene la traducción junto con dos o más tokens) y si es así, responde a la solicitud con un mensaje, manteniendo un token y enviando el resto. Al recibir la primera respuesta con la traducción y los tokens, la

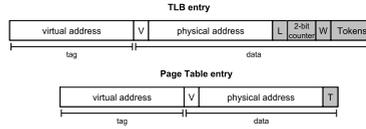


Fig. 1: Formatos de entrada para la TLB y la tabla de páginas. Los campos sombreados representan los campos adicionales que se requieren.

consulta a la tabla de páginas es cancelada, los tokens son anotados de forma privada en la entrada de TLB correspondiente y el acceso de memoria puede proceder. De ésta forma, el tráfico de respuesta en las TLBs es contenido, ya que solo se permite que una TLB responda en el caso general (la que ha adquirido la traducción más recientemente). Además, el acceso a la página se desbloquea antes comparado a otros mecanismos similares [9, 18], mejorando por tanto el tiempo de ejecución (como se ve en la Sección VI-A).

Los tokens se almacenan en un campo adicional de la TLB llamado *Tokens*, como se ven en la Figura 1. Ese campo añade $\log_2(N)$ bits a cada entrada (por ejemplo, solo 4 bits adicionales serían necesarios en un CMP de 16 núcleos) en la TLB. Cuando todos los tokens son cedidos, es el bit de válido/inválido (V) de la propia entrada de TLB el encargado de rastrear su estado. En el caso de la tabla de páginas, no se requiere hardware dedicado adicional, sino tan solo un bit adicional por cada entrada (T), indicando si posee o no todos los tokens. Éste bit adicional se puede tomar de los bits reservados de la entrada de la tabla de páginas.

Los fallos de TLB alojan una entrada en el Miss Status Holding Register (MSHR), el cual se desaloja tras adquirir la entrada de traducción de la página y al menos un token para la misma. Por tanto, si el fallo se resuelve en la tabla de páginas sin responder con tokens (éstos están actualmente almacenados en otras TLBs del sistema) debemos esperar a la primera respuesta que contenga tokens. El acceso a la página no se puede desbloquear sin la información de compartición (es decir, los tokens).

Cuando una página es escrita por primera vez (es decir, el bit W no está activo), éste bit necesita activarse en todas las copias de la traducción almacenadas en otras TLBs. Éste bit permanece activo hasta que el *tiempo de generación global* (el tiempo que pasa desde que la página es accedida por primera vez en una TLB hasta el momento en el que es desalojada de la última TLB del sistema) [9] termina para dicha página. Si la escritura acontece en una página privada no se requieren acciones adicionales. En caso contrario, un mensaje de difusión es enviado para actualizar el bit W en todas las TLBs que contengan tokens, lo cual hace que la página transite a compartida-escrita (SW). La información de escritura se envía junto con los tokens como parte de la información de compartición de la página en cada respuesta a fallos de TLB.

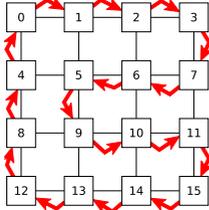


Fig. 2: Camino que sigue los mensajes con tokens tras desalojos en la TLB para una malla 2D de 16 núcleos.

C. Cesión de tokens para la Corrección

Los tokens no se crean ni se destruyen, se transfieren. Esto implica que el sistema debe garantizar en todo momento la existencia de N tokens para cualquier traducción de página. La tabla de páginas contiene o bien todos los N tokens o ninguno.

Al desalojar una entrada de TLB que contiene un subconjunto de tokens, se envía un mensaje buscando un nuevo propietario. Por contra, si una entrada de TLB contiene todos los N tokens, éstos se envían de vuelta a la tabla de páginas. En consecuencia, los desalojos en la TLB deben ser no silenciosos, lo cual añade consumo de red adicional. Sin embargo, los desalojos no silenciosos no dañan las prestaciones del sistema, ya que están fuera del camino crítico para los accesos de memoria. De hecho, los desalojos no silenciosos garantizan una clasificación más dinámica y son la clave para una pronta reclasificación a privado.

Como se ha indicado previamente, un subconjunto de tokens en una entrada de TLB en desalojo deben ser transferidas a un nuevo propietario. A tal fin, se envía un mensaje a otra TLB (*token_evict*) conteniendo tokens únicamente. En una situación ideal, una TLB que reciba un mensaje *token_evict* puede solo aceptar tokens si ya almacena una traducción para dicha página o si tiene alojada una entrada en el MSHR debido a un fallo TLB (Sección III-B). En cualquier otro caso, el mensaje se envía al siguiente TLB designado. Cuando el mensaje *token_evict* encuentra un nuevo propietario, los tokens se anotan en la nueva TLB, la cual responde con un ACK al emisor original.

Para poder evitar potenciales interbloqueos y minimizar el tráfico, la búsqueda de un nuevo propietario requiere una exploración completa de la red. Para ello añadimos el concepto de un anillo lógico, es decir, un camino que cubre todos los nodos de la red, minimizando el número de enlaces a atravesar. El camino seguido es dependiente de la topología. La Figura 2 ilustra un anillo de ejemplo para una malla 4x4. Éste camino representa una de las rutas circulares mínimas que recorren todos los nodos de la red de interconexión.

Sin embargo, los interbloqueos aún puede ocurrir si todas las TLBs que contienen una página (y todos sus tokens) intentan desalojar simultáneamente

la entrada asociada. Para solucionar dicho problema, los tokens contenidos en un mensaje *token_evict* pueden ser almacenados en la correspondiente entrada del MSHR de un núcleo cuya TLB este involucrada en un proceso de desalojo, siempre y cuando su ID (identificador) sea mayor que el ID del núcleo que inició el envío.

Cabe observar que la condición clave para evitar la inanición en un escenario en el que múltiples páginas de TLB que son desalojadas ceden indefinidamente los tokens es la comparación entre IDs de núcleo. Si todas las TLBs desalojan simultáneamente la misma página, todos los N tokens serán finalmente alojados en el MSHR de la TLB del núcleo con el ID mayor, el cual enviará todos los tokens de vuelta a la tabla de páginas.

En resumen, una TLB desalojando tokens puede simultáneamente almacenarlos. Por tanto, cuando una TLB recibe una confirmación indicando que los tokens han sido adquiridos por otra TLB, debe comprobar su propio MSHR de nuevo. Si éste contiene un subconjunto de tokens para dicha página, envía un nuevo mensaje buscando un nuevo propietario. En el caso de que el MSHR no contenga más tokens, el proceso de desalojo finaliza.

Por último, una solicitud entre TLBs puede fallar a adquirir tokens tras un fallo si todas las TLBs que sean propietarias de la traducción (entradas que contienen más de un token) han sido desalojadas poco antes de recibir la solicitud, y por tanto sus tokens están actualmente en vuelo. Dicha situación es prevenida a través de un tiempo de vencimiento que se configura tras un fallo de TLB, el cual reenvía el mensaje de difusión si no se consigue adquirir tokens tras el vencimiento. Éste evento no ocurre con frecuencia y no causa un incremento notable en el tráfico de red, como se muestra en la Sección VI-A.

D. Token Predictor Buffer

En la búsqueda de un mecanismo de clasificación más escalable, TokenTLB reduce el tráfico de respuesta de TLB y la duplicación de traducciones. Sin embargo, un mensaje de difusión es enviado aún tras cada fallo de traducción. Esto es debido a que las TLBs no tienen información previa de posibles propietarios en el momento del fallo, por tanto dichos fallos deben ser resueltos inundando la red. No obstante, algunos fallos de TLB ocurren poco después de una invalidación, y por tanto se podría predecir un potencial propietario. Por tanto, el tráfico TLB se puede reducir aún más, contribuyendo a un mecanismo de clasificación más escalable. Para dicho fin, presentamos un predictor que se encarga de identificar otras TLBs como potenciales propietarios. Acertar en el predictor tras un fallo de TLB envía un mensaje punto a punto, por tanto reduce el tráfico de petición. Este mecanismo basado en la historia reciente es similar al propuesto por Martin *et al.* [30]. Otros trabajos también se benefician de ésta observación con diferentes objetivos [31–33].

El predictor consiste en un pequeño buffer de da-

Parámetros de memoria			
Frecuencia del procesador	2.8GHz	Jerarquía TLB	Exclusive
L1 TLBs separadas instr & datos	8 conjuntos, 4 vías	L1 TLB tiempo de acierto	1 ciclo
L2 TLB combinada	128 conjuntos, 4 vías	L2 TLB tiempo de acierto	2 ciclos
Timeout adquisición de tokens	1200 ciclos	TPB	32 conjuntos, 4 vías
TPB tiempo de acierto	1 ciclo	Tamaño de página	4KB (64 bloques)
Jerarquía caché	No inclusiva	Tamaño de bloque caché	64 bytes
L1 cachés separadas instr & data	64KB, 4 vías	L2 cache combinada compartida	1MB/tila, 8 vías
L1 cache tiempo de acierto	1 (tag) y 2 (tag+datos) ciclos	L2 cache tiempo de acierto	2 (tag) y 6 (tag+datos) ciclos
Caché de directorio	256 sets, 4 ways	Directorio tiempo de acierto	1 ciclo
Tiempo de acceso a memoria	160 ciclos		
Parámetros de red			
Topología	Malla 2D (4x4)	Mecanismo de encaminamiento	Determinístico X-Y
Tamaño de flit	16 bytes		
Tiempo de encaminamiento, switch y enlace	2, 2 y 2 ciclos	Tamaño de mensajes de datos y control	5 flits y 1 flit

TABLA II: Parámetros del sistema base.

tos situado en paralelo con la L2 TLB llamado Token Predictor Buffer (TPB) que almacena la ID del proceso, la dirección virtual y la ID del núcleo. Tras ceder tokens en desalijos no silenciosos, el receptor se convierte en un propietario potencialmente conocido y es almacenado en la TPB. Si ocurre un fallo en la L1 TLB, se consultan en paralelo la L2 TLB y la TPB. Si la L2 TLB falla y la TPB mantiene información de un potencial propietario para la misma página, se envía un mensaje punto a punto y la entrada de TPB es desalojada. Si la TLB que recibe la solicitud es aún propietaria, responde con la traducción y la información de compartición. En caso contrario, si la TLB consultada no es ya propietaria, responde negativamente y el mecanismo habitual es invocado, inundando la red de interconexión y consultando en paralelo la tabla de páginas.

IV. DESACTIVACIÓN DE LA COHERENCIA CON TOKENTLB

Para probar los beneficios del esquema de clasificación que proporciona TokenTLB, lo aplicamos al mecanismo de *Desactivación de la Coherencia*, el cual ha demostrado una mejora en el uso del directorio bajo un esquema de clasificación no adaptativo [6]. De acuerdo con este esquema, tan solo los bloques pertenecientes a página compartidas y escritas (SW) requieren que se mantenga la coherencia, lo cual incrementa dramáticamente los accesos a datos cuya coherencia no es necesaria.

Como se ha indicado previamente, la clasificación en TokenTLB puede evolucionar naturalmente de compartido a privado y viceversa múltiples veces en cada tiempo de generación de cada página en cada TLB. A diferencia de otros mecanismos de clasificación, TokenTLB detecta transiciones a privado inmediatamente (tan pronto como la TLB obtiene todos los N tokens), lo cual permite detectar eficientemente periodos privados de vidas globales.

Sin embargo, al aplicar TokenTLB a la desactivación de la coherencia, hay que tener especial cuidado con sus implicaciones. Incluso en un estado coherente (SW), la clasificación podría transitar de nuevo a no coherente durante el mismo tiempo de generación de la página, siempre que una entrada recupere todos sus tokens y se convierta de nuevo en privada (PW).

Por tanto, en una reclasificación de privado o solo lectura a compartido y escrito, todas las copias no

coherentes de dicha página deben ser desalojadas de las caches privadas de todos los núcleos compartidores. Por contra, al reclasificar a no coherente, no se requieren acciones especiales.

V. ENTORNO DE SIMULACIÓN

Nuestra propuesta esta evaluada a través de simulación de sistema completo usando Virtutech Simics [34] junto con el conjunto de herramientas de Wisconsin GEMS [35], lo que habilita la simulación detallada de sistemas multiprocesador. La red de interconexión se ha modelado mediante el simulador GARNET [36]. La simulación consiste en una arquitectura CMP de 16 núcleos implementando un mecanismo de coherencia caché basado en directorio con los parámetros mostrados en la Tabla II, en la arquitectura considerada base para la evaluación. La latencia de la L2 TLB asume cuatro referencias a memoria para atravesar la tabla de páginas, como ocurre con el espacio de direcciones virtual de 28 bits del x86-64. Las latencias de las cachés y la TLB, así como su consumo, se han calculado usando la herramienta CACTI [37] asumiendo tecnología de 32nm. A través de la experimentación se ha determinado un tiempo de 1200 ciclos para la adquisición de tokens, lo cual ofrece un buen equilibrio entre prestaciones y tráfico de red.

Para la evaluación se han utilizado una amplia variedad de cargas de trabajo paralelas de varias suites, cubriendo así diferentes patrones y grados de compartición. *Barnes* (8192 bodies, 4 time steps), *Cholesky* (tk15.O), *FFT* (64K complex doubles), *Ocean* (258 × 258 ocean), *Radiosity* (room, -ae 5000.0 -en 0.050 -bf 0.10), *Raytrace-opt* (teapot), *Vobrend* (head), and *Water-NSQ* (512 molecules, 4 time steps) son de la SPLASH-2 benchmark suite [38]. *Tomcatv* (256 points, 5 time steps) y *Unstructured* (Mesh.2K, 5 time steps) son dos benchmarks científicos. *FaceRec* (script), and *SpeechRec* (script) pertenecen al ALPBenchs suite [39]. *Blackscholes* (simmedium), *Swaptions* (simmedium) y *x264* (simsmall) pertenecen a PARSEC [40]. Finalmente, *Apache* (1000 HTTP transactions) y *SPEC-JBB* (1600 transactions) son dos cargas de trabajo comerciales [41]. Raytrace-opt optimiza la aplicación Raytrace original al eliminar la adquisición del cierre de exclusión para un ID de ray que no esta en uso actualmente. Todos los resultados experimentales mostrados

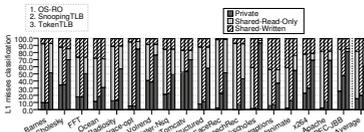


Fig. 3: Proporción de fallos de L1 de datos clasificadas.

dos corresponden a las fases paralelas de las aplicaciones.

VI. RESULTADOS DE LA EVALUACIÓN

Primeramente los resultados muestran cómo el mecanismo de clasificación TokenTLB se comporta comparado a anteriores mecanismos, incluyendo un estudio de sensibilidad para el Token Predictor Buffer y hasta cuatro tamaños distintos. Seguidamente, se introduce la desactivación de la coherencia en el análisis para comparar los beneficios que se obtienen al aplicar los diversos mecanismos del estudio.

A. Clasificación completamente Adaptativa

Ésta sección muestra la precisión y eficiencia de la clasificación con TokenTLB comparada con propuestas previas.

Datos Privados y de solo Lectura. El porcentaje de páginas privadas y compartidas es una buena métrica general para medir la bondad de un mecanismo de clasificación. Sin embargo, ésta métrica es injusta para los mecanismos de clasificación adaptativos, donde las páginas se reclasifican frecuentemente a privado, ya que dicha reclasificación no se muestra en las gráficas. Además, esta situación se ve favorecida por el hecho de que TokenTLB desbloquee el acceso a la página tras la primera respuesta de otra TLB, lo cual acelera la resolución de fallos de TLB, pero que puede resultar en accesos compartidos a páginas con tokens en vuelo.

A pesar de que un mecanismo adaptativo puede reclasificar páginas libremente, los bloques no son reclasificados individualmente durante su tiempo de generación local. Por tanto, cuanto más tiempo se mantiene una página como privada o de solo lectura, más fallos en la caché L1 serán clasificados como tal. La Figura 3 muestra la clasificación de los accesos a la caché L1 de datos. Se puede observar como los fallos en la caché de datos considerados como privados se incrementa ámpliamente al usar *TokenTLB*, ya que, a diferencia de *SnoopingTLB*, la reclasificación ocurre de forma natural durante el tiempo de generación de la página. En concreto, *TokenTLB* es capaz de clasificar un 61.1% de fallos en la caché L1 de datos de media como privado, hasta un 40.8% más que *SnoopingTLB*. También es importante observar que, a diferencia de *SnoopingTLB*, *TokenTLB* y *OS-RO* pueden reconocer páginas de solo lectura, lo cual representa un 24.4% de los fallos de la caché L1 en el caso de *TokenTLB*, mejorando enormemente la precisión de la clasificación.

Intercambio de tokens entre TLBs. Uno de

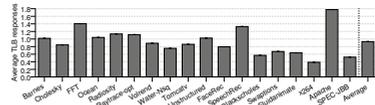


Fig. 4: Proporción de respuestas de TLB tras un fallo en la L2 TLB.

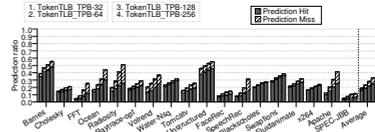


Fig. 5: Ratio de acierto para predicciones de la TPB.

los beneficios clave de TokenTLB sobre mecanismos de clasificación previos es cómo maneja las transferencias entre TLBs, de forma que obtiene sus beneficios al mismo tiempo que limita el número de respuestas requeridas. Como resultado, el tráfico TLB se reduce y se evita el bloqueo del sistema esperando respuestas. La Figura 4 representa el número de respuestas enviadas tras cada fallo de TLB al usar *TokenTLB*. Mientras que los mecanismos de resolución entre TLBs como *SnoopingTLB* requieren $N - 1$ respuestas invariablemente, *TokenTLB* requiere de media solamente 0.93 respuestas por fallo de TLB. Dicha media es menor a uno debido al hecho de que, al usar *TokenTLB*, no se esperan respuestas si los tokens están en la tabla de páginas. En algunos casos, como *Apache* o *SpeechRec* la media supera ámpliamente a una respuesta por fallo. *TokenTLB* permite que exista más de un propietario de la traducción en otras TLBs, ya que los desalojos pueden ser aceptados por la primera TLB válida en el anillo de desalojo, y por tanto en estos casos, dos o más respuestas puede enviarse como consecuencia de un fallo TLB.

Efectividad del predictor de tokens. Esta sección analiza brevemente el Token Predictor Buffer (TPB), que es concebido para evitar recurrir a mensajes de difusión tras fallos de TLB, siempre que haya un potencial propietario conocido. Por tanto, el TPB tiene un impacto sobre el tráfico y el consumo de la red, pero no en el tiempo de ejecución. La TPB es sencillamente un pequeño buffer de 4 vías, evaluado con hasta cuatro tamaños distintos (32, 64, 128 y 256 entradas).

La Figura 5 muestra la proporción de predicciones exitosas y fallidas (es decir, la cantidad de TLBs remotas que son propietarias de tokens -o no- en el momento de ser solicitadas en una petición punto a punto) con respecto al total de fallos de TLB. Se puede observar que al incrementar el tamaño de la TPB tiene un efecto positivo en la precisión de las predicciones. En concreto, una TPB de 256 entradas evita un 24.7% de difusiones de un total de casi 33% de intentos de predicción de media. En algunos casos, como *Barnes* o *Unstructured*, alrededor de un 45% de difusiones de TLB son prevenidas al usar la TPB.

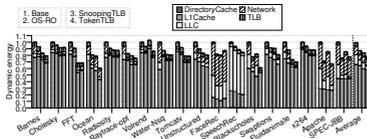


Fig. 10: Consumo dinámico de energía normalizado al sistema base.

en el consumo de energía dinámica de la caché, como se muestra en la Figura 10. *TokenTLB* reduce el consumo global hasta un 21.9% comparado al sistema base, particularmente el consumo de la L1 caché, ya que la presión se reduce a través de la desactivación de la coherencia. El consumo de la red también se reduce significativamente, aunque proporcionalmente al consumo total el impacto de la misma se diluye. En comparación, *TokenTLB* reduce el consumo dinámico en un 5.7% con respecto a *SnoopingTLB* y un 9% con respecto a *OS-R0*.

En resumen, aplicar *TokenTLB* a la desactivación de la coherencia mejora la escalabilidad del sistema. Reduce enormemente la utilización del directorio debido a sus propiedades. Como consecuencia, *TokenTLB* permite estructuras de directorio más pequeñas y eficientes, reduciendo los fallos en la caché de L1, el tráfico de la red, el consumo dinámico y mejora también el tiempo de ejecución sobre cualquier otro mecanismo de clasificación previo.

VII. CONCLUSIONES

Este trabajo propone *TokenTLB*, un mecanismo de clasificación novedoso basado en las TLBs y en la cuenta e intercambio de tokens a través de mensajes entre TLBs, en los que solo las TLBs que son propietarias de la traducción deben contestar. Al usar los tokens para la clasificación en lugar de usarlos para la coherencia se evita la necesidad de peticiones persistentes. Además, la cuenta de tokens es muy eficiente al realizar una clasificación adaptativa en un esquema privado-compartido, detectando de forma natural fases privadas de páginas compartidas durante su tiempo de generación. Finalmente, la dicotomía clásica de la clasificación es extendida al permitir la detección de escritura por vez primera en un mecanismo adaptativo. Al predecir los propietarios de los tokens y enviar mensajes punto a punto se mejora la escalabilidad del sistema. *TokenTLB* presenta todas las características deseables para un esquema de clasificación: ésta es conocida antes del acceso a la caché (*a-priori*), detecta accesos de solo lectura (*read-only*), es *adaptativa* y *precisa*.

AGRADECIMIENTOS

Este trabajo ha sido financiado por MINECO y la Comisión Europea (fondos FEDER) bajo el proyecto TIN2015-66972-C5-1-R/3-R y la *Fundación Séneca, Agencia de Ciencia y Tecnología de la Región de Murcia* bajo el proyecto *Jóvenes Líderes en Investigación* 18956/JLI/13.

REFERENCIAS

- [1] Blas Cuesta, Alberto Ros, María E. Gómez, Antonio Robles, and José Duato, "Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks," in *38th Int'l Symp. on Computer Architecture (ISCA)*, June 2011, pp. 93–103.
- [2] Michael Ferdman, Pejman Lotfi-Kamran, Ken Balet, and Babak Falsafi, "Cuckoo directory: A scalable directory for many-core systems," in *17th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2011, pp. 169–180.
- [3] Daniel Sanchez and Christos Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in *18th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2012, pp. 129–140.
- [4] Jason Zebchuk, Vijayalakshmi Srinivasan, Moinuddin K. Qureshi, and Andreas Moshovos, "A tagless coherence directory," in *42nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2009, pp. 423–434.
- [5] Mohammad Alisafaei, "Spatiotemporal coherence tracking," in *35th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2012, pp. 341–350.
- [6] Blas Cuesta, Alberto Ros, María E. Gómez, Antonio Robles, and José Duato, "Increasing the effectiveness of directory caches by avoiding the tracking of non-coherent memory blocks," *IEEE Transactions on Computers (TC)*, vol. 62, no. 3, pp. 482–495, Mar. 2013.
- [7] Mahdad Davari, Alberto Ros, Erik Hagersten, and Stefanos Kaxiras, "An efficient, self-contained, on-chip, directory: DIR-SISD," in *24th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2015, pp. 317–330.
- [8] Socrates Demetriades and Sangyeun Cho, "Stash directory: A scalable directory for many-core coherence," in *20th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2014, pp. 177–188.
- [9] Albert Esteve, Alberto Ros, María E. Gómez, Antonio Robles, and José Duato, "Efficient tlb-based detection of private pages in chip multiprocessors," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 27, no. 3, pp. 748–761, Mar. 2016.
- [10] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki, "Reactive NUCA: Near-optimal block placement and replication in distributed caches," in *36th Int'l Symp. on Computer Architecture (ISCA)*, June 2009, pp. 184–195.
- [11] Hemayet Hossain, Sandhya Dwarakadas, and Michael C. Huang, "POPS: Coherence protocol optimization for both private and shared data," in *20th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2011, pp. 45–55.
- [12] Daehoon Kim, Jeongseob Ahn, Jaehong Kim, and Jaehyuk Huh, "Subspace snooping: Filtering snoops with operating system support," in *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2010, pp. 111–122.
- [13] Konstantinos Koukos, Alberto Ros, Erik Hagersten, and Stefanos Kaxiras, "Building heterogeneous unified virtual memories (uvms) without the overhead," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 1, pp. 1:1–1:22, Mar. 2016.
- [14] Yong Li, Ahmed Abousamra, Rami Melhem, and Alex K. Jones, "Compiler-assisted data distribution for chip multiprocessors," in *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2010, pp. 501–512.
- [15] Yong Li, Rami Melhem, and Alex K. Jones, "PS-TLB: Leveraging page classification information for fast, scalable and efficient translation for future cmps," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, pp. 28:1–25:21, Jan. 2013.
- [16] Yong Li, Rami G. Melhem, and Alex K. Jones, "Practically private: Enabling high performance cmps through compiler-assisted data classification," in *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2012, pp. 231–240.
- [17] Seth H. Pugsley, Josef B. Spjut, David W. Nellans, and Rejeev Balasubramonian, "SWL: Hardware cache coherence protocols to map shared data onto shared caches," in *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2010, pp. 465–476.
- [18] Alberto Ros, Blas Cuesta, María E. Gómez, Antonio Robles, and José Duato, "Temporal-aware mechanism to

- detect private data in chip multiprocessors,” in *42nd Int'l Conf. on Parallel Processing (ICPP)*, Oct. 2013, pp. 562–571.
- [19] Alberto Ros, Mahdad Davari, and Stefanos Kaxiras, “Hierarchical private/shared classification: the key to simple and efficient coherence for clustered cache hierarchies,” in *21th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2015, pp. 186–197.
- [20] Alberto Ros and Alexandra Jimborean, “A dual-consistency cache coherence protocol,” in *29th Int'l Parallel and Distributed Processing Symp. (IPDPS)*, May 2015, pp. 1119–1128.
- [21] Alberto Ros and Alexandra Jimborean, “A hybrid static-dynamic classification for dual-consistency cache coherence,” *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. PP, no. 99, Feb. 2016.
- [22] Jason Zebchuk, Babak Falsafi, and Andreas Moshovos, “Multi-grain coherence directories,” in *46th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2013, pp. 359–370.
- [23] Milo M.K. Martin, *Token Coherence*, Ph.D. thesis, University of Wisconsin-Madison, Dec. 2003.
- [24] Milo M.K. Martin, Mark D. Hill, and David A. Wood, “Token coherence: Decoupling performance and correctness,” in *30th Int'l Symp. on Computer Architecture (ISCA)*, June 2003, pp. 182–193.
- [25] Michael R. Marty, Jesse D. Bingham, Mark D. Hill, Alan J. Hu, Milo M.K. Martin, and David A. Wood, “Improving multiple-CMP systems using token coherence,” in *11th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2005, pp. 328–339.
- [26] Alberto Ros and Stefanos Kaxiras, “Complexity-effective multicore coherence,” in *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2012, pp. 241–252.
- [27] Bogdan F. Romanescu, Alvin R. Lebeck, Daniel J. Sorin, and Anne Bracy, “UNified instruction/translation/data (UNITD) coherence: One protocol to rule them all,” in *16th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2010, pp. 1–12.
- [28] Shekhar Srikantaiah and Mahmut Kandemir, “Synergistic tlbs for high performance address translation in chip multiprocessors,” in *43rd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2010, pp. 313–324.
- [29] Arun Raghavan, Colin Blundell, and Milo M.K. Martin, “Token tenure: PATCHing token counting using directory-based cache coherence,” in *41th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Nov. 2008, pp. 47–58.
- [30] Milo M.K. Martin, Pacia J. Harper, Daniel J. Sorin, Mark D. Hill, and David A. Wood, “Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors,” in *30th Int'l Symp. on Computer Architecture (ISCA)*, June 2003, pp. 206–217.
- [31] Manuel E. Acacio, José González, José M. García, and José Duato, “Owner prediction for accelerating cache-to-cache transfer misses in cc-NUMA multiprocessors,” in *ACM/IEEE Conf. on Supercomputing (SC)*, Nov. 2002, pp. 1–12.
- [32] Alberto Ros, Manuel E. Acacio, and José M. García, “DiCo-CMP: Efficient cache coherency in tiled CMP architectures,” in *22nd Int'l Parallel and Distributed Processing Symp. (IPDPS)*, Apr. 2008, pp. 1–11.
- [33] Alberto Ros, Manuel E. Acacio, and José M. García, “Dealing with traffic-area trade-off in direct coherence protocols for many-core cmps,” in *8th Int'l Conf. on Advanced Parallel Processing Technologies (APPT)*, Aug. 2009, pp. 11–27.
- [34] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner, “Simics: A full system simulation platform,” *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.
- [35] Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, “Multifacet’s general execution-driven multiprocessor simulator (GEMS) toolset,” *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Sept. 2005.
- [36] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraaj K. Jha, “GARNET: A detailed on-chip network model inside a full-system simulator,” in *IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2009, pp. 33–42.
- [37] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P. Jouppi, “Cacti 5.1,” Tech. Rep. HPL-2008-20, HP Labs, Apr. 2008.
- [38] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” in *22nd Int'l Symp. on Computer Architecture (ISCA)*, June 1995, pp. 24–36.
- [39] Man-Lap Li, Ruchira Sasanka, Sarita V. Adve, Yen-Kuang Chen, and Eric Debes, “The ALPbench benchmark suite for complex multimedia applications,” in *Int'l Symp. on Workload Characterization (IISWC)*, Oct. 2005, pp. 34–45.
- [40] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li, “The PARSEC benchmark suite: Characterization and architectural implications,” in *17th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2008, pp. 72–81.
- [41] Alaa R. Alameldeen, Carl J. Mauer, Min Xu, Pacia J. Harper, Milo M.K. Martin, Daniel J. Sorin, Mark D. Hill, and David A. Wood, “Evaluating non-deterministic multi-threaded commercial workloads,” in *5th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, Feb. 2002, pp. 30–38.

Modelando de forma precisa el subsistema de memoria de una GPU

Francisco Candel, Salvador Petit, Julio Sahuquillo y José Duato¹

Resumen— Actualmente, la investigación relacionada con la arquitectura de las GPU es muy activa debido a que estas arquitecturas ofrecen mucha más potencia computacional por vatio que las arquitecturas CPU. Esta es la principal razón por la que el despliegue masivo de multiprocesadores GPU es considerado una de las soluciones más factibles para alcanzar capacidades de computa *exascale*. En este contexto, la investigación sobre GPUs es necesaria para hacer más fácil su programación y mejorar la integración de arquitecturas CPU y GPU en el mismo *die*. Uno de los campos de investigación más importantes en las GPUs actuales es la jerarquía de memoria, ya que sus objetivos de diseño son muy diferentes a los de la jerarquía de memoria en las CPUs. Para explorar nuevos diseños destinados a mejorar las prestaciones de computo general en las GPUs (computación GPGPU), así como para mejorar el rendimiento de los sistemas heterogéneos CPU/GPU, los investigadores a menudo necesitan usar simuladores que modelen detalladamente las arquitecturas y los subsistemas de memoria. Sin embargo, debido a la alta velocidad a la que estas arquitecturas evolucionan, la precisión de los simuladores del estado del arte se ve afectada negativamente. En este artículo se identifican 3 aspectos principales que deben ser modelados con más precisión: i) el banco de MSHRs, ii) la combinación de los accesos generados por las instrucciones de memoria vectoriales en el propio *pipeline* y iii) la implementación de instrucciones de escritura en memoria no bloqueantes. En este sentido, se ha extendido el simulador heterogéneo de procesadores CPU/GPU MultiSim para modelar estos aspectos con la suficiente precisión. Los resultados experimentales muestran que si estos aspectos no son considerados en el entorno de simulación, la desviación de las prestaciones puede crecer en algunas aplicaciones hasta un 70 %, 75 % y un 60 %, respectivamente.

Palabras clave— GPU, MSHR, sistema de memoria, simulación.

I. INTRODUCCIÓN

RECIENTEMENTE ha habido un continuo aumento en el uso de las GPU (Graphics Processing Units) para computo de propósito general. Esto es debido a que las GPUs ofrecen mayor eficiencia energética que los sistemas tradicionales. Es decir, por la misma cantidad de energía, son capaces de aportar más capacidad de cómputo, especialmente en la ejecución de cargas masivamente paralelas. Debido a este hecho, la mayoría de los supercomputadores entre los 10 más potentes en la lista del top500.org[1] usan GPUs. Por ejemplo, el supercomputador Titan, que ocupaba el segundo lugar en la lista de top500.org en noviembre de 2015, fue construido con GPUs Nvidia K20x; y el primero de la lista, Titanhe-2 hace uso de las tarjetas Intel Xeon Phi, las cuales incorporan una GPU que ocupa gran parte del chip. Sin embargo, la programación para

GPUs es muy diferente de la de los sistemas tradicionales. Para hacer frente a este hecho, los arquitectos de computadores están intentando adaptar diferentes técnicas (por ejemplo, caches y prebúsqueda) que han funcionado exitosamente en CPUs facilitando la programación y también aumentando la potencia de cómputo.

La enorme potencia de cálculo de las GPUs viene proporcionada por el funcionamiento de cientos de elementos de procesamiento en paralelo. Para mantener ocupados todos estos elementos de procesamiento con datos, los accesos a memoria deben ser controlados debidamente. Esto significa que la jerarquía de memoria debe proporcionar mucho más ancho de banda que los sistemas de memoria convencionales de las CPUs multinúcleo. Por otro lado, las aplicaciones de GPU se caracterizan por su masivo paralelismo (miles de hilos lógicos). Basándose en este hecho la jerarquía de memoria en las GPUs no se diseña para reducir la latencia la de las CPUs sino para tolerar un alto número de accesos simultáneos. De esta forma, permite ocultar la mayor parte de la latencia a memoria principal.

La importancia de facilitar la programabilidad de las GPUs para ejecutar computo GPGPU (*General-Purpose computing on Graphics Processing Units*), así como la integración de CPUs y GPUs, las cuales presentan patrones de acceso a memoria distintos, en el mismo chip está impulsando la investigación sobre la jerarquía de memoria de las GPUs en estos momentos. Para explorar y evaluar las nuevas propuestas y mejoras en el subsistema de memoria, los investigadores utilizan complejos entornos de simulación. Estos entornos son abstracciones del hardware y modelan su funcionalidad, centrándose en los componentes hardware que tienen un impacto significativo en el rendimiento del sistema.

En particular, este trabajo mejora la precisión del Multi2sim modelando tres aspectos clave del subsistema de memoria de las GPUs actuales: i) el directorio de MSHRs, ii) combinar los accesos generados por las instrucciones de memoria vectoriales en el propio *pipeline*, iii) instrucciones de escritura en memoria no bloqueantes. El primer mecanismo permite estimar el efecto del directorio MSHR sobre las prestaciones. La segunda extensión combina las peticiones a memoria, emitidas por la misma instrucción vectorial, en el *pipeline* del procesador, el cual proporciona un modelo más realista de los patrones de acceso a memoria principal. Por último, la tercera mejora evita que el *pipeline* se bloquee cuando una instrucción de escritura se encuentra a la cabeza del *buffer* de instrucciones de acceso a memoria vectorial.

¹Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: francanma@inf.upv.es

Los resultados experimentales muestran que: i) modelar el banco MSHR puede reducir las prestaciones hasta 3 veces con respecto a asumir un banco MSHR ilimitado; ii) combinar los accesos en el *pipeline* puede aumentar el tiempo de hasta un 30% en algunas aplicaciones; iii) las escrituras no bloqueantes mejoran el rendimiento en todas las aplicaciones y hasta un 60% en algunos casos. En resumen, no modelar estos mecanismos realistas puede provocar una desviación importante en los resultados obtenidos.

El resto de este trabajo se organiza como sigue. La sección II presenta varios simuladores de GPUs relevantes. La sección III describe la arquitectura *Southern Islands* y su modelo de programación. En la sección V, se describen con detalle las extensiones propuestas. La sección V presenta los resultados experimentales. Por último, en la sección VI se resumen las conclusiones.

II. TRABAJO RELACIONADO

Los simuladores de GPUs son relativamente nuevos y aun están madurando. De hecho, el número de entornos de simulación de GPUs disponibles hoy en día es mucho menor que el de simuladores de CPUs. La razón principal de la falta de herramientas es la poca información que dan los diseñadores de GPUs, así como el hecho de que las arquitecturas de las GPUs modernas están en constante evolución y cambian rápidamente, lo que dificulta el diseño de los simuladores de GPUs, los cuales requieren para su desarrollo un modelo establecido y bien conocido. A pesar de este hecho, debido a la creciente utilización de las GPUs, algunos entornos de simulación han ido apareciendo recientemente. A continuación, se describen un conjunto representativo de ellos.

GPGPU-Sim [2], [3] es actualmente uno de los simuladores GPU más referenciados. Es un simulador detallado ciclo a ciclo que es compatible con la versión de CUDA 3.1. Modela una microarquitectura GPU similar a la Nvidia GeForce 8x, 9x, y la serie Fermi. GPGPU-Sim simula también la red de interconexión entre los núcleos SIMT (Single Instruction Multiple Thread) y los módulos de memoria. Recientemente, la plataforma de simulación por eventos Gem5 [4] se ha combinado con GPGPU-Sim para implementar un simulador heterogéneo completo. Por otra parte, GPGPU-Sim integra desde la versión 3.2.0 un modelo energético basado en McPAT [5] denominado GPUWatch [6]. Sin embargo, debido a su dependencia de los controladores de Nvidia, los cuales sólo soportan OpenCL 1.1, GPGPU-Sim no es apropiado para evaluar aplicaciones escritas en OpenCL como las proporcionadas por AMD [7].

Barra [8] es un simulador funcional de GPUs. Se basa en UNISIM y implementa un emulador del controlador de CUDA, así como, un simulador de la GPU Nvidia Tesla. En este sentido, Barra puede ejecutar directamente programas CUDA y generar estadísticas a nivel de micro-instrucciones. Sin embargo, presenta dos inconvenientes: i) sólo soporta CU-

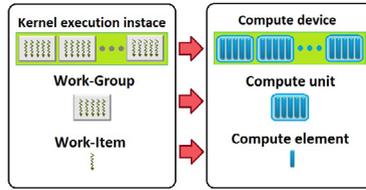


Fig. 1. Relaciones entre el modelo de plataforma y de ejecución de OpenCL.

DA 2.2 (la versión actual de CUDA es la 7.5) y no simula la microarquitectura de la GPU, por lo tanto, no proporciona el soporte para evaluar posibles mejoras en el subsistema de memoria.

Multi2Sim [9], [10] es un simulador de sistemas CPU-GPU heterogéneos con precisión ciclo a ciclo y controlado por eventos. Está disponible tanto la versión estable como la de desarrollo. Modela un subsistema de memoria totalmente configurable con varios niveles de cache y la red de interconexión. Multi2Sim implementa varias arquitecturas de GPU de AMD (Evergreen, Southern-Islands) y Nvidia (Fermi), así como arquitecturas de CPU como X86, MIPS y ARM-32. El equipo de desarrollo de Multi2Sim está modelando actualmente la arquitectura heterogénea HSA [11], donde la CPU y la GPU comparten el mismo espacio de memoria. Por último, Multi2Sim incluye su propia implementación de los controladores de OpenCL y CUDA. De esta manera, puede proporcionar información dinámica sobre la interacción entre CPU y GPU mediante el análisis de las llamadas de OpenCL o CUDA.

Para el desarrollo de este trabajo, se ha elegido Multi2Sim debido a que: i) simula un sistema completo ciclo por ciclo, ii) implementa la microarquitectura GPU de AMD más reciente llamada GCN [12], iii) incluye su propio controlador para OpenCL y CUDA y iv) el soporte para la arquitectura HSA está en fase de desarrollo.

III. LA ARQUITECTURA SOUTHERN ISLANDS Y EL MODELO DE PROGRAMACIÓN OPENCL

Esta sección describe la arquitectura y el modelo de programación de una GPU reciente. Para este fin, se ha seleccionado la GPU de AMD Southern Islands, presentada en 2012 y modelada por Multi2Sim. Esta GPU esta compuesta por varios núcleos de procesamiento que comparten el subsistema de memoria. La arquitectura del núcleo de procesamiento y del subsistema de memoria se describen a continuación. Adicionalmente, la plataforma OpenCL, la cual es usada para programar las aplicaciones GPGPU para las GPUs Southern Islands, también sera introducida.

A. OpenCL

Hay dos plataformas principales para programar aplicaciones GPGPU: CUDA de Nvidia y OpenCL

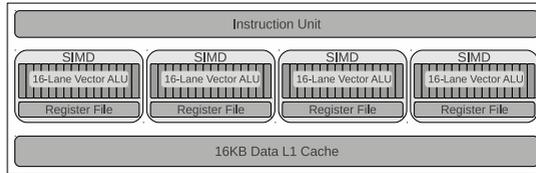


Fig. 2. Microarquitectura Graphics Core Next.

del grupo Khronos. Mientras que CUDA sólo es soportada por GPUs diseñadas por Nvidia, distintas marcas, como Intel, AMD, ARM, o incluso la misma Nvidia, implementan el estándar OpenCL.

El modelo de plataforma de OpenCL define los conceptos *Compute Device*, *Compute Unit* (CU), y *Processing Element* (PE) para hacer referencia a la GPU como tal, a un núcleo individual dentro de la GPU, y por último al nodo de cómputo donde cada hilo es procesado, respectivamente, como muestra la figura 1.

El modelo de ejecución de OpenCL define varios niveles para organizar los hilos de ejecución. Cada hilo individual se define como un *work-item*. Los *work-items* se agrupan en *work-groups*. Una aplicación GPU (comúnmente conocida como *kernel*) se compone de varios *work-groups*. La figura 1 representa la relación entre los modelos de ejecución y plataforma de OpenCL.

B. Microarquitectura Graphics Core Next

Una GPU Southern Islands puede tener hasta 32 CUs que implementan la microarquitectura de AMD Graphics Core Next (GCN). Como se puede apreciar en la figura 2, una CU dispone de 4 unidades vectoriales *Single Instruction Multiple Data* (SIMD), cada una con capacidad de realizar hasta 16 operaciones en paralelo. Una instrucción vectorial lanza 64 *work-items* (agrupación denominada *wavefront*) que se ejecutan en el mismo SIMD. Para ello, el *wavefront* se divide en 4 *subwavefronts* de 16 *work-items*, que tardan 1 ciclo en procesarse cada uno en el SIMD. Por tanto, para procesar una instrucción completa en un SIMD se necesitan cada 4 ciclos. Adicionalmente, cada CU incluye una unidad para el procesamiento de datos escalares y varias unidades de lectura/escritura.

C. Subsistema de memoria

En las GPUs Southern Islands, una instrucción de acceso a memoria puede generar hasta 64 peticiones en un *wavefront*. Para reducir el número de accesos al subsistema de memoria, el mecanismo de *coalesce* combina en el mismo acceso las peticiones de lectura que hacen referencia al mismo bloque de memoria.

Las peticiones de escritura no se combinan de la misma forma que las lecturas, sino en colas de acceso a los distintos módulos del subsistema de memoria. Esta distinción entre las lecturas y las escrituras

es específica de la arquitectura Southern Islands. Otras arquitecturas (p.e. la arquitectura *Evergreen* de AMD [13]) no presentan este comportamiento.

Una vez que las peticiones a memoria han sido combinadas, se emiten los accesos resultantes a una caché de 16KB (ver figura 2), la cual representa el primer nivel de la jerarquía de memoria. Solo aquellos accesos que fallen en la caché L1 acceden a la caché L2 a través de una *crossbar*. Adicionalmente, cada CU tiene disponibles 64 KB de memoria local.

La caché de L2 está dividida en bancos con una organización de direcciones entrelazada (el grano de entrelazamiento es de 4 bloques de caché). Los bancos se conectan a los módulos de memoria principal a través de controladores de memoria de doble canal, siendo cada canal de 64 bits. Las GPUs Southern Islands pueden incluir hasta 6 controladores de memoria. Así, una GPU puede incluir hasta 12 módulos DRAM.

IV. EXTENSIONES PROPUESTAS PARA MULTI2SIM

El modelo del subsistema de memoria de la GPU de Multi2Sim es común al de la CPU, lo que facilita el modelado de procesadores heterogéneos CPU-GPU y permite una implementación más genérica. Sin embargo, también es una de las razones por la que algunos aspectos particulares de las GPUs no se modelan con suficiente detalle o de forma más precisa (p.e. combinar las peticiones de lectura en el *pipeline* en vez de hacerlo en las colas de acceso a los módulos de la jerarquía de memoria). Como se muestra en la sección V, estas variaciones incurren en un impacto significativo positivo o negativo en el rendimiento del sistema.

Con el fin de mejorar la precisión del modelado de la GPU en Multi2Sim, se han implementado 3 extensiones, detalladas a continuación: i) el banco de MSHRs, ii) la combinación de los accesos generados por las instrucciones de memoria en el propio *pipeline* y iii) instrucciones de escritura en memoria no bloqueantes.

A. Modelo del banco de MSHRs

Las GPUs generan una inmensa cantidad de accesos a memoria, pero en un momento dado el subsistema de memoria solo soporta un número limitado de peticiones en vuelo. Por esta razón, las actuales caches no bloqueantes implementan bancos de *Miss Status Holding Registers* (MSHRs). Tras un fallo de

cache, el banco de MSHRs se examina para comprobar si el bloque que se está pidiendo a la cache ya está siendo accedido por otro acceso. En ese caso, el acceso a memoria se encola en un MSHR asociado al bloque accedido.

Hay que tener en cuenta que un MSHR dado está a cargo del seguimiento de todos los accesos a memoria a un bloque de cache concreto (es decir, todos los accesos cuya dirección está dentro del mismo bloque de memoria). Por lo tanto, el número máximo de accesos a memoria pendientes está limitado por el número de MSHRs. En consecuencia, si todos los MSHRs están ocupados y el acceso que ha fallado requiere un nuevo bloque de cache que no está siendo ya accedido, el acceso se detiene hasta que se libera un MSHR.

A.1 Modelo de MSHRs de Multi2Sim

En Multi2Sim, se pueden distinguir dos partes principales en cualquier modelo de CPU o GPU: el pipeline del procesador y el subsistema de memoria. El pipeline modela el hardware más relacionado con las etapas de la ruta de datos excepto las cache de primer nivel. Todas las caches se modelan en el subsistema de memoria, que comprende además la memoria principal. En este contexto, una petición de acceso a memoria se introduce en el subsistema de memoria tan pronto como es emitida por el pipeline para acceder el primer nivel de la jerarquía de memoria.

Multi2Sim sólo modela el banco de MSHRs en el pipeline de la CPU, pero no en el de la GPU. Además, el modelo sólo considera los fallos de cache de primer nivel. Por tanto, en el modelo de GPU de Multi2Sim, el número de fallos en vuelo gestionados por cualquier cache L2 es ilimitado. Como la sección V mostrará, esta implementación es causa de importantes desviaciones en las prestaciones.

A.2 Extensión del modelo del MSHRs

Se propone desacoplar el modelado de los MSHRs del pipeline del procesador y asociar un banco de MSHRs a cada cache del subsistema de memoria. Esta implementación ofrece una simulación más precisa tanto para procesadores CPU como para GPU, ya que comparten el código fuente que modela el subsistema de memoria. Nuestra implementación permite que los bancos de MSHRs de distintas caches tengan diferentes números de entradas, imitando la implementación real.

Nuestra implementación funciona del siguiente modo. Cuando un acceso a una cache dada se resuelve como un fallo, el banco de MSHRs es accedido. Si se encuentra un MSHR asociado al mismo bloque, el acceso se encola en este. En caso contrario, se ocupa un MSHR libre y el acceso se propaga a L2. Si la copia del bloque solicitado en L2 está involucrada en otras operaciones (p.e. está siendo reemplazada), entonces se devuelve un mensaje *nack* a la cache de L1, la cual repetirá el acceso más tarde. Cuando se da esta situación, para hacer un uso eficiente del banco

de MSHRs, la entrada MSHR es liberada y el acceso se registra en una cola especial para ser reintentado posteriormente. Cuando finalmente el bloque accedido es almacenado en la cache de L1, el MSHR asociado a su acceso es liberado y todos los accesos encolados en él son satisfechos.

Cuando hay un fallo en la cache de L2, el mecanismo descrito es aplicado de forma recursiva si procede en los niveles inferiores de la jerarquía de memoria (p.e. L3). Finalmente, si no hay MSHRs disponibles cuando se solicitan, los accesos esperan en la cola de acceso al módulo de cache correspondiente, desde donde se relanzarán por orden de llegada según se vayan liberando MSHRs.

B. Mecanismos para combinar peticiones a memoria

La agrupación de los hilos de ejecución en wavefronts ayuda a mejorar las prestaciones del sistema de memoria. Por ejemplo, en la arquitectura GCN, uno de los principales factores que limitan la cantidad de accesos en vuelo a memoria es el tamaño del *Vector Memory Instruction Buffer* (VMB) que se encuentra en la unidad de acceso a memoria vectorial (VMU) de cada CU. Cada instrucción de acceso a memoria vectorial puede generar hasta 64 peticiones al subsistema de memoria. Las instrucciones de acceso a memoria vectorial se almacenan en el VMB hasta que todas sus peticiones al subsistema de memoria hayan finalizado. Asumiendo que el VMB tiene 32 entradas (este es el valor por defecto usado en nuestro experimentos debido a la ausencia de información disponible), puede haber hasta 2048 (32×64) peticiones a memoria en vuelo en un momento dado por cada CU. Esta situación causa que el subsistema de memoria sea un importante cuello de botella en las prestaciones de las GPUs.

Para reducir la carga del sistema de memoria, las arquitecturas GPUs agrupan las peticiones a memoria del mismo tipo (lecturas y escrituras) que acceden a la mismo bloque de cache en un solo acceso. Así, se reduce el número efectivo de accesos a memoria y consecuentemente la carga de la jerarquía de memoria.

B.1 Coalescing y merging

Dos enfoques principales, o una combinación de ellos, son usados en las GPUs actuales para reducir la cantidad de accesos a memoria: *coalescing* y *merging*. El enfoque *coalescing* implementa la lógica necesaria para combinar múltiples peticiones a memoria pertenecientes a la misma instrucción de acceso a memoria en un solo acceso a la cache. Esta lógica actúa en la unidad de acceso a memoria vectorial justo antes de que el acceso se envíe al subsistema de memoria, por lo que está sincronizada con la ejecución de las instrucciones.

Por otra parte, el enfoque *merging* se implementa en las colas de acceso a los módulos del subsistema de memoria, desvinculado de la VMU. A diferencia del enfoque anterior, las peticiones de memoria de la misma instrucción vectorial pueden llegar al subsiste-

ma de memoria en diferentes instantes. Por ejemplo, en la arquitectura GCN cada subwavefront emite sus peticiones a memoria en diferentes ciclos. Por lo tanto, las peticiones de una misma instrucción pueden generar distintos accesos, incluso si estas peticiones acceden al mismo bloque.

Las diferentes GPU comerciales implementan uno de estos enfoques o una combinación de ambos. AMD Evergreen soporta coalescing tanto para lecturas como para escrituras [14], [13]. Sin embargo, en la arquitectura Southern Islands, las lecturas siguen un enfoque coalescing mientras que las escrituras siguen un enfoque merging.

B.2 Combinación de peticiones a memoria en Multi2Sim

Multi2Sim implementa un enfoque merging genérico para los accesos a la L1 tanto de la CPU como de la GPU. Este modelo puede combinar múltiples accesos independientemente de la cantidad y de si han sido producidos por la misma instrucción o no, aunque se le aplican algunas restricciones para cumplir con el protocolo de coherencia y con el modelo de consistencia. Sin embargo, el enfoque coalescing no está implementado en Multi2Sim. Como se muestra en la sección V, usar uno u otro enfoque puede dar lugar a diferencias significativas en las prestaciones.

B.3 Extensión del enfoque coalescing

Se ha extendido el Multi2Sim con una implementación mixta con coalescing y merging que permite usar cualquier enfoque o una combinación de ambos. Además de las características de la implementación original, la extensión propuesta puede detectar si los accesos han sido emitidos por la misma instrucción para de esta forma saber si pueden ser combinados o no.

C. Escrituras no bloqueantes

Como se explicó anteriormente, una instrucción de memoria vectorial es almacenada en el VMB hasta que todos sus accesos a memoria han finalizado. Además, debido al diseño en orden del pipeline de las CUs, las instrucciones de memoria vectoriales liberan las entradas del VMB en orden de programa. Sin embargo, las implementaciones comerciales pueden optimizar este comportamiento permitiendo a las instrucciones de escritura liberar su entrada tan pronto como sus accesos a memoria haya sido emitidos, siempre que las escrituras previas, siguiendo el orden de programa, ya hayan emitido sus escrituras. Esta optimización se puede realizar en las GPUs debido al modelo de consistencia de memoria relajado soportado por OpenCL.

C.1 Modelo de consistencia de memoria de OpenCL

Esta sección resume el modelo de consistencia de memoria relajado de OpenCL para analizar las restricciones que impone al hardware real. Téngase en cuenta que los autores de este artículo no han encontrado ninguna publicación que detalle el soporte

hardware al modelo de consistencia de memoria de OpenCL.

El modelo de consistencia de memoria de OpenCL está organizado jerárquicamente distinguiendo entre un work-item, varios work-items en el mismo work-group, y entre work-groups de la siguiente manera [15]:

1. Dentro de un work-item (es decir, hilo de ejecución) dos lecturas y escrituras a la misma dirección no pueden ser reordenadas.
2. Para diferentes work-items que pertenecen al mismo work-group, la consistencia de memoria se garantiza solo mediante barreras.
3. La consistencia no se garantiza entre diferentes work-groups.

Para garantizar la primera condición, no hay necesidad de forzar que las escrituras a la misma dirección finalicen en orden sino que sean emitidas en orden al subsistema de memoria. La razón de esto es que el controlador de la cache procesa las peticiones al mismo bloque en orden. Esto significa que una escritura solo puede ser procesada cuando los accesos al mismo bloque hayan finalizado.

En cuanto a la segunda y tercera condición, no imponen un orden determinado para las escrituras de distintos work-items entre barreras de sincronización, sin tener en cuenta si están en el mismo o diferentes work-groups. Así, las escrituras de diferentes work-groups que se ejecuten concurrentemente en la misma CU no necesitan seguir un orden de escritura concreto.

C.2 Implementación bloqueante de las escrituras en Multi2Sim

Multi2Sim, en su implementación original, no permite liberar la entrada asociada a una instrucción de escritura en el VMB hasta que todos sus accesos correspondientes finalicen. Como se explicó anteriormente, este comportamiento, el cual puede ser necesario para soportar otros modelos de consistencia más estrictos (como los de la mayoría de CPU comerciales), es demasiado restrictivo para las GPUs. Como se muestra en la sección V esta implementación puede ocasionar un impacto importante en las prestaciones.

C.3 Extensión de escrituras no bloqueantes

La ampliación propuesta se basa en las observaciones del análisis expuesto anteriormente. Esta extensión acelera la ejecución mediante la liberación anticipada de entradas del VMB ocupadas por instrucciones de escritura. Más concretamente, las instrucciones de escritura liberan sus entradas tan pronto como sus peticiones son emitidas al subsistema de memoria. Sin embargo, se asegura que las escrituras pertenecientes al mismo work-item no son reordenadas.

TABLA I
 JERARQUÍA DE MEMORIA Y CONFIGURACIÓN DE LA GPU.

GPU AMD HD 7770	
GCN Configuration	
Compute Units	10
Work-groups per CU	10
Wavefronts per Work-group	4
Work-items per Wavefront	64
LDS Unit	64 KB, 1 cycle, 32 ports
SIMD Unit	4 per CU, 16 lines, 4 cycles per instruction
Scalar Unit	1 per CU, 1 cycle per instruction
Vector Memory Unit	1 per CU, VMB of 32 entries
Cache Hierarchy	
All Caches	LRU, 64B line, 2 ports, directory latency 1 cycles
L1 Scalar Cache	3 caches (shared by 4, 3, and 3 CUs) 16KB, 4 way, 1 cycle
L1 Texture Cache	1 per CU 1 per CU, 16KB, 4 way, 1 cycles
L2 Cache	2 modules each module is 128KB, 16 way, 10 cycles
Main Memory	2 channels per L2 module, 100 cycles

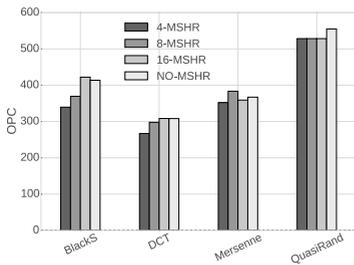


Fig. 3. Impacto del número de MSHRs sobre el OPC ($OPC \geq 200$).

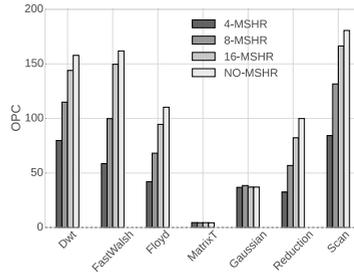


Fig. 4. Impacto del número de MSHRs sobre el OPC ($OPC < 200$).

V. RESULTADOS EXPERIMENTALES

En esta sección se evalúan las extensiones propuestas y se analiza su impacto en las prestaciones del sistema. Los experimentos se han realizado con el simulador Multi2Sim version 4.2 con y sin las extensiones propuestas.

Para obtener los resultados, se ha modelado la arquitectura de GPU Southern Islands. La tabla I resume los principales parámetros de la arquitectura simulada.

Los resultados se han obtenido a partir de los benchmarks de AMD OpenCL SDK 2.5 adaptados para Multi2Sim [16]. Estos benchmarks son un subconjunto de los incluidos en el APP-SDK (*Application Parallel Programming — Software Development Kit*) de AMD.

Las prestaciones han sido evaluadas y comparadas en función de las Operaciones Por Ciclo (OPC) alcanzadas. Esta métrica cuenta el número de operaciones escalares realizadas por cada instrucción du-

rante la ejecución. Por ejemplo, si una instrucción vectorial realiza 64 operaciones escalares individuales esta métrica contará 64 en vez de 1 como haría el IPC.

A. Variación del tamaño de banco de MSHRs

Esta sección investiga el impacto del número de MSHRs sobre el rendimiento del procesador. Se han lanzado experimentos variando el número de MSHRs (4, 8, 16) y se han comparado con la configuración base donde no se modela el banco (es decir, la configuración base soporta un número virtualmente infinito de fallos de cache).

No se ha encontrado información sobre el tamaño de los bancos de MSHRs implementados en las GPUs comerciales. Por lo tanto, se han utilizado los valores obtenidos en [17]. En este trabajo los autores calculan mediante la realización de un test que cada CU tiene un banco de MSHRs de 6 entradas, similar a procesadores como el Pentium 4, el cual implementa

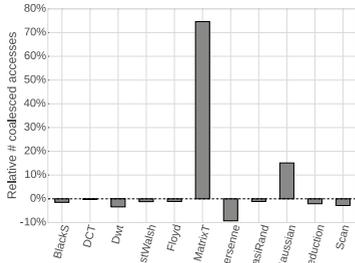


Fig. 5. Número relativo de accesos combinados por coalescing con respecto a merging.

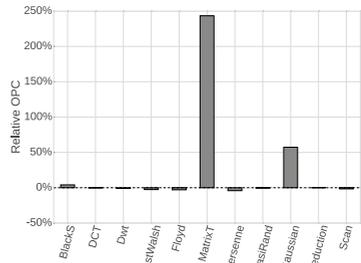


Fig. 6. Incremento del OPC relativo de coalescing con respecto a merging.

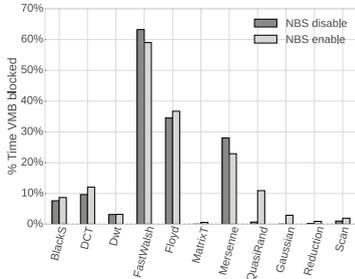


Fig. 7. Impacto de las escrituras no bloqueantes sobre el tiempo de bloqueo del VMB.

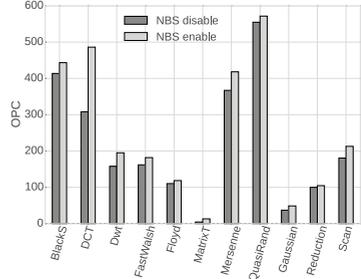


Fig. 8. Impacto de las escrituras no bloqueantes sobre el OPC.

3 entradas en su LI de datos [18].

Las figuras 3 y 4 representan el rendimiento medido en OPC para los benchmarks con un OPC bajo (≥ 200) y alto (< 200), respectivamente. Como se observa, el tamaño del banco de MSHRs tiene una gran influencia en las prestaciones en la mayoría de benchmarks, aunque el impacto (en porcentaje) es más alto en las aplicaciones con bajo OPC. Mientras que los benchmarks con alto OPC la diferencia mas grande es Blacks (20%), en las que tienen un OPC bajo no limitar el número de fallos en vuelo (NO-MSHR) puede aumentar el OPC más de 3 veces en algunos casos

La lógica detrás de estos resultados es que un alto OPC significa que el procesador es capaz de obtener un alto paralelismo a nivel de operación en las unidades SIMD, y que el subsistema de memoria no es el cuello de botella más importante. Por lo tanto, limitar la cantidad de peticiones en vuelo a un cantidad relativamente baja (p.e. 8 o 16) tendrá un ligero impacto en las prestaciones. Incluso bajando este valor hasta 4 tiene un impacto escaso en la mayor parte de los benchmarks con OPC alto. De forma análoga, un OPC bajo significa que el procesador no es capaz de extraer un buen paralelismo a nivel de

operación. Por lo tanto, es probable que el subsistema de memoria esté estrangulando las prestaciones. Como consecuencia, si el número de fallos en vuelo permitido se reduce las prestaciones del sistema se reducirán de forma dramática.

B. Impacto de coalescing con respecto a merging

Las figuras 5 y 6 muestran la cantidad relativa de accesos a la cache que se han combinado y el OPC relativo, respectivamente, del enfoque coalescing sobre merging. Se puede observar que el numero de accesos a la cache es muy similar en 9 de los 12 benchmarks. Sin embargo, aparecen diferencias importante entre ambos enfoques en algunos benchmarks que crecen hasta un 15% para Gaussian y 75% para MatrixT. Estas diferencias se convierten en importantes cambios en las prestaciones, que crecen 3,4 y 1,6 veces, respectivamente.

C. Impacto de las escrituras no bloqueantes

La figura 7 muestra el porcentaje de tiempo que el VMB esta bloqueado en la configuracion base (NBS disabled) y cuando aplicamos el mecanismo de escrituras no bloqueantes (NBS). Como se observa, el VMB se bloquea durante mas tiempo si activamos

el mecanismo *NBS*, en la mayoría de aplicaciones. En una primera aproximación esto podría parecer contradictorio ya que las escrituras no bloqueantes permiten liberar entradas del VMB antes. Sin embargo, la liberación temprana de entradas del VMB puede desbloquear los work-groups que están esperando que sus instrucciones de escritura lleguen a la primera posición del VMB para finalizar, permitiéndoles: i) continuar emitiendo nuevas instrucciones de memoria; ii) finalizar su ejecución de manera que un nuevo work-group podrá empezar a ejecutarse y a emitir nuevas instrucciones de acceso a memoria. En ambos casos, el VMB se bloqueará de nuevo debido a las nuevas instrucciones que requieren entrada en el VMB.

La figura 8 muestra el OPC conseguido para ambas configuraciones. Se puede apreciar que activar el mecanismo *NBS* mejora las prestaciones en todas las aplicaciones estudiadas. En algunas de ellas como *Mersenne*, el OPC crece al rededor de un 10% pero en otras como *DCT* el rendimiento del procesador crece hasta un 60%. Esto es debido a que el mecanismo *NBS* aumenta significativamente el paralelismo de memoria en estas aplicaciones.

VI. CONCLUSIONES

Este artículo ha presentado tres extensiones para el simulador CPU/GPU heterogéneo Multi2Sim. Estas extensiones mejora la precisión del modelo de la jerarquía de memoria de la GPU, el cual es actualmente un campo de investigación muy activo dentro del diseño de las GPU. La primera extensión modela el banco de MSHRs de las caches, el cual presenta un impacto mayor en las GPUs del que tiene en las CPUs debido al alto paralelismo de memoria que requieren las primeras. La segunda extensión modela de forma más precisa cómo se combinan los accesos a memoria en las GPUs. Finalmente, la tercera extensión aumenta el rendimiento de las escrituras, evitando que las instrucciones de escritura obstruyan el *pipeline* mientras se espera que finalicen.

Los resultados experimentales muestran que: i) modelar el banco de MSHRs puede reducir las prestaciones hasta 3 veces con respecto a asumir una cantidad de MSHRs infinita; ii) combinar las peticiones de memoria en el *pipeline* puede disminuir el tiempo de ejecución más de un 30% en algunas aplicaciones; iii) las escrituras no bloqueantes mejoran las prestaciones en todas las aplicaciones estudiadas y hasta un 60% en algunos casos. En resumen, no modelar estos mecanismos hardware realistas puede dar como resultado importantes desviaciones en el rendimiento.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad (MINECO) y por los fondos del Plan E bajo la subvención TIN2015-66972-C5-1-R, por el programa Intel Early Career Faculty Honor Program Award, y por la Generalitat Valenciana bajo la subvención AICO/2016/059.

REFERENCIAS

- [1] "Top500 Supercomputer Sites." <http://top500.org>.
- [2] W.W.L. Fung, I. Sham, G. Yuan, and T.M. Aamodt, "Dynamic warp formation and scheduling for efficient gpu control flow." in *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, Dec 2007, pp. 407-420.
- [3] A. Bakthoda, G.L. Yuan, W.W.L. Fung, H. Wong, and T.M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator." in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, April 2009, pp. 163-174.
- [4] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoabi, Nilay Vaish, Mark D. Hill, and David A. Wood, "The gem5 simulator." *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1-7, aug 2011.
- [5] Sheng Li, Jung Ho Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multi-core and manycore architectures." in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, Dec 2009, pp. 469-480.
- [6] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi, "Gpuwatch: Enabling energy optimizations in gpgpus." *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 487-498, June 2013.
- [7] "AMD Accelerated Parallel Processing (APP) Software Development Kit (SDK)" <http://developer.amd.com/sdks/amdappsdk/>.
- [8] S. Collange, M. Daumas, D. Defour, and D. Farello, "Barra: A parallel functional simulator for gpgpu." in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, Aug 2010, pp. 351-360.
- [9] Rafael Ubal, Julio Sahuquillo, Salvador Petit, and Pedro Lopez, "Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors." in *Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on*, oct. 2007, pp. 62-68.
- [10] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli, "Multi2sim: A simulation framework for cpu-gpu computing." in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, New York, NY, USA, 2012, PACT '12, pp. 335-344, ACM.
- [11] "Heterogeneous System Architecture foundation" <http://www.hsafoundation.com/standards/>.
- [12] AMD Radeon Graphics Technology, "AMD Graphics Cores Next (GCN) Architecture White Paper," June 2012.
- [13] Aaftab Munshi, Benedict Gaster, Timothy G. Mattson, James Fung, and Dan Ginsburg, *OpenCL Programming Guide*, Addison-Wesley Professional, 2.7 edition, 2013.
- [14] "Evergreen Family Instruction Set Architecture" http://developer.amd.com/wordpress/media/2012/10/AMD_Evergreen-Family_Instruction_Set_Architecture.pdf.
- [15] Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, and Dana Schaa, *Heterogeneous Computing with OpenCL*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2011.
- [16] "AMD Accelerated Parallel Processing (APP) Software Development Kit (SDK)".
- [17] C. Nugteren, G.-J. van den Braak, H. Corporaal, and H. Bal, "A detailed gpu cache model based on reuse distance theory." in *High Performance Computer Architecture (HPCA), 2014. IEEE 20th International Symposium on*, Feb 2014, pp. 37-48.
- [18] D. Boggs, A. Baktha, J. Hawkins, D. Marr, J. A. Miller, P. Roussel, R. Singhal, B. Toll, and K. Venkatraman, "The microarchitecture of the intel pentium 4 processor on 90nm technology." *Intel Technology Journal*, vol. 8, no. 1, February 2004.

Gestión de los Reemplazos de Bloques Limpios en Protocolos de Coherencia Basados en Directorio

Ricardo Fernández-Pascual, Alberto Ros y Manuel E. Acacio¹

Resumen— Asegurar el mantenimiento de la coherencia de las cachés en arquitecturas con cientos o incluso miles de núcleos no es tarea sencilla. De entre las diversas alternativas propuestas hasta la fecha, los protocolos basados en directorio representan la solución más escalable, y gracias al conocimiento que sobre estos protocolos se ha ido acumulando durante los últimos cuarenta años, a día de hoy existe un consenso general sobre el impacto que la mayoría de sus aspectos de diseño tienen sobre el rendimiento, el consumo de energía y el coste. Sin embargo, a día de hoy existe aún un aspecto de diseño sutil pero importante, para el cual no hay unanimidad en los trabajos de investigación recientes. Más concretamente, mientras que algunos trabajos asumen una política de reemplazos *silenciosos* para bloques limpios expulsados del último nivel de cachés privadas, otros implementan para estos casos justo lo contrario, una política de reemplazos *ruidosos*, e incluso algunos otros trabajos ni siquiera mencionan la manera en la que se gestionan los reemplazos de bloques limpios. En este trabajo demostramos que este aspecto puede tener una influencia significativa sobre el rendimiento y el consumo de energía de un protocolo de coherencia basado en directorio. El utilizar la política de reemplazos *silenciosos* puede ahorrar una gran cantidad de tráfico de red (más de un 25% en varios casos, 9,6% de media) en comparación con el uso de la política de reemplazos *ruidosos*. Dado que en las arquitecturas multinúcleo con un gran número de núcleos que están por venir se espera que la red de interconexión dentro del chip represente una fracción importante del consumo de energía total, la utilización de la política de reemplazos *silenciosos* para bloques limpios implicará ahorros en consumo de energía no despreciables. Sin embargo, lo que es más importante es que hemos comprobado que en función de cómo se organice la estructura de directorio, el uso de la política de reemplazos *silenciosos* podría afectar o no al rendimiento, lo que indica que la asunción de la política de reemplazos *ruidosos* no está justificada siempre, ya que incrementarían de forma innecesaria el tráfico de red sin que ello supusiese ninguna ventaja en cuanto a rendimiento se refiere.

Palabras clave— Arquitecturas multinúcleo, protocolo de coherencia de cachés, reemplazos, bloques limpios, directorio, rendimiento, tráfico.

I. INTRODUCCIÓN

La práctica totalidad de las arquitecturas multinúcleo de propósito general actuales ofrecen como paradigma de programación de bajo nivel la abstracción de memoria compartida. El paradigma de memoria compartida se ha popularizado gracias a su cercanía con la programación secuencial, la facilidad de mapear determinados tipos de algoritmos y la amplia disponibilidad de aplicaciones basadas en POSIX Threads y OpenMP. Además, ha sido la

abstracción usada tradicionalmente en la construcción de los sistemas operativos. De esta forma, a menos que suceda un cambio radical (algo realmente improbable), las arquitecturas multinúcleo que están por venir seguirán empleando el modelo de memoria compartida e implementarán a nivel hardware un mecanismo que asegure la coherencia de los niveles privados de caché de cada núcleo [1]. De esta forma, comunicación y sincronización (esta última implementada normalmente usando posiciones de la memoria compartida) ocurrirán bajo el control del protocolo de coherencia de cachés.

La mejor manera de asegurar la coherencia de las cachés privadas cuando el número de núcleos de procesamiento es grande es a través de un protocolo de coherencia de cachés basado en directorio. En estas soluciones se emplea una estructura de directorio distribuida que almacena información sobre qué cachés privadas mantienen copia de qué bloques de datos. De esta forma, los fallos en la cada caché privada de último nivel son enviados al módulo de directorio correspondiente, el cual estará encargado de realizar las acciones de coherencia oportunas (por ejemplo, la invalidación de todas las copias compartidas del bloque de datos en un fallo de escritura).

Desde que fueron propuestos por primera vez a finales de la década de los 70 [2], los protocolos de coherencia basados en directorio vienen siendo empleados como una solución escalable al problema de la coherencia de cachés tanto en el ámbito académico como diseños comerciales [3], [4]. De esta manera, existe a día de hoy un conocimiento profundo sobre el impacto que la mayoría de sus aspectos de diseño tienen sobre el rendimiento, el consumo de energía y el coste. Sin embargo, a pesar del conocimiento acumulando durante los últimos cuarenta años, existe aún un aspecto de diseño sutil pero importante, para el cual parece no haber unanimidad en los trabajos de investigación recientes y que tiene que ver sobre cómo se han de manejar los reemplazos de los bloques que no han sido modificados localmente (bloques limpios)¹. Más concretamente, mientras que algunos trabajos asumen una política de reemplazos *silenciosos* para bloques limpios expulsados del último nivel de cachés privadas [5], [6], [7], [8],

¹A partir de una revisión que hemos realizado de la mayoría de los artículos sobre coherencia de caché aparecidos en las cinco últimas ediciones de las conferencias ISCA, HPCA, PACT and MICRO, hemos encontrado que de 36 artículos, la política de reemplazos *ruidosos* se asume en 14, la de reemplazos *silenciosos* en 8, mientras que en otros 14 artículos no se mencionaba cómo se trataban este tipo de reemplazos.

¹Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: {rfernandez, ar, maecacio}@ditec.um.es

otros implementan para estos casos justo lo contrario, una política de reemplazos *ruidosos* [9], [10], [11], [12], [13], e incluso algunos otros trabajos ni siquiera mencionan la manera en la que se gestionan los reemplazos de bloques limpios [14], [15].

En este trabajo demostramos que este aspecto puede tener una influencia significativa sobre el rendimiento y el consumo de energía de un protocolo de coherencia basado en directorio. El utilizar la política de reemplazos *silenciosos* puede ahorrar una gran cantidad de tráfico de red (más de un 25% en varios casos, 9.6% de media) en comparación con el uso de la política de reemplazos *ruidosos*. Dado que en las arquitecturas multinúcleo con un gran número de núcleos que están por venir se espera que la red de interconexión dentro del chip represente una fracción muy importante del consumo de energía total [16], [17], la utilización de la política de reemplazos *silenciosos* para bloques limpios implicará ahorros en consumo de energía significativos. Sin embargo, lo que es más importante es que hemos comprobado que en función de cómo se organice la estructura de directorio, el uso de la política de reemplazos *silenciosos* podría afectar o no al rendimiento, lo que indica que la asunción de la política de reemplazos *sucios* no está justificada siempre, ya que incrementaría de forma innecesaria el tráfico de red sin que ello supusiese ninguna ventaja en cuanto a rendimiento se refiere. Para ahondar en este último aspecto, en este trabajo mostramos dos casos de uso en los que en cada uno es mejor aplicar una política sobre la contraria.

El resto del documento se organiza como sigue. Empezamos discutiendo en la sección II las ventajas e inconvenientes de cada una de las políticas de gestión de reemplazos de bloques limpios comentadas con anterioridad. Después, en la sección III, mostramos el entorno de evaluación que hemos utilizado y los casos de uso evaluados, y a continuación, en la sección IV, proporcionamos resultados detallados en términos de tiempo de ejecución y tráfico de red, demostrando la importancia que tiene el manejo adecuado de los reemplazos de bloques limpios. Por último, en la sección V resumimos las principales conclusiones de este trabajo.

II. REEMPLAZOS DE BLOQUES LIMPIOS: ¿SILENCIOSOS O RUIDOSOS?

Cada vez que se produce un fallo de caché, y una vez resuelto el fallo, el bloque solicitado es llevado a la caché. Si no hay espacio en caché para dicho bloque, otro bloque tiene que ser reemplazado de la caché. Los reemplazos, por tanto, son tan frecuentes como los fallos de cache (asumiendo que la caché esté llena).

El bloque que se reemplaza puede haber sido modificado localmente (bloque *sucio*) o no (bloque *limpio*). Ante un reemplazo de un bloque sucio, el siguiente nivel de caché debe ser actualizado. Sin embargo, ante un reemplazo de un bloque limpio no hay que actualizar nada, y por tanto se puede realizar de forma *silenciosa*. No obstante, en sistemas multipro-

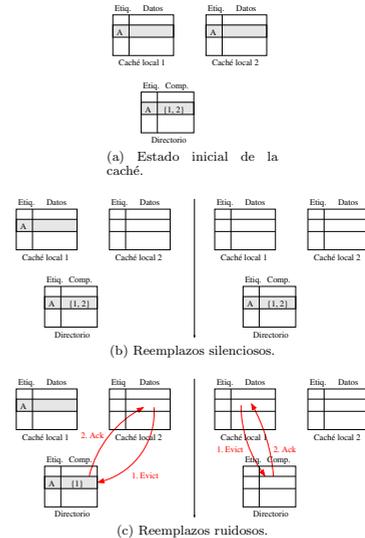


Fig. 1: Ejemplos de reemplazos.

cesador hay veces que es conveniente informar del reemplazo de bloques limpios para actualizar la información de directorio. En este artículo llamaremos a estos reemplazos *ruidosos*.

Las siguientes secciones detallan las ventajas e inconvenientes del uso de reemplazos silenciosos y reemplazos ruidosos y analizan cuándo es mejor utilizar cada uno de ellos.

A. Ejemplos de reemplazos silenciosos y ruidosos

La Fig. 1 muestra el comportamiento de los reemplazos silenciosos y ruidosos. En los ejemplos se muestran tan solo dos cachés locales y uno de los bancos de directorio, que guarda información acerca de los bloques cacheados. La situación inicial de la caché es la misma en ambos casos y se muestra en la Fig. 1a: las dos cachés almacenan una copia del bloque A y el directorio contiene tal información (en el campo *Comp.*).

En la Fig. 1b los reemplazos se realizan de forma silenciosa. La figura de la izquierda muestra el reemplazo del bloque A en la *caché local 2*. Este reemplazo se realiza sin la necesidad de emitir ningún mensaje de coherencia. El dato es simplemente descartado de la caché local. El directorio, por tanto, mantiene su información intacta, por lo que no será precisa.

Es importante resaltar que no puede ocurrir ningún comportamiento incorrecto debido a esta imprecisión, siempre y cuando el directorio incluya en el conjunto de compartidores a todos los compartidores actuales (puede incluir nodos adicionales). Si el

directorio necesita invalidar todos los compartidores de un bloque, algunas cachés podrán recibir mensajes de invalidación innecesarios que requerirán respuesta, pero esto sólo se traduce en tráfico de coherencia extra, no en un comportamiento incorrecto.

La Fig. 1b (derecha) muestra otro reemplazo silencioso del mismo bloque A, en este caso de la *caché local 1*. Como se puede observar, la entrada del bloque A se mantiene ocupada en el directorio a pesar de no haber ninguna caché que almacene una copia de ese bloque.

La Fig. 1c muestra como se realizan los reemplazos ruidosos. En este caso, es necesario enviar mensajes de coherencia al directorio. En primer lugar, la caché que reemplaza el bloque envía la notificación al directorio. Cuando el directorio recibe, actualiza el campo *Comp.*, eliminando la caché que ha reemplazado el dato de la lista de compartidores. Por último, el directorio confirma a la caché el reemplazo y la transacción de coherencia acaba. La Fig. 1c (izquierda) muestra el reemplazo ruidoso del bloque A de la *caché local 2*.

La Fig. 1c (derecha) muestra un reemplazo del bloque A en la *caché local 1*. El reemplazo ocurre de la misma forma que para la *caché local 2*, pero en este caso el directorio ya no necesita almacenar información acerca de los compartidores, debido a que ninguna caché tiene copia del dato. Por tanto, el directorio puede liberar la entrada del bloque A, permitiendo así hacer un mejor uso del mismo.

B. Consecuencias de la política de reemplazo

La primera consecuencia de la política de reemplazo (silenciosa o ruidosa) es la cantidad de *tráfico* generada por el protocolo de coherencia de caché. Informar al directorio sobre el reemplazo de un bloque limpio implica dos mensajes de control, tal y como se muestra en la Fig. 1c. Observe que también serían necesarios dos mensajes de control para invalidar una copia limpia ya reemplazada en el caso en el que los reemplazos fueran silenciosos. Sin embargo estos mensajes sólo se producirían si los bloques sufrieran un fallo de escritura o un reemplazo de la información de directorio, es decir, una invalidación. Por tanto, los bloques de solo lectura o los bloques limpios que se reemplazan y cachean frecuentemente sin ninguna invalidación entre ambos eventos se beneficiarán de una política de reemplazo silenciosa. Como hemos observado que estas situaciones son frecuentes, el tráfico adicional generado por el uso de reemplazos ruidosos puede representar una gran parte del tráfico global generado por el protocolo de coherencia de cachés.

La segunda consecuencia de implementar reemplazos ruidosos para bloques limpios es la *exactitud* de la información de directorio. Esto afecta al protocolo de coherencia de dos formas diferentes. Por un lado, se reduce la cantidad de información que el directorio necesita almacenar. Esto significa que ante un fallo de escritura o un reemplazo en el directorio se emiten menos mensajes de invalidación, y por lo tanto se re-

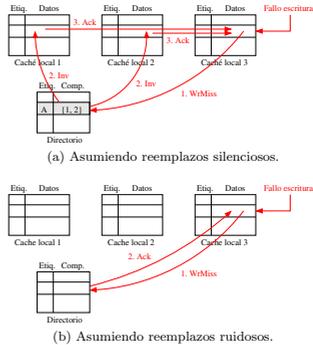


Fig. 2: Ejemplo de fallo de escritura dependiendo de la política de reemplazo.

cibirán también menos mensajes de confirmación. El envío de un menor número de invalidaciones reduce tanto el tráfico como la latencia de fallos de escritura. Informar de los reemplazos de datos limpios puede ser también beneficioso cuando cada entrada de directorio sólo puede almacenar un número limitado de compartidores (por ejemplo, en un esquema de punteros limitados). De esta forma se reutiliza naturalmente el almacenamiento del directorio, reduciendo, por tanto, el número de situaciones de desbordamiento que de lo contrario podrían surgir. Por otro lado, y más importante, la ocupación de directorio se reduce, ya que las entradas de directorio para los bloques que han sido reemplazados de todas las cachés pueden ser eliminadas del directorio. La reducción de la presión en el directorio conduce en efecto a un menor número de reemplazos de directorio y, en consecuencia, a un menor número de invalidaciones, que a su vez da lugar a un menor número de fallos de caché.

El efecto de tener información de directorio precisa ante un fallo de escritura se muestra en la Fig. 2. Supongamos por ejemplo, que ocurre un fallo de escritura para el bloque A después de los dos reemplazos de la Fig. 1. En el caso de haber usado reemplazos silenciosos (Fig. 2a), el directorio tiene que enviar invalidaciones a las L1 cachés a pesar de que ya no tienen una copia del bloque. Esto se debe a que el directorio no fue notificado cuando el bloque se reemplazó. El resultado es que los dos mensajes ahorrados por los reemplazos silenciosos son generados ahora por la invalidación, con un coste extra en latencia en el camino crítico del fallo. Sin embargo, en el caso de reemplazos ruidosos (Figura 2b), el directorio no realiza invalidaciones innecesarias, sino que directamente da permiso de escritura a la caché que generó el fallo.

C. Cuándo usar reemplazos silenciosos o ruidosos

A partir de los ejemplos anteriores se puede deducir que los dos tipos de políticas de reemplazo tie-

nen ventajas y desventajas. Los reemplazos silenciosos pueden reducir el tráfico de coherencia debido a los fallos de escritura como los reemplazos ruidosos pueden reducir la latencia de los fallos de escritura y mejorar la eficiencia de directorio. La mejor política dependerá tanto de las características de la aplicación como del protocolo de coherencia de caché empleado.

Los inconvenientes de los reemplazos silenciosos son las invalidaciones adicionales al resolver tanto los fallos de escritura como los reemplazos de directorio. Los datos de sólo lectura, por lo tanto, se beneficiarían de los reemplazos silenciosos si con frecuencia son reemplazados y solicitados de nuevo sin que el directorio reemplace la entrada correspondiente. Del mismo modo, los datos de lectura y escritura se verían favorecidos por los reemplazos silenciosos si los fallos de lectura dominan claramente a los fallos de escritura. De lo contrario, los reemplazos ruidosos pueden ser preferibles debido a una menor latencia de las escrituras.

Las propiedades del protocolo de coherencia de caché empleado tienen aún más importancia a la hora de tomar una decisión sobre la implementación de la política de reemplazo. Por ejemplo, un protocolo basado en listas enlazadas [18], [19], [20] no podría emplear reemplazos silenciosos ya que la lista de compartidores debe ser actualizada ante un reemplazo [21]. Del mismo modo, un protocolo basado en tokens [22], [23] no puede usar reemplazos silenciosos ya que los tokens que acompañan al bloque en caché deben transferirse al directorio o a otro compartidor.

Esta decisión es especialmente relevante para protocolos que emplean directorios donde el número de entradas utilizadas por bloque de memoria depende del número de compartidores (por ejemplo, asignación dinámica de punteros [24] o SCD [13]). En estos casos, una mejor precisión proporcionada por el uso de reemplazos ruidosos puede permitir una mejor utilización del directorio y, en consecuencia, reducir el número de reemplazos de directorio. Esto es menos importante en directorios tradicionales, ya que, en ese caso, las entradas de directorio solo se eliminan cuando el número de compartidores llega a cero.

III. ENTORNO DE EVALUACIÓN

Hemos utilizado los simuladores PIN [25] y GEMS 2.1 [26], conectándolos de forma similar a como se propone en [27]. Para modelar la red de interconexión, usamos SiCoSys [28]. La arquitectura simulada corresponde a un multiprocesador en un único chip (*tiled-CMP*) con 64 núcleos. Los principales parámetros de evaluación se muestran en la Tabla I.

Evaluamos el impacto que tiene la política de reemplazos de bloques limpios en dos configuraciones CMP de 64 núcleos. La primera configuración utiliza un directorio disperso que utiliza un vector de bits no escalable para codificar los compartidores de cada entrada (la primera barra de todas las gráficas representa a esta configuración con reemplazos silenciosos,

TABLA I: Parámetros del sistema.

Parámetros de la memoria	
Tamaño de bloque	64 bytes
Caché L1 de datos e instr.	32 KiB, 4 vías
Latencia de acceso a L1	1 ciclo
Caché L2 compartida	256 KiB/celda, 16 vías
Latencia de acceso a L2	6 ciclos más latencia de red
Organización L1 y L2	Inclusiva
Información de directorio	Incluida en L2
Tiempo de acceso a memoria	160 ciclos
Parámetros de la red	
Topología	Malla 2-D de 8x8 nodos
Técnica de conmutación	Wormhole
Técnica de enrutamiento	Determinista X-Y
Multicast/Broadcast	No soportado
Tamaño de mensajes	4 flits (datos), 1 flit (control)
Tiempo de enrutamiento	1 ciclo
Tiempo de conmutador	1 ciclo
Tiempo de enlace	2 ciclos
Tamaño del buffer	6 flits
Ancho de banda	1 flit por ciclo

mientras que la segunda usa reemplazos ruidosos). La segunda configuración representa a la qcientemente propuesta arquitectura de directorio SCD [13] (que aparece usando reemplazos silenciosos en la tercera barra de las gráficas y usando reemplazos ruidosos en la cuarta).

Para nuestras imulaciones, hemos usado una gran variedad de aplicaciones de los conjuntos de aplicaciones SPLASH-2 [29] y PARSEC 2.1 [30]. *Barnes*, *Cholesky*, *FFT*, *Ocean*, *Radix*, *Raytrace*, *Volrend* y *Water-NSQ* pertenecen a SPLASH-2 y emplean los mismos tamaño de entrada recomendados en el artículo [29]. *Bodytrack*, *Canneal*, *Streamcluster* y *Swaptions* pertenecen a PARSEC 2.1 y usan tamaño de entrada *simmedium*. Hemos tenido en cuenta la variabilidad en las aplicaciones paralelas tal y como se describe en [31]. Todos los resultados mostrados en este trabajo corresponden a la parte paralela de las aplicaciones evaluadas.

IV. RESULTADOS

El objetivo de nuestra evaluación es mostrar el impacto de los reemplazos silenciosos frente a los ruidosos. Comenzamos nuestro análisis observando la frecuencia de los reemplazos de L1, clasificándolos según el tipo del bloque expulsado. Para cada aplicación, la figura 3 muestra el número de reemplazos por instrucción ejecutada, clasificando los reemplazos en cuatro categorías: *Sucio*, *LimpioExclusivo*, *LimpioCompartidoRuidoso* y *LimpioCompartidoSilencioso*. Los reemplazos Sucios se corresponden con los de bloques en estado M (modificado) y los LimpioExclusivo con bloques en estado E. Los reemplazos de bloques en estado S se han clasificado como LimpioCompartidoRuidoso cuando el protocolo utiliza la política de reemplazos ruidosos y como LimpioCompartidoSilencioso si usa reemplazos silenciosos.

En promedio, los bloques en estado E no están involucrados en una parte significativa de los reemplazos, aunque hay excepciones en aplicaciones como *Canneal* y *FFT*. La mayoría de los reemplazos de caché L1 (67% en SCD y 73% en el directorio basado en vector de bits, de media) corresponden a

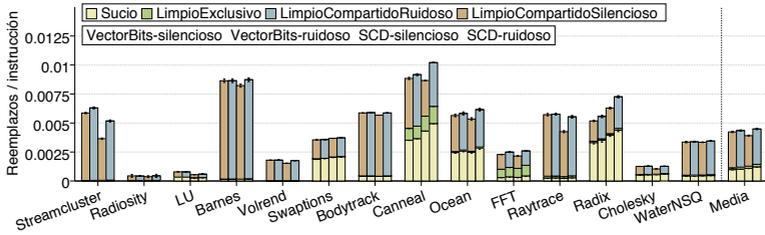


Fig. 3: Reemplazos de LI por instrucción, categorizados.

bloques en estado S. Este es el primer indicio de la importancia que tiene la implementación eficiente de este tipo de reemplazos.

Los reemplazos ruidosos incrementan la precisión de la información de directorio al eliminar del conjunto de compartidores apuntado en el directorio aquellos nodos que expulsan el bloque de su caché local. En un directorio disperso tradicional, las entradas de directorio se liberan únicamente cuando el número de compartidores llega a 0. Sin embargo, en directorios en los que el código de compartición se distribuye entre varias entradas, tales como SCD, el uso de reemplazos ruidosos ayuda a reducir la presión en el directorio porque permite liberar algunas entradas tan pronto como dejan de ser necesarias, lo que reduce la ocupación del directorio. Con el objetivo de analizar este efecto, la Figura 4 muestra el número de reemplazos de directorio por instrucción ejecutada. De media, cuando se usan reemplazos ruidosos, el número de reemplazos de directorio se reduce en un 1.3% en el caso del vector de bits y en un 30% en el caso de SCD.

Otro efecto esperado del uso de los reemplazos silenciosos en una reducción en la latencia de los fallos de escritura debido a que se necesita enviar menos invalidaciones. La Figura 5 muestra la latencia de los fallos de escritura para cada protocolo. Cada barra está dividida en cinco partes: tiempo accediendo a la L1 (En_{L1}); tiempo hasta llegar a la L2 ($Hasta_{L2}$), tiempo de espera en la L2 hasta que ésta puede atender la petición, debido principalmente a que el controlador está ocupado atendiendo a otras peticiones (En_{L2}); tiempo de acceso a la memoria ($Memoria$) y tiempo desde que el fallo deja la L2 hasta que la L1 recibe la respuesta y se resuelve ($Hasta_{L1}$) el fallo. En realidad, sólo se aprecian muy pequeñas variaciones en la latencia; se reduce muy ligeramente para algunos benchmarks en el caso del vector de bits o se incrementa también muy ligeramente para otros en el caso de SCD. Esto ocurre así porque en la práctica el número de invalidaciones por fallo es similar y, lo que es más, todas las invalidaciones se envían en paralelo y sólo el procesamiento de los mensajes de reconocimiento se beneficia de la reducción, ya que esto tiene que ser realizado secuencialmente por el

petionario. Por otro lado, el ligero incremento en tráfico y reemplazos de directorio puede explicar el ligero incremento en latencia.

Por otro lado, el efecto en la latencia de los fallos de lectura se puede ver en la Figura 6. Los fallos de lectura se manejan exactamente igual tanto en el caso de los reemplazos ruidosos como en el de los silenciosos, por lo cualquier diferencia se debe explicar principalmente por la diferente cantidad de tráfico que viaja por la red y por el diferente número de reemplazos de directorio debido a la mayor precisión en la información de compartidores proporcionada por los reemplazos ruidosos. Por ejemplo, la latencia de *Streamcluster* con SCD se reduce cuando se usan reemplazos ruidosos en lugar de silenciosos. Esto es porque el tiempo En_{L2} es menor, lo que es consecuencia de que hay menos reemplazos de directorio manteniendo ocupado al controlador. Igual que en caso de las escrituras, vemos que las diferencias en promedio son mínimas, con un incremento casi insignificante de la latencia cuando se usan reemplazos ruidosos.

El impacto más importante de la política de reemplazos de datos limpios está en el tráfico de la red de interconexión. La Figura 7 muestra el tráfico de coherencia medido en flits y normalizado respecto al directorio basado en vector de bits con reemplazos silenciosos. Se ha dividido el tráfico en las siguientes categorías: mensajes de datos debidos a fallos de caché (*Data*), mensajes de datos debidos a reemplazos (*WBData*), mensajes de control debidos a fallos de caché (*Control*), y mensajes de control debidos a reemplazos (*WBControl*). Vemos que, aunque un protocolo con reemplazos silenciosos genera en algunas ocasiones un poco más de tráfico de mensajes de control debidos a fallos de caché, un protocolo con reemplazos ruidosos genera siempre una cantidad significativamente mayor de mensajes de control debidos a reemplazos. En el caso del directorio vector de bits, los reemplazos ruidosos casi siempre incrementan el tráfico total (excepto en *FTT* y *Radix*), pero el incremento es menos pronunciado en el caso de SCD y hay más excepciones (*FTT* y *Radix* de nuevo, pero también *Cannal*, *Cholesky*, *LU* y *Streamcluster*). Esto ocurre porque los reemplazos ruidosos pueden

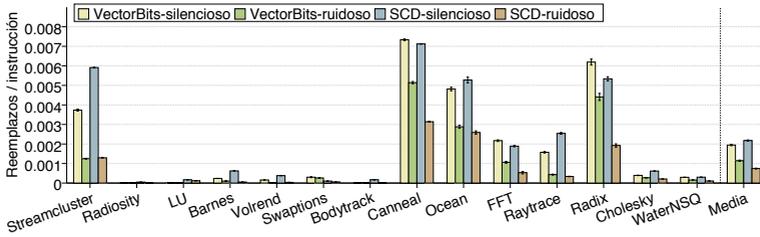


Fig. 4: Reemplazos de directorio por instrucción.

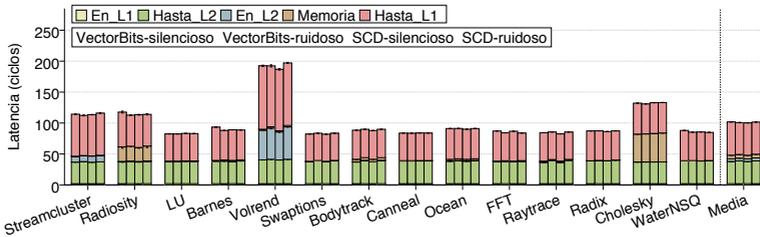


Fig. 5: Latencia de los fallos de escritura en L1.

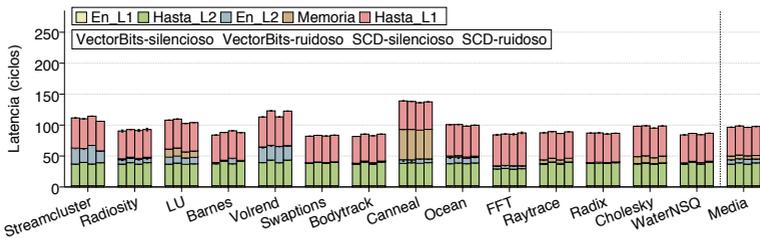


Fig. 6: Latencia de los fallos en L1.

reducir el número de reemplazos de directorio (como se muestra en la figura 4). Estos reemplazos de directorio requieren la invalidación de cualquier copia del bloque de datos que estuviera en alguna caché local, por lo que su reducción reduce también el número de fallos que, a su vez, reduce la cantidad de mensajes de control y, lo que es más importante, la cantidad de mensajes de datos (que son más grandes). De media, los reemplazos ruidosos incrementan el tráfico total de la red en un 9.6% para un directorio basado en vector de bits y en un 4.1% para el directorio SCD.

Para terminar la evaluación, mostramos también el impacto en el tiempo de ejecución de la política

de reemplazos en la Figura 8. Todos los resultados se han normalizado respecto a un directorio basado en vector de bits que usa reemplazos silenciosos. De media, el uso de reemplazos ruidosos no afecta en nada al tiempo de ejecución de un protocolo de directorio basado en vector de bits respecto al uso de reemplazos silenciosos. Dado que los reemplazos ruidosos incrementan el tráfico en un 9.6%, podemos concluir que cuando se usa un directorio basado en vector de bits, es mejor utilizar reemplazos silenciosos. El efecto en el tiempo medio de ejecución del directorio SCD es muy pequeño: los reemplazos ruidosos reducen el tiempo de ejecución en un 1.5% a

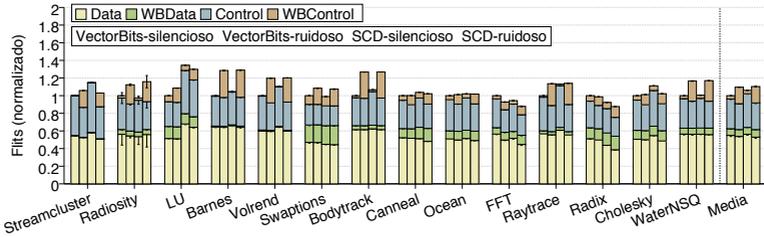


Fig. 7: Tráfico en la red de interconexión.

cambio de un incremento del 4.1 % en tráfico. Por tanto, el uso de reemplazos ruidosos en un protocolo como SCD puede estar justificado si el diseñador quiere conseguir el máximo rendimiento sin que importe el coste energético.

V. CONCLUSIONES

En este trabajo ponemos el foco sobre un aspecto de diseño de los protocolos de coherencia basados en directorio que creemos que está siendo subestimado en la actualidad: cómo conviene gestionar los reemplazos de los bloques limpios en el último nivel de cachés privadas. Discutimos las dos alternativas que son posibles (tener reemplazos *silenciosos* frente a reemplazos *ruidosos*), enfatizando las ventajas e inconvenientes de cada una de ellas. A continuación consideramos dos estructuras alternativas para el directorio y mostramos, teniendo en cuenta el rendimiento (tiempo de ejecución) y la cantidad de tráfico de red, que para cada una de ellas es preferible usar una política distinta.

Más concretamente, nuestros resultados demuestran que la utilización de la política de reemplazos *ruidosos* no está justificada cuando se emplea un directorio que utiliza vectores de bits como código de compartición, ya que se incrementaría el tráfico en más de un 25 % en varios casos (9.6 % de media) y no se obtendría ningún beneficio desde el punto de vista del tiempo de ejecución. El hecho de que todos los compartidores tengan que haber reemplazado sus copias del bloque de datos para que la entrada de directorio pueda ser liberada, limita mucho el número de ocasiones en las que la entrada podría ser reasignada a otra dirección distinta como consecuencia de las notificaciones de reemplazo. Además, hemos observado también que hay ocasiones en las que un bloque limpio se reemplaza varias veces antes de que sea invalidado (como consecuencia de un fallo de escritura o del reemplazo de la entrada de directorio asignada a la dirección del bloque), lo que ocasiona bastante tráfico extra cuando se usa la política de reemplazos *ruidosos*. Una conclusión importante, por lo tanto, que se podría extraer de este trabajo es que cuando se asume como sistema base en una comparativa un directorio que utiliza vectores de bits, este debería

utilizar la política de reemplazos *silenciosos*, ya que de otra manera se estaría realizando la comparativa partiendo de una configuración base subóptima.

Por otro lado, también hemos constatado que la habilidad que tiene la política de reemplazos *ruidosos* de mejorar la precisión de la información de directorio puede tener efectos balsámicos en algunas aplicaciones cuando el directorio utiliza un código de compartición que se distribuye entre distintas entradas, tal y como sucede en SCD. En estos casos, la utilización de reemplazos *ruidosos* ayuda a reducir la presión sobre el directorio, ya que permite liberar algunas entradas en el momento en el que los pocos compartidores que codificaban han reemplazado su copia del bloque de datos. Esto al final conlleva reducciones en la ocupación del directorio y, gracias a este efecto, el tiempo de ejecución puede ser reducido (hasta un 9.8 %, 2.7 % de media) si se emplea la política de reemplazos *ruidosos*.

AGRADECIMIENTOS

Este trabajo ha sido financiado por la Fundación Séneca-Agencia de Ciencia y Tecnología de la Región de Murcia mediante el proyecto "19295/PI/14".

REFERENCIAS

- [1] Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin, "Why on-chip cache coherence is here to stay," *Communications of the ACM*, vol. 55, no. 7, pp. 78–89, July 2012.
- [2] Lucien M. Censier and Paul Feautrier, "A new solution to coherence problems in multicache systems," *IEEE Transactions on Computers (TC)*, vol. 27, no. 12, pp. 1112–1118, Dec. 1978.
- [3] Daniel J. Sorin, Mark D. Hill, and David A. Wood, *A Primer on Memory Consistency and Cache Coherence*, Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2011.
- [4] David E. Culler, Jaswinder P. Singh, and Anoop Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers, Inc., 1999.
- [5] Dana Vreantea, Mikko H. Lipasti, and Nathan Binkert, "Atomic coherence: Leveraging nanophotonics to build race-free cache coherence protocols," in *17th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2011, pp. 132–143.
- [6] Blas Cuesta, Alberto Ros, María E. Gómez, Antonio Robles, and José Duato, "Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks," in *38th Int'l Symp. on Computer Architecture (ISCA)*, June 2011, pp. 93–103.

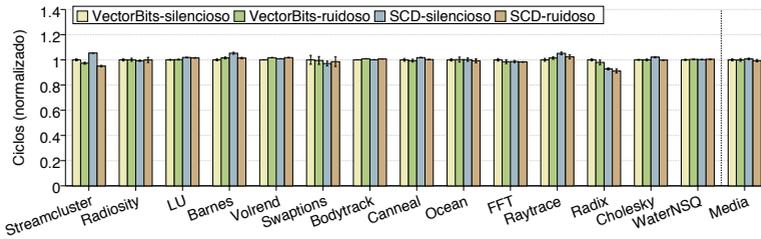


Fig. 8: Tiempo de ejecución.

[7] Marco Elver and Vijay Nagarajan, "TSO-CC: Consistency directed cache coherence for tso," in *20th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2014, pp. 165–176.

[8] Meng Zhang, Jesse D. Bingham, John Erickson, and Daniel J. Sorin, "PVCohere: Designing flat coherence protocols for scalable verification," in *20th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2014, pp. 392–403.

[9] Jason Zebchuk, Babak Falsafi, and Andreas Moshovos, "Multi-grain coherence directories," in *46th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2013, pp. 359–370.

[10] Soemates Demetriades and Sangyeon Cho, "Stash directory: A scalable directory for many-core coherence," in *20th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2014, pp. 177–188.

[11] Lucia G. Menezes, Valentin Puenente, and José Ángel Gregorio, "Flask coherence: A morphable hybrid coherence protocol to balance energy, performance and scalability," in *21th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2015, pp. 198–209.

[12] Minshu Zhao and Donald Yeung, "Studying the impact of multicore processor scaling on directory techniques via reuse distance analysis," in *21th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2015, pp. 590–602.

[13] Daniel Sanchez and Christos Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in *18th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2012, pp. 129–140.

[14] Guowei Zhang, Webb Horn, and Daniel Sanchez, "Exploiting commutativity to reduce the cost of updates to shared data in cache-coherent systems," in *48th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2015, pp. 13–25.

[15] Yaosheng Fu, Tri M. Nguyen, and David Wentzlaff, "Coherence domain restriction on large scale systems," in *48th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2015, pp. 686–698.

[16] Thomas Moscibroda and Omur Mutlu, "A case for bufferless routing in on-chip networks," in *36th Int'l Symp. on Computer Architecture (ISCA)*, June 2009, pp. 196–207.

[17] Shekhar Borkar, "Thousand core chips: a technology perspective," in *44th Design Automation Conference (DAC)*, June 2007, pp. 746–749.

[18] David V. James, Anthony T. Landrie, Stein Gjessing, and Gurinder S. Sohi, "Scalable coherent interface," *IEEE Computer*, vol. 23, no. 6, pp. 74–77, June 1990.

[19] Tom Lovett and Russell Clapp, "STING: A cc-NUMA computer system for the commercial marketplace," in *23rd Int'l Symp. on Computer Architecture (ISCA)*, June 1996, pp. 308–317.

[20] Radhika Thekkath, Amit P. Singh, Jaswinder P. Singh, Susan John, and John L. Hennessy, "An evaluation of a commercial cc-NUMA architecture: The CONVEX Exemplar SPP1200," in *11th Int'l Symp. on Parallel Processing (IPPS)*, Apr. 1997, pp. 8–17.

[21] Ricardo Fernández-Pascual, Alberto Ros, and Manuel E. Acacio, "Optimization of a linked cache coherence protocol for scalable manycore coherence," in *Proc. of 29th International Conference on Architecture of Computing Systems (ARCS 2016)*, 2016, pp. 100–112.

[22] Milo M.K. Martin, Mark D. Hill, and David A. Wood, "Token coherence: Decoupling performance and correctness," in *30th Int'l Symp. on Computer Architecture (ISCA)*, June 2003, pp. 182–193.

[23] Michael R. Marty, Jesse D. Bingham, Mark D. Hill, Alan J. Hu, Milo M.K. Martin, and David A. Wood, "Improving multiple-CMP systems using token coherence," in *11th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2005, pp. 328–339.

[24] Richard Simoni and Mark A. Horowitz, "Dynamic pointer allocation for scalable cache coherence directories," in *Int'l Symp. on Shared Memory Multiprocessing*, Apr. 1991, pp. 72–81.

[25] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klausner, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *2005 ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, June 2005, pp. 190–200.

[26] Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Sept. 2005.

[27] Matteo Monchiero, Jung Ho Ahn, Ayose Falcón, Daniel Ortega, and Paolo Faraboschi, "How to simulate 1000 cores," *Computer Architecture News*, vol. 37, no. 2, pp. 10–19, July 2009.

[28] Valentin Puenente, José A. Gregorio, and Ramón Beivide, "SICOSYS: An integrated framework for studying interconnection network in multiprocessor systems," in *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, Jan. 2002, pp. 15–22.

[29] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *22nd Int'l Symp. on Computer Architecture (ISCA)*, June 1995, pp. 24–36.

[30] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *17th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2008, pp. 72–81.

[31] Alaa R. Alameldeen and David A. Wood, "Variability in architectural simulations of multi-threaded workloads," in *9th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2003, pp. 7–18.

Gestión de Contenidos en Caches Operando a Bajo Voltaje

Alexandra Ferrerón¹, Jesús Alastruey Benedé¹, Darío Suárez Gracia¹,
Teresa Monreal Arnal², Pablo Ibáñez¹ y Víctor Viñals Yúfera¹

Resumen—La eficiencia energética de las caches en chip puede mejorarse reduciendo su voltaje de alimentación (V_{dd}). Sin embargo, este escalado de V_{dd} está limitado a una tensión $V_{dd,min}$ por debajo de la cual algunas celdas SRAM (*Static Random Access Memory*) puede que no operen de forma fiable.

Block disabling (BD) es una técnica microarquitectónica para implementar caches tolerantes a fallos. BD permite operar a tensiones muy bajas (por debajo de $V_{dd,min}$) desactivando aquellas entradas que contienen alguna celda defectuosa, aunque a cambio de reducir la capacidad efectiva de la cache. El ahorro potencial es mayor en caches de último nivel (LLC). Sin embargo, para algunas aplicaciones, el incremento de consumo debido a los accesos a memoria fuera del chip no compensa el ahorro energético conseguido en la LLC.

Este trabajo aprovecha recursos existentes en los multiprocesadores, como son la jerarquía de memoria en chip y el mecanismo de coherencia, para mejorar las prestaciones de BD. En concreto, proponemos explotar la redundancia natural existente en una jerarquía de cache inclusiva para mitigar la pérdida de rendimiento debida a la reducción en la capacidad de la LLC.

También proponemos una nueva política de gestión de contenidos consistente de la existencia de entradas de cache defectuosas. Utilizando la información de retiso, el algoritmo de reemplazo asigna entradas de cache operativas a aquellos bloques con más probabilidad de ser referenciados.

Las técnicas propuestas llegan a reducir el MPKI hasta en un 37% respecto a block disabling, mejorando su rendimiento entre un 2% y un 13%.

Palabras clave—Computación cerca de la tensión umbral, tolerancia a fallos, gestión de contenidos en LLC

I. INTRODUCCIÓN

La densidad de potencia es el principal factor que limita las prestaciones de las tecnologías CMOS actuales. La ley de Moore se sigue cumpliendo, por lo que cada nueva tecnología de fabricación duplica el número de transistores y la escala de integración. Sin embargo, el escalado de Dennard ha llegado a su fin [1], y ya no es posible mantener la densidad de potencia constante de generación en generación. Las restricciones en el consumo impiden la activación simultánea de todos los transistores disponibles, lo que se denomina *dark silicon* [2].

Durante años, la industria ha confiado en la reducción de la tensión de alimentación (V_{dd}) para disminuir el consumo de energía. Pero, desde la generación de 90 nm, esta tendencia ha cambiado, debido principalmente al incremento de las corrientes de fuga. Al

reducir V_{dd} hay que reducir asimismo la tensión umbral (V_{th}) para mantener la fiabilidad y velocidad en la conmutación. Sin embargo, una V_{th} baja incrementa el consumo estático de energía en mayor medida que los ahorros obtenidos al disminuir V_{dd} . Reducir la tensión de operación de los transistores a valores cercanos a V_{th} sin pérdida de fiabilidad permitiría minimizar los consumos estático y dinámico de energía (corrientes de fuga y conmutación, respectivamente). La reducción de consumo resultante podría usarse para activar más recursos del chip y potencialmente obtener mejores prestaciones [3].

Desafortunadamente, el escalado de V_{dd} está limitado por los ajustados márgenes de operación que tienen las celdas SRAM utilizadas para construir las caches en chip. Las variaciones en los parámetros de las celdas SRAM limitan la reducción de V_{dd} a una tensión, $V_{dd,min}$, por debajo de la cual las celdas SRAM no tienen un funcionamiento fiable. $V_{dd,min}$ normalmente determina la tensión mínima del procesador, que en las tecnologías actuales es del orden de 0.7–1.0 V cuando se utilizan celdas SRAM convencionales (de 6 transistores o 6T). La fluctuación aleatoria de dopantes (*random dopant fluctuation*, RDF) dentro del dado es una de las principales causas de los fallos en las celdas que funcionan a voltajes ultra bajos, lo que resulta en diferentes valores de V_{th} en los dispositivos SRAM considerados. Esto da lugar a una distribución aleatoria de celdas defectuosas cuyo comportamiento a tensiones inferiores a $V_{dd,min}$ es equivalente al de un fallo de operación (*hard fault*).

En la literatura se han propuesto diferentes soluciones a los problemas que aparecen cuando la cache opera a baja tensión. A nivel circuital, puede aumentarse la resiliencia de una celda SRAM aumentando el tamaño de sus transistores o añadiendo circuitería especial [4], [5]. Su principal desventaja es que aumentan el área y el consumo del dispositivo. Procesadores comerciales, como por ejemplo los de la familia Intel Nehalem, utilizan celdas con circuitería adicional (celdas SRAM 8T) en las caches de primer nivel para hacerlas más resilientes [6]. Las caches de primer nivel en los multiprocesadores en chip (*chip multiprocessors*, CMPs) ocupan poca superficie para no limitar la frecuencia del procesador. Por otra parte, las caches de último nivel (*last-level caches*, LLCs) tienen tamaños y asociatividades mucho mayores, por lo que ocupan gran parte de la superficie del chip. Por tanto, para construir las LLCs son preferibles las estructuras SRAM formadas por celdas 6T, ya que son más densas que las formadas por celdas más

¹gaZ. Dpto. de Informática e Ing. de Sistemas. Inst. de Investigación en Ing. de Aragón. Universidad de Zaragoza, e-mail: {ferreron, jalastru, dario, imarin, victor}@unizar.es;
²Dpto. de Arquitectura de Computadores. Universitat Politècnica de Catalunya, email: teresa@ac.upc.edu

grandes o con circuitería adicional.

A nivel microarquitectónico, los diseños de caches tolerantes a fallos se basan en la deshabilitación de los recursos defectuosos usando distintas granularidades [7], en la corrección de bits defectuosos mediante complejos códigos de corrección de error (*error correction codes*, ECCs) [8], o en la replicación distribuida de bloques [9], [10]. *Block disabling* (BD) [11] es una técnica sencilla que desconecta toda una entrada de cache cuando en ella se detecta al menos un bit defectuoso¹. Esta técnica se implementa en procesadores modernos para protegerlos contra los *hard faults* [12]. Sin embargo, debido a que las celdas defectuosas se distribuyen de forma aleatoria, la capacidad de la cache queda comprometida. Técnicas complejas basadas en sofisticados ECCs o en la combinación de recursos defectuosos permiten utilizar más capacidad de cache, pero requieren costes adicionales de almacenamiento y a veces complicados mecanismos de mapeo de direcciones de memoria que penalizan la latencia de acceso a cache.

Este trabajo propone una nueva estrategia para mitigar el impacto de los fallos que causan las variaciones de parámetros en las celdas SRAM de las LLCs. Se basa en la técnica *block disabling* y utiliza estructuras ya existentes en un CMP. Hemos identificado una fuente natural de redundancia de datos dentro del chip que surge por la replicación de bloques en jerarquías de cache en inclusión y la hemos aprovechado mediante nuevas técnicas de gestión de contenidos.

En este trabajo se hacen las siguientes contribuciones. En primer lugar, evaluamos varias técnicas de deshabilitación de bloques en un CMP que ejecuta cargas de trabajo multiprogramadas en un entorno de simulación con un completo y detallado modelo de memoria. En segundo lugar, presentamos una técnica que mantiene operativas las etiquetas de las entradas de cache deshabilitadas, y por tanto, sus funcionalidades de control del directorio de coherencia. De esta forma, un bloque que físicamente no está almacenado en la LLC puede estar presente, y disponible, en las caches privadas. Finalmente, proponemos una técnica de gestión de contenidos consciente de las entradas defectuosas que reduce la tasa de fallos de cache. Esta técnica, basándose en el patrón de uso de los bloques de cache, predice su utilidad y controla su asignación a entradas de cache.

El resto de este artículo está organizado como sigue. La sección II presenta el modelo de fallos de celdas SRAM cuando operan a bajo voltaje. La sección III introduce las técnicas de deshabilitación de entradas de cache (*block disabling*) y su impacto en la LLC. En la sección IV proponemos aprovechar la organización de la jerarquía de memoria en inclusión para operar a voltajes bajos y en la sección V exploramos técnicas de gestión de contenidos en este contexto. La sección VI describe la metodología utilizada. En la

¹En este trabajo distinguimos entre bloque y entrada de cache: bloque es la unidad de correspondencia y transferencia, el contenido per se; entrada es el conjunto de celdas físicas que almacenan un bloque.

sección VII se detallan los resultados obtenidos. La sección VIII recoge el trabajo relacionado y, finalmente, la sección IX concluye.

II. MODELO DE FALLOS DE CELDAS SRAM

Las estructuras SRAM son especialmente vulnerables a fallos causados por variaciones paramétricas cuando operan a tensiones de alimentación muy bajas o cercanas al umbral. Normalmente, las estructuras SRAM tienen un voltaje mínimo que garantiza la operación de las celdas de forma fiable, $V_{dd,min}$. En la actualidad este voltaje se sitúa entre los 0.7 y 1.0 V cuando se utilizan celdas 6T.

Diversos autores han analizado la fiabilidad de estas celdas por debajo de $V_{dd,min}$. En concreto, Zhou *et al.* estudian seis tamaños de celdas SRAM 6T en una escala de integración de 32 nm y sus probabilidades de fallo conforme disminuye V_{dd} [5]. Según este estudio, a 0.5 V la probabilidad de fallo de una celda SRAM (P_{fail}) varía entre 10^{-3} y 10^{-2} . El uso de celdas más grandes reduce la probabilidad de fallo, ya que se incrementan los márgenes de lectura y escritura. Por contra, celdas más grandes dan como resultado caches menos densas que consumen más energía.

La Tabla 1 recoge las seis celdas SRAM de este estudio (C1, C2, C3, C4, C5 y C6) con su área relativa a la celda más pequeña (C1), así como los porcentajes de entradas operativas de caches implementadas con cada tipo de celda a 0.5 V, asumiendo que las entradas tienen un tamaño de 64 bytes. Se considera que una entrada no está operativa si alguna de sus celdas falla.

Como muestra la Tabla 1, menos del 10% de las entradas están operativas cuando se utilizan las celdas C1 o C2 a 0.5 V. Si se utilizan celdas más robustas, por ejemplo, la C6, el porcentaje de entradas operativas sube al 60%, pero con un coste añadido de incrementar el área un 58% (relativo a C1), lo que no es una opción viable para una estructura tan grande como la LLC.

En este artículo utilizamos este estudio como referencia y nos centramos en las celdas C2-C6 a 0.5 V, ya que una cache construida con celdas C1 prácticamente no tiene entradas operativas.

TABLA I
 PORCENTAJE DE ENTRADAS OPERATIVAS EN UNA CACHE A 0.5 V (ENTRADAS DE 64 BYTES) PARA LOS DISTINTOS TAMAÑOS DE CELDAS Y SU ÁREA RELATIVA AL TAMAÑO DE LA CELDA C1.

Celda	C1	C2	C3	C4	C5	C6
Área relativa	1.00	1.12	1.23	1.35	1.46	1.58
% sin fallos	0.0	9.9	27.8	35.8	50.6	59.9

III. BLOCK DISABLING APLICADO A LAS CACHES DE ÚLTIMO NIVEL

Una opción sencilla para mitigar los fallos de operación consiste en deshabilitar aquellos recursos que no operan de forma fiable. La técnica *block disabling* (BD) deshabilita estos recursos usando granularidad de bloque [11]. Si se detecta un fallo en una celda de memoria, se desactiva la entrada de cache en la

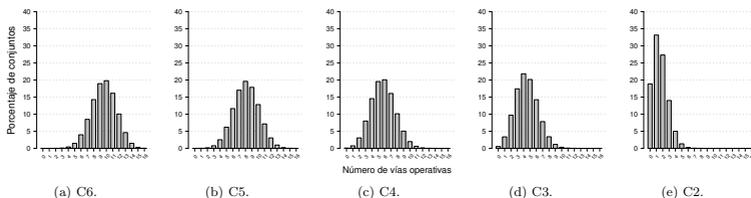


Fig. 1. Vías operativas, no defectuosas, en un banco de cache de 16 vías (celdas C6-C2) con tamaño de bloque 64 bytes a 0.5 V.

que se encuentra, por lo que no podrá almacenar bloque alguno. Esta técnica ya está implementada en procesadores actuales con protección contra *hard faults* [12].

Debido a su sencilla implementación y bajo coste, se ha estudiado la aplicación de BD a caches operando a tensiones bajas [13]. Desde el punto de vista de su implementación, un único bit basta para marcar una entrada de cache como defectuosa. La principal desventaja, tal como se mostró en la Tabla I, es que a medida que la probabilidad de fallo aumenta, la capacidad de la cache disminuye de forma sustancial. La cantidad total de celdas defectuosas en la cache no es elevada (menos del 1%), pero dada la aleatoriedad en su distribución, la capacidad efectiva de la cache disminuye rápidamente. Utilizar BD como técnica de tolerancia a fallos a tensiones bajas da lugar a caches con una asociatividad variable, que viene determinada por el número y distribución de los fallos.

La interacción entre BD y la organización de la jerarquía de cache en el chip también juega un papel importante en el rendimiento del sistema. Actualmente, algunos procesadores comerciales, como Intel Core i7, implementan jerarquías en inclusión para gestionar la coherencia de una forma sencilla. En tales jerarquías, todos los bloques almacenados en los niveles privados también lo están en la LLC. La información de coherencia va integrada en la propia LLC, es decir, para cada bloque se guarda su estado y un vector binario de presencia para identificar la ubicación de todas sus copias compartidas. Para garantizar la propiedad de inclusión, cuando se expulsa un bloque de la LLC, se invalidan todas las copias del bloque presentes en las caches privadas. A estos bloques invalidados se les llama víctimas por inclusión [14].

Las jerarquías en inclusión no se comportan bien cuando la capacidad agregada de las caches privadas es similar a la de la LLC [15]. BD agrava este problema al disminuir de manera significativa la asociatividad y capacidad de la LLC. La Figura 1 muestra el porcentaje de vías operativas (no defectuosas) en un banco de cache de 16 vías y tamaño de bloque 64 bytes, usando las celdas descritas a 0.5 V. La cantidad de vías defectuosas por conjunto sigue una distribución binomial $B(n, p)$, donde n es la asociatividad y p la probabilidad de fallo de una entrada de cache. Esta

pérdida de asociatividad se traduce directamente en un aumento importante del número de víctimas por inclusión. Como ejemplo, a 0.5 V el número de invalidaciones en una cache con celdas C3 se multiplica por 10 en relación a una cache implementada con celdas SRAM libres de fallos.

Este último dato sugiere que las jerarquías en inclusión no son óptimas en sistemas que implementan BD como técnica de tolerancia a fallos en situaciones en las que se dan un gran número de entradas defectuosas. Sin embargo, para gestionar la coherencia, sólo es estrictamente necesario disponer de inclusión de directorio: basta con conservar en la cache compartida la etiqueta y el vector de presencia de los bloques existentes en las caches privadas, sin necesidad de tener una copia de los datos [13]. Esta observación es la base de la propuesta que planteamos en este trabajo.

IV. APROVECHAR LA COHERENCIA PARA OPERAR DE MANERA EFICIENTE A BAJO VOLTAJE

Block disabling utiliza un bit por entrada de cache para identificar las entradas defectuosas en el *array* de datos (una o más celdas defectuosas). Las entradas defectuosas se excluyen de la búsqueda asociativa en etiquetas y del reemplazo, lo que conlleva una reducción en el número de vías útiles y el incremento de víctimas por inclusión.

Sin embargo, desde el punto de vista de la coherencia, identificar los bloques que se encuentran en caches privadas mediante el almacenamiento de su etiqueta en el *array* de etiquetas de la LLC es suficiente para garantizar la inclusión de directorio. Esta idea es la base de nuestra primera técnica, a la que llamamos *Block Disabling with Operational Tags* (BDOT). Para mantener la inclusión de directorio utilizamos las etiquetas de las entradas defectuosas, de forma que se incluyen en las operaciones de búsqueda y reemplazo. Al habilitar estas etiquetas, la asociatividad vista por los niveles privados de la jerarquía se mantiene. La etiqueta de una entrada defectuosa, si es válida, nos indica que un bloque puede estar presente en las caches privadas, pero que no se puede almacenar en la cache compartida.

Para poder implementar este esquema, debemos garantizar que no existan celdas defectuosas en el *array* de etiquetas. Esto puede hacerse utilizando celdas

robustas (por ejemplo, celdas con más transistores) o mediante sofisticados mecanismos ECC. Como las etiquetas ocupan un área relativamente pequeña en comparación con los datos (en nuestro diseño, Tabla II, alrededor del 6%), incrementar el tamaño de la celda un 33% (si asumimos celdas 8T) tiene un impacto en el área total de la cache del 2%. Nos decantamos por esta solución porque utilizar complejos ECCs puede incrementar la latencia de acceso a las etiquetas, mientras que utilizar celdas más robustas implica un menor coste añadido. Otros trabajos también utilizan esta alternativa [9], [10].

El protocolo de coherencia también tiene que adaptarse a esta nueva situación, en donde entradas defectuosas sólo contienen etiqueta, información de presencia y estado de coherencia. Desde el punto de vista de la implementación, sólo se necesita un bit que indique si la entrada contiene alguna celda defectuosa o no (como en el caso de BD). Este bit permite distinguir dos tipos de entradas en la LLC:

- *T*: sólo puede almacenar etiqueta (la entrada de datos tiene alguna celda defectuosa).
- *D*: puede almacenar etiqueta y dato (la entrada de datos es operativa).

Cuando llega una petición de un bloque almacenado en una entrada de tipo *T*, la petición debe enviarse al siguiente nivel de la jerarquía (en este caso fuera del chip). Lo mismo ocurre con bloques sucios, que deben enviarse a memoria una vez se expulsan de la cache privada.

Protegiendo la información del directorio, el incremento en el número de víctimas por inclusión desaparece, ya que la asociatividad de la cache se mantiene. De todas formas, la capacidad de la cache se sigue viendo comprometida, con el consecuente aumento del tráfico fuera del chip.

La principal limitación de este esquema es que la asignación de bloques a entradas de cache no tiene en cuenta el tipo de entrada, es decir, si puede almacenar sólo etiqueta (*T*) o etiqueta y dato (*D*). En el caso de que un bloque con un patrón de reuso (es decir, que tras ser expulsado de la L1 el procesador vuelva a solicitarlo en un periodo corto de tiempo) sea asignado a una entrada *T*, posteriores peticiones a la LLC se redirigen a memoria. Por tanto, es necesario implementar BDOT junto con una política de gestión de contenidos capaz de identificar dichas situaciones, así como de aprovechar las diferencias en cuanto a los dos tipos de contenedores existentes.

V. ASIGNACIÓN DE VÍAS BASADA EN REÚSO OPERANDO A VOLTAJES MUY BAJOS

Las políticas de reemplazo para caches convencionales (libres de fallos) no cumplen los requerimientos impuestos por BDOT, ya que implícitamente asumen que todas las entradas pueden almacenar bloques. Esta asunción es falsa en caches con deshabilitación de entradas pero etiquetas operacionales (que implementan BDOT).

En BDOT, cada conjunto de la cache con aso-

ciatividad *N* contiene entradas *T* que solo pueden almacenar la etiqueta del bloque y entradas *D* capaces de almacenar tanto etiquetas como datos, siendo $Num_T + Num_D = N$. Para mejorar las prestaciones del sistema es necesario acomodar de manera distinta bloques en ambos tipos de entradas. En concreto, proponemos incorporar a la política de reemplazo dos objetivos relacionados con la gestión diferenciada de entradas *T* y *D*:

1. Asignar entradas *D* a los bloques con mayor probabilidad de ser usados en el futuro.
2. Entre los bloques con alta probabilidad de volver a ser usados en el futuro, asignar entradas *D* preferiblemente a aquellos bloques no presentes en caches privadas (L1).

Una petición sobre un bloque alojado en una entrada *T* pero presente en una cache L1 puede ser servido dentro del chip. Sin embargo, una petición sobre un bloque alojado en una entrada *T* no presente en ninguna L1 siempre es servida por la memoria principal, con la consecuente penalización en tiempo y consumo. Estos nuevos objetivos pueden ser añadidos a cualquier política de reemplazo. En este artículo usaremos *Not Recently Reused*, NRR, como algoritmo de reemplazo base ya que es simple, efectivo, y nuestra propuesta puede aprovechar la información existente en NRR. A continuación describimos en qué se basa el algoritmo NRR. En la subsección V-B se detalla la implementación de nuestra propuesta utilizando NRR.

A. Política de reemplazo base

Los algoritmos de reemplazo basados en reuso han demostrado ser eficientes en caches compartidas de último nivel [16]. La localidad de reuso dice que si un bloque ha sido usado por lo menos dos veces, este bloque tiende a ser reusado muchas veces en el futuro cercano; además, indica que los bloques más recientemente reusados son más útiles que los que fueron reusados con anterioridad.

Por otra parte, en las jerarquías inclusivas los bloques solicitados por el procesador se almacenan tanto en las caches privadas como en la cache compartida (LLC). El reemplazo de un bloque de la LLC implica su invalidación en las caches privadas, lo que suele conllevar una merma del rendimiento porque estos bloques están siendo usados por el procesador [15].

NRR se basa en estas dos observaciones y almacena el estado de todos los bloques presentes en la cache LLC según el diagrama de la Figura 2. Cuando la memoria sirve un bloque al procesador, su estado en LLC pasa a ser NR-C (NonReused, CACHED). Si el bloque es expulsado de la cache privada, su estado en LLC pasa a NR-NC (NonReused, NonCached). Y si un procesador vuelve a pedir el bloque, éste se copia a su cache privada y su estado en LLC pasa a R-C (Reused, CACHED). Finalmente, cuando el bloque es expulsado de la cache privada su estado en LLC pasa a R-NC (Reused, NonCached). Con las posteriores peticiones y expulsiones de L1, el bloque cambiará

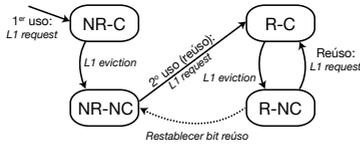


Fig. 2. Estados según reuso e inclusión de un bloque en LLC. NR, R, C y NC representan: no reusado, reusado, presente en cachas privadas, y no presente en cachas privadas, respectivamente.

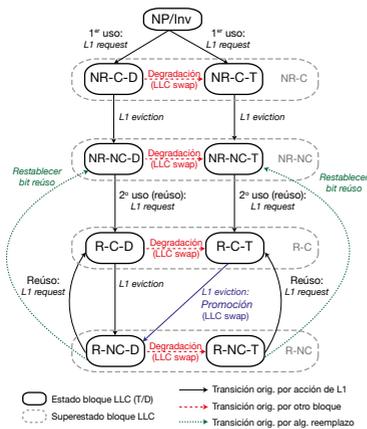


Fig. 3. Estados según reuso e inclusión de un bloque en LLC con BDOT.

entre los estados R-NC y R-C.

NRR asigna la máxima protección a los bloques presentes en las cachas privadas y entre los no presentes, a aquellos que han demostrado reuso, lo que se traduce en el siguiente orden de reemplazo: NR-NC, R-NC, NR-C, R-C. Siguiendo el principio de localidad de reuso, cuando todos los bloques NC han sido reusados (R-NC), el algoritmo de reemplazo restablece su estado a no reusado (NR-NC), de forma que bloques reusados recientemente obtienen mayor protección (transición *Restablecer bit reuso* en la Figura 2).

La información de presencia en cachas privadas puede obtenerse de diferentes formas [15], pero una manera sencilla es utilizar del vector de presencia que gestiona el protocolo de coherencia, asumiendo que los reemplazos en L1 son no silenciosos. NRR requiere un bit por cada bloque de la cache para recordar su reuso [16].

B. Política de reemplazo consciente de los fallos

Para dar soporte a BDOT, es necesario desdoblarse en dos cada estado de la política de reemplazo base: uno para entradas de etiquetas, T , y otro para entradas de etiquetas y datos, D , tal y como muestra la Figura 3. Mientras un bloque recorre los estados NR-C, NR-NC y R-C, puede estar alojado en una entrada T o D , en función de la vía que inicialmente le asigna el algoritmo de reemplazo. Sin embargo, para proteger a los bloques con reuso, cuando un bloque en estado R-C es expulsado de la cache privada, le aseguramos una entrada a defectuosa tipo D (siempre pasa a estado R-NC-D). Es decir, si residía en una entrada defectuosa tipo T (estado R-C-T), el bloque se mueve a una entrada D (transición *Promoción* en la Figura 3).

Esta transición normalmente requiere que otro bloque sea degradado de una entrada D a la entrada T que queda libre, lo que implica la pérdida del bloque en la cache de último nivel. Por lo tanto, se requiere otro algoritmo de reemplazo que seleccione un bloque en D para ser degradado a T (transición *Degradación: LCC swap* en la Figura 3). El objetivo de este algoritmo es maximizar los contenidos dentro del chip, degradando prioritariamente a bloques presentes en cachas privadas. Como objetivo secundario degradará preferiblemente a los bloques sin reuso. Por tanto, el orden de mayor a menor probabilidad de ser degradado es: NR-C-D, R-C-D, NR-NC-D, R-NC-D.

VI. METODOLOGÍA

En esta sección presentamos la metodología utilizada, incluyendo las características del sistema modelado, el entorno de simulación, y las métricas utilizadas para cuantificar los beneficios de nuestros mecanismos.

A. Sistema modelado

Nuestro sistema de referencia es un chip multiprocesador (CMP) teselado con una jerarquía de cache inclusiva de dos niveles. El segundo nivel (LLC) es compartido y esta distribuido entre los ocho núcleos. Las teselas están interconectadas por medio de una malla. Cada tesela tiene un núcleo y un primer nivel de cache (L1) dividido en instrucciones y datos, y un banco de la cache compartida, todo ello conectado al router (Figura 4). El CMP incluye dos controladores de memoria colocados en los bordes del chip. La Tabla II recoge los parámetros de configuración del sistema de referencia, la jerarquía de memoria y la red de interconexión.

Assumimos una frecuencia de 1 GHz a 0.5 V (nuestro V_{th} cerca del umbral). El voltaje del módulo de DRAM no escala con el resto del sistema, y por tanto la velocidad relativa de la memoria principal con respecto al chip incrementa conforme se baja el voltaje. Este modelo es consistente con trabajos previos [9], [10].

El protocolo de coherencia se implementa sobre un directorio *full-map* con estados MESI (Modificado, Exclusivo, Compartido e Invalido). Utilizamos

TABLA II
CARACTERÍSTICAS PRINCIPALES DEL CMP MODELADO.

Núcleos	8. Ultrasparc III Plus, en orden, 1 instr/cycle, mono-hilo, 1 GHz a V_{dd} 0.5 V
Protocolo de Coherencia	MESI, directorio (full-map) distribuido en los bancos de LLC
Modelo de Consistencia	Secuencial
L1 cache	Privada, 64 KB datos e inst., 4-way, 64 B tamaño de bloque, LRU, 2 ciclos acceso en acierto
LLC cache	Compartida, inclusiva, entrelazado por dirección de bloque, 1 banco/tesela, 1 MB/banco, 16-way 64 B tamaño de bloque, NRU, 8 ciclos acceso en acierto (4 ciclos acceso etiquetas)
Memoria	2 controladores de memoria (en los bordes del chip); Double Data Rate (DDR3 1333 MHz) 2 canales, 8 Gb/canal, 8 bancos, 8 KB tamaño de página, política <i>open page</i> ; acceso <i>row</i> 50 ciclos
Red on-chip	Malla, 2 redes virtuales (VN); peticiones y respuestas 16-byte <i>flit</i> , 1 ciclo latencia salto, routing en dos etapas

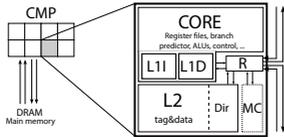


Fig. 4. Multiprocesador de 8 núcleos modelado.

notificación explícita de expulsión de bloques tanto compartidos como exclusivos. Las caches L1 están construidas con celdas robustas y funcionan a voltajes bajos sin errores inducidos por variaciones paramétricas [6]. Los bancos de LLC están contruidos con celdas convencionales (6T SRAM) y, por lo tanto, son sensibles a fallos. Como en estudios previos [9], [10], asumimos que el *array* de etiquetas es robusto utilizando, por ejemplo, celdas 8T [17].

B. Entorno de simulación y cargas de trabajo

Con respecto al entorno de simulación, usamos Simics [18] con GEMS para modelar la jerarquía de memoria on-chip y la red de interconexión [19], y DRAMSim2 para un modelo detallado de DDR3 DRAM [20]. Para modelar latencias utilizamos McPAT [21].

Utilizamos un conjunto de 20 cargas de trabajo multiprogramadas que son combinaciones aleatorias de los 29 programas que componen SPEC CPU 2006 [22], sin distinguir entre los intensivos en cálculo con enteros y con coma flotante. Cada aplicación aparece en media 5.5 veces con una desviación estándar de 2.5. Para localizar el final de la fase de inicialización de cada aplicación, los programas corrieron con la entrada de referencia en una máquina real usando contadores de rendimiento. Para asegurarnos que ninguna aplicación se encuentra en su fase de inicialización, cada carga multiprogramada ejecutó tantas instrucciones como la fase de inicialización más larga, y se creó un *checkpoint* en ese lugar. A partir de ahí, corremos simulaciones detalladas que incluyen 300 millones de ciclos para calentar la jerarquía de memoria y 700 millones de ciclos de generación de estadísticas.

Creamos mapas de fallos aleatorios y ejecutamos simulaciones Monte Carlo para asegurarnos que los resultados se encuentran dentro de un error del 5% con un nivel de confianza $(1-\alpha) = 0.95$. El número de

muestras se incrementa si es necesario para alcanzar el error. Calculamos si una celda falla o no de forma aleatoria e independientemente del resto de las celdas. De esta manera obtenemos mapas de fallos con un número similar de celdas defectuosas, pero en diferentes localizaciones. Para poder compararnos con *block disabling* en un escenario poco favorable (C2 ó C3), asumimos que al menos una de las vías es robusta (no tiene fallos), aunque nuestras técnicas funcionan incluso si no hay ninguna entrada operativa.

C. Métricas

Por último, para cuantificar los beneficios de nuestros mecanismos, vamos a utilizar dos métricas de rendimiento: fallos de la LLC por cada mil instrucciones (MPKI) y *speedup*.

Nuestro sistema de referencia (*Base*) es un sistema donde la LLC está construida con celdas robustas, es decir, que no presentan fallos al operar a bajo voltaje, y que no tienen ninguna penalización con respecto a las celdas C2. Podemos decir que esta configuración es “no realista”, ya que para conseguir una LLC con celdas robustas necesitaríamos incrementar el tamaño de las celdas o utilizar celdas con más transistores. Esta configuración representa una cota máxima del rendimiento alcanzable.

El MPKI nos muestra el aumento del tráfico a la memoria fuera de chip. Para calcular el MPKI relativo de cada mezcla, sumamos el número total de fallos (peticiones a memoria) en la LLC para una configuración dada y lo dividimos entre el MPKI del sistema base.

Para entender cómo afectan las diferentes técnicas al rendimiento global del sistema, calculamos el *speedup* para cada mezcla como se muestra en la Ecuación 1, donde IPC_i^A es el número de instrucciones ejecutadas por ciclo para el programa i cuando corre en el sistema A , e IPC_i^{Base} es el número de instrucciones ejecutadas por ciclo cuando corre en el sistema de referencia *Base*.

$$Speedup = \sqrt[n]{\prod_{i=1}^n \frac{IPC_i^A}{IPC_i^{Base}}} \quad (1)$$

Como las diferentes cargas de trabajo ejecutan el mismo número de ciclos tanto en el sistema *Base* como en las distintas configuraciones, utilizamos la media aritmética para calcular el rendimiento medio

(MPKI y *speedup*) de todas las mezclas para una configuración dada.

VII. RESULTADOS

En esta sección presentamos los resultados de la evaluación de nuestras técnicas de gestión de contenidos utilizando la metodología expuesta en la sección VI.

Como ya hemos comentado en la sección VI, ofrecemos datos relativos a un sistema base que posee una LLC ideal, implementada con celdas robustas, capaces de operar sin fallos a muy bajo voltaje sin penalización de ningún tipo. Las figuras muestran la media de los resultados obtenidos para cada una de las 20 mezclas. Añadimos como referencias los resultados obtenidos por *block disabling* y reemplazo NRU [23], solución con coste similar a nuestras propuestas. También nos comparamos con *Word disabling* [10], solución más compleja y de mayor coste.

En la subsección VII-A se presentan los datos de MPKI y en la subsección VII-B se analiza el *speedup*. En cada sección mostramos datos de los siguientes modelos:

- BASE o Celda Robusta: sistema con LLC construida con celdas robustas. Es nuestra base de comparación (todos los datos son relativos a este sistema) y representa una cota superior para el rendimiento.
- BD-NRU: sistema real con *block disabling*, reemplazo NRU [23] (sección III).
- BDOT-NRU: sistema real con *block disabling with operational tags*, reemplazo NRU (sección IV).
- BDOT-NRR: sistema real con *block disabling with operational tags*, reemplazo NRR [16] (sección V-A).
- BDOT-NRR-FA: sistema real con *block disabling with operational tags*, reemplazo NRR Fault Aware (sección V-B)
- WD: *Word disabling* [10].

A. Análisis de resultados en términos de MPKI

En esta sección presentamos, para cada configuración de LLC, los datos de MPKI normalizados al sistema BASE. La Figura 5 muestra la media para las 20 mezclas descritas en sección VI.

BD-NRU puede ser una solución válida para caches con pocas entradas defectuosas como las construidas con celdas C6, en las que produce una penalización media en términos de MPKI del 23,9%. Sin embargo, esta penalización aumenta rápidamente al aumentar el número de entradas defectuosas llegando a un 136% para celdas C2. El uso de las entradas defectuosas de LLC para guardar la información de coherencia de bloques presentes en los niveles privados, permite a BDOT-NRU disminuir significativamente el número de fallos respecto a BD-NRU en caches con muchos bloques defectuosos (C2). Pero, BDOT-NRU va peor que BD-NRU para el resto de celdas.

Los algoritmos de reemplazo basados en reuso han demostrado ser eficientes en caches compartidas de último nivel [16]. BDOT-NRR consigue mejorar los

resultados de BDOT-NRU en el contexto de caches con entradas defectuosas, pero su rendimiento todavía se encuentra lejos del obtenido por BD-NRU, excepto para la celda C2. Recordemos que, en este modelo, la asignación de bloques a entradas es ciega, solo depende del algoritmo de reemplazo NRR y no es consciente de qué entradas están sanas y cuales son defectuosas. La asignación de un bloque con reuso a una entrada de LLC defectuosa provoca que todas las peticiones a dicho bloque dirigidas a LLC se conviertan en peticiones a memoria principal. Además, debido al reemplazo basado en reuso, este bloque permanecerá mucho tiempo en la entrada defectuosa.

BDOT-NRR-FA soluciona este problema añadiendo la información de bloques defectuosos al algoritmo de reemplazo/colocación. La penalización en términos de MPKI disminuye drásticamente respecto a BD-NRU, pasando de 23,9%, 30,6%, 44,1%, 56,7% y 136% en BD-NRU para las celdas C6, C5, C4, C3 y C2 respectivamente, a 5,8%, 10,9%, 21%, 27,9% y 48,3% en BDOT-NRR-FA.

El MPKI conseguido por WD es casi invariante entre las distintas celdas, pese a la gran diferencia de celdas defectuosas en cada una de ellas. Dos razones explican este comportamiento: i) una sola celda defectuosa clasifica a la entrada entera como defectuosa y obliga a combinarla con otra, y ii) el número de celdas defectuosas por entrada es siempre relativamente pequeño (tres en media para la peor celda: C2 [24]), por lo que casi siempre se puede conseguir un almacenar un bloque combinando dos entradas defectuosas. Por consiguiente, el número medio de vías por conjunto conseguido por WD es de 8. En la comparación con WD, BDOT-NRR-FA consigue menor MPKI con celdas C6, C5, C4 y C3 con diferencias de 20%, 16,1%, 8,5% y 3,4% respectivamente. WD solo supera a BDOT-NRR-FA en caches con muchos bloques defectuosos como las fabricadas con celdas C2, donde el 90% de las entradas tienen al menos un fallo. Sin embargo, la complejidad y el coste en el área de WD son mayores que los de BDOT-NRR-FA, en donde sólo se necesita un bit para indicar si la entrada es defectuosa (*block disabling*) y un bit para señalar el reuso de los bloques (NRR).

B. Análisis de resultados en términos de *speedup*

En esta sección presentamos, para cada configuración de LLC, los datos de *speedup* normalizados al sistema BASE. La Figura 6 muestra la media para las 20 mezclas descritas en la sección VI. Los resultados en términos de *speedup* correlan perfectamente con los de MPKI analizados en la sección previa. BD provoca pérdidas respecto a la cache ideal que varían entre el 3,3% para C6 y el 17,7% para C2. BDOT-NRR-FA reduce las pérdidas hasta el 1,3%, 2%, 3,4%, 4,3% y 6,9% para las celdas C6, C5, C4, C3 y C2, respectivamente. Al igual que en MPKI, WD solo mejora a BDOT-NRR-FA con la celda C2 en un 2%.

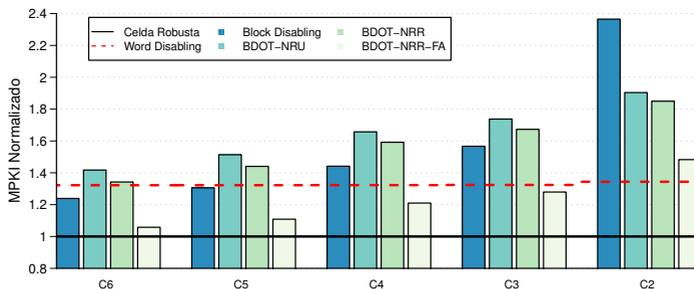


Fig. 5. MPKI normalizado (media mezclas SPEC) con respecto a una celda robusta para las distintas propuestas.

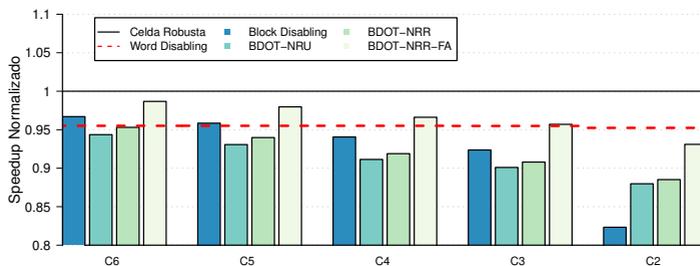


Fig. 6. Speedup normalizado (media mezclas SPEC) con respecto a una celda robusta para las distintas propuestas.

VIII. TRABAJO RELACIONADO

Las celdas SRAM con 6 transistores no funcionan de forma fiable cuando el voltaje escala a valores cerca de la tensión umbral, ya que no se pueden garantizar los márgenes necesarios para la estabilidad de escrituras y lecturas, especialmente en un contexto de V_{th} variable. Las propuestas previas para mitigar el impacto de los fallos de SRAM debidos a variaciones paramétricas se pueden dividir en dos grupos: técnicas a nivel de circuito y técnicas a nivel arquitectónico.

Las soluciones a nivel de circuito incluyen métodos que hacen que la celda sea más fiable incrementando su tamaño [5], o añadiendo circuitería adicional (i.e., más transistores) [17], [4]. El principal problema de incrementar el tamaño de los transistores o el número de transistores por celda es el coste en área (disminuye la densidad) y el aumento del consumo. Por tanto, estas técnicas no se suelen aplicar en la LLC.

Las soluciones a nivel de micro-arquitectura incluyen técnicas que añaden redundancia a través de códigos de corrección de errores, deshabilitar contenido o mecanismos basados en la duplicación (y combinación). Nuestra propuesta se enmarca en esta categoría.

Los códigos de corrección de errores (ECC) se uti-

lizan para protección ante errores *soft*. Algunos trabajos proponen utilizar ECC para protección ante errores *hard* cuando las caches operan a voltajes ultrabajos [25], [8]. Los ECC se optimizan normalmente para minimizar el espacio que ocupan a costa de una lógica más compleja en las fases de detección y corrección de errores. Modificar el ECC para que sea capaz de detectar y corregir más errores implica lógica más compleja en la fase de decodificación, o incrementar el número de bits almacenados [8]. Nuestra propuesta es ortogonal al uso de ECC para proporcionar más entradas funcionales; de hecho se adapta al número de entradas de datos funcionales y no funcionales.

En cuanto a *block disabling* [11], Lee *et al.* estudian la degradación en rendimiento al deshabilitar entradas de cache, conjuntos, vías, puertos, o el banco de cache al completo, en un entorno mono procesador [7]. Ladas *et al.* proponen añadir una cache víctima para compensar por la asociatividad perdida [13]. Nuestro mecanismo también se basa en *block disabling*, pero no requiere ninguna estructura adicional.

Ghasemi *et al.* utilizan tamaños de celda heterogéneos, de modo que cuando la cache opera a voltajes bajos las vías o conjuntos con celdas SRAM pequeñas se desactivan cuando empiezan a fallar [26].

En la misma línea, Khan *et al.* proponen el diseño de una cache en donde una pequeña partición se implementa con celdas robustas y el resto con celdas no robustas (de menor tamaño). Las celdas robustas se utilizan para almacenar bloques sucios, para lo que modifican la política de reemplazo, guiando la inserción de los bloques en función de si la petición es un *load* o un *store* [27]. Zhou *et al.* combinan celdas de repuesto, tamaños de celdas heterogéneas y ECC para crear un diseño híbrido que mejora la eficiencia de dichas técnicas aplicadas de forma individual [5]. Al contrario que estos trabajos, nosotros no asumimos que existen celdas robustas y guiamos la inserción de bloques basándonos en el reúso.

Las entradas de la cache se pueden deshabilitar a una granularidad más fina (por ejemplo, a nivel de palabra), a cambio de incrementar la complejidad de acceso a cache. *Word disabling* monitoriza fallos a nivel de palabra y combina dos entradas consecutivas para obtener una entrada sin fallos, lo que divide entre dos la asociatividad y la capacidad de la cache resultante [10]. Abella *et al.* también deshabilitan contenido a nivel de palabra, redirigiendo los accesos a palabras deshabilitadas al siguiente nivel de la jerarquía; esta técnica tiene sentido sólo en el primer nivel de la jerarquía en donde los accesos son a nivel de palabra [28]. Palfman *et al.* siguen una política similar buscando palabras almacenadas en celdas defectuosas en otras estructuras micro-arquitectónicas, como por ejemplo la *store queue* o el *miss status holding register* [29]. Ferreron *et al.* proponen comprimir los bloques y almacenarlos en entradas con fallos, lo que permite utilizar la totalidad de la cache [24]. Esquemas más complejos combinan entradas con defectos utilizando un mecanismo de remapeo que se basa en la ejecución de un algoritmo para encontrar entradas que se puedan combinar y las correspondientes estructuras para almacenar el remapeo [9], [30], [31]. El principal inconveniente de estas técnicas es que los mecanismos de remapeo añaden un nivel de indirectación al acceso a cache (incrementando la latencia), y combinar entradas para obtener un bloque añade complejidad. Además, se necesitan varios accesos a la cache para obtener un bloque completo, lo que aumenta el consumo de energía y/o la latencia por acceso a bloque. Nuestra propuesta no necesita ninguna estructura adicional o mecanismo de remapeo, sólo cambios mínimos en el protocolo de coherencia y la política de reemplazo.

En el contexto de ejecución a voltajes ultra-bajos, Keramidas *et al.* usan un predictor espacial indexado por PC para las decisiones de reemplazo entre entradas con o sin fallos en el primer nivel de la jerarquía [32]. Nuestro mecanismo de inserción no necesita ninguna predicción, lo que simplifica el hardware de manera considerable, y no contempla el uso de entradas parcialmente deshabilitadas.

En lo que se refiere a la implementación de nuestros mecanismos, debemos referirnos al trabajo de Jaleel *et al.* [15]. En jerarquías en inclusión, las caches privadas filtran la localidad temporal y bloques que

están siendo usados por el procesador son degradados en la pila de reemplazo de la LLC para ser finalmente expulsados. Jaleel *et al.* abordan este problema protegiendo bloques en las caches privadas para prevenir que sean reemplazados en la LLC a través de varias técnicas, que incluyen: enviar pistas a la LLC, identificar la localidad temporal por medio de invalidación temprana y consultar a las caches privadas acerca de la presencia de bloques. Nosotros también protegemos las copias privadas en todas las configuraciones (incluyendo *Base*) utilizando la información del protocolo de coherencia y asumiendo que las expulsiones de L1 no son silenciosas.

Albericio *et al.* basan las decisiones de reemplazo en la localidad de reúso [16]. Proponen el algoritmo *Not-Recently Reused* (NRR) que protege los bloques presenten en caches privadas y los bloques que han demostrado reúso en la LLC. Su implementación sencilla pero eficiente mejora el rendimiento de otras técnicas más complejas como RRIP [33].

IX. CONCLUSIONES

Uno de los mayores desafíos para continuar mejorando el rendimiento y la eficiencia energética de los procesadores es reducir V_{dd} . El mayor freno a la hora de hacerlo son las celdas de memoria, ya que pequeñas disminuciones de V_{dd} rápidamente las vuelven inestables e inválidas para almacenar cualquier contenido.

Disminuir la capacidad de las memorias caches mediante la invalidación de celdas permite reducir V_{dd} a costa de empeorar el rendimiento y la eficiencia energética del procesador porque aumenta considerablemente la actividad de la memoria principal. Sin embargo si se explota adecuadamente la replicación de contenidos en jerarquías de memoria inclusiva se puede paliar este problema. Además, con políticas de gestión de contenidos conscientes de los fallos en celdas se puede casi alcanzar el rendimiento de un sistema trabajando con un V_{dd} mayor a la par que se reduce el consumo energético. Este trabajo fusiona ambas propuestas con un aprovechamiento inteligente de los mecanismos de coherencia existentes para permitir operar a la memoria a un V_{dd} bajo.

En un entorno de multiprocesador homogéneo las técnicas propuestas consiguen mejorar *block disabling* reduciendo MPKI hasta un 37,3%, lo que se traduce en una ganancia de rendimiento del 13%. Respecto a una celda ideal, la pérdida de rendimiento se encuentra entre el 1,2 y el 6,9%, frente al 3,3 y el 17,7% de *block disabling*.

AGRADECIMIENTOS

Este trabajo ha sido financiado en parte por los proyectos gaZ: T48 grupo de investigación (Gobierno de Aragón y Unión Europea) y TIN2013-46957-C2-1-P, TIN2015-65316-P, y Consolider NoE TIN2014-52608-REDC (Gobierno de España y Unión Europea).

REFERENCIAS

- [1] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *Proceedings of the IEEE*, vol. 87, no. 4, pp. 668-678, April 1999.
- [2] M. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8-19, Sept. 2013.
- [3] R. Dreslinski, M. Wiczkowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proc. of the IEEE*, vol. 98, no. 2, pp. 253-266, Feb. 2010.
- [4] J. Kulkarni, K. Kim, and K. Roy, "A 160mV robust schmitt trigger based subthreshold SRAM," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 10, pp. 2303-2313, Oct. 2007.
- [5] S.-T. Zhou, S. Katariya, H. Ghasemi, S. Draper, and N. S. Kim, "Minimizing total area of low-voltage SRAM arrays through joint optimization of cell size, redundancy, and ECC," in *IEEE Int. Conf. on Computer Design*, 2010, pp. 112-117.
- [6] R. Kumar and G. Hinton, "A family of 45nm IA processors," in *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*, 2009, pp. 58-59.
- [7] H. Lee, S. Cho, and B. Childers, "Performance of graceful degradation for cache faults," in *IEEE Computer Society Annual Symp. on VLSI*, 2007, pp. 409-415.
- [8] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving cache lifetime reliability at ultra-low voltages," in *42nd Annual IEEE/ACM Int. Symp. on Microarchitecture*, 2009, pp. 89-99.
- [9] A. Ansari, S. Feng, S. Gupta, and S. Mahlke, "Archipelago: A polymorphic cache design for enabling robust near-threshold operation," in *IEEE 17th Int. Symp. on High Performance Computer Architecture*, 2011, pp. 539-550.
- [10] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off cache capacity for reliability to enable low voltage operation," in *35th Annual Int. Symp. on Computer Architecture*, 2008, pp. 203-214.
- [11] G. Sohi, "Cache memory organization to enhance the yield of high performance VLSI processors," *IEEE Trans. on Computers*, vol. 38, no. 4, pp. 484-492, Apr. 1989.
- [12] J. Chang, M. Huang, J. Shoemaker, J. Denoit, S.-L. Chen, W. Chen, S. Chitt, R. Gaesman, J. Leong, Y. Lukka, S. Russi, and D. Srivastava, "The 65-nm 16-MB Shared On-Die L3 Cache for the Dual-Core Intel Xeon Processor 7100 Series," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, pp. 846-852, Apr. 2007.
- [13] N. Ladas, Y. Sazeides, and V. Desmet, "Performance-effective operation below Vcc-min," in *IEEE Int. Symp. on Performance Analysis of Systems Software*, 2010, pp. 223-234.
- [14] J. L. Baer and W. Wang, "On the inclusion properties for multi-level cache hierarchies," in *15th Annual Int. Symp. on Computer Architecture*, 1988, pp. 73-80.
- [15] A. Jaleel, E. Borch, M. Bhandaru, S. C. Steely Jr., and J. Emer, "Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (TLA) cache management policies," in *43rd Annual IEEE/ACM Int. Symp. on Microarchitecture*, 2010, pp. 151-162.
- [16] J. Albericio, P. Ibáñez, V. Viñals, and J. M. Llaberia, "Exploiting reuse locality on inclusive shared last-level caches," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 38:1-38:19, Jan. 2013.
- [17] L. Chang, R. Montoye, Y. Nakamura, K. Batson, R. Eickemeyer, R. Dennard, W. Haensch, and D. Janssek, "An ST-SRAM for variability tolerance and low-voltage operation in high-performance caches," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 4, pp. 956-963, Apr. 2008.
- [18] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50-58, Feb. 2002.
- [19] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92-99, Nov. 2005.
- [20] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAM-Sim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16-19, Jan. 2011.
- [21] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd Annual IEEE/ACM Int. Symp. on Microarchitecture*, 2009, pp. 469-480.
- [22] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1-17, Sept. 2006.
- [23] S. Microsystems, "UltraSPARC T2 supplement to the UltraSPARC architecture," Draft d1.4.3, Sun Microsystems Inc., 2007.
- [24] A. Ferrerón, D. Suarez, J. Alastruey, T. Monreal, and P. Ibáñez, "Concertina: Squeezing in cache content to operate at near-threshold voltage," *IEEE Trans. on Computers*, vol. 65, no. 3, pp. 755-769, Mar. 2016.
- [25] A. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu, "Energy-efficient cache design using variable-strength error-correcting codes," in *38th Annual Int. Symp. on Computer Architecture*, 2011, pp. 461-471.
- [26] H. Ghasemi, S. Draper, and N. S. Kim, "Low-voltage on-chip cache architecture using heterogeneous cell sizes for high-performance processors," in *IEEE 17th Int. Symp. on High Performance Computer Architecture*, 2011, pp. 38-49.
- [27] S. M. Khan, A. R. Alameldeen, C. Wilkerson, J. Kulkarni, and D. A. Jimenez, "Improving multi-core performance using mixed-cell cache architecture," in *IEEE 19th Int. Symp. on High Performance Computer Architecture*, 2013, pp. 119-130.
- [28] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. González, "Low Vccmin fault-tolerant cache with highly predictable performance," in *42nd Annual IEEE/ACM Int. Symp. on Microarchitecture*, 2009, pp. 111-121.
- [29] D. J. Palfreman, N. S. Kim, and M. H. Lipasti, "iPatch: Intelligent fault patching to improve energy efficiency," in *IEEE 21st Int. Symp. on High Performance Computer Architecture*, 2015, pp. 428-438.
- [30] C.-K. Koh, W.-F. Wong, Y. Chen, and H. Li, "Tolerating process variations in large, set-associative caches: The buddy cache," *ACM Trans. on Architecture and Code Optimization*, vol. 6, no. 2, pp. 8:1-8:34, July 2009.
- [31] T. Mahmood, S. Kim, and S. Hong, "Macho: A failure model-oriented adaptive cache architecture to enable near-threshold voltage scaling," in *IEEE 19th Int. Symp. on High Performance Computer Architecture*, 2013, pp. 532-541.
- [32] G. Keramidas, M. Mavropoulos, A. Karvouniari, and D. Nikolos, "Spatial pattern prediction based management of faulty data caches," in *Conference on Design, Automation & Test in Europe*, 2014, pp. 60:1-60:6.
- [33] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," in *37th Annual Int. Symp. on Computer Architecture*, 2010, pp. 60-71.

Arquitecturas del procesador, multiprocesadores y chips multinúcleo

Exploración y optimización energética de arquitecturas heterogéneas con el framework gem5

Marcos Horro¹, Gabriel Rodríguez¹, Juan Touriño¹

Resumen— El futuro de la arquitectura de computadores presenta retos importantes. El actual ritmo de miniaturización nos acerca cada vez más a los límites físicos del tamaño de los transistores y, por lo tanto, es fundamental desarrollar nuevas alternativas para sobreponerse al estancamiento del “escalado” en los *chips*. Así, los sistemas heterogéneos se presentan como una solución factible. El framework gem5 permite la simulación de todo tipo de arquitecturas y configuraciones de memoria. De esta manera, hemos adaptado e integrado memorias *scratchpad* al simulador y, haciendo uso de un sistema de programación lineal, hemos obtenido resultados satisfactorios a nivel de eficiencia energética y temporal respecto a arquitecturas que carecen de esta configuración.

Palabras clave— memorias *scratchpad*, memorias caché, arquitecturas heterogéneas, dark silicon, power wall, memory wall, gem5

I. INTRODUCCIÓN

EL fin del escalado de Dennard ha frenado de manera notable las mejoras de rendimiento en los procesadores, en beneficio del desarrollo de arquitecturas paralelas. Aun con este cambio de paradigma, se espera que los sistemas paralelos o *multicore* vean limitado su crecimiento por el *power wall*, es decir, por los límites térmicos y energéticos. Se ha presentado así un futuro dominado por arquitecturas que integrarán un gran número de módulos especializados tales como unidades vectoriales, jerarquías de memoria configurables e incluso la integración de diferentes tecnologías dentro del mismo *chip*. En este nuevo modelo, usuarios y compiladores en conjunto deberán cooperar para determinar qué módulo o qué partes del procesador utilizar en un momento dado. El área apagada de un *chip* en un momento determinado se conoce como *dark silicon* [1]. El concepto es adecuarse a un coste energético máximo, con el fin de ejecutar una aplicación dada bajo unas restricciones de rendimiento mínimo o QoS.

Este trabajo se centra en el análisis y simulación de arquitecturas heterogéneas haciendo uso del framework gem5, un simulador ampliamente reconocido y desarrollado por entidades y organizaciones de la categoría de Intel, ARM o Google. Una de las grandes ventajas de esta herramienta es la posibilidad de crear nuevos módulos o modificar los ya creados habilitando la exploración arquitectural. En este trabajo se proponen extensiones con el fin de simular memorias *scratchpad* en el framework y así evaluar el *trade-off* energético de las diferentes tecnologías y

posibles configuraciones en la jerarquía de memoria. Para ello, se ha desarrollado un modelo matemático que decide en qué módulo de memoria se debería alojar una determinada variable de programa. Con este motivo se analiza la localidad que presenta la susodicha variable y, en caso de que su reuso dé lugar a conflictos y no sea explotable por el algoritmo de reemplazo LRU de las memorias caché, si las restricciones de tamaño lo permitiesen se alojaría en *scratchpad*. De esta forma, el algoritmo también permitiría escoger entre diferentes memorias *scratchpad* cuál es la que tiene un mejor rendimiento teórico o mayor eficiencia energética.

Los experimentos realizados haciendo uso de memorias *scratchpad* y caché con tecnologías SRAM y STT-RAM avalan la fiabilidad de esta aproximación, como se expondrá en las siguientes secciones.

II. ESTADO DEL ARTE

Las memorias *scratchpad* destacan por su programabilidad, es decir, por la posibilidad de controlar los datos alojados en su región mediante software, y contrastan con las memorias caché, cuyos contenidos se gestionan de forma automática, habitualmente por un algoritmo de tipo LRU. Su uso está extendido en sistemas empotrados, como por ejemplo sistemas de tiempo real, en los que el uso de caché es prohibitivo dado que el patrón de accesos al flujo de datos no tiende a favorecer la localidad, incluso provocando que la caché sea contraproducente en algunos casos [2] [3]. Sin embargo, las memorias caché están ampliamente extendidas en procesadores de propósito general, véase arquitecturas convencionales como x86. Con todo, existen arquitecturas comerciales de propósito específico que han implementado estas memorias programables o *scratchpad*:

- Cell IBM [4]: esta arquitectura se utilizó en los nodos del supercomputador MareNostrum del BSC y en las consolas PlayStation 3.
- PlayStation 2 [5]: incluía unas pequeñas memorias *scratchpad* a las cuales podían acceder tanto la GPU como la CPU.
- Sistemas de procesamiento digital (DSP) [6]: la arquitectura SODA incluye tanto memorias *scratchpad* privadas para cada unidad de procesamiento como global a la que diferentes unidades pueden acceder.
- Arquitectura *Knights Landing* de Intel Xeon Phi [7]: esta arquitectura hace uso de unas memorias llamadas MCDRAM. La idea es permi-

¹Dpto. de Electrónica y Sistemas, Universidade da Coruña, e-mail: {marcos.horro,grodriguez,juan}@udc.es

tir un acceso a memoria direccionable con un gran ancho de banda al procesador, por lo que el concepto es similar al del uso de una memoria scratchpad, pero a otra escala (Intel Xeon Phi es una arquitectura *manycore*).

Con todo, Cyrix 6X86MX [8], un procesador de propósito general de finales de los años noventa manufacturado por IBM, incluía una memoria “scratchpad RAM”. Este procesador del 1996 implementaba una memoria caché “primaria” de 64kB. Ésta caché podía modificar su comportamiento para manejar el contenido explícitamente como una memoria *scratchpad*. El área dedicada para este funcionamiento funcionaba como una memoria privada para la CPU.

Por otro lado, gem5 es una plataforma que permite la simulación de arquitecturas hardware. Los simuladores juegan un papel fundamental en el desarrollo y exploración de arquitecturas, dado que permiten, con un costo muy bajo en comparación a soluciones litográficas o la programación de FPGAs, una simulación razonablemente precisa. Por ello, han surgido diferentes soluciones, con diferentes granularidades, enfocadas a subsistemas concretos y otras de carácter holístico. En la Tabla I se resumen y comparan diferentes simuladores tanto de licencia privada, por ejemplo Simics, como de licencia *opensource* (en sus diferentes versiones), que son el resto.

TABLA I
 DIFERENCIAS ENTRE LOS SIMULADORES DE ACUERDO A LOS SIGUIENTES CRITERIOS: SI LA PLATAFORME SIGUE EN DESARROLLO, CONJUNTO DE INSTRUCCIONES SOPORTADAS (ISAs), TIPO DE LICENCIA Y PRECISIÓN. DATOS EXTRAÍDOS DE [9] [10]

Sim.	Des.	ISA(s)	Precisión
Simics	Sí	Alpha, ARM, M68k, MIPS, PowerPC, SPARC, x86	Funcional
SimFlex	No	SPARC, x86 (requiere Simics)	Ciclo
GEM5	No	SPARC (requiere Simics)	Timing
m5	No	Alpha, MIPS, SPARC	Ciclo
MARSS	No	x86 (requiere QEMU)	Ciclo
OVPsim	Sí	ARM, MIPS, x86	Funcional
PTLsim	No	x86 (requiere Xen y KVM/QEMU)	Ciclo
Simple Scalar	No	Alpha, ARM, PowerPC, x86	Ciclo
gem5	Sí	Alpha, ARM, x86, SPARC, PowerPC, MIPS	Ciclo

Los principales motivos de selección de gem5 frente

a otras alternativas han sido: (i) su actual desarrollo y nivel de implicación de organizaciones y entidades como pueden ser Google, Intel o ARM, además de usuarios; (ii) su licencia *opensource*; y (iii) la posibilidad de simular diferentes ISAs. Además, la modularidad de la plataforma permite integrar componentes de una manera sencilla.

III. IMPLEMENTACIÓN DE MEMORIAS SCRATCHPAD EN GEM5

El objetivo fundamental de la inclusión de las memorias scratchpad en el sistema es reducir el consumo energético. En los procesadores actuales, una gran parte del consumo se debe al uso de las cachés [11] [12], y es que estas memorias necesitan activar la región de *tags* y de datos para obtener la línea buscada, mientras que con una referencia a memoria sólo hace falta un acceso. Además, los fallos caché también implican penalizaciones en el consumo.

gem5 dispone de dos modos de funcionamiento: *full system* y *emulation system*. Este último realiza simulaciones a nivel de programa, lo cual nos interesa para realizar benchmarks sin tener que cargar una imagen de sistema operativo. En cuanto a la simulación de la jerarquía de memoria, existen dos modos: clásico y Ruby. Las ventajas del clásico son su simplicidad frente al Ruby, que está más orientado al análisis del impacto de los protocolos de coherencia caché.

Por otra parte, las memorias *scratchpad* no difieren demasiado en concepto de unas memorias convencionales, véase una memoria RAM. La única diferencia es la programabilidad de ellas. Por ello, para integrar las memorias *scratchpad* (SPM) en gem5, hemos partido de los módulos de memoria ya creados (SimpleMemory) y hemos modificado sutilmente sus parámetros e implementación para añadir latencias.

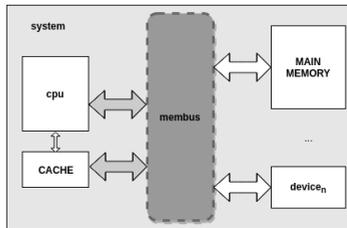


Fig. 1. Esquema arquitectura tradicional en gem5

Una vez creadas, las hemos conectado al sistema. Dado que habilitar un puerto único y específico en la CPU para SPM nos parecía una solución poco portable, dado que existen diferentes tipos de CPU en gem5, hemos decidido hacer uso de las crossbars existentes. Haciendo referencia al sistema de memoria clásico de gem5, las CPUs tienen tres puertos: un puerto principal y los dos puertos de caché de da-

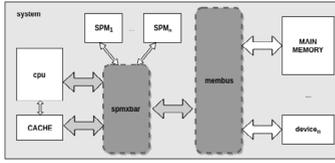


Fig. 2. Esquema arquitectural que incluye SPM en gem5

tos e instrucciones. El puerto principal, por defecto, se conecta al bus principal del que cuelgan todos los controladores de memoria, e.g. memoria principal, dispositivos de entrada/salida. De esta manera, hemos decidido modificar estas conexiones. Así hemos creado una nueva crossbar que no introduce ninguna latencia ni sobrecoste, de la cual cuelgan las memorias scratchpad. A esta crossbar se conecta la CPU y las memorias caché, y el bus original. Las Figuras 1 y 2 muestran las diferencias entre la configuración por defecto de gem5 y la arquitectura modificada.

El siguiente paso para la integración de estas memorias en el sistema es la posibilidad de acceder a ellas mediante instrucciones. Para ello se ha modificado la ISA original (en nuestro caso hemos escogido la x86, por familiaridad), de modo que se ha añadido una instrucción para reservar explícitamente memoria en esa región. Para facilitar esta reserva, se asigna de manera explícita un rango fijo de direcciones física a la SPM, que se corresponde con la siguiente dirección física a la de memoria principal hasta la suma de la capacidad de la memoria. Así, esta instrucción mapea dinámicamente la región indicada, devolviendo una referencia en el programa. Como nota adicional, esta instrucción ha sido generalizada en su modelado, para que dentro de un mismo programa se pueda referenciar a más de una memoria scratchpad.

IV. MODELO MATEMÁTICO

El hecho de que la memoria sea programable en esta arquitectura implica la toma de decisiones sobre dónde alojar una variable de programa. Es decir, decidir si una variable se aloja en memoria principal o en una de las memorias scratchpad.

Así hemos pensado que lo ideal sería plantear un problema de programación lineal, que tenga como objetivo minimizar la energía dinámica en nuestro caso. Una de las ventajas de estos sistemas es que se podrían plantear la optimización de otros parámetros cambiando, a priori, sólo la función objetivo. La función objetivo de este sistema sería la suma de los accesos a las variables, tanto de lectura como de escritura, multiplicado por el parámetro de energía de consumo de cada memoria y por la variable de decisión. Las restricciones básicas serían: (i) una variable sólo puede estar en un módulo a la vez, y (ii) la suma del tamaño de las variables que se alojen en scratchpad tienen que ser menores o iguales que el tamaño de la memoria en cuestión. Las variables, en principio, tienen un tamaño de ocho bytes considerando

variables de tipo *double*, mientras que las memorias scratchpad tienen una capacidad desde 1MB hasta los 16MB que veremos en la sección siguiente. La traducción está plasmada en las siguientes ecuaciones (ver Figura 3).

$\psi \in \Psi \equiv$ conjunto de variables

$$\begin{aligned} \min E_{DYN} &= \sum_{\psi \in \Psi} \left(M(\psi)h(\psi) + \sum_{i=1}^n S_i(\psi)f_i(\psi) \right) \\ S_i(\psi) &= \begin{cases} 0 & \psi \text{ no en } SPM_i \\ 1 & \psi \text{ sí en } SPM_i \end{cases} \\ M(\psi) &= \begin{cases} 0 & \psi \text{ no en mem principal} \\ 1 & \psi \text{ sí en mem principal} \end{cases} \\ M(\psi) + \sum_{i=1}^n S_i(\psi) &= 1; \forall \psi \in \Psi \\ \sum_{\psi \in \Psi} S_i(\psi)|\psi| &\leq |SPM_i|, \forall i \in [1, n] \\ f_i(\psi) &= \sum_{\psi \in \Psi} E_{SPM,r}(i)N_r(\psi) + E_{SPM,w}(i)N_w(\psi) \\ h(\psi) &= \sum_{\psi \in \Psi} E_{MM,r}N_r(\psi) + E_{MM,w}N_w(\psi) \end{aligned}$$

Fig. 3. Sistema de programación lineal

Dado que estamos comparando la energía dinámica que consumiría una scratchpad con la energía dinámica que consumiría la memoria principal, en principio si todas las variables de programa cupiesen en las memorias scratchpad, entonces el decisor siempre alojaría en scratchpad, con la premisa de que el uso de memoria principal es más costoso. Esta decisión puede ser contraproducente, e.g. si una caché LRU pudiera aprovechar de forma eficiente la localidad presente en los accesos a una variable determinada (acceso *cache-friendly*). Por ello, a mayores decidimos introducir una nueva restricción: forzar que una determinada variable se aloje en memoria principal. Esta restricción queda reflejada en la Ecuación (1).

$$\begin{aligned} C(\psi) &= \begin{cases} 0 & \psi \text{ no } cache\text{-friendly} \\ 1 & \psi \text{ } cache\text{-friendly} \end{cases} \quad (1) \\ M(\psi) &\geq C(\psi) \quad \forall \psi \in \Psi \end{aligned}$$

El valor de esta variable permite introducir información externa, e.g. datos obtenidos de hacer *profiling* o análisis en tiempo de compilación. En nuestro caso, con el fin de evitar un *profiling* exhaustivo, hemos decidido darle valor a esta variable en función del patrón de acceso al vector. Así hemos determinado lo siguiente: el acceso a una variable es *cache-friendly* ssi dos accesos consecutivos a un array k -dimensional para todo $k > 1$ en un conjunto de bucles anidados (o un simple bucle) se encuentran en la misma línea caché. A modo de corolario, se exponen y comentan las siguientes expresiones:

$A[i, j] \equiv$ regular y *cache-friendly*
 $A[i, B[j]] \equiv$ irregular por lo tanto no *cache-friendly*
 $A[i, j * 8] \equiv$ regular pero no necesariamente *cache-friendly*

Es destacable este último ejemplo. No sería *cache-friendly* teniendo en cuenta que trabajamos con valores que ocupan ocho bytes (tipo *double*) y que el tamaño de línea es, normalmente, de 64 bytes. Así, podemos reescribir un acceso *cache-friendly* como:

$$\exists a_i / \text{addr}(a_{i+1}) - \text{addr}(a_i) \leq |cb| \quad (2)$$

Donde a_i y a_{i+1} son dos accesos consecutivos, $\text{addr}(x)$ sería la dirección de x y $|cb|$ es el tamaño de la línea caché. Con lo anterior, hemos programado el sistema con el lenguaje de programación R.

V. RESULTADOS EXPERIMENTALES

A. Configuración experimental

Una vez implementada la arquitectura propuesta para el experimento, es importante probar su rendimiento y compararlo con otras arquitecturas. Con este propósito, hemos escogido la suite PolyBench/C [13] por ser de código abierto y tener unos kernels fáciles de analizar para nuestro modelo, i.e. unos bucles con accesos afines y fácil cálculo de número de lecturas y escrituras.

Otro aspecto importante para la correcta configuración de nuestras arquitecturas, es la selección de características de consumo de energía y latencias de nuestros módulos. Para ello hemos hecho uso de CACTI [14] y de NVSim [15]. En realidad, NVSim es una extensión de CACTI, pues funciona haciendo uso de la misma, añadiendo otras características y tecnologías. Estas herramientas nos han servido para validar las configuraciones de nuestras arquitecturas.

Así, hemos planteado dos arquitecturas principales: arquitectura de propósito general y arquitectura modificada con memoria scratchpad. La arquitectura de propósito general es, básicamente, un procesador con conjunto de instrucciones x86 y con de una jerarquía de memoria tradicional: cachés en varios niveles y memoria principal (ver Figura 1). En cuanto a la arquitectura modificada (ver Figura 2), no es más que la anterior con la integración de memorias scratchpad en la jerarquía de memoria.

En nuestro caso, hemos creado esta arquitectura sustituyendo las memorias caché de último nivel en la jerarquía tradicional. Este cambio se hace ocupando un área equivalente, es decir, se sustituye la caché L2 que ocupa un área determinada, por una memoria scratchpad que ocupe ese área. Dado que la tecnología tradicional de estas memorias caché es normalmente SRAM, exploramos otras tecnologías como STT-RAM buscando una mejora en términos de energía y de capacidad. En la Tabla II se muestran los resultados de las ejecuciones realizadas con NVSim.

La Tabla II muestra las diferentes configuraciones que hemos obtenido para las memorias. Como hemos comentado y para comparar justamente las arquitecturas, hemos ido calculando el área de la memoria caché para una capacidad determinada, y modelando una memoria RAM con tecnología STT-RAM que se adaptase a ese área. De esta manera, hemos obtenido cinco configuraciones diferentes tanto de memoria caché como de memoria scratchpad.

En cuanto a la potencia y energía de estas configuraciones, la Tabla II resume la energía por acceso de lectura y escritura y también la corriente de fuga o *leakage power*. Es destacable la diferencia de corriente de fuga en ambas tecnologías.

Así, la configuración de nuestro experimento nos permite hacer uso de memorias de mayor capacidad gracias al cambio de tecnología. Además, la potencia de la corriente de fuga se mejora por un factor de aproximadamente un cuarto, aunque también en detrimento de las latencias de acceso. Los resultados de este experimento no son más que el reflejo del *trade-off* energético y temporal propuesto en este artículo.

B. Resultados

A lo largo de este experimento hemos hablado de diferentes características: área, latencias y energía, fundamentalmente. En la sección anterior hablábamos de las diferentes configuraciones de las memorias, haciendo referencia también a su tecnología. Pero las suposiciones teóricas pueden no verse reflejadas en la ejecución de los programas.

Para la medición de tiempos en nuestras ejecuciones, hemos creado *breakpoints* en las regiones de interés del programa, pudiéndonos centrar en el *kernel* o núcleo del benchmark a ejecutar. Así, gem5 nos proporciona las mediciones para esas regiones determinadas, además del consumo en pJ para la memoria principal.

Con el fin de conseguir unos resultados finales lo más homogéneos posibles, hemos usado gem5 para obtener el número de referencias a la memoria SPM tanto de lectura como de escritura, así como del tiempo de ejecución de la región determinada. Así podemos calcular la energía dinámica consumida por las memorias scratchpad (ver Ecuación 4). Con todo, los resultados energéticos de gem5 son un poco limitados en cuanto a memorias se refiere, y por ello para obtener datos energéticos del consumo de éstas hemos utilizado McPAT [16].

A fin de adaptar la salida de gem5 a la entrada de McPAT, hemos desarrollado un pequeño analizador léxico y sintáctico (gem5McPATparse [17]), que busca en los ficheros de salida de gem5 los parámetros y estadísticas los valores que son entrada en McPAT, basándonos en la tabla de traducción desarrollada en el trabajo por Endo [18]. Así, la energía consumida por las cachés la obtenemos como el producto de la suma de la potencia dinámica y corrientes de fuga y el tiempo de ejecución (ver Ecuación 5).

Por último, la energía estática es calculada con los parámetros obtenidos de NVSim y el tiempo de

TABLA II
COMPARACIÓN DE LAS DIFERENTES TECNOLOGÍAS. CADA TUPLA CORRESPONDE A UNA CONFIGURACIÓN DE MEMORIA CACHÉ O SCRATCHPAD

		SRAM (cachés)			Energías		
caché	mm ²	Latencias (ns)			Hit/miss (pJ)	Esctr. (pJ)	Leak (mW)
kB		Lect.	Miss	Esctr.			
256	0.229	2.258	0.083	1.588	72	25	336.330
512	0.380	2.669	0.107	1.996	112	21	600.112
1024	0.741	3.452	0.144	2.773	214	36	1180.407
2048	1.343	9.989	0.149	7.941	378	24	2141.436
4096	2.619	11.52	0.222	9.037	383	290	4288.790
		STT-RAM (scratchpads)			Energía		
SPM	mm ²	Latencias (ns)			Read (pJ)	Write (pJ)	Leak (mW)
kB		Lect.	Miss	Esctr.			
1024	0.183	2.221	N.A.	5.686	195.251	205.024	84.809
2048	0.348	2.364	N.A.	5.744	228.512	242.614	146.194
4096	0.696	2.499	N.A.	5.812	276.137	290.231	292.389
8192	1.311	3.055	N.A.	6.038	388.324	383.871	568.592
16384	2.488	5.036	N.A.	7.739	516.687	465.678	640.935

ejecución del programa en cuestión (ver Ecuación 3). La suma de todas las anteriores queda reflejada en la Ecuación 6.

$$E_{STATIC}(t) = t_{exec} * P_{leakage} \quad (3)$$

$$E_{SPM} = N_r * E_{SPM,r} + N_w * E_{SPM,w} \quad (4)$$

$$P_{cache} = P_{cache,LEAK} + P_{cache,DYN}$$

$$E_{DYNcache} = P_{cache} * t_{exec} \quad (5)$$

$$E_{DYN}(t) = E_{MM} + E_{DYNcache} + E_{SPM}$$

$$E_{TOT}(t) = E_{STATIC}(t) + E_{DYN}(t) \quad (6)$$

Las ejecuciones de los *benchmarks* se han realizado en el clúster Pluton [19]. Así, hemos observado los siguientes fenómenos en las ejecuciones de los *benchmarks*:

1. *Rendimiento temporal*: las arquitecturas con *scratchpad* presentan peores resultados con tamaños de matrices pequeños, es decir, con un *working set* pequeño. Esto se debe a que el decisor considera que los vectores unidimensionales y vectores k-dimensionales con accesos no *cache-friendly* se deben alojar en *scratchpad*. La consecuencia es que, en los casos en los que la memoria caché de primer nivel podría almacenar estos datos sin perjudicar la localidad es que, el uso de *scratchpad* aumenta la latencia y la energía dinámica consumida. Es por ello que en una nueva aproximación, sería conveniente incluir este caso en el sistema a fin de considerar más escenarios. Con todo, cuando el tamaño de los vectores aumenta, el decisor juega un papel crítico, dado que la caché de primer nivel ya no tiene capacidad para alojar todos los elementos y se producen conflictos. Aquí, el uso de memorias *scratchpad* consigue mitigar en diferentes casos las penalizaciones de la memoria caché tanto por los accesos no *cache-friendly* como por

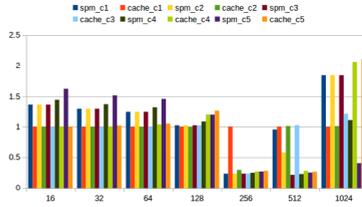


Fig. 4. Tiempo ejecución normalizado 2MM benchmark

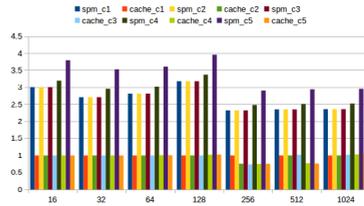


Fig. 5. Tiempo ejecución normalizado BICG benchmark

capacidad. Estos eventos se pueden observar en la Figura 4. Con todo, también existen casos en los que este comportamiento no se replica de la misma forma, véase la Figura 5. En este caso el decisor no consigue mejoras temporales dado que todo el conjunto de variables funciona bien a través de caché y perder la L2 es contraproducente (i.e., no estamos usando el SPM o sí lo estamos usando pero su contenido hubiese cabido en la memoria de primer nivel, e.g., vectores unidimensionales).

2. *Energía dinámica*: de manera similar al rendimiento temporal, la energía dinámica en arquitecturas con *scratchpad* es mayor para *working sets* pequeños, dado que la memoria caché de primer nivel tiene una energía dinámica de ac-

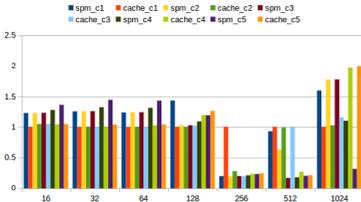


Fig. 6. Energía dinámica normalizada 2MM benchmark

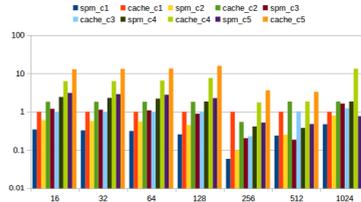


Fig. 9. Energía estática normalizada 2MM benchmark

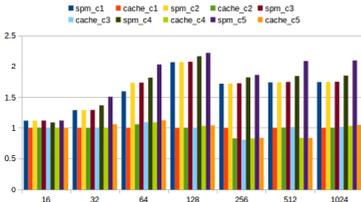


Fig. 7. Energía dinámica normalizada BICG benchmark

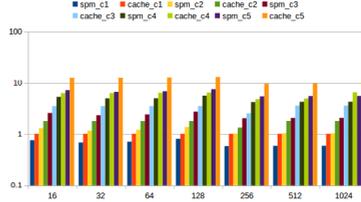


Fig. 10. Energía estática normalizada BICG benchmark

ceso menor. De manera análoga, al aumentar el tamaño de los vectores, las arquitecturas sin *scratchpad* realizan más accesos a memoria principal, aumentando la energía dinámica consumida (ver figuras 6 y 7). Con todo, también se puede observar que para el benchmark BICG, donde no existe mejora temporal, la energía consumida también es notablemente mayor, y es que la penalización de memoria principal es tanto temporal como en términos de energía dinámica consumida. Así, podemos observar que las causas de incremento de energía dinámica se identifican con la pérdida de rendimiento. De hecho, la correlación entre el tiempo de ejecución y la energía dinámica tiende a uno en todos los casos, como se puede observar en la Figura 8.

3. *Energía estática*: como era de esperar, independientemente de las diferencias temporales, en general las memorias *scratchpad* consumen menos energía estática debido a la tecnología usada

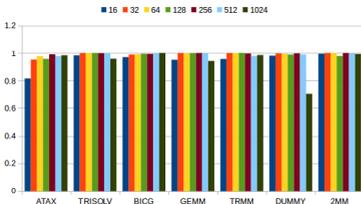


Fig. 8. Correlación tiempo ejecución - energía dinámica

(STT-RAM frente a SRAM) (ver figuras 9 y 10).

A modo de resumen, se ilustran las demás ejecuciones de los benchmarks (ver Figura 11). Es fácilmente observable la repetición de los patrones anteriormente comentados en la inmensa mayoría de ejecuciones.

VI. CONCLUSIONES

La imperiosa necesidad de hacer un uso eficiente de los recursos disponibles presenta retos interesantes. En este artículo se analizan arquitecturas de propósito general con diferentes jerarquías de memoria. Se han utilizado para este propósito herramientas de modelado y simulación ampliamente usadas y conocidas para demostrar la fiabilidad de los resultados. Se ha desarrollado un nuevo sistema para decidir en qué módulo se aloja una determinada variable del programa con el fin de conseguir una mejora en el consumo de energía. Hemos comparado memorias caché y RAM equivalentes en área con tecnologías SRAM y STT-RAM respectivamente. Los resultados experimentales demuestran que (i) sin penalizaciones temporales excesivas, se puede mejorar la eficiencia energética debido, en gran parte, a la baja corriente de fuga de las memorias *scratchpad* con tecnología STT-RAM; (ii) el uso de memorias *scratchpad* con un algoritmo de decisión sencillo también puede mejorar el rendimiento temporal; (iii) se han encontrado casos para los que el algoritmo no escoge bien la localización de las variables, aunque es fácilmente corregible teniendo en cuenta el tamaño de memoria caché de primer nivel; (iv) a medida que crece el tamaño del *working set*, las mejoras energéticas son más notables, debido en algunos casos al acierto del decisor, pero principalmente a la mayor ponderación

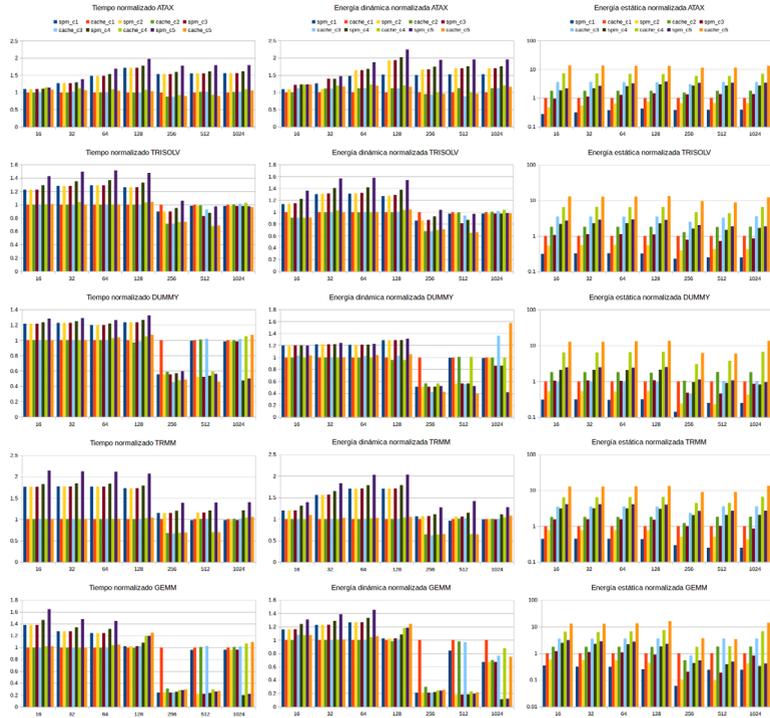


Fig. 11. Resultados de ejecución

de la corriente de fuga.

VII. AGRADECIMIENTOS

Este trabajo está financiado por el Ministerio de Economía y Competitividad de España y fondos FEDER de la UE (Proyecto TIN2013-42148-P).

REFERENCIAS

- [1] H. Esmailzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, New York, NY, USA, 2011, pp. 365–376.
- [2] Vivvy Suhendra, Chandrashekar Raghavan, Tulika Mitra, "Integrated Scratchpad Memory Optimization and Task Scheduling for MPSoC Architecture," *School of computing. National University of Singapore*, 2006.
- [3] B. Anuradha and C. Vivekanandan, "Usage of scratchpad memory in embedded systems 2014; state of art," in *Third International Conference on Computing Communication Networking Technologies (ICCCNT)*, July 2012, pp. 1–5.
- [4] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, "Synergistic processing in cell's multicore architecture," *IEEE Micro*, vol. 26, no. 2, pp. 10–24, March 2006.
- [5] T. M. Conte, *Computer Architecture: A Quantitative Approach. Appendix E: Embedded Systems*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.
- [6] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "Soda: A low-power architecture for software radio," *SIGARCH Comput. Archit. News*, vol. 34, no. 2, pp. 89–101, May 2006.
- [7] A. Sodani, "Intel® Xeon® Phi™ Processor "Knights Landing" Architectural Overview," <https://www.nercsc.gov/assets/Uploads/KNL-ISC-2015-workshop-Keynote.pdf>, 2015.
- [8] IBM, "IBM 6X86MX Microprocessor," http://datasheets.chipdb.org/IBM/x86/6x86MX/mx_full.pdf, 1998.
- [9] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli, "Accuracy evaluation of gem5 simulator system," in *7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, July 2012, pp. 1–7.
- [10] A. Gutierrez, J. Pusedesir, R. G. Dreslinski, T. Mudge, C. Sudanthi, C.D. Emmons, M. Hayenga, and N. Paver, "Sources of error in full-system simulation," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2014, pp. 13–22.
- [11] R. Bankar, S. Steinke, L. Bo-Sik, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: Design alternative for cache on-chip memory in embedded systems," in *Proceedings of the 10th International Symposium on*

- Hardware/Software Codesign*, pp. 73–78, 2002.
- [12] Gabriel Rodríguez, Juan Touriño, and Mahmut T. Kandemir, “Volatile STT-RAM Scratchpad Design and Data Allocation for Low Energy,” *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 38:1–38:26, Dec. 2014.
 - [13] L. Pouchet, “PolyBench/C; the Polyhedral Benchmark suite,” <http://web.cse.ohio-state.edu/~pouchet/software/polybench/>, 2015.
 - [14] S. J. E. Wilton and N. P. Jouppi, “CACTI: an enhanced cache access and cycle time model,” *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, May 1996.
 - [15] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, “NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, July 2012.
 - [16] S. Li, J. Ho Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multi-core and Manycore Architectures,” in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42)*, 2009, pp. 469–480.
 - [17] M. Horro, “gem5McPATparse,” <https://github.com/markshorro/gem5McPATparse>, 2016.
 - [18] F.A. Endo, *Online Auto-Tuning for Performance and Energy through Micro-Architecture Dependent Code Generation*, Theses, Université Grenoble Alpes, Sept. 2015.
 - [19] R. Rey, “Website del clúster Pluton,” <http://pluton.des.udc.es/wordpress/>, 2015.

Irrevocabilidad Relajada para Memoria Transaccional Hardware

Ricardo Quisilant, Eladio Gutiérrez, Emilio L. Zapata y Óscar Plata¹

Resumen— Los sistemas comerciales que ofrecen memoria transaccional (TM) implementan un sistema hardware best-effort (BE-HTM) con limitaciones. Es necesario programar un fallback software basado en cerrojos para asegurar el progreso de la aplicación, lo que añade complejidad al paradigma.

En este artículo se propone un nuevo tipo de irrevocabilidad hardware (un modo transaccional que marca las transacciones como no abortables) para hacer frente a las limitaciones de los sistemas BE-HTM de una manera más eficiente, y para liberar al usuario de tener que programar un fallback. Se basa en el concepto de suscripción relajada utilizada en el contexto de la programación de fallbacks basada en cerrojos, donde la transacción se suscribe al cerrojo al final de la misma en lugar de al principio. El mecanismo de irrevocabilidad relajada hardware no involucra cambios en el protocolo de coherencia y se compara con su homólogo software, que proponemos como un fallback con suscripción relajada de espera escapada. También proponemos la irrevocabilidad relajada con anticipación, un mecanismo que no se puede implementar en software, y que mejora el rendimiento de las aplicaciones con múltiples reemplazos de bloques transaccionales de caché.

La evaluación de las propuestas se lleva a cabo con el simulador Simics/GEMS junto con la suite de benchmarks STAMP, y se obtiene una mejora de rendimiento sobre el fallback del 14% al 28% para algunos benchmarks.

Palabras clave— Memoria Transaccional Hardware; Best-effort; Irrevocabilidad Relajada; Fallback

I. INTRODUCCIÓN

LA memoria transaccional (TM) es un paradigma de programación cuyo propósito es facilitar la programación concurrente y maximizar el paralelismo en multiprocesadores de memoria compartida. Después de veinte años de su publicación por Herlihy y Moss [1], la TM hardware (HTM) se incluye en los procesadores Intel Haswell [2] e IBM BlueGene/Q [3] en 2013. IBM también lanza dos sistemas HTM diferentes en sus procesadores System z [4] y Power 8 [5]. Estos sistemas son una forma best-effort de HTM (BE-HTM) donde las transacciones que no encuentran impedimentos se ejecutan en hardware, pero las que encuentran alguna limitación, como fallos de capacidad, interrupciones o conflictos persistentes, tienen que ser ejecutadas por un fallback para asegurar el progreso de la aplicación.

El fallback de una transacción tiene que ser programado por el usuario en la mayoría de estos sistemas, lo que va en contra del propósito principal del paradigma TM, la simplicidad. Sin embargo, BlueGene/Q elimina la necesidad de fallback por medio de un runtime software que implementa irrevocabilidad [6]. Siempre que una transacción no pueda fi-

nalizar después de un número de reintentos se hace irrevocable para que no pueda ser abortada. En este modo, el sistema puede ser incapaz de mantener información transaccional de la transacción por lo que los demás hilos de ejecución deben parar.

En este artículo se propone la *irrevocabilidad relajada* como un modo de irrevocabilidad mejorado basado en el concepto de suscripción relajada [7] que se usa en la programación de fallbacks en sistemas BE-HTM [8]. En una implementación de fallback simple con un solo cerrojo global las transacciones se suscriben al cerrojo leyéndolo al principio. La suscripción relajada se hace al final de la transacción permitiendo más paralelismo. La ejecución es segura ya que la suscripción al cerrojo, el aislamiento de las transacciones y el aislamiento fuerte con respecto al código no transaccional [9] abortan las transacciones que hacen algún acceso indebido.

Las contribuciones de este artículo son las siguientes:

- Un mecanismo de irrevocabilidad relajada: Se permite paralelismo entre las transacciones normales y la irrevocable, pero las primeras han de parar antes de finalizar y esperar a que acabe la irrevocable. De esta manera se preserva el cómputo realizado a no ser que un conflicto las haga abortar. El modo irrevocable se arbitra por un protocolo basado en token independiente del protocolo de coherencia.
- Un fallback de suscripción relajada con espera escapada: Es el homólogo software del mecanismo anterior. Se trata del fallback propuesto en [8] con una espera escapada que retrasa la finalización de la transacción hasta que acabe la transacción que está en el fallback. El sistema transaccional ha de permitir instrucciones escapadas dentro de las transacciones [10].
- Un mecanismo de irrevocabilidad relajada con anticipación de reemplazo: Se realizan ligeras modificaciones al protocolo de coherencia para anticipar un reemplazo de un bloque transaccional de caché, de manera que se pide la irrevocabilidad antes de que se produzca. Este mecanismo no se puede implementar en software.

Con estas propuestas se pretende tener una solución BE-HTM que no requiera un esfuerzo extra por parte del usuario y que maximice el paralelismo. La evaluación con el sistema simulado Simics [11]/GEMS [12] y con la suite de benchmarks STAMP [13] muestra un rendimiento mejorado para ciertos benchmarks de hasta el 28%.

¹Dpto. de Arquitectura de Computadores, Univ. Málaga, e-mail: {quisilant, eladio, zapata, oplata}@uma.es.

II. TRABAJO RELACIONADO

Intel Haswell [2] e IBM System z [4] y Power 8 [5] incluyen sistemas BE-HTM que requieren un fallback software para garantizar el progreso de las aplicaciones transaccionales. Estos sistemas usan las cachés privadas para mantener los datos transaccionales y se basan en el protocolo de coherencia para detectar conflictos. En cuanto a las instrucciones de escape [10], sólo Power 8 permite una forma de instrucciones no transaccionales dentro de transacciones a través de su modo transaccional suspendido [14].

IBM Blue Gene/Q [3] usa una solución diferente que utiliza la caché L2 para almacenar los datos transaccionales, gracias a su caché asociativa por conjuntos con multiversión, donde un bloque puede estar a su vez en forma transaccional y no transaccional. Para asegurar el progreso de las transacciones utiliza un modo de irrevocabilidad implementado en un runtime software. Si una transacción supera un número de abortos dada entra en irrevocabilidad, y su identificador se inserta en una tabla hash para que sucesivas ejecuciones de la misma sólo permitan un aborto antes de entrar en irrevocabilidad. BG/Q aborta la transacción en un fallo de capacidad a diferencia de nuestro sistema con anticipación de reemplazo. Además, puede sufrir el problema de contención debido al cerrojo con el que implementa la irrevocabilidad.

En lo que se refiere a fallbacks software, Calciu et al. [8] proponen el fallback con suscripción relajada para sistemas como Haswell, que proporcionan aislamiento fuerte, y hardware sandboxing. Su fallback favorece el paralelismo entre las transacciones que no han tomado el fallback y aquella que lo ha tomado. Pero si a la hora de finalizar una transacción se comprueba en la suscripción al cerrojo que el fallback sigue en ejecución se aborta la transacción.

La irrevocabilidad en el contexto de HTM fue propuesta en un principio para asegurar el progreso en el sistema BE-HTM TCC [15]. Blundell et al. [16] la introducen para ejecutar las transacciones que exceden los recursos. Su sistema OneTM-Serialized es una implementación simplificada que no permite la ejecución de transacciones que no han excedido los recursos ni la ejecución de código transaccional en paralelo con la irrevocable. Ofrecen otra versión de su sistema llamada OneTM-Concurrent que permite más concurrencia, pero para su implementación necesitan incluir información transaccional en memoria principal para que se puedan detectar conflictos entre el código (no) transaccional y la transacción irrevocable. Nuestra propuesta de irrevocabilidad consigue una concurrencia similar sin necesidad de una modificación tan drástica.

III. ARQUITECTURA DEL SISTEMA

La arquitectura del sistema que hemos utilizado en este artículo se muestra en la Fig. 1. El sistema usa la caché L1 privada para almacenar los valores nuevos de las escrituras transaccionales, mientras que los

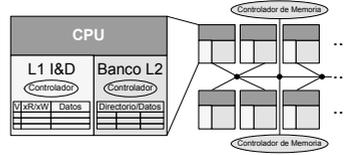


Fig. 1. Arquitectura del sistema base BE-HTM.

valores antiguos se mantienen en la L2. Se guardan un par de bits de lectura y escritura transaccionales (xR/xW) por bloque de caché L1. Estos bits se pueden borrar en un solo ciclo cuando se aborta o se finaliza una transacción. En caso de abortos, los bloques cuyo bit de escritura transaccional está a 1 también son invalidados. El protocolo de caché mantiene aislamiento fuerte [9] e implementa una política de detección de conflictos eager. La política de resolución de conflictos es requester-wins, es decir, el que pide el dato aborta al que lo tiene.

El protocolo de caché está basado en directorio y está modificado para soportar el sistema transaccional. Las modificaciones son:

- *Copiar en la primera escritura transaccional:* Un bloque modificado en L1 tiene que ser guardado en L2 antes de que una transacción escriba en él para que la L2 mantenga la copia antigua más reciente.
- *Abortar en reemplazos:* Un reemplazo de un bloque transaccional supone abortar la transacción, incluso si el reemplazo en L1 es debido a un reemplazo en L2 por la propiedad de inclusión.
- *Servir los datos antiguos desde la L2:* Si una transacción requiere un dato de una transacción que fue abortada debe de ser servido por la caché L2. Un paquete reenviado desde la L2 a una L1 será devuelto por esta a la L2 para que sirva el dato.

Por último, el sistema BE-HTM permite instrucciones de escape [10] y es implícitamente transaccional, lo que quiere decir que todas las instrucciones incluidas en una transacción son tomadas como transaccionales, excepto aquellas que estén escapadas.

IV. IRREVOCABILIDAD RELAJADA

El mecanismo de irrevocabilidad relajada implica cambios en los procedimientos de inicio y finalización de una transacción, e involucra un protocolo de comunicación de irrevocabilidad para indicar a todos los núcleos del multiprocesador el cambio a modo irrevocable por parte de una transacción. Sin embargo, no se modifica el protocolo de coherencia ni el conjunto de instrucciones.

Al iniciar una transacción primero hemos de comprobar si la transacción se tiene que ejecutar en modo

irrevocable. Usualmente, esto consiste en verificar que la transacción haya llegado a su límite de abortos. Si la transacción tiene que entrar en irrevocabilidad se pide a través del protocolo de comunicación (Véase la Sección IV-B). Otra transacción en modo irrevocable puede hacernos esperar. Una vez que se nos permite entrar en modo irrevocable la transacción se ejecuta con el sistema de aislamiento transaccional desactivado, es decir, como si fuera código no transaccional. De esta manera no se puede abortar. Cuando se llega al final de la transacción lo único que tenemos que hacer es comunicar el final de la irrevocabilidad.

En caso de que no tengamos que entrar en irrevocabilidad todavía, el inicio de la transacción se ejecuta con normalidad, activando el aislamiento transaccional, lo que llevará a que cada lectura y escritura marque su respectivo bit de bloque de la caché (xR/xW). Cuando la transacción llega a la fase de finalización se comprueba la existencia de una transacción irrevocable en el sistema. Si la hay, la transacción se para hasta que finalice la irrevocable. Nótese que una transacción puede ser abortada estando en el estado de espera.

A. Correctitud

El mecanismo de irrevocabilidad relajada asegura una ejecución correcta debido a que se cumplen tres propiedades en el sistema BE-HTM:

- Aislamiento transaccional: esta propiedad asegura que cualquier escritura transaccional sólo será visible por la transacción que la ejecutó.
- Aislamiento fuerte: define la relación entre los accesos transaccionales y el código no transaccional. Con aislamiento fuerte, si una instrucción no transaccional escribe una posición de memoria que está en el conjunto de datos de una transacción, se abortará la transacción.
- Finalización postergada: el mecanismo de irrevocabilidad posterga la finalización de la transacción siempre que haya una transacción irrevocable en el sistema.

La Fig. 2 muestra los cuatro posibles casos de ejecución de una transacción normal e irrevocable. El primero muestra una transacción irrevocable en el primer hilo (Th1) que empieza después de una normal en el segundo (Th2). Pero la transacción de Th2 finaliza antes que la irrevocable. En este caso se debería abortar la transacción de Th2 ya que ha leído la posición Y (LD Y) escrita por la irrevocable (ST Y) y finaliza antes, lo que compromete la serializabilidad [17]. Sin embargo, a diferencia del fallback con suscripción relajada descrito en la Sección II, nuestro mecanismo de irrevocabilidad no aborta la transacción, si no que posterga su finalización hasta que acabe la irrevocable. En el peor caso, la transacción será abortada un poco más tarde debido a un conflicto con otra transacción o con la irrevocable por aislamiento fuerte. Con suerte, la transacción podrá finalizar mejorando así la concu-

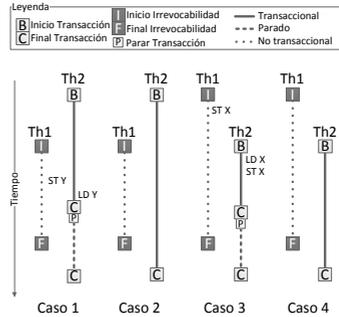


Fig. 2. Casos de ejecución para evaluar la correctitud de la irrevocabilidad relajada.

rencia.

El Caso 2 muestra un escenario correcto donde la irrevocable acaba antes que la transacción normal. De hecho, el Caso 1 es una adaptación artificial al Caso 2 gracias a la finalización postergada. La misma adaptación se realiza con el Caso 3 y el 4 donde la serializabilidad podría romperse si Th1 modifica X antes de que la transacción de Th2 lo lea. Postergar la finalización soluciona el problema.

B. Protocolo de Comunicación de Irrevocabilidad

Para la comunicación de la irrevocabilidad proponemos un protocolo basado en token donde sólo el núcleo que posee el token puede ejecutar una transacción en modo irrevocable. Cada núcleo tiene un bit, I, que indica si hay una transacción irrevocable corriendo en el sistema. Se necesita otro bit, T, para señalar si se posee el token. Junto con este par de bits (I,T), cada núcleo posee un contador (C) que mantiene el número de reintentos de la transacción en curso. El núcleo pide la irrevocabilidad cuando C llega a cero.

Dependiendo del valor del par de bits (I,T) el controlador de L1 de cada núcleo actúa en consecuencia:

- $(I, T) = (0, 0)$: No hay transacciones irrevocables y no tenemos el token. Si tenemos que pedir irrevocabilidad se difunde un mensaje de petición de token que será respondido por el núcleo que lo posee. Si ese núcleo acaba de iniciar una transacción irrevocable no mandará el token y quedaremos a la espera. Si recibimos el token ponemos el bit T a 1 y difundimos un mensaje de irrevocabilidad para que el resto de núcleos ponga su bit I a 1. Cuando estén a 1 podremos continuar en modo irrevocable, (1,1). En caso de encontrar (0,0) al finalizar una transacción es que no hay transacciones irrevocables y no tenemos que postergar la finalización.

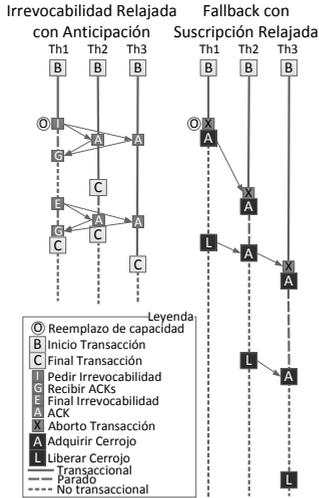


Fig. 3. Escenario de ejecución de irrevocabilidad relajada con anticipación contra fallback con suscripción relajada.

- $(I, T) = (0, 1)$: El núcleo tiene el token por lo que puede informar de que va a entrar en irrevocabilidad directamente ya que nadie más está en modo irrevocable.
- $(I, T) = (1, 0)$: Otro núcleo está en modo irrevocable. Si tenemos que pedir irrevocabilidad pararemos hasta que I sea 0. De la misma manera, si tenemos que finalizar una transacción también pararemos hasta recibir el mensaje de fin de la irrevocabilidad. En ese momento se finalizará la transacción.

El caso en el que varias transacciones piden el token al mismo tiempo se arbitra por medio de las colas de mensajes que poseen los routers de la red de interconexión y de la cola del controlador de memoria del núcleo que posee el token.

El uso de este protocolo de comunicación evita el efecto de la contención por cerrojo (véase la Sección VII-C). Además, no es necesario alojar el cerrojo y el contador en la L1, lo que podría causar abortos innecesarios.

V. IRREVOCABILIDAD RELAJADA CON ANTICIPACIÓN DE REEMPLAZO

La anticipación a un reemplazo de un bloque transaccional consiste en pedir irrevocabilidad antes de que se produzca el reemplazo para continuar la transacción en modo irrevocable sin tener que perder el cómputo hecho hasta el momento. Esta optimización no se puede realizar en fallbacks software.

La Fig. 3 muestra cómo se comportaría la irrevocabilidad relajada con anticipación de reemplazo en contraste con el fallback con suscripción relajada. El hilo 1, Th1, encuentra un reemplazo de caché que causaría un aborto. Pero en el caso de la irrevocabilidad se pide irrevocabilidad y se retiene el reemplazo. En cuanto al fallback, Th1 aborta y reintenta la transacción tras adquirir el cerrojo del fallback. Las otras transacciones continúan debido a la suscripción relajada. Sin embargo, Th2 finaliza antes que el fallback por lo que tiene que abortar. Después de abortar trata de adquirir el cerrojo pero tiene que esperar a que lo libere Th1. La transacción de Th3 finaliza después de la de Th1 y Th2 pero tiene que abortar porque Th2 adquirió el cerrojo un poco antes. Con la irrevocabilidad relajada se espera hasta recibir el mensaje de final de irrevocabilidad, que es cuando Th2 finaliza su transacción. En este caso, la transacción de Th3 que finalizaba después de las anteriores no tiene que esperar.

A. Implementación

La implementación de la anticipación de reemplazo requiere ligeras modificaciones del protocolo de coherencia de caché puesto que hay que intervenir las acciones realizadas en los eventos de reemplazo. La Tabla 1 muestra dichas modificaciones sombreadas. El reemplazo de bloques no transaccionales, $\neg(xRv\bar{x}W)$, se realiza sin modificación. Sin embargo, si el bloque es transaccional, $xRv\bar{x}W$, se comprueba el contador de reintentos (C) para ver si la transacción ha llegado a su límite. Si no ha llegado, la transacción aborta y se decrementa el contador. El bloque cambia su estado a inválido, I. Nótese que un reemplazo de caché puede no ser un evento persistente. Por lo tanto, la transacción se reintenta hasta que llega al límite menos uno, $C \leq 1$, de manera que se anticipa el último reintento y se pide irrevocabilidad. El mensaje que causó el reemplazo es reciclado en la cola de la CPU hasta que entremos en irrevocabilidad, parando así al procesador. Un paso importante que tiene que realizar el protocolo de irrevocabilidad es limpiar los bits xR y xW como si se tratara de finalizar la transacción. De esta manera, el mensaje reciclado que causó el reemplazo transaccional causará ahora un reemplazo no transaccional.

Los reemplazos de bloques de L1 debidos a reemplazos en L2 y la propiedad de inclusión se muestran como **Reempla L2** en la Tabla I. Los reemplazos de L2 no transaccionales se dejan igual, se reconoce el reemplazo y se invalida el dato. Se manda el dato también en caso de que se haya modificado. Sin embargo, cuando el dato a reemplazar por la L2 es transaccional en la L1, $xRv\bar{x}W$, abortamos la transacción y decrementamos el contador de reintentos siempre que no hayamos llegado al límite de abortos o haya otra transacción en modo irrevocable, $(I, T) = (1, 0)$. Esta última condición, que no está presente en los reemplazos de L1, se necesita para abortar la transacción, ya que puede haber un interbloque si es la irrevocable la que necesita reemplazar

TABLA I
 MODIFICACIONES DEL PROTOCOLO DE COHERENCIA DE CACHÉ L1 PARA LA ANTICIPACIÓN DE REEMPLAZO.

Estado	Eventos					
	Reempla L1 $\neg(xRv \times W)$	Reempla L1 $(xRv \times W) \wedge (C > 1)$	Reempla L1 $(xRv \times W) \wedge (C \leq 1)$	Reempla L2 $\neg(xRv \times W)$	Reempla L2 $(xRv \times W) \wedge (C > 1)$	Reempla L2 $(xRv \times W) \wedge (C \leq 1)$
I	-	-	-	ACK	-	-
S	- / I	Aborto, C-1 / I	Irre, Z	ACK / I	Aborto, C-1 / I	Irre, Zz
E	PUT(no datos) / I	Aborto, C-1 / I	Irre, Z	ACK / I	Aborto, C-1 / I	Irre, Zz
M	PUT+Datos / I	Aborto, C-1 / I	Irre, Z	ACK+Data / I	Aborto, C-1 / I	Irre, Zz

Irre: pedir irrevocabilidad.
 Z, Zz: reciclar las colas de CPU y de red, respectivamente.
 (#,#): par de bits (I,T).

el bloque de L2 para continuar. De esta manera, un hilo sólo puede esperar a entrar en irrevocabilidad si su transacción llegó al límite de reintentos y ninguna transacción del sistema está en modo irrevocable, $(0,-) \wedge (C \leq 1)$. En el caso de que una transacción acabe de pedir irrevocabilidad pero todavía no hemos sido informados, permaneceremos parados reciclando el mensaje de reemplazo y cuando los bits $(I,T) = (0,-)$ cambian a $(1,0)$ se lanzará el evento anterior y nuestra transacción será abortada.

VI. FALLBACKS SOFTWARE

La Fig. 4 muestra el código de un fallback con suscripción relajada. Definimos dos primitivas para empezar una transacción: (i) TAKE_XACT_CHECKPOINT toma un checkpoint para guardar el punto al que volver tras un aborto, pero no empieza el aislamiento transaccional (línea 3); y (ii) BEGIN_XACT inicia el sistema transaccional de manera que cada lectura y escritura sea aislada adecuadamente (línea 8). Así es posible insertar código no transaccional entre las dos primitivas para comprobar si tenemos que ejecutar el fallback.

El procedimiento de inicio de una transacción (líneas 2–10) comienza tomando el punto de retorno e incrementando la variable local que mantiene el número de reintentos de la transacción (línea 4). Si la transacción llegó al límite de reintentos definido globalmente en RETRY_LIMIT se intenta adquirir el cerrojo global (línea 6). Si no, el hilo ejecuta de forma transaccional (línea 8). El procedimiento de finalización (líneas 11–17) comprueba la variable local que almacena el número de reintentos para saber si se adquirió el cerrojo. Si no, se realiza la suscripción al cerrojo (líneas 13 y 14) comprobando si el cerrojo está a cero. Se trata de la suscripción relajada descrita en la Sección II. Si está a uno abortamos explícitamente. Si venimos de ejecutar el fallback liberamos el cerrojo (línea 16).

En la Fig. 4 se puede ver nuestra propuesta de fallback con suscripción relajada con espera escapada que es el homólogo software de la irrevocabilidad relajada sin anticipación de reemplazo. Consiste en sustituir la cláusula condicional por un bucle en el que se permanecerá hasta que el cerrojo se libere. Pero esta espera debe estar escapada para no incluir el cerrojo en el conjunto de lectura de la transacción, ya que de otro modo la liberación del cerrojo abor-

```

1 localRetries = 0;
2 void beginTransaction(localRetries) {
3     TAKE_XACT_CHECKPOINT; //Punto de retorno
4     localRetries++; //Incrementar reintentos
5     if (localRetries > RETRY_LIMIT) //Fallback?
6         while(!lockAcquire(globalLock)); //Adquirir cerrojo
7     else { //Ejecucion transaccional
8         BEGIN_XACT;
9     }
10 }
11 void endTransaction(localRetries) {
12     if (localRetries <= RETRY_LIMIT) {
13         if (lock != 0) //Suscripción relajada
14             ABORT_XACT;
15         COMMIT_XACT;
16     } else lockRelease(globalLock);
17 }
18
19 BEGIN_ESCAPE; //Espera escapada
20 while(lock != 0) ;
21 END_ESCAPE;

```

Fig. 4. Código de fallback con suscripción relajada y con espera escapada.

taría la transacción debido al aislamiento fuerte.

VII. EVALUACIÓN

A. Metodología

Hemos usado el simulador Simics [11] junto con el módulo GEMS [12] que implementa el simulador de memoria de un multiprocesador. Hemos simulado el BE-HTM de la sección III con las siguientes características: 16 núcleos escalares, con caché L1 privada de 32KB y 4 vías. Caché L2 unificada y compartida dividida en 16 bancos de 512KB y 8 vías. Directorio de vector de bits completo y red Garnet [18]. El número de hilos está limitado a 15 para que el sistema no haga migraciones.

Los benchmarks son de la suite de la universidad de Stanford STAMP [13]. La Tabla II muestra los parámetros que se han usado para cada benchmark y las principales características transaccionales como el número de transacciones, el porcentaje de tiempo dentro de transacción y la media del tamaño de los conjuntos de lectura y escritura de las transacciones, en bloques de caché.

B. Resultados

En esta sección se analizan los resultados obtenidos con nuestros mecanismos de irrevocabilidad, comparándolos con los resultados del fallback homólogo al mecanismo de irrevocabilidad sin anticipación. También se analizan los resultados del fallback con suscripción relajada. La Fig. 5 muestra

TABLA II
 BENCHMARKS: PARÁMETROS Y CARACTERÍSTICAS TRANSACCIONALES.

Benchmark	Entrada	# Xact	Tiempo en Xact	avg RS	avg WS
Bayes	-v32 -r1024 -n2 -p20 -i2 -e2 -s1	654	94%	87.64	48.91
Genome	-g512 -s32 -n32768	19496	85%	23.34	3.58
Intruder	-a10 -l16 -n4096 -s1	54933	92%	9.87	3.06
Kmeans-high	-m15 -n15 -t0.05 -i random-n2048-d16-c16	8235	46%	6.23	1.75
Kmeans-low	-m40 -n40 -t0.05 -i random-n2048-d16-c16	10980	12%	6.23	1.75
Labyrinth	-i random-x32-y32-z3-n96	222	100%	139.34	95.12
SSCA2	-s14 -i1.0 -u1.0 -l9 -p9	93721	13%	3.00	2.00
Vacation-high	-n4 -q60 -u90 -r16384 -t4096	4095	95%	63.20	10.16
Vacation-low	-n2 -q90 -u98 -r16384 -t4096	4095	93%	48.17	8.60
Yada	-a20 -i633.2	5447	100%	62.45	38.21

los resultados de rendimiento sobre la aplicación secuencial para los sistemas mencionados. Todos los experimentos fueron realizados con el contador de reintentos configurado a cinco repeticiones, que es un valor frecuentemente usado en la bibliografía [4], [2].

Los resultados muestran tres benchmarks que no escalan: Bayes, Labyrinth y Yada. Estos benchmarks dan un rendimiento como el de la aplicación secuencial. Además, con un sólo hilo los resultados son incluso peores que el secuencial. Esta degradación se debe al número de reintentos antes de la irrevocabilidad. Las transacciones que desbordan la caché abortan cinco veces hasta que se entra en irrevocabilidad lo que hace que se rinda peor que el secuencial. El mecanismo de irrevocabilidad que anticipa el último aborto rinde un poco mejor que los demás debido a que hay un aborto menos.

El siguiente grupo de benchmarks escala adecuadamente y muestra poca diferencia entre los distintos métodos estudiados. Este grupo se compone de Kmeans-low, SSCA2 y Vacation-low. Estos benchmarks tienen transacciones pequeñas en media, Kmeans-low y SSCA2 pasan al reedor de un 13% del tiempo dentro de transacción (véase la Tabla II) y todos ellos muestran baja contención. La Tabla III muestra el número de transacciones irrevocables desglosadas por causa y podemos ver que el porcentaje de transacciones irrevocables está en torno al 5% para 15 hilos. Además, Kmeans y SSCA2 no muestran transacciones irrevocables debido a reemplazos de caché por lo que el sistema con anticipación no produce ningún beneficio. Por el contrario, Vacation sí muestra transacciones irrevocables debido a reemplazos, debido a su conjunto de lectura más grande, pero no son suficientes como para suponer una mejora de rendimiento. Sin embargo, con respecto al fallback relajado nuestras propuestas muestran una mejora notable para 15 hilos ya que al haber baja contención los abortos del fallback relajado penalizan el rendimiento para ningún benchmark.

El último grupo de benchmarks muestra una mejora en el rendimiento cuando se usa la irrevocabilidad en lugar del fallback software: Genome, Intruder, Kmeans-high y Vacation-high. Se puede observar que la irrevocabilidad relajada con y sin anticipación pueden dar un resultado muy parecido

TABLA III
 NÚMERO DE TRANSACCIONES IRREVOCABLES POR CAUSA:
 DEBIDO A UN REEMPLAZO DE L1, L2, O CONFLICTO.

Bench	# Transacciones Irrevocables (L1/L2/Conflicto)	
	8 th's	15 th's
Bayes	211(62/0/149)	226(34/0/191)
Genome	1119(922/0/197)	1434(1165/0/268)
Intruder	6397(42/0/6355)	16619(89/0/16530)
Kmeans-high	1043(0/0/1043)	1986(0/0/1986)
Kmeans-low	291(0/0/291)	567(0/0/567)
Labyrinth	106(29/0/77)	117(22/0/96)
SSCA2	283(0/0/283)	527(0/0/527)
Vacation-high	336(268/0/68)	428(302/0/126)
Vacation-low	103(69/0/33)	247(181/3/63)
Yada	1863(581/0/1282)	1883(540/2/1342)

entre sí y mejor que el fallback con espera escapada, homólogo de la irrevocabilidad sin anticipación, sobre todo para Intruder, Kmeans-high y Vacation-high con un 14%, un 25% y un 13% de mejora respectivamente. Esto ocurre cuando la causa principal de entrar en irrevocabilidad son los conflictos, y los reemplazos no tienen una posición dominante, como se puede ver en la Tabla III. Vacation-high, sin embargo, muestra un mayor número de transacciones irrevocables debidas a reemplazos, y el mecanismo de irrevocabilidad con anticipación realmente muestra mejores resultados que el que no tiene anticipación, pero no parece algo significativo. La Sección VII-C explica el efecto que la contención de cerrojos tiene en el fallback homólogo a nuestro mecanismo de irrevocabilidad, que hace que el fallback pierda rendimiento. Por último, Genome muestra una mejora de rendimiento notable de un 28% con respecto al fallback con espera escapada y al mecanismo de irrevocabilidad sin anticipación. En este caso, las transacciones irrevocables de Genome son principalmente debidas a fallos de capacidad, ya que tiene un numeroso grupo de transacciones con un conjunto de lectura grande. En cualquier caso, nuestros mecanismos de irrevocabilidad no penalizan el rendimiento.

C. Efecto de la Contención de Cerrojos

Hemos visto como el mecanismo de irrevocabilidad relajada sin anticipación puede dar mejores resultados que su homólogo software, incluso cuando se im-

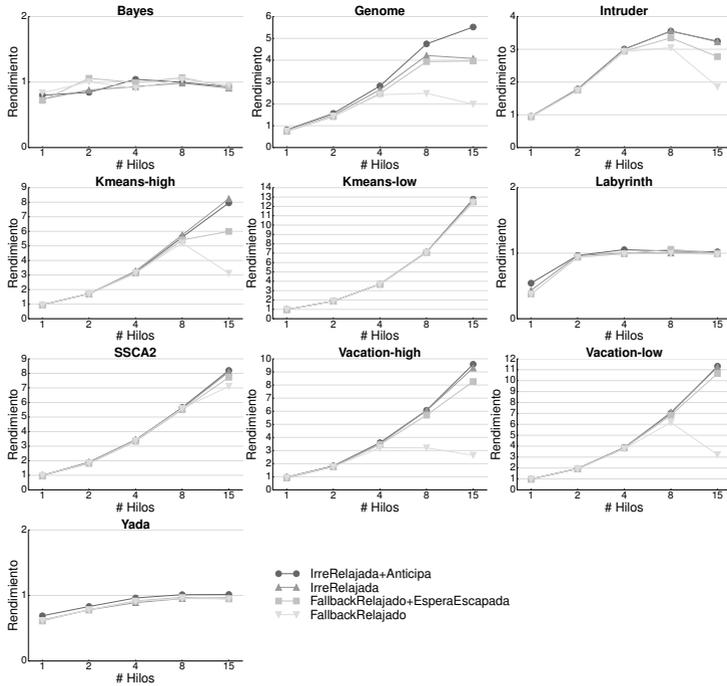


Fig. 5. Rendimiento sobre la aplicación secuencial de los mecanismos de irrevocabilidad relajada con y sin anticipación, junto con los fallbacks con suscripción relajada con y sin espera escapada.

plementan esencialmente de la misma manera. Sin embargo, el fallback utiliza cerrojos que introducen un efecto de contención del que carece nuestro sistema de irrevocabilidad, que utiliza el protocolo de comunicación basado en token.

Con el uso de cerrojos se puede dar la situación de que un grupo de transacciones esté esperando para finalizar sus transacciones debido a la transacción que está en el fallback. Están postergando su finalización con la espera escapada. Por otro lado podemos tener un grupo de transacciones que está esperando a que termine la transacción que está en el fallback, para entrar uno de ellos. Cuando la transacción que está en el fallback libera el cerrojo, se invalidan todas las copias privadas del mismo que residen en las cachés L1, y los dos grupos de transacciones esperando por distinto motivo piden acceso al cerrojo. Si la que obtiene el acceso primero es una de las que quería entrar al fallback, esta pondrá el cerrojo a uno y todas las demás transacciones encontrarán el cerrojo adquirido nuevamente. Por lo tanto, aquellas que estaban postergando su finalización, seguirán en la

espera escapada, con el riesgo de poder ser abortadas.

Este efecto de contención debido al cerrojo no se da en nuestro protocolo de comunicación de irrevocabilidad ya que las transacciones que estaban postergando la finalización finalizan inmediatamente al recibir el mensaje de fin de irrevocabilidad. El efecto de contención es más probable a medida que se incrementa el número de hilos y en consecuencia la contención transaccional.

VIII. CONCLUSIONES

Recientemente podemos encontrar en el mercado multiprocesadores con extensiones transaccionales best-effort que necesitan de un fallback software para garantizar el progreso de las aplicaciones y superar las limitaciones hardware. En este artículo proponemos un nuevo tipo de irrevocabilidad que garantiza el progreso de las aplicaciones transaccionales y esconde las limitaciones del hardware al usuario a la vez que maximiza el paralelismo.

Se propone la irrevocabilidad relajada basada en el concepto de suscripción relajada de cerrojos en fall-

backs software. El mecanismo favorece la concurrencia postergando la finalización de las transacciones hasta la finalización de la transacción irrevocable y no necesita modificar el protocolo de coherencia. Proponemos un fallback homólogo basado en cerrojo con espera escapada que obtiene un rendimiento peor en algunos casos debido al efecto de la contención de cerrojos. Nuestro mecanismo de irrevocabilidad evita ese efecto con un protocolo de comunicación de irrevocabilidad basado en token.

También se propone un mecanismo de irrevocabilidad con anticipación de reemplazo que requiere ligeras modificaciones al protocolo de coherencia y que no puede ser implementado en software. Esta solución mejora el rendimiento de aquellas aplicaciones cuyas transacciones irrevocables sean causadas principalmente por reemplazos de caché.

La evaluación de las propuestas se lleva a cabo con el sistema simulado Simics/GEMS y con la suite de benchmarks STAMP, y obtenemos mejoras de rendimiento de hasta el 28% sobre las soluciones basadas en fallback.

AGRADECIMIENTOS

Este trabajo ha sido realizado gracias a los proyectos TIN2013-42253-P, del Ministerio de Economía y Competitividad del gobierno de España, y P12-TIC-1470, de la Junta de Andalucía.

REFERENCIAS

- [1] M. Herlihy and J.E.B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *20th Ann. Int'l. Symp. on Computer Architecture (ISCA'93)*, 1993, pp. 289–300.
- [2] Richard M Yoo, Christopher J Hughes, Konrad Lai, and Ravi Rajwar, "Performance Evaluation of Intel Transactional Synchronization Extensions for High-performance Computing," in *Int'l Conf. on High Performance Computing, Networking, Storage and Analysis (SC'13)*, 2013, pp. 19:1–19:11.
- [3] Amy Wang, Matthew Gaudet, Peng Wu, José Nelson Amaral, Martin Ohmacht, Christopher Barton, Raul Silveira, and Maged Michael, "Evaluation of Blue Gene/Q hardware support for transactional memories," in *21st Int'l Conf. on Parallel Architectures and Computation Techniques (PACT'12)*, 2012, pp. 127–136.
- [4] Christian Jacobi, Timothy Siegel, and Dan Greiner, "Transactional Memory Architecture and Implementation for IBM System z," in *45th Ann. Int'l. Symp. on Microarchitecture (MICRO'12)*, Dec. 2012, pp. 25–36.
- [5] Allon Adir, Charles Meissner, Amir Nahir, Randall R. Pratt, Mike Schiffli, Brett St. Onge, Brian Thompto, Elena Tsanko, Avi Ziv, Dave Goodman, Daniel Hershkovich, Oz Hershkovitz, Bryan Hickerson, Karen Holtz, Wisam Kadry, Anatoly Kofman, John Ludden, and Brett St Onge, "Verification of Transactional Memory in POWERS," in *51st Ann. Design Automation Conf. (DAC'14)*, 2014, pp. 1–6.
- [6] Adam Welc, Saha Bratin, and Ali-Reza Adl-Tabatabai, "Irreversible transactions and their applications," in *20th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'08)*, June 2008, pp. 285–296.
- [7] Luke Dalessandro, François Carouge, Sean White, Yossi Lev, Mark Moir, Michael L. Scott, and Michael F. Spear, "Hybrid NOrec: A Case Study in the Effectiveness of Best Effort Hardware Transactional Memory," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 39, Jun 2012.
- [8] Irina Calciu, Tatiana Shpeisman, Gilles Pokam, and Maurice Herlihy, "Improved Single Global Lock Fallback for Best-effort Hardware Transactional Memory," in *9th Workshop on Transactional Computing (TRANSACTIONAL'14)*, 2014.
- [9] Milo M K Martin, Colin Blundell, and E Lewis, "Subtleties of Transactional Memory Atomicity Semantics," *IEEE Computer Architecture Letters*, vol. 5, no. 2, pp. 17, 2006.
- [10] Michelle J Moravan, Jayaram Bobba, Kevin E Moore, Luke Yen, Mark D Hill, Ben Liblit, Michael M Swift, and David A Wood, "Supporting Nested Transactional Memory in logTM," in *12th Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'06)*, 2006, pp. 359–370.
- [11] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, and B. Werner, "Simics: A full system simulation platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, 2002.
- [12] M.M.K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, and D.A. Wood, "Multifacet's general execution-driven multiprocessor simulator toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.
- [13] C.C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun, "STAMP: Stanford Transactional Applications for Multi-Processing," in *IEEE Int'l Symp. on Workload Characterization (IISWC'08)*, 2008, pp. 35–46.
- [14] Harold W Cain, Maged M Michael, Brad Frey, Cathy May, Derek Williams, and Hung Le, "Robust Architectural Support for Transactional Memory in the Power Architecture," in *40th Ann. Int'l. Symp. on Computer Architecture (ISCA'13)*, 2013, pp. 225–236.
- [15] L. Hammond, V. Wong, M. Chen, B.D. Carlstrom, J.D. Davis, B. Hertzberg, M.K. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun, "Transactional memory coherence and consistency," in *31th Ann. Int'l. Symp. on Computer Architecture (ISCA'04)*, 2004, pp. 102–113.
- [16] Colin Blundell, Joe Devietti, E Christopher Lewis, and Milo M K Martin, "Making the Fast Case Common and the Uncommon Case Simple in Unbounded Transactional Memory," in *34th Ann. Int'l. Symp. on Computer Architecture (ISCA'07)*, 2007, ISCA '07, pp. 24–34.
- [17] Tim Harris, James Larus, and Ravi Rajwar, *Transactional Memory*, 2nd edition, Morgan & Claypool Publishers, 2010.
- [18] N. Agarwal, T. Krishna, Li-Shiuan Peh, and N.K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *IEEE Int'l. Symp. on Performance Analysis of Systems and Software (ISPASS'09)*, April 2009, pp. 33–42.

Reduciendo el consumo dinámico de energía con Tag Filter Cache

Joan J. Valls¹, Julio Sahuquillo¹, Alberto Ros², María E. Gómez¹

Resumen—

El consumo de energía en los actuales multiprocesadores en chip (CMPs) de altas prestaciones se ha convertido en una de las mayores preocupaciones a la hora de realizar nuevos diseños. La tendencia actual del aumento del número de cores no hace sino agravar este problema. Las caches on-chip consumen una fracción significativa del consumo total del chip. La mayoría de las técnicas propuestas para reducir el consumo de energía de estas estructuras de memoria suelen conllevar una pérdida de prestaciones, lo cual puede ser inaceptable para CMPs de altas prestaciones. Estas caches en sistemas multinúcleo se implementan con un alto grado de asociatividad para poder incrementar sus prestaciones. Incluso las caches de primer nivel se están implementando actualmente con ocho vías. El acceso concurrente a todas las vías en el conjunto de la cache es bastante costoso en términos de energía. En este paper proponemos un diseño de cache energéticamente eficiente, la arquitectura Tag Filter Cache (TF-Cache), que filtra algunas de las vías de un conjunto durante los accesos a la cache, permitiendo el acceso únicamente de un subconjunto de ellos sin afectar a las prestaciones. Nuestra cache almacena para cada vía los bits de etiqueta de menor peso en un bit array auxiliar y estos bits se utilizan para filtrar las vías cuyos bits no coincidan con los de la etiqueta del bloque buscado. Los resultados experimentales muestran que, de media, la arquitectura TF-Cache reduce el consumo de energía dinámico hasta un 74.9% y 85.9% cuando se aplica a una cache L1 y L2, respectivamente, para las aplicaciones evaluadas.

Palabras clave— Multiprocesadores en chip; Memorias cache; Consumo de energía; Aplicaciones multihilo

I. INTRODUCCIÓN

CONFORME se avanza en la escala de integración y se aumentan los recursos de silicio, el número de núcleos en las nuevas generaciones de multiprocesadores en chip (CMPs) se va incrementando progresivamente. Estos CMPs aceleran el acceso a memoria introduciendo uno o más niveles de cache, siendo éstas responsables de un porcentaje significativo del área ocupada [1] y de una parte importante del consumo de energía total. Gran parte de este consumo es debido a consumo dinámico (los cambios de nivel de los transistores durante los accesos). Los diseñadores de cache deben ofrecer nuevos diseños en los que se llegue a un compromiso entre prestaciones, coste, tamaño y disipación de energía.

El primer nivel de la jerarquía de memoria es el que domina sobre el consumo dinámico ya que es accedida con mayor frecuencia que el último nivel de cache (LLC), por ejemplo, caches L3, a las cuales se accede bastante menos. Además, este elevado consumo también es debido al hecho de que se accede a etiquetas y datos en paralelo, ya que los primeros niveles de cache tienen un alto grado de influencia sobre las prestaciones del procesador. Esto resulta incluso aún más importante en CMPs que en procesadores monolíticos ya que las caches pueden ser accedidas tanto desde el lado del procesador como desde

la red de interconexión (para peticiones de coherencia), aumentando así el número de accesos a la cache.

Debido al interés de mantener unas prestaciones elevadas, estas caches se implementan con un alto grado de asociatividad. En los microprocesadores de altas prestaciones, todas las vías del conjunto se acceden concurrentemente durante un acceso a la cache. Por tanto, el grado de asociatividad define el número de etiquetas que se tienen que mirar en paralelo durante cada acceso. Las caches incluyen un comparador por vías y necesitan realizar tantas comparaciones de etiquetas como número de vías. Como consecuencia directa, la energía dinámica disipada por acceso se ve incrementada con la asociatividad de la cache.

Generalmente, el diseño de caches de bajo consumo dinámico se centran en minimizar la actividad interna de los transistores durante un acceso a la cache. Esta actividad procede de la lectura y comparación de etiquetas en los tag arrays y de la lectura o escritura de datos en los data array. Idealmente, en un acierto en cache, la cache tan solo necesitaría leer y comparar una única etiqueta y acceder a una entrada de datos. Además, en caso de fallo, idealmente una cache no tendría por que necesitar acceder ni al tag array ni al data array. De hecho, para detectar un fallo, la cache no tiene la necesidad de acceder a un campo de etiqueta completo, ya que la diferencia de un único bit es suficiente para detectarlo.

Muchas propuestas para la reducción de energía en las caches se han centrado en procesadores monolíticos (tales como Cache Decay [2], Drowsy Caches [3] y Way Guard [4]). Algunas de ellas, por ejemplo [5], fueron diseñadas originalmente para reducir el tiempo de acceso a la cache, pero investigaciones posteriores han demostrado que estos esquemas también ofrecen importantes ahorros de consumo. No obstante, ya que estos esquemas no se aplican directamente a CMPs o pueden ser mejorados, la investigación actual se centra en el ahorro de energía en CMPs ejecutando cargas paralelas.

En este trabajo proponemos la Tag Filter Cache (TF-Cache), una arquitectura cache que reduce el número de etiquetas y bloques de datos que se comprueban cuando se accede a la jerarquía de memoria. La TF-Cache se puede aplicar a cualquier nivel de la jerarquía de memoria con el objetivo de reducir el consumo de energía dinámico de estas estructuras. Se basa en la utilización de los bits de menor peso (LSB) de la etiqueta para discernir que vías de una cache asociativa por conjuntos puede contener el bloque buscado. Únicamente se accede a aquellas vías que puedan estar conteniendo el bloque referenciado, consiguiendo así ahorrar el consumo de energía necesario para acceder al resto de vías.

La TF-Cache se puede implementar con una complejidad de hardware mínima. Además, se puede acceder en

¹DISCA, Universitat Politècnica de València, e-mails: joavalma@fiv.upv.es, jsahuqui@disca.upv.es, megomez@disca.upv.es

²DITEC, Universidad de Murcia, e-mail: aros@ditec.um.es

paralelo a los tag y data arrays así que no se origina ninguna degradación de prestaciones, lo cual es una de las mayores preocupaciones en las cache L1. A diferencia de otras propuestas como [6] no es necesario realizar ningún alineamiento de vías.

Los resultados experimentales muestran que la arquitectura TF-Cache puede reducir el consumo de energía dinámico hasta un 74.9% y 85.9% para las caches L1 y L2, respectivamente, obteniendo así mejores resultados que otros trabajos recientes.

El resto de este trabajo se organiza de la siguiente forma:

La sección II describe las razones principales que motivan esta investigación. La sección III describe el trabajo relacionado. La sección IV presenta la propuesta. La sección V describe el entorno experimental. La sección VI presenta los resultados obtenidos. Finalmente, la sección VII presentará unas conclusiones finales.

II. MOTIVACIÓN

Las memorias cache, especialmente las caches de primer y segundo nivel, se acceden frecuentemente, dado que las instrucciones de memoria representan un porcentaje significativo de las instrucciones ejecutadas. Una fracción significativa del consumo total de energía es consumida normalmente por las caches on-chip, tal y como sucede en el procesador Niagara2 [7], donde el 44% del consumo del chip lo consume la cache L2. La reducción del consumo de energía dinámico en las caches de los CMPs es un problema actual que está siendo investigado [8] [6]. Para dar solución a este problema, este trabajo propone una arquitectura que pretende aprovechar la distribución homogénea de los bits menos significativos de la etiqueta en las vías de un conjunto en una cache asociativa por conjuntos.

Lanzamos varios experimentos para verificar esta hipótesis en las cargas evaluadas. La figura 1 muestra la distribución media de los bloques en una cache L1 de 8 vías y una cache L2 de 16 vías para un sistema CMP con 16 núcleos¹.

Como se puede ver en las figuras 1(a) y 1(b), de media hay 1 y 2 vías en estado inválido, bajo un protocolo de coherencia MOESI, para la L1 y L2, respectivamente. Mientras tanto, el resto de vías mantienen una distribución bastante homogénea considerando los bits de menor peso de los bloques almacenados. No hay ninguna necesidad de acceder a todas las vías de un conjunto si existe un mecanismo que permite filtrar el acceso a un subconjunto de vías que pueda estar conteniéndolo. Una distribución homogénea como la mencionada anteriormente (los bits de menor peso) es, por tanto, un método muy interesante para un mecanismo de filtrado.

El objetivo de este trabajo es ahorrar energía dinámica reduciendo el número de consultas que se hacen durante cada acceso a la cache. Concretamente, la propuesta ahorra energía accediendo únicamente a aquellas vías cuyos bits de menor peso coinciden con los bits de menor peso del bloque solicitado.

¹El entorno experimental, los parámetros del sistema y la jerarquía de memoria se describen en la sección V.

III. TRABAJO RELACIONADO

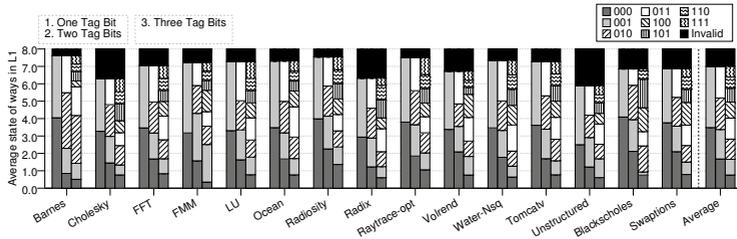
Este trabajo presenta un diseño de cache energéticamente eficiente que utiliza los bits de menor peso de las etiquetas de los bloques referenciados por las aplicaciones para filtrar vías y reducir el consumo. Por tanto, esta sección comentará algunos trabajos relacionados sobre el diseño de caches que pretenden reducir el consumo de energía.

El consumo de las caches proviene tanto del leakage (o consumo estático) como del dinámico. Con respecto al ahorro del leakage, Powell *et al.* [9] proponen Gated-Vdd, una técnica que pretende reducir el consumo estático de las caches de instrucciones reconfigurándolas y desconectando líneas no utilizadas. Kaxiras *et al.* [2] proponen Cache Decay, una propuesta que reduce el leakage de las caches del procesador apagando aquellas líneas que se predicen como muertas, es decir, aquellas que no van a volver a ser referenciadas por el procesador antes de su expulsión. Alternativamente, Flautner *et al.* [3] explotan el hecho de que durante un periodo de tiempo concreto solo se accede a un subconjunto de las líneas de la cache para proponer las Drowsy Caches. A diferencia de las propuestas anteriores, el voltaje es reducido pero no eliminado para aquellas líneas que no están siendo accedidas. Consecuentemente, no se preserva el contenido de la línea de la cache.

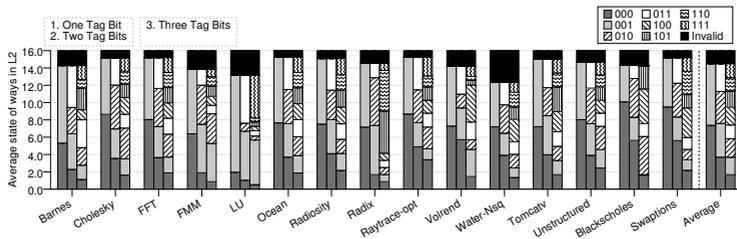
Mientras que las técnicas que intentan reducir el leakage se centran en reducir (o cortar) el voltaje, las técnicas de ahorro de energía dinámica intentan minimizar el número de datos leídos y escritos en cada acceso a la cache. Por ejemplo, Albonesi [10] propone Selective Cache Ways, un diseño de cache que activa tan solo un subconjunto de las vías de la cache cuando la actividad de la cache no es muy elevada. La predicción de vías ya fue previamente propuesta por Calder *et al.* [5] para reducir el tiempo de acceso a caches asociativas por conjunto. Esta propuesta funciona bien en caches L1 con un grado de asociatividad relativamente pequeño ya que presenta unos patrones de acceso muy predecibles. No obstante, como mostramos en la evaluación, este mecanismo presenta unos resultados pobres para niveles inferiores de cache ya que la localidad queda ocultada por los niveles anteriores.

Ghost *et al.* [4] proponen Way Guard, un mecanismo para grandes caches asociativas por conjunto que utilizan bloom filters para reducir el consumo de energía dinámico al evitarse el acceso a aquellas vías que no contengan los datos solicitados a partir de la información del bloom filter. Este esquema requiere añadir un decodificador grande y dos campos por vía: un bloom filter segmentado, propuesto previamente por los mismos autores para filtrar el acceso de toda la cache [11], y otro bloom filter para filtrar el acceso a las vías. Esto puede suponer un sobre coste excesivo y un incremento notable de la complejidad. Se realiza una comparación cuantitativa con Way Guard en la sección de evaluación.

Valls *et al.* [12] proponen PS-Cache, un mecanismo que filtran las vías que se miran en cada acceso a la cache al clasificar cada bloque como privado o compartido, según la información de la tabla de páginas. En cada acceso a la cache, solo se accede a las vías que contienen



(a) Número medio de vías de cada tipo en un conjunto en la cache L1



(b) Número medio de vías de cada tipo en un conjunto en la cache L2

Fig. 1. Número medio de vías de cada tipo en un conjunto en la jerarquía de cache.

bloques cuyo tipo coinciden con el de la clasificación. También se ha realizado una comparación con esta propuesta.

Finalmente, otras técnicas propuestas recientemente se centran en reducir los dos tipos de consumo, por ejemplo, reduciendo el área de las etiquetas, como en TLB Index-Based Tagging [8], utilizando caches de mapeado directo junto mecanismos para eliminar fallos por conflicto, como en ASCIB [13], o realizando particionamiento en tiempo de ejecución, como en Cooperative Caching [6] o ReCaC [14].

IV. TAG FILTER CACHE

El objetivo principal de Tag Filter Cache es reducir el número de etiquetas que se comparan en cada acceso a la cache y también del número de vías que se acceden en paralelo en el data array, de forma que se pueda reducir el consumo de energía dinámica en estas estructuras. En un acceso a cache típico, para comprobar si el bloque se encuentra en la cache, es necesario comparar todas las etiquetas en todas las vías del conjunto y, como mucho, una de esas comparaciones tendrá éxito, mientras que el resto fallarán. En las caches de primer nivel, todas las vías del conjunto en el data array se acceden simultáneamente, antes de saber si el bloque se encuentra finalmente en el conjunto. Este trabajo propone filtrar el acceso de aquellas vías (tanto en etiquetas como en datos) en las que se espera que la comparación de etiquetas vaya a fallar.

La figura 2 muestra un diagrama de bloques de la propuesta para una cache L1. El filtro consiste en comparar únicamente un subconjunto de X bits de los bits de las etiquetas. Con este propósito, el tag array se desacopla en dos estructuras: una de X_s bits y otras de $N - X$ bits. La TF-Cache utiliza los bits de menor peso de la etiqueta, que se encuentran almacenados en la estructura delgada de X_s bits, para reducir el número de vías accedidas.

El mecanismo realiza la comparación de etiquetas en dos fases. En la primera fase, tan solo se comparan los X_a bits de menor peso de la etiqueta del bloque solicitado con los bits de menor peso de todas las vías del conjunto. El reducido número de bits que se usan en el mecanismo permiten que esta primera comparación sea rápida e introduzca una penalización de tiempo despreciable. En la segunda fase, el resto de bits de etiqueta se comparan con los bits correspondientes de la dirección virtual del bloque que se está buscando. Esta segunda comparación, que requiere de un número mayor de bits, solo se realiza en aquellas vías que hayan tenido acierto durante la primera comparación.

De igual forma, la TF-Cache solo accede a aquellas vías en el data array cuya primera comparación hayan tenido éxito. Hay que tener en cuenta, que en una cache convencional, todos los bits de etiqueta y todos los datos se leen en paralelo para todas las vías del conjunto. Gracias al filtrado, y como se demuestra en la sección de evaluación, el mecanismo consigue importantes ahorros

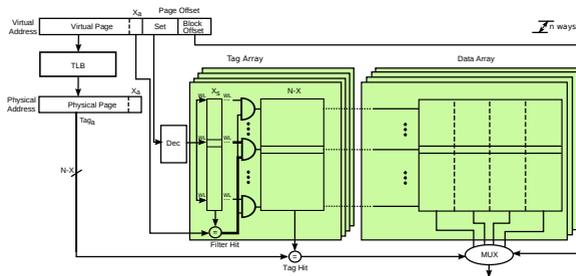


Fig. 2. La arquitectura TF-Cache para caches L1.

de energía.

Para que este mecanismo funcione en caches VIPT (virtuality indexed physically-tagged) como las de los procesadores Intel, esta primera comparación se tiene que realizar antes de que la salida de la TLB se conozca. Para ello, asumimos que el sistema operativo es el responsable de asegurar que los X bits de menor peso de la dirección virtual coincidan con aquellos de la dirección física. Esto es una suposición razonable ya que se espera una distribución de direcciones de página uniforme y las capacidades de memoria principal son de cuatro órdenes de magnitud superiores que las del tamaño de páginas (por ejemplo 32GB de memoria principal [15] y un tamaño de página de 4KB), lo que permite que el OS tenga cierta flexibilidad de ubicación. Bajo este supuesto, la primera comparación puede realizarse en cuanto esté el resultado de la decodificación del conjunto, mientras la traducción de la dirección en la TLB está aún siendo procesada. En cuanto la TLB termina la traducción, los $N - X$ bits de las vías que no hayan sido filtradas en la primera comparación se comparan con los $N - X$ que devuelve la TLB.

En cuanto a complejidad, la propuesta requiere un hardware mínimo para adaptarse a las caches actuales. Como se menciona arriba, el tag array se desacopla en dos estructuras independientes. Se necesita una lógica sencilla para dirigir la señal del wordline (WL) tanto a la estructura de etiquetas de $N - X$ bits como al data array. Como se puede apreciar en la figura 2, el wordline activa únicamente aquellas vías que hayan tenido éxito durante la primera comparación. Hay que tener en cuenta que las puertas AND no cortan el suministro de potencia pues esto no preservaría el contenido de los datos.

Este diseño permite reducir el consumo de energía dinámico de forma significativa, ya que, como los resultados experimentales revelan, tan solo una pequeña fracción de las vías se acceden en la mayoría de los accesos.

V. ENTORNO DE SIMULACIÓN

Evaluamos nuestra propuesta con un simulador de sistema completo utilizando Virtutech Simics [16] y el toolset Wisconsin GEMS [17], que permite una simulación detallada de sistemas multiprocesador. Para modelar la red de interconexión se utiliza GARNET [18], un simulador detallado de redes incluido en el toolset de GEMS.

TABLA I
 PARÁMETROS DEL SISTEMA

Parámetros de Memoria	
Cache hierarchy	Non-inclusive
Cache block size	64 bytes
Split L1 I & D caches	64KB, 8-way
L1 cache access time	2 cycles
Shared single L2 cache	512KB/tile, 16-way
L2 cache access time	6 cycles (2 if only tag accessed)
Directory cache	256 sets, 4 ways (same as L1)
Directory cache hit time	2 cycles
Memory access time	160 cycles
Parámetros de Red	
Topology	2-dimensional mesh (4x4)
Routing technique	Deterministic X-Y
Flit size	16 bytes
Data and control message size	5 flits and 1 flit
Routing, switch, and link time	2, 2, and 2 cycles

La tabla I muestra los valores de los parámetros del sistema principal correspondientes a nuestro sistema base, que es una arquitectura CMP de 16 tiles. Utilizamos la herramienta CACTI 6.5 [19] para estimar los tiempos de acceso, los requisitos de área y el consumo de energía de las diferentes estructuras cache para una tecnología de 32nm y transistores de altas prestaciones.

Nuestra evaluación analiza una jerarquía cache con L1 privadas para cada núcleo y una L2 NUCA distribuida entre todos los tiles. Se utiliza un protocolo de coherencia basado en directorio para mantener la coherencia de los datos almacenados en las caches privadas.

Se evalúa la propuesta con una amplia gama de aplicaciones científicas. *Barnes* (16K particles), *Cholesky* (tk15), *FFT* (64K complex doubles), *FMM* (16K particles), *LU* (512x512 matrix), *Ocean* (514x514 ocean), *Radiosity* (room, -ae 5000.0 -en 0.050 -bf 0.10), *Radix* (512K keys, 1024 radix), *Raytrace* (teapot -optimizada-), *Volrend* (head), y *Water-Nsq* (512 molecules) pertenecen al benchmark suite SPLASH-2 [20]. *Tomcatv* (256 points) y *Unstructured* (Mesh.2K) son dos benchmarks

científicos. *Blackscholes* (simmedium) y *Swaptions* (simmedium) pertenecen al suite PARSEC [21]. Los resultados experimentales se corresponden con la fase paralela de los benchmarks evaluados.

VI. EVALUACIÓN EXPERIMENTAL

En esta sección se explican los esquemas empleados para la evaluación comparativa y se analizan los resultados experimentales obtenidos.

A. Esquemas Comparados

Comparamos el esquema propuesto con otras propuestas que también reducen el consumo dinámico de energía accediendo a un subconjunto de las vías en lugar de a todas ellas. Estos esquemas son Way Prediction y dos propuestas más recientes: Way Guard y PS-Cache.

Las técnicas de predicción de vías [5, 10] predicen que vía tiene que se accedida por adelantado, habitualmente la vía que contiene el bloque más recientemente usado (MRU), y solo accede a esa vía en primer lugar. El problema surge cuando la predicción falla, en cuyo caso, el resto de las vías deben buscarse en una segunda fase para intentar dar con el bloque referenciado. Esto significa que tras cada fallo de predicción se está malgastando energía y se está incrementando la latencia, ya que se precisan ciclos adicionales para resolver la petición de memoria.

Way Guard [4] ha demostrado que tiene un funcionamiento eficiente en caches altamente asociativas. Este mecanismo implementa un counting bloom filter asociado a cada vía. Funciona de la siguiente forma: primero, se aplica una función hash a un subconjunto de los bits de la dirección del bloque. La salida de la función hash es un índice de m bits que se decodifica para acceder al vector de bloom filters de $2^m - 1$ entradas. Si el bit está activo a 1 entonces se accede a la vía de la cache asociada (tanto etiquetas como datos). En caso contrario, no se mira en esa vía. Cada entrada en el bloom filter tiene asociado un contador ascendente-descendente (de 3 bits en el trabajo original), que se decrementa cada vez que se expulsa una línea de la cache cuya dirección se mapea a esa posición y se incrementa cuando un bloque se almacena en la cache. En el trabajo original, se muestran los resultados para un valor de m igual a cuatro veces el número de bloques en una cache. Esta propuesta requiere de un decodificador con 4 veces más salidas que las que ya vienen implementadas en la cache para indexar el conjunto.

La PS-Cache [12] marca los bloques en tiempo de ejecución como compartido o privado siguiendo un sencillo mecanismo de clasificación basado en la información de la tabla de páginas. Durante un acceso a la PS-Cache, tan solo se miran aquellas vías cuyo tipo coincide con el del bloque requerido.

B. Resultados Experimentales

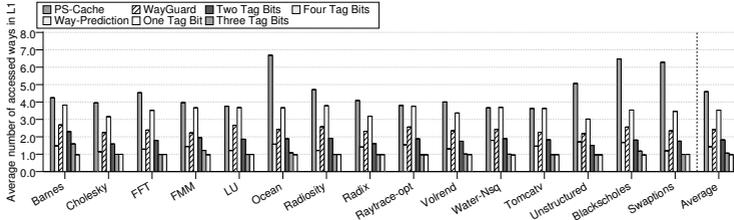
Los beneficios de esta propuesta dependen del número medio de vías que se acceden en cada acceso a la cache. Este número varía bastante en función del nivel de cache al que se accede (L1 o L2) y del comportamiento de las aplicaciones.

La figura 3(a) muestra el número medio de vías accedidas en una cache L1 de 8 vías para las distintas técnicas

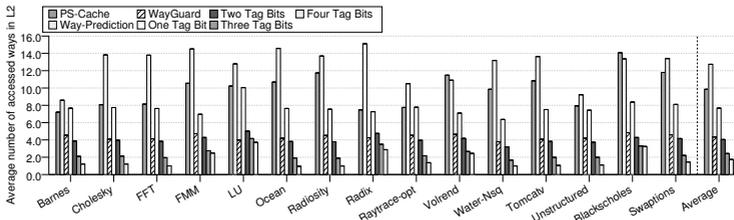
estudiadas. Como se puede observar, a mayor número de bits (de 1 a 4) que se utilice en el bit-array para filtrar las vías, menor el número de vías accedidas. De media, para una cache de 8 vías, se accede a 3.53 vías con un solo bit de etiqueta, 1.82 con dos bits, 1.06 con tres bits y 0.98 con cuatro. Esto significa que los accesos siguen una distribución de acceso uniforme considerando los bits de menor peso. Como era de esperar, el uso de tres bits es suficiente para limitar el número de vías accedidas necesarias a una sola, dado que nuestro primer nivel de cache tiene 8 vías. Por tanto, esto permite que el consumo de una cache asociativa por conjuntos sea similar a la de una cache de mapeado directo. No hay diferencias significativas en los resultados obtenidos para las distintas aplicaciones para un mismo número de bits de etiqueta para el filtrado. Puede darse el caso de tener un número medio de vías accedidas inferior a 1, ya que es posible que ninguno de los bits de menor peso de la etiqueta tengan una coincidencia en el bit-array. En este caso, no se accede a ninguna vía y el fallo de cache se dispara un poco antes de lo habitual.

Comparando con la PS-Cache, la propuesta ya consigue obtener mejores resultados con un único bit de etiqueta. La PS-Cache accede de media a 4.6 vías, aunque los resultados varían considerablemente entre las distintas aplicaciones. En algunos casos como *Ocean* apenas se consigue una reducción de vías accedidas, mientras que en otras (como *Tomcatv*) puede llegar a reducirlos aproximadamente en un 50%. El patrón de acceso a bloques privados y compartidos es muy distinto de una aplicación a otro, de ahí estos resultados. Way Guard y Way-Prediction acceden de media a 2.41 y 1.43 vías, respectivamente, lo cual se mantiene bastante constante en todas las aplicaciones. Obtienen mejores resultados que la propuesta con un único bit. Dos bits son suficientes para superar a Way Guard y se requiere de un tercero para mejorar a Way-Prediction. Utilizar la predicción de la vía más recientemente usada obtiene buenos resultados debido a su alta tasa de aciertos.

La figura 3(b) muestra el número medio de vías buscadas en una cache L2 de 16 vías. El número medio de vías accedidas en este nivel de la jerarquía de memoria es de 7.68, 4.04, 2.43 y 1.74 para un valor de X_s de 1, 2, 3 y 4 bits respectivamente. Como sucedía en las caches de primer nivel, descritas anteriormente, la reducción se distribuye de forma similar para todas las aplicaciones. La tendencia muestra que aún hay margen de mejora, pero hay un límite en cuan grande puede ser la pequeña estructura de filtrado. En comparación, la PS-Cache, Way-Prediction y Way Guard acceden a 9.85, 12.7 y 4.34, respectivamente. Way-Prediction, que funcionaba realmente bien en caches L1, funciona de forma más pobre en niveles inferiores de la jerarquía cache, ya que la L1 filtra la mayoría de los accesos del procesador, por tanto, la localidad de la aplicación empieza a quedar ofuscada conforme se va descendiendo en la jerarquía. Cuando la predicción acierta, solo se accede a una vía y cuando falla se tienen que acceder al resto. La figura muestra una tasa de acierto de LLC bastante baja. Es interesante mencionar, que un fallo de la predicción también supone ciclos adicionales para poder obtener la información de los datos.



(a) Número medio de vías accedidas en una cache L1 de 8 vías.



(b) Número medio de vías accedidas en una cache L2 de 16 vías

Fig. 3. Número medio de vías accedidas en la jerarquía de memoria.

Way-Prediction es por tanto perjudicial cuando se aplica a este nivel. Tanto Way-Prediction como la PS-Cache obtienen peores resultados que la propuesta incluso con un único bit, mientras que Way Guard obtiene resultados similares a TF-Cache cuando se utiliza un X_s de 2 bits.

La figura 4(a) muestra el consumo de energía dinámico de la cache de primer nivel evaluada en este trabajo. Los resultados se han normalizado con respecto a los de una cache asociativa por conjuntos en la que se accede a todas las vías. La TF-Cache es capaz de reducir el consumo de energía dinámico en un 48.1%, 65.8%, 73.2%, y 74.9% para un filtrado de 1, 2, 3 y 4 bits, respectivamente. Se pueden apreciar como los beneficios de los bits adicionales van menguando conforme vamos aumentando su número. Podemos asumir que los resultados con un filtro de 5 bits no serían muy diferentes a los de uno de 4. Como era de esperar por los resultados anteriores, Way-Prediction consigue los mejores resultados, ya que es capaz de reducir el consumo hasta en un 82.1%. Mientrastanto, la PS-Cache obtiene los peores resultados, ya que es el esquema que accede a un mayor número de vías.

De forma análoga, la figura 4(b) muestra los mismos resultados pero para la cache L2. La TF-Cache es capaz de reducir el consumo en 51.8%, 72.2%, 81.1% y 85.9% para un filtrado de etiquetas de 1, 2, 3 y 4 bits, respectivamente. De nuevo se puede apreciar como la diferencia entre el ahorro en consumo va menguando conforme aumentamos el tamaño del filtro. Way Guard obtiene unas reducciones similares a una TF-Cache de 2

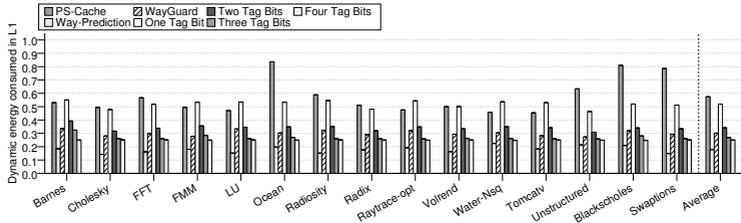
bits, mientras que PS-Cache y Way-Prediction no muestran esas mejoras al compararnos con la arquitectura propuesta, reduciendo el consumo en tan solo un 38.4% y 20.4%, respectivamente.

En resumen, la Tag Filter Cache obtiene unas ganancias de energía significativas, a mayor número de bits de etiqueta mejores, que consiguen superar a otras propuestas recientes como Way Guard, Way-Prediction y PS-Cache. Además, esta idea se puede aplicar a cualquier nivel de la jerarquía de memoria sin incurrir en una degradación de las prestaciones, como le sucede a Way-Prediction en niveles inferiores de cache.

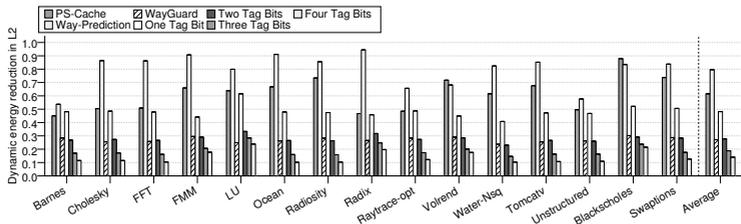
VII. CONCLUSIONES

Una de las mayores preocupaciones en el diseño de los multiprocesadores en chip de altas prestaciones es el consumo de energía, el cual va en aumento conforme se incrementa el número de núcleos. Las caches on-chip consumen una parte importante del consumo total, y en consecuencia, son un foco importante de investigación. Muchas de las técnicas propuestas consiguen reducir el consumo en estas estructuras de memoria, pero a costa de una pérdida de prestaciones.

En este trabajo se propone TF-Cache, un diseño de cache energéticamente eficiente que solo accede a un subconjunto de las vías del conjunto sin ningún tipo de degradación de prestaciones. La propuesta divide el tag array en dos estructuras diferentes. Una de ellas con uno pocos de los bits de menor peso y otra con el resto de los



(a) Energía dinámica consumida en una cache L1 de 8 vías normalizada con una cache convencional.



(b) Energía dinámica consumida en una cache L2 de 16 vías normalizada con una cache convencional.

Fig. 4. Energía dinámica consumida en la jerarquía de cache.

bits de la etiqueta. Para realizar el filtrado de las vías del conjunto, se realizan dos comparaciones. En la primera comparación, los bits de menor peso de la etiqueta del bloque buscado se comparan con los bits de menor peso de la estructura reducida del tag array. Esta comparación se puede hacer de manera rápida y sin necesidad de esperar a la salida de la TLB. Una vez se tiene el resultado de la comparación, se realiza una segunda comparando el resto de bits de etiqueta, aunque únicamente en aquellas vías en las que haya habido un acierto en la comparación previa. En el caso de que accedamos al data array en paralelo con el tag array, tan solo se buscaría en esas vías que han tenido un acierto en la primera comparación. De esta forma filtramos los accesos de la cache y se obtienen unas reducciones importantes del consumo dinámico de energía. Esta elección de filtrado es apropiado, ya que como los resultados muestran, hay una distribución homogénea a lo largo de las distintas vías de los bits de menor peso. Este diseño de cache puede implementarse en cualquier nivel de la jerarquía de cache, aunque aquellos que se acceden con mayor frecuencia obtendrán los mejores ahorros de energía. Además, a mayor asociatividad implementada en la cache, mayores son los beneficios potencial que puede aportar la TF-Cache.

Los resultados han mostrado que la TF-Cache puede reducir hasta un 87.75 % y 89.13 % el número medio de vías que se buscan cuando se aplica a una cache L1 y una L2, respectivamente. Esto se traduce en ahorros de energía. Concretamente, la arquitectura TF-Cache reduce

el consumo dinámico en un 74.9 % y 85.9 % cuando se aplica a la cache L1 y L2, respectivamente. Comparada con otros esquemas del estado del arte, la TF-Cache obtiene mejores resultados que las arquitecturas estudiadas, con la única excepción de Way-Prediction en caches de primer nivel por un pequeño margen. Por desgracia, Way-Prediction ha demostrado ser completamente inefectivo cuando se aplica a otros niveles de la jerarquía de cache, mientras que la propuesta funciona adecuadamente en cualquier nivel.

AGRADECIMIENTOS

Este trabajo se ha realizado conjuntamente con el apoyo de MINECO y la Comisión Europea (FEDER funds) bajo el proyecto TIN2015-66972-C5-1-R y por la *Fundación Seneca-Agencia de Ciencia y Tecnología de la Región de Murcia* bajo el proyecto *Jóvenes Líderes en Investigación 18956/JLI/13*.

REFERENCIAS

- [1] Rajeev Balasubramonian, Norman Paul Jouppi, and Naveen Muralimohanar. *Multi-Core Cache Hierarchies*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2011.
- [2] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. "Cache decay: Exploiting generational behavior to reduce cache leakage power." in *28th Int'l Symp. on Computer Architecture (ISCA)*, June 2001, pp. 240–251.
- [3] Krisztián Flautner, Nam Sung Kim, Steve Martin, David Blaauw, Trevor Mudge, Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. "Drowsy caches: Simple techniques for reducing leakage power." in *29th Int'l Symp. on Computer Architecture (ISCA)*, May 2002, pp. 148–157.
- [4] Mrinmoy Ghosh, Emre Özer, Simon Ford, Stuart Biles, and

- Hsien-Hsin S. Lee, "Way guard: A segmented counting bloom filter approach to reducing energy for set-associative caches," in *Int'l Symp. on Low Power Electronics and Design (ISLPED)*, Aug. 2009, pp. 165–170.
- [5] Brad Calder and Dirk Grunwald, "Predictive sequential associative cache," in *2nd Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 1996, pp. 244–253.
- [6] Karthik T. Sundararajan, Vasileios Porpodas, Timothy M. Jones, Nigel P. Topham, and Björn Franke, "Cooperative partitioning: Energy-efficient cache partitioning for high-performance cmps," in *18th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2012, pp. 311–322.
- [7] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2009, pp. 469–480.
- [8] Jongmin Lee, Seokin Hong, and Soontae Kim, "Tlb index-based tagging for cache energy reduction," in *17th Int'l Symp. on Low Power Electronics and Design (ISLPED)*, Aug. 2011, pp. 85–90.
- [9] Michael Powell, Se hyun Yang, Babak Falsafi, Kaushik Roy, and T. N. Vijaykumar, "Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories," in *Int'l Symp. on Low Power Electronics and Design (ISLPED)*, July 2000, pp. 90–95.
- [10] David H. Albonesi, "Selective cache ways: On-demand cache resource allocation," in *32nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 1999, pp. 248–259.
- [11] Mrinmoy Ghosh, Emre Özer, Stuart Biles, and Hsien-Hsin S. Lee, "Efficient system-on-chip energy management with a segmented bloom filter," in *19th Int'l Conf. on Architecture of Computing Systems (ARCS)*, Mar. 2006, pp. 283–297.
- [12] Joan J. Valls, Alberto Ros, Julio Sahuquillo, and María Engracia Gómez, "PS-cache: An energy-efficient cache design for chip multiprocessors," in *22nd Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2013, pp. 407–408.
- [13] Alberto Ros, Polychronis Xekalakis, Marcelo Cintra, Manuel E. Acacio, and José M. García, "Ascibi: Adaptive selection of cache indexing bits for reducing conflict misses," in *Int'l Symp. on Low Power Electronics and Design (ISLPED)*, July 2012, pp. 51–56.
- [14] Kamil Kedzierski, Francisco J. Cazorla, Roberto Gioiosa, Alper Buyuktosunoglu, and Mateo Valero, "Power and performance aware reconfigurable cache for cmps," in *2nd Int'l Forum on Next-Generation Multicore/Manycore Technologies*, June 2010, pp. 1–12.
- [15] "27-inch imac, technical specifications, available online (nov, 2014) at <http://www.apple.com/imac/specs/>."
- [16] Peter S. Magnusson, Magnus Christensson, and Jesper Eskilson, et al., "Simics: A full system simulation platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.
- [17] Milo M.K. Martin, Daniel J. Sorin, and Bradford M. Beckmann, et al., "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Sept. 2005.
- [18] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2009, pp. 33–42.
- [19] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi, "Cacti 6.0," Tech. Rep. HPL-2009-85, HP Labs, Apr. 2009.
- [20] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *22nd Int'l Symp. on Computer Architecture (ISCA)*, June 1995, pp. 24–36.
- [21] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *17th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2008, pp. 72–81.

Docencia en Arquitectura y Tecnologías de Computadores

WepSIM: simulador integrado de microprogramación y programación en ensamblador

Javier Prieto Cepeda,¹ Félix García-Carballeira,²
Alejandro Calderón Mateos³ y Saúl Alonso Monsalve⁴

Resumen— En este artículo se presenta WepSIM, un simulador portable de microprogramación y programación en ensamblador, que intenta mejorar la enseñanza y comprensión de la asignatura Estructura de Computadores. Este simulador incorpora en una misma herramienta la microprogramación y la programación en ensamblador. Permite la definición del formato completo de un juego de instrucciones máquina, la definición del microcódigo asociado al mismo, la creación de programas en ensamblador que usan dicho juego de instrucciones y la simulación y ejecución paso a paso de programas en ensamblador. La ejecución de programas se puede realizar instrucción a instrucción o microinstrucción a microinstrucción. WepSIM permite a los estudiantes entender el funcionamiento de un procesador elemental de una forma fácil e intuitiva a través de su interfaz. Los estudiantes podrán ver la evolución y generación del código, de una forma incremental desde el diseño del microcódigo y generación de la memoria de control, hasta la generación del binario de un programa en ensamblador y la visualización de su ejecución ciclo a ciclo.

Palabras clave— Estructura de Computadores, microprogramación, programación en ensamblador.

I. INTRODUCCIÓN

Es muy frecuente utilizar distintas herramientas para las prácticas de la asignatura de Estructura de Computadores, en función de la parte del temario con la que se esté trabajando. Esto supone un tiempo extra para la enseñanza de cada una de estas herramientas. También supone aprender herramientas específicas lo que facilita la pérdida de la visión global. Todo ello suma complejidad para los alumnos en la comprensión del funcionamiento de un procesador a bajo nivel junto a su unidad de control. Otro factor a tener en cuenta, es que la mayor parte de las herramientas están pensadas para ejecutarse en PC con distintos sistemas operativos, pero cada día la enseñanza usa plataformas con más movilidad y estas herramientas difícilmente están adaptadas a estas nuevas plataformas.

Para ayudar a solucionar estos problemas proponemos WepSIM, que facilita la integración de la enseñanza de la microprogramación y la programación en ensamblador, ofreciendo simulaciones tanto a nivel de microinstrucción como a nivel de instrucción, de forma que sea la única herramienta

utilizada a lo largo del curso. Ello permite una enseñanza incremental sobre la misma plataforma.

Este simulador permite la definición de diferentes juegos y formatos de instrucciones, la definición del microcódigo asociado a dicho juego de instrucciones y la ejecución de programas en ensamblador que utilizan las instrucciones máquina definidas previamente. Es posible definir instrucciones de distintos tipos de procesador (MIPS, ARM, Intel, Z80, etc.) lo que facilita posteriormente cargar un programa que use el conjunto de instrucciones definido. El simulador proporciona inicialmente un subconjunto de las instrucciones del MIPS, pero el alumno puede definir instrucciones máquina de otros conjuntos de forma similar.

Además, debido a la gran variedad de dispositivos con los que cuentan los alumnos hoy día, y solventando el problema de portabilidad, WepSIM es una herramienta multiplataforma, estando disponible en versión Web, y se está trabajando en una aplicación móvil para dispositivos Android y Windows Phone 10.

El resto del documento se estructura de la siguiente forma: la sección II describe el estado del arte; la sección III presenta el simulador WepSIM y sus distintas visiones. Por último la sección IV presenta las principales conclusiones y trabajos futuros.

II. ESTADO DEL ARTE

Para la programación en ensamblador, los simuladores más conocidos para labores docentes son SPIM y Mars. SPIM [1] es un simulador de un procesador MIPS de 32 bits que permite ejecutar (y depurar) programas en ensamblador para esta arquitectura. Este simulador creado por James R. Larus tiene versiones compiladas para Windows, Mac OS X y Unix/Linux e incluso tiene una versión básica para Android [2]. Desafortunadamente, no permite su uso para microprogramación, no está pensado su diseño para dispositivos móviles (la versión existente busca parecerse a la versión para ordenador) y se basa en el juego de instrucciones de MIPS32.

Mars [3] es también un simulador de un procesador MIPS de 32 bits similar a SPIM desarrollado en Java. Sus autores destacan la interfaz gráfica, la edición integrada y la reciente incorporación de una utilidad para acceder al hardware desde ensamblador. De igual forma que antes, no ofrece una visión integrada, no se encuentra versión alguna para dispositivos móviles y se centra en MIPS32.

¹Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: javprieto@pa.uc3m.es.

²Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: fgcarbal@inf.uc3m.es.

³Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: acaldero@inf.uc3m.es.

⁴Dpto. de Informática, Universidad Carlos III de Madrid

Simulador	SPIM	MARS	PC88110	WepSIM	P8080E	MicMac
Ensamblador	✓	✓	✓	✓		
Microprogramación				✓	✓	✓
Multiplataforma	✓			✓		
Interactivo				✓		
Juego de instrucciones				✓	✓	
Personalizable						

TABLA I
 COMPARACIÓN DE WEPsim CON OTROS SIMULADORES.

Otro simulador conocido en entornos docentes es el pc88110 descrito en [4], y como en casos anteriores, se centra en una arquitectura específica (MC88110) y no ofrece una ayuda en la movilidad del estudiante. Aunque integra el efecto de un procesador superescalar, no integra la microprogramación. Este simulador presenta un trabajo muy interesante descrito en [5], donde se muestra la metodología que hay detrás del simulador pc88110 para ser usado en prácticas. Ello incluye la descripción del entorno de prácticas, el sistema de entrega, el corrector automático, la detección de copias, etc.

Fuera del ámbito docente hay simuladores/emuladores como por ejemplo OVPSim [6] o GXemul [7] que permiten trabajar con distintas arquitecturas como por ejemplo ARM, MIPS, PowerPC, etc. En el caso de OVPSim, se ha usado para la investigación de plataformas de computación paralelas [8], co-diseño de hardware/software [9], etc. y aunque es posible su uso para labores docentes, por su nivel de detalle es posible que no sea la mejor herramienta con la que empezar a aprender.

Para labores docentes en microprogramación, destacamos el simulador P8080E [10] desarrollado en el DATSI de la Facultad de Informática de la Universidad Politécnica de Madrid. A diferencia del P8080E nuestra propuesta (a) presenta una interfaz gráfica portable e interactiva, y (b) el ensamblador a utilizar se define en el microcódigo de forma que luego es posible compilarlo y generar el binario asociado al mismo. En el P8080E esta última labor se hace a mano, en la definición de las instrucciones no se incluye su formato y no está pensado para poder usarse para enseñar tanto microprogramación como ensamblador con la misma herramienta. Otros simuladores para microprogramación son el UT1000 [11] de 1989 y MicMac [12] de 1987. MicMac está pensado para usar su propio código máquina denominado Mac1 (no tanto diseñar nuevos), y UT1000 es descrito por sus autores como un simulador de una CPU de 16 bits con un secuenciador de AMD2910. Desafortunadamente UT1000 tras más de 25 años, es un proyecto a día de hoy no accesible.

Por tanto, como podemos observar en la tabla I, WepSIM aúna las características de las diversas herramientas comentadas anteriormente, facilitando el aprendizaje de forma iterativa en un único simulador.

III. WEPsim

WepSIM¹, es una plataforma, que integra el uso de la microprogramación con la programación en ensamblador. Permite la definición del formato completo de un juego de instrucciones máquina, la definición del microcódigo asociado al mismo, la creación de programas en ensamblador que usan dicho juego de instrucciones y la simulación y ejecución paso a paso de programas en ensamblador. La ejecución de los programas se puede realizar instrucción a instrucción o microinstrucción a microinstrucción.

WepSIM se basa un procesador elemental de 32 bits [13], que direcciona la memoria por bytes, (véase la Figura 1) con una unidad de control microprogramada (véase la Figura 2), que utiliza secuenciamiento implícito. Utiliza un bus interno y dispone de un banco con 32 registros. Para simplificarlo, se ha decidido que únicamente conste de una Unidad Aritmético Lógica de números enteros. El mapa de memoria y de entrada/salida se encuentran separados.

En WepSIM, una vez ensamblado un programa en ensamblador, se pueden realizar simulaciones del código visualizando la activación de las señales del procesador y el tránsito de datos a través de los buses y registros. De esta forma, WepSIM, ofrece 3 funcionalidades diferentes: la definición de un juego de instrucciones y su microcódigo, la creación de programas en ensamblador que utilizan dicho juego de instrucciones, y la depuración y simulación del microcódigo diseñado y de los programas en ensamblador creados.

La explicación de estas tres funcionalidades va a contemplantar:

1. La definición de un determinado juego de instrucciones y su microcódigo.
2. La definición y carga de programas en ensamblador.
3. Los principales aspectos relacionados con el uso del simulador (ejecución paso a paso de instrucciones o microinstrucciones, puntos de ruptura, señales interactivas, etc.)

A. Microprogramación en WepSIM

WepSIM permite el diseño del conjunto de instrucciones que conforman el ensamblador a utilizar

¹El simulador se encuentra disponible en <http://www.arcos.inf.uc3m.es/~ec-2ed>

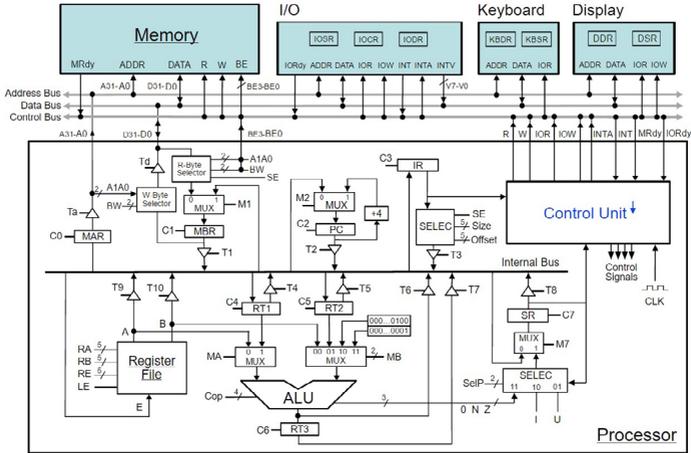


Fig. 1. Procesador elemental de WepSIM

en la plataforma. De esta forma, se pueden indicar las instrucciones que constituyen el juego de instrucciones y su formato, junto con el microprograma que conforma cada instrucción máquina. Esta descripción, se realiza en un fichero de texto en el cual se describe el formato de cada instrucción, los nombres simbólicos de los registros del procesador y las pseudoinstrucciones que se desean añadir para la programación en ensamblador. WepSIM, permite la carga de un microcódigo de ejemplo, que se puede modificar con el editor, guardar y cargar de nuevo. La figura (3), muestra un ejemplo de definición para la instrucción *li* (*load immediate*), que almacena un valor inmediato en un registro.

El fichero con el cronograma asociado a *fetch* y todos los cronogramas de las instrucciones, definen el microcódigo para la plataforma WepSIM. De esta forma, el simulador permite la definición de diferentes juegos y formatos de instrucciones. El simulador proporciona un subconjunto de las instrucciones del MIPS, pero los alumnos pueden definir juegos de instrucciones de otros conjuntos de forma similar. La posibilidad de asignar códigos simbólicos a los registros del banco de registros, permite que en los programas escritos en ensamblador se puedan usar dichos símbolos.

En la figura 3, observamos los diferentes componentes que forman la definición de la instrucción máquina:

- Formato: indica el formato que tiene la instrucción en ensamblador para su posterior uso. Incluye el nombre de la instrucción y los campos de dicha instrucción.
- Código de operación: el código de operación, se

especifica con la etiqueta "co". Será especificado en un número binario de 6 bits, lo que permite definir hasta 64 instrucciones distintas. En el caso de las operaciones aritméticas, se pueden utilizar los últimos 4 bits de la instrucción para especificar la operación de la ALU a realizar, de forma que todas las instrucciones de este tipo, constan del mismo "co", ahorrando el consumo de instrucciones y permitiendo hasta un máximo de 79 instrucciones (63 + 16).

- Número de palabras: este campo indica el número de palabras que precisa la instrucción para su definición y carga en memoria. Una palabra en WepSIM son 4 bytes.
- Definición de los campos de la instrucción: cada uno de los campos que definen la instrucción, han de ser descritos para indicar el tipo de campo y número de bits que ocupan en la instrucción. Estos campos pueden ser de 3 tipos: registro, valor inmediato o dirección. Un campo de tipo registro necesita un campo de 5 bits en la instrucción para la especificación de un registro del banco de registros. Un campo de tipo inmediato indica que se especificará un valor, que ocupará un determinado número de bits en la instrucción. Un campo de tipo dirección se utilizará para direcciones de memoria. Para ello, se debe definir el tipo de direccionamiento, relativo o absoluto. Si el direccionamiento es relativo, se indicará mediante la etiqueta "rel" tras el paréntesis de cierre de la especificación de bits del campo. Por el contrario, si el direccionamiento es absoluto, se indicará con la etiqueta "abs".

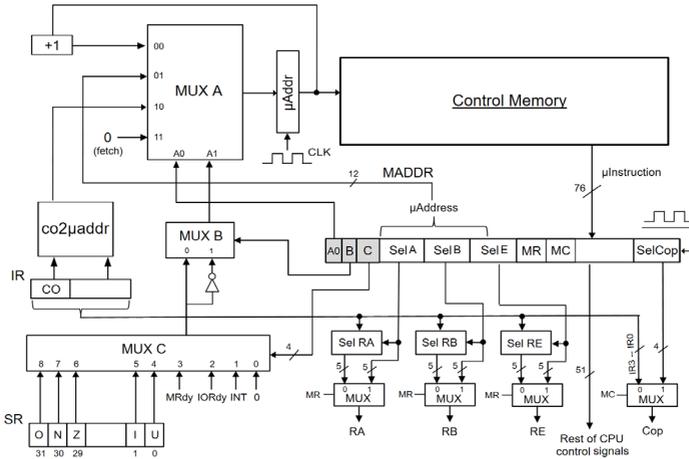


Fig. 2. Unidad de control del procesador WepSIM

Una vez especificados todos los campos que conforman la instrucción, se procede a describir el microprograma asociado. De esta forma, se indicarán las señales que se activan en cada ciclo de reloj cuando son de un bit, y para las señales de más de un bit, se indicará su valor en binario.

```

li reg val {
    co=000010,
    memords=1,
    reg=reg(25,21),
    val=inm(15,0),
    {
        (SE=0, OFFSET=0, SIZE=10000, SE=1, T3=1,
        LE=1, MR=0, SELE=10101, A0=1, B=1, C=0)
    }
}
    
```

Fig. 3. Definición de la instrucción li en WepSIM.

En la figura 4 se puede observar el formato de instrucción asociado a la instrucción definida en la Figura 3.

En cuanto a la asignación de nombres simbólicos a los registros del banco de registros, WepSIM, permite

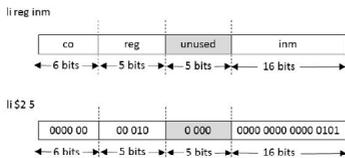


Fig. 4. Formato de la instrucción li

la asignación de nombres a los 32 registros existentes. Además, es necesario indicar el registro que será utilizado para el puntero de pila.

B. Ensamblador en WepSIM

Una vez definido el juego de instrucciones del ensamblador deseado, se pueden realizar programas escritos en este ensamblador. WepSIM, permite al igual que en el caso del microcódigo, la carga de un fichero de ejemplo, pudiendo ser modificado con el editor, guardado y cargado posteriormente. En él, se pueden definir dos segmentos: datos y código.

En el segmento de datos, se podrán definir las variables deseadas para usar posteriormente en el segmento de código. Las directivas admitidas por WepSIM, son las siguientes:

- **.byte**: define un byte, permitiendo el formato octal, hexadecimal, decimal y carácter.
- **.half**: define media palabra (2 bytes), permitiendo el formato octal, hexadecimal y decimal.
- **.word**: define una palabra (4 bytes), permitiendo el formato octal, hexadecimal y decimal.
- **.ascii**: define una cadena de caracteres sin incluir el carácter '\0' al final de la cadena.
- **.asciiz**: define una cadena de caracteres incluyendo el carácter '\0' al final de la cadena.
- **.space**: define una reserva de espacio en memoria del número de bytes indicado en formato decimal.
- **.align**: alinea en memoria el dato definido a continuación.

Una vez definido el segmento de datos, se pasa a definir el segmento de código. WepSIM, permite el

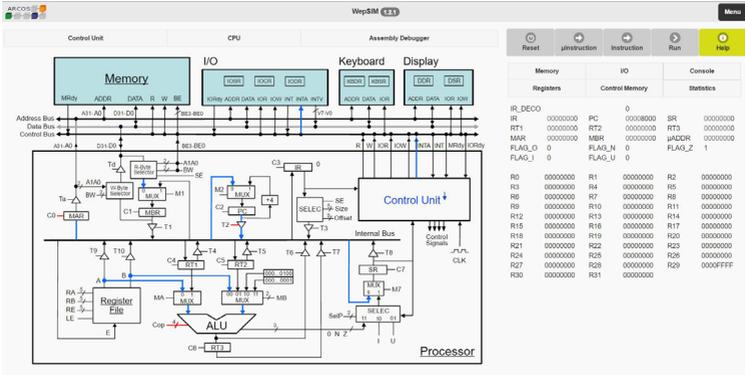


Fig. 5. Visualización del procesador y microejecución/ejecución paso a paso.

uso de etiquetas para el uso de subrutinas y saltos condiciones o incondicionales. Además, se pueden usar tanto los nombres simbólicos dados a los registros (ej: \$zero) en la definición del microcódigo, como el número de registro deseado (ej: \$0). WepSIM, además, permite que para referenciar a direcciones de memoria, se puedan usar tanto las etiquetas, que indican la dirección a la que se referencia, como el valor de la dirección en los diferentes formatos permitidos (decimal, octal y hexadecimal).

Una vez desarrollado el código ensamblador, se puede compilar y cargar en memoria principal, mediante el botón correspondiente.

C. Ejecución de un microprograma y programa en ensamblador en WepSIM

Una vez definido el juego de instrucciones, y ensamblado y cargado en memoria principal el código ensamblador, se podrá realizar la simulación de la ejecución de éste en WepSIM. Esta simulación, puede realizarse tanto a nivel de microinstrucción, como a nivel de instrucción máquina. WepSIM, permite mediante el esquema de la arquitectura del procesador, ver en tiempo de ejecución, las señales activadas en cada ciclo de reloj. Esto resulta muy interesante, puesto que ayuda a trazar la ejecución de las instrucciones a nivel de ciclo. Se puede visualizar la activación de las señales de control en la arquitectura, tanto a nivel de procesador, como de unidad de control (véase las figuras 5 y 7).

WepSIM ofrece, además de la simulación visual, la realización de simulaciones observando únicamente el código escrito en ensamblador, y el avance de la ejecución del mismo, permitiendo el uso de puntos de ruptura (*breakpoint*) para detener la ejecución en las instrucciones deseadas (véase la figura 8). Cuando se detiene la ejecución en un "breakpoint", se detendrá antes de que la instrucción comience a ejecutar el *fetch* correspondiente. De este modo, se podrá ver

```
.data
age1: .word 0x20,20
age2: .word 20,10
resultado: .word 0
# 32-bit word initialized with decimal
texto: .ascii "Hola!"
texto2: .ascii "Hola!"
hueco: .space 16

.text
.globl main
main: li $10,45
      li $3,5
      li $4,1
      li $5,0
      d: sub $3,$4,$4
         bne $3,$5,d
         lw $1,age1
         lw $2,age2
         add $4,$1,$2
         exit: sw $4,resultado
```

Fig. 6. Ejemplo de programa ensamblador.

el estado de la máquina en el momento previo a la ejecución de la instrucción.

Durante las simulaciones, WepSIM ofrece la visualización del estado de diferentes parámetros del procesador en tiempo de ejecución. Algunos de estos parámetros son el contenido de la memoria principal, el estado de los diferentes registros y flags, la memoria de control (señalando la microinstrucción que se encuentra en ejecución), etc. Además, WepSIM incorpora un módulo de entrada/salida, que permite la interacción del usuario con el procesador, de modo que en instrucciones de tipo 'in/out', se pueda visualizar en el monitor de WepSIM los valores que se desean imprimir, e introducir mediante el teclado de WepSIM los valores deseados.

Otro de los aspectos que es importante destacar de WepSIM, es que permite la generación de interrupciones, pudiendo analizar durante la ejecución el número de interrupciones generadas de cada uno de los tipos definidos.

En WepSIM se pueden configurar diferentes opciones, de forma que se puede elegir el formato de los valores mostrados durante la simulación (Decimal, Octal, Hexadecimal), los colores a usar para las señales, la velocidad de simulación, la nomenclatura de las direcciones (numérica o etiquetas) y la acti-

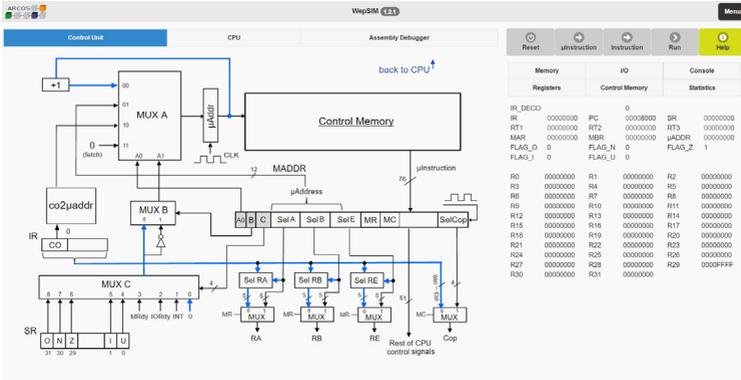


Fig. 7. Visualización de la unidad de control y microejecución/ejecución paso a paso.

La imagen muestra la interfaz de usuario del simulador WepSIM. A la izquierda, una tabla de 'breakpoints' con columnas para 'address', 'assembly' y 'pseudo-instructions'. A la derecha, una tabla de 'Registers' que muestra el estado de los registros de propósito general (R0-R31) y los registros de control (PC, PSR, MBR, FLAG_O, FLAG_I).

Registers	Control Memory	Console			
IR_DECO	0	Statistics			
SR	00000000	PC	00008000	SR	00000000
RT1	00000000	RT2	00000000	RT3	00000000
MAR	00000000	MBR	00000000	μADDR	00000000
FLAG_O	0	FLAG_N	0	FLAG_Z	1
FLAG_I	0	FLAG_U	0		
R0	00000000	R1	00000000	R2	00000000
R3	00000000	R4	00000000	R5	00000000
R6	00000000	R7	00000000	R8	00000000
R9	00000000	R10	00000000	R11	00000000
R12	00000000	R13	00000000	R14	00000000
R15	00000000	R16	00000000	R17	00000000
R18	00000000	R19	00000000	R20	00000000
R21	00000000	R22	00000000	R23	00000000
R24	00000000	R25	00000000	R26	00000000
R27	00000000	R28	00000000	R29	00000000
R30	00000000	R31	00000000		

Fig. 8. Visualización del programa ensamblador y su ejecución paso a paso.

vación del modo interactivo.

El modo interactivo de WepSIM, permite modificar en tiempo de ejecución el valor de una señal de control en el ciclo detenido. De esta forma, se puede corregir en tiempo de ejecución un error encontrado en el microcódigo y exportar el fichero de microcódigo con la corrección realizada.

IV. CONCLUSIONES Y TRABAJOS FUTUROS

Este trabajo presenta un nuevo simulador que es intuitivo, portable y que incorpora en una misma herramienta la microprogramación y la programación en ensamblador. Permite definir diferentes juegos de instrucciones, y ejecutar y depurar código fuente escrito en dicho juego de instrucciones. Proporciona una gran simplicidad de uso, lo que hace que sea muy útil y potente como herramienta para las prácticas y la docencia en Estructura de Computadores. WepSIM, permite a los estudiantes, en-

tender el funcionamiento de un procesador elemental de una forma fácil e intuitiva a través de su interfaz. Los estudiantes, podrán ver la evolución y generación del código, de una forma incremental, desde el diseño del microcódigo y generación de la memoria de control, hasta la generación del binario de un programa en ensamblador y la visualización de la ejecución de éste ciclo a ciclo.

Hay varias líneas de trabajos futuros en las que actualmente se está trabajando:

- Completar con una herramienta de pruebas basada en el motor de WepSIM que permita ejecutar un microcódigo con varios programas en ensamblador y un programa en ensamblador con varios microcódigos.
- Actualmente, estamos trabajando en el desarrollo de una aplicación móvil basada en el proyecto Apache Cordova para iOS, de forma que el alumno sea capaz de trabajar incluso sin

conexión a internet.

- Introducir más elementos hardware, como una memoria caché.
- Diseñar un sistema operativo mínimo adaptado a WepSIM, que permita la simulación total de una máquina en la plataforma.

REFERENCIAS

- [1] James R. Larus, "SPIM," Feb. 2016.
- [2] Jim Matak, "Assembly Emulator," Feb. 2016.
- [3] Pete Sanderson and Ken Vollmar, "MARS," Feb. 2016.
- [4] José M. Pérez Villadeamigo, Santiago Rodríguez de la Fuente, Rafael Méndez Cavanillas, and M. Isabel García Clemente, "The em88110: Emulating a super-scalar processor," *SIGCSE Bull.*, vol. 29, no. 4, pp. 45–50, Dec. 1997.
- [5] Antonio García Dopico, Santiago Rodríguez de la Fuente, and Francisco Javier Rosales García, "Automatización de prácticas en entornos masificados," in *Actas de las IX Jornadas de Enseñanza universitaria de la Informática*, Spain, 2003, Jenni 2003, pp. 119–126, Thonson-Paraninfo.
- [6] Imperas Software, "Open Virtual Platforms simulator," Mar. 2016.
- [7] Anders Gavare, "GXemul," Mar. 2016.
- [8] Christian Pinto, Shivani Raghav, Andrea Marongiu, Martino Ruggiero, David Atienza, and Luca Benini, "Gpgpu-accelerated parallel and fast simulation of thousand-core platforms," in *CCGRID*. 2011, pp. 53–62, IEEE Computer Society.
- [9] I. Nita, V. Lazarescu, and R. Constantinescu, "A new hw/sw co-design method for multiprocessor system on chip applications," in *ISSCS*. 2009, pp. 1–4, IEEE Computer Society.
- [10] DATSLFLUPM.ES, "P8080E," Apr. 2016.
- [11] F. Cornett, "The ut1000 microprogramming simulator: An educational tool," *SIGARCH Comput. Archit. News*, vol. 17, no. 4, pp. 111–118, June 1989.
- [12] John L. Donaldson, "Micmac: A microprogram simulator for courses in computer organization," *SIGCSE Bull.*, vol. 19, no. 1, pp. 428–431, Feb. 1987.
- [13] Félix García Carballeira, Jesús Carretero Pérez, José Daniel García Sánchez, and David Expósito Singh, *Problemas resueltos de estructura de computadores, segunda edición*, vol. 1, pp. 1–307, Ediciones Paraninfo, 2015.

MIPSfpga: Una infraestructura para la enseñanza de asignaturas del área de Arquitectura y Tecnología de Computadores

Sarah Harris¹, Robert Owen², Enrique Sedano³ y Daniel Chaver⁴

Resumen— En este artículo presentamos un curso basado en MIPSfpga, una infraestructura basada en un soft-core MIPS, puesta a disposición de la comunidad académica por la empresa Imagination Technologies. El curso consta de una primera parte en la que se estudia la arquitectura MIPS y una serie de conceptos básicos de microarquitectura, y una segunda parte más extensa, en la que se realizan una serie de prácticas centradas en el diseño de Systems on Chip (SoC) y en el co-diseño Hardware-Software. Dichas prácticas comienzan con una parte básica en la que se guía al estudiante en la carga y ejecución del sistema MIPSfpga en una Field-Programmable Gate Array (FPGA) y la posterior ejecución y depuración de programas sobre dicho sistema. A continuación, en la segunda parte de las prácticas se explica cómo modificar la infraestructura MIPSfpga para incluir nuevos periféricos. El curso se completa con el desarrollo de un proyecto por parte del alumno.

Palabras clave— MIPS, Docencia, Arquitectura y Tecnología de Computadores, co-diseño Hardware-Software, Field-Programmable Gate Array (FPGA), Soft-Core, Procesador, Entrada/Salida mapeada en Memoria, System-on-Chip (SoC).

I. INTRODUCCIÓN

La infraestructura MIPSfpga [1, 2], basada en un procesador soft-core con arquitectura MIPS, y puesta a disposición de la comunidad académica por la empresa Imagination Technologies de forma gratuita, permite a los estudiantes aprender múltiples conceptos de diseño de Systems on Chip (SoC) y de co-diseño hardware-software, en un procesador MIPS comercial. Los estudiantes pueden utilizar una única herramienta hardware, una Field-Programmable Gate Array (FPGA), para mejorar, de forma práctica, sus conocimientos en las áreas de diseño digital o arquitectura de computadores.

Si bien hay múltiples procesadores soft-core disponibles desde hace décadas, MIPSfpga es el primer procesador soft-core MIPS comercial que se pone a disposición de la comunidad académica. En multitud de Universidades se utiliza la arquitectura MIPS para la docencia de asignaturas del área de Arquitectura y Tecnología de Computadores. MIPSfpga permite reducir la brecha existente entre el uso de procesadores MIPS reales y sus herramientas de soporte, y los procesadores MIPS “de juguete” típicamente empleados en docencia.

En el resto del paper, se proporciona en primer lugar un resumen del curso y del sistema MIPSfpga en las Secciones II y III respectivamente. A continuación, en las secciones IV y V, describimos las prácticas y el proyecto final, y se ilustran, en la Sección VI, diversas experiencias prácticas con MIPSfpga. Por último, se describe el trabajo relacionado y el trabajo futuro en las Secciones VII y VIII respectivamente, y se concluye el paper con las Secciones IX (conclusiones) y X (agradecimientos).

II. RESUMEN DEL CURSO

El curso comienza exponiendo conceptos básicos de arquitectura de computadores y finaliza con prácticas en las que el estudiante llega a modificar el sistema MIPSfpga. Se requiere que el alumno disponga de algunos conocimientos previos de diseño digital, y es recomendable que tenga cierta experiencia en programación, aunque, de no ser así, ésta también se puede enseñar de forma paralela.

La enseñanza de fundamentos, organización y estructura, arquitectura de computadores o microarquitectura se realiza tomando como base la arquitectura MIPS y empleando como texto principal [3]. Tras desarrollar las prácticas propuestas en dicho libro, en las que se llega a construir un procesador con arquitectura MIPS muy básico, se muestra a los estudiantes el funcionamiento del sistema MIPSfpga por medio de la *MIPSfpga Getting Started Guide*, proporcionada como parte del paquete básico de MIPSfpga [4]. A continuación, los estudiantes realizan las prácticas incluidas en *MIPSfpga Fundamentals*, otro paquete de la infraestructura proporcionada por Imagination Technologies [4]. El curso finaliza con la realización de un proyecto final por parte de los alumnos.

III. RESUMEN DE MIPSFPGA

Las prácticas del curso están centradas en torno a un procesador soft-core comercial MIPS, suministrado por medio de una serie de ficheros Verilog incluidos en el paquete *MIPSfpga Getting Started* [4]. Las herramientas de programación y depuración (Codescape MIPS SDK Essentials y OpenOCD) también van incluidas en el paquete. En esta sección proporcionamos un resumen del core y del sistema MIPSfpga.

A. Core MIPS

El core incluido en MIPSfpga es una versión reducida del core `micrAptiv-UP`, utilizado en el microcontrolador de Microchip PIC32MZ, y más recientemente en el módulo de Samsung Artik-1, un dispositivo de bajo consumo orientado a aplicaciones de Internet of Things (IoT). Dicho core implementa la Instruction Set Architecture (ISA) MIPS32r3 en un pipeline de 5 etapas [5]. El core incluye una Memory Management Unit (MMU) con un Translation Lookaside Buffer (TLB), caches de instrucciones y datos, y diversos interfaces (como EJTAG), tal y como ilustra la Figura 1. El bus utiliza el protocolo AMBA 3 AHB-Lite [6]. En la hoja de especificaciones del core [5] se pueden encontrar el resto de especificaciones del mismo.

B. Sistema MIPSfpga

La Figura 2 muestra el Sistema MIPSfpga, que incluye el core, los periféricos, y el bus AHB-Lite de

¹ Univ. de Nevada, Las Vegas, USA / Sarah.Harris@unlv.edu

² Imagination Technologies / Robert.Owen@imgtec.com

³ Imagination Technologies / Enrique.Sedano@imgtec.com

⁴ Univ. Complutense de Madrid / dani02@uicm.es

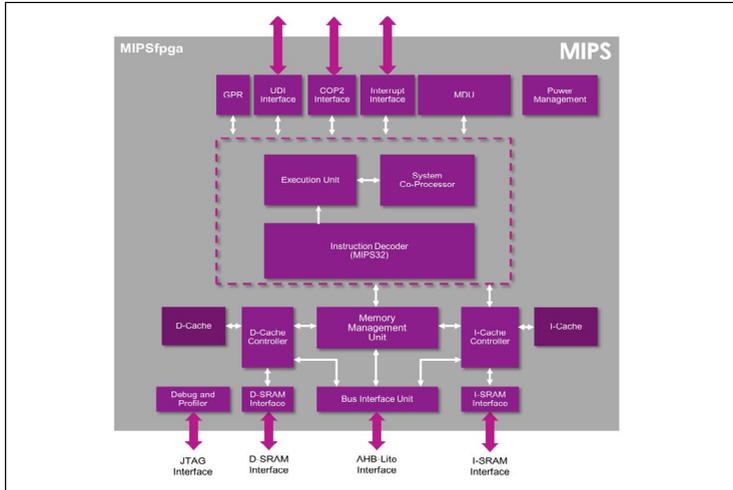


Fig. 1. Diagrama de bloques del procesador incluido en MIPSfpga (figura obtenida de la MIPSfpga Getting Started Guide [2])

comunicación entre ambos. Los periféricos incluyen la memoria principal, implementada en la RAM de bloque de la FPGA, y el General-Purpose I/O (GPIO), con el que se gestiona la comunicación con los LEDs, los interruptores, los botones o los displays de 7-segmentos de la placa FPGA. El sistema necesita una señal de reloj (SI_ClkIn) y una de reset (SI_Reset_N), activa en baja. Además, aunque su uso no es imprescindible, existe un interfaz EJTAG que facilita la carga y permite la depuración de programas en el sistema MIPSfpga.

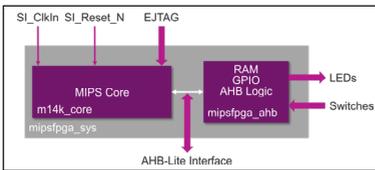


Fig. 2. Sistema MIPSfpga

Se incluyen dos bloques de memoria, uno a partir de la dirección física 0x0 (Code/Data RAM) y otro a partir de la dirección física 0x1fc00000 (Reset RAM), como ilustra la Figura 3. Tras el reset, el procesador comienza buscando instrucciones a partir de la dirección 0x1fc00000. Así pues, como mínimo, esa dirección debe contener instrucciones. Normalmente, dichas instrucciones corresponden a código de arranque que se encarga de inicializar el sistema. Tras ello, se salta al código de usuario, ubicado en el otro bloque de memoria. Aunque este es el procedimiento habitual, en códigos simples que no utilizan ciertos componentes del sistema (como las caches o la TLB), es posible

prescindir del código de arranque y ubicar el código de usuario a partir de la dirección física 0x1fc00000 para comenzar la ejecución inmediatamente tras el reset.

MIPSPfga incluye por defecto un bloque de memoria RAM de arranque (Reset RAM en la Figura 3) de 128KB, y un bloque de memoria RAM de programa (Code/Data RAM en la Figura 3) de 256KB. Este sistema está concebido para trabajar con las placas Digilent Nexys4 DDR o Terasic DE2-115. No obstante, si trabajamos con placas más reducidas (por ejemplo con una Digilent Basys3 o una Terasic DE0), el sistema puede reconfigurarse para incluir bloques de memoria más pequeños.

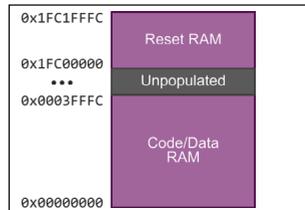


Fig. 3. Memoria física en MIPSfpga

IV. PRÁCTICAS

Tras estudiar la arquitectura MIPS y el procesador y el sistema de MIPSfpga, los estudiantes realizan ocho prácticas guiadas donde configuran MIPSfpga, interactúan con el sistema realizando prácticas de complejidad creciente, y extienden el sistema de distintas maneras. Una novena práctica describe cómo

portar MIPSfpga a otras placas FPGA. Esta sección describe los requisitos hardware y software para completar las prácticas, así como una revisión de cada práctica.

A. Requisitos hardware

Los requisitos hardware para realizar las prácticas son una placa FPGA y una placa Bus Blaster, tal y como aparece en la Tabla I. Para las prácticas 7 y 8, que son opcionales, también se necesitan un timbre y un LCD.

Los estudiantes pueden hacer las prácticas usando una Nexys4 DDR [7] o una DE2-115 [8], placas FPGA basadas en tecnologías de Xilinx y Altera respectivamente. Ambas compañías ofrecen además alternativas de bajo coste como la Basys3 o la DE0. Estas placas integran FPGAs más pequeñas pero más que suficientes para albergar el sistema MIPSfpga completo. El sistema base de MIPSfpga utiliza el 15%, 42%, 13% y 98% de la lógica de las placas Nexys4 DDR, Basys3, DE2-115 y DE0, respectivamente. El resto de la lógica puede usarse para hardware personalizado y/o ampliaciones del sistema MIPSfpga. Por medio de una placa Bus Blaster, disponible a través de SEED Technology, los estudiantes pueden descargar y depurar programas directamente en el sistema MIPSfpga.

TABLA I
REQUISITOS HARDWARE

Nombre	Opciones de placa FPGA		
	FPGA	Precio	Página web
Nexys4 DDR	Xilinx Artix-7	\$159 (académico)	digilentinc.com
		\$320 (no académico)	
Basys3	Xilinx Artix-7	\$79 (académico)	digilentinc.com
		\$149 (no académico)	
DE2- 115	Altera Cyclone IV	\$309 (académico)	de2- 115.terasic.com
		\$595 (no académico)	
DE0	Altera Cyclone III	\$81 (académico) \$119 (no académico)	de0.terasic.com
Nombre	Placa de programación para MIPSfpga		
Bus Blaster	Precio	Página web	
	\$43.95	www.seeedstudio.com/depot/B us- Blaster-V3c-for-MIPS-Kit-p- 2258.html	
Nombre	Hardware Extra		
Timbre (Prac. 7)	Precio	Página web	
	\$2	https://www.digikey.com/prod uct-search/en/audio- products/buzzers/720967?k=10 2-1153-ND	
LCD (Prac. 8)	\$16	https://www.digikey.com/prod uct-search/en?keywords=1481- 1063-ND	

Además de este hardware, necesario para todas las prácticas, los estudiantes necesitan componentes adicionales para llevar a cabo las prácticas 7 y 8. La Tabla I propone dos recomendaciones asequibles.

B. Requisitos software

El software necesario para realizar las prácticas consiste en: (1) una herramienta de CAD para hacer simulaciones y cargar el sistema MIPSfpga en una FPGA, (2) herramientas de programación y (3) software para cargar y depurar programas en el procesador, como muestra la Tabla II. Vivado o Quartus II son las herramientas de Xilinx y Altera, respectivamente, con las que cargar un diseño hardware en una FPGA. Imagination ofrece Codescape, un kit de desarrollo software (SDK, Software Development Kit) para compilar y depurar ensamblador MIPS y programas C. OpenOCD es un depurador on-chip de código abierto que se usa junto a Bus Blaster para cargar y depurar programas en el sistema MIPSfpga.

TABLA II
REQUISITOS SOFTWARE

Nombre	Herramientas de CAD	
	Descripción	Página web
Xilinx Vivado WebPack	Software para simulación y programación de FPGAs Xilinx	www.xilinx.com/su pport/download.ht ml
Altera Quartus II Web Edition	Software para simulación y programación de FPGAs Altera	http://dl.altera.com/ ?edition=web
Codescape MIPS SDK Essentials	Kit de desarrollo de software de Imagination Technologies para programar y depurar MIPSfpga. Incluye gcc y gdb	Instalador disponible como parte del Kit de Iniciación de MIPSfpga
OpenOCD	Depurador on-chip de código abierto que permite cargar y depurar programas en un procesador MIPS	Instalador disponible como parte del Kit de Iniciación de MIPSfpga

C. Descripción de las prácticas

Hemos desarrollado nueve prácticas ahora disponibles desde la web de Imagination Technologies como el pack *MIPSfpga Fundamentals* [4], donde se guía a los estudiantes a través de la configuración del hardware, la programación y depuración sobre un procesador software MIPS, y cómo extender dicho procesador para construir un SoC basado en MIPSfpga. La novena práctica describe cómo portar MIPSfpga a otras placas FPGA. La Tabla III ofrece un breve resumen de dichas prácticas.

TABLA III
PRÁCTICAS INCLUIDAS EN MIPSPFGA FUNDAMENTALS

Número	Descripción
1	Proyecto en Vivado o Quartus II: Creación de un proyecto para el sistema MIPSfpga usando Vivado (para la placa Nexys4 DDR) o Quartus II (para la placa DE2-115)
2	Programación en C: Aprendizaje de cómo escribir, compilar, depurar y ejecutar programas en C en el sistema MIPSfpga.
3	Programación en Ensamblador MIPS: Aprendizaje de cómo escribir, compilar, depurar y ejecutar programas en ensamblador MIPS en el sistema MIPSfpga.
4	Más Prácticas de Programación (opcional): Creación de dos programas adicionales en C que implementan un hipnotizador de bolsillo y un juego de memoria.
5	Entrada/Salida Mapeada a Memoria - Display de 7 Segmentos: Expansión del sistema MIPSfpga para incorporar acceso a los ocho displays de 7 segmentos usando entrada/salida mapeada a memoria.
6	Entrada/Salida Mapeada a Memoria - Contador: Incorporación de un contador de milisegundos mapeado a memoria al sistema MIPSfpga.
7	Entrada/Salida Mapeada a Memoria - Timbre: Incorporación de un timbre mapeado a memoria al sistema MIPSfpga y escritura de un programa que toque una canción usando el timbre.
8	Entrada/Salida Mapeada a Memoria - SPI y LCD: Incorporación de puerto de interfaz de periféricos serie (SPI, Serial Peripheral Interface) al sistema MIPSfpga con el que programar un LCD.
9	Adaptación a otras FPGAs: Aprendizaje de cómo portar el sistema MIPSfpga a otras placas FPGA como por ejemplo la Basys3 de Digilent o la DE0 de Terasic.

La primera práctica describe cómo crear un proyecto con MIPSfpga para una FPGA usando el software Vivado de Xilinx o Quartus II de Altera como herramienta principal. La disponibilidad del código fuente de MIPSfpga en Verilog ofrece a los estudiantes la oportunidad de ver el código no ofuscado, así como de modificar, ampliar y revisar el interior de un soft-core MIPS.

En las prácticas 2 y 3 se explica cómo usar el SDK Codescape, que incluye gcc y gdb para MIPS, y la placa Bus Blaster para compilar, descargar, ejecutar y depurar programas en C y ensamblador MIPS que correrán en la FPGA. La práctica 4 es optativa y propone prácticas adicionales de programación.

La práctica 5 es la primera de las prácticas que muestra cómo utilizar la entrada/salida mapeada a memoria para

incorporar periféricos a MIPSfpga y diseñar un SoC. En él se muestra cómo mapear a memoria las señales necesarias para utilizar los displays 7 segmentos de las placas FPGA, así como modificar y extender el sistema MIPSfpga para dar soporte a las direcciones de memoria mapeadas y escribir programas en C o ensamblador que utilizan los displays 7 segmentos.

Las prácticas 6 a 8 guían a los estudiantes para que añadan otros periféricos extra: un contador mapeado a memoria para temporización, un timbre para tocar música, y una interfaz SPI para utilizar un display LCD. Estas prácticas, junto con la 5, hacen hincapié en el diseño SoC y el codiseño hardware-software. Los profesores pueden elegir incluirlos todos o solo algunos de ellos en sus cursos.

La práctica 9 se ofrece para aquellos profesores/estudiantes que quieran emplear una FPGA distinta a la Nexys 4 DDR o la DE2-115 para trabajar con MIPSfpga, ya sea porque tienen otra disponible o porque se decantan por una opción más económica. La práctica muestra cómo ajustar el sistema MIPSfpga para ser cargado en la placa Basys3 de Digilent (que incluye una FPGA Artix-7 de Xilinx) y una DE0 de Terasic (que contiene una FPGA Cyclone III de Altera) a modo de ejemplo. El proceso general mostrado en esta práctica puede ser extrapolado a otras placas FPGA.

Todas las prácticas pueden usar la placa FPGA que se quiera, siempre y cuando el sistema MIPSfpga entero quepa en ella. Por ejemplo, un profesor puede elegir emplear las placas Basys3 o DE0, más asequibles económicamente. En ese caso, el profesor puede utilizar la práctica 9 para hacer ligeras modificaciones a la práctica 1, de modo que éste hable de la FPGA adecuada. Dependiendo de las características de la FPGA, el resto de prácticas tal vez necesiten algún ligero ajuste para hablar de la placa adecuada. Por ejemplo, si se usan la Basys3 o la DE0, placas más pequeñas, las prácticas 2 a 4 y 6 no necesitan modificación alguna. La práctica cinco requiere cambios menores dado que estas placas tienen una cantidad menor de displays 7 segmentos (4 en vez de 8). Las prácticas 7 y 8 necesitan cambios solo en cuanto a qué pines de entrada/salida de propósito general (GPIO) se utilizan para conectar la FPGA con el timbre o el display LCD.

V. PROYECTO FINAL

Tras completar las prácticas, los estudiantes llevan a cabo un proyecto final no guiado de cinco semanas. Los temas para estos proyectos finales pueden incluir, entre otros, los indicados en la Tabla IV. Los estudiantes pueden ampliar su conocimiento de diseño de SoC y codiseño hardware-software añadiendo interfaces adicionales, como I²C o Emisor/Receptor Asíncrona Universal (UART, Universal Asynchronous Receiver-Transmitter) para comunicarse con otros periféricos adicionales. Por ejemplo, la interfaz UART puede usarse para leer y escribir texto en una terminal. Los profesores pueden pedir a los estudiantes que incluyan varias interfaces nuevas en sus proyectos finales.

TABLA IV
TEMAS PARA EL PROYECTO FINAL DE MIPSPFGA.

Título	Descripción
I ² C	Incorporar una interfaz I ² C al sistema MIPSPfga.
UART	Incorporar una interfaz UART al sistema MIPSPfga
DDR	Añadir una interfaz con DDR a MIPSPfga para acceder a memoria off-chip.

Otras propuestas más avanzadas, posiblemente demasiado ambiciosas para un proyecto no guiado de cinco semanas, podrían profundizar en distintas partes de la microarquitectura. Por ejemplo, los estudiantes pueden añadir un predictor de saltos o una unidad de cálculo en punto flotante, añadir instrucciones definidas por el usuario (a través del módulo de UDI), probar distintas políticas de reemplazo cache, y otras modificaciones similares.

Una última propuesta, tan compleja como atractiva, y que podría servir como temática para un proyecto de máster, es la de construir un sistema multi-core basado en MIPSPfga. Para esto haría falta llevar a cabo una serie de mejoras importantes al sistema, entre las que se incluye ampliar el protocolo AHB-Lite, dar soporte para coherencia en las caches e implementar un gestor de coherencia.

VI. EXPERIENCIAS PRÁCTICAS

Aunque MIPSPfga se lanzó de forma pública hace poco tiempo (en junio de 2015), son varias las universidades que ya han diseñado laboratorios y proyectos de investigación usando el procesador y las prácticas que presentamos aquí. Por ejemplo, los estudiantes de un curso de grado de Arquitectura de Computadores de uno de los autores de este artículo llevaron a cabo los proyectos indicados en la Tabla IV. Igualmente, otros dos de los autores del artículo están actualmente dirigiendo un proyecto de fin de grado que persigue llevar a cabo una comparativa de políticas de reemplazo cache. Creemos firmemente que este tipo de actividades prácticas basadas en MIPSPfga permiten que los estudiantes asimilen los conceptos teóricos de asignaturas del área de Arquitectura y Tecnología de Computadores más ampliamente y con mayor profundidad.

De forma adicional, Imagination Technologies está ofreciendo multitud de workshops prácticos de un día de duración alrededor del mundo, los cuales han contado ya con más de 400 participantes. En estos eventos se explican los conceptos básicos de MIPSPfga y los participantes completan algunas de las prácticas incluidas en *MIPSPfga Fundamentals* (en particular, las prácticas 1, 2, 3 y 5). En general, los académicos han dado buena acogida a la disponibilidad de MIPSPfga para uso educativo y en el ámbito de la investigación, y han ofrecido un feedback muy valioso para futuras mejoras y ampliaciones del material ofrecido.

VII. TRABAJO RELACIONADO

Hoy en día, existe una gran disponibilidad de procesadores soft-core. Por ejemplo, las principales

compañías de desarrollo y comercialización de FPGAs, como Altera o Xilinx, ofrecen sus propios soft-cores, Nios/NiosII [9] y MicroBlaze [10] respectivamente, configurados específicamente para sus propias FPGAs. Estas alternativas presentan diversas desventajas: no son de código abierto, lo cual limita su uso significativamente; no están basados en soft-cores industriales/comerciales ni en un ISA comercial; y carecen de material docente extenso y de calidad.

Por su parte, ARM también proporciona una opción de código no abierto, el Cortex M0 Design Start [11], un soft-core muy básico (compuesto únicamente de 8K puertas) y de bajo rendimiento. Se trata de un soft-core con el código ofuscado, con un soporte para depuración muy limitado (no incluye EJTAG). Además, no proporciona la posibilidad a la comunidad académica de integrarlo en silicio y, al igual que los dos anteriores, carece de buen material docente.

Existe también disponibilidad de soft-cores de código abierto. Dos opciones muy conocidas, ambas basadas en el ISA SPARC RISC, son la familia OpenSPARC [12] y la familia LEON [13], desarrollados actualmente por Oracle (originalmente por Sun Microsystems) y por Aeroflex Gaisler (originalmente por la European Space Agency), respectivamente. Si bien son alternativas interesantes, al igual que las descritas anteriormente no incluyen buen material docente, y están basadas en un ISA menos extendido en el mundo académico que el ISA de MIPS o el de ARM. Por último, otras dos alternativas que debemos mencionar son RISC-V [14], desarrollada originalmente por la Universidad de California, Berkeley, y openRISC, desarrollada por opencores.org [15]. Estos soft-cores están basados en ISAs no comerciales y, al igual que los anteriores, proporcionan escaso material docente.

MIPSPfga, por su parte, soluciona todas las limitaciones anteriores. Incluye un procesador soft-core industrial, de código no ofuscado, utilizado en diversos dispositivos comerciales, como el conocido PIC32MZ de Microchip o el reciente módulo Artik1 de Samsung. Dicho soft-core utiliza el release 3 del ISA de MIPS, ampliamente empleado en el mundo académico y con disponibilidad de multitud de documentación y soporte docente. MIPSPfga también proporciona gran cantidad de documentación, entre la que se incluye mucho material docente teórico y práctico. Toda esta documentación está disponible en 5 idiomas (castellano entre ellos). Además, MIPSPfga incluye soporte para diversas FPGAs, tanto de Xilinx como de Altera, y es fácilmente extensible a otras FPGAs siguiendo los pasos explicados en el guión de la práctica 9 del paquete *MIPSPfga Fundamentals*.

VIII. EXTENSIÓN DE MIPSPFGA

A. MIPSPfga-SoC

El tercer paquete de MIPSPfga, denominado *MIPSPfga SoC* [2], hecho público recientemente también desde la empresa Imagination Technologies, incluye un SoC centrado en torno al procesador soft-core incluido con la infraestructura MIPSPfga, y ampliado con diversos dispositivos periféricos, como una memoria DDR, un UART16550, un I²C, un interfaz Ethernet, o un

controlador de interrupciones. En este paquete, al igual que en los dos anteriores, se proporciona gran cantidad de material docente, en el que se muestra detalladamente el proceso para sintetizar y cargar el SoC en la FPGA, así como el proceso para portar y ejecutar el kernel de Linux en dicho sistema.

B. MIPSfpga v2.0

A comienzos del año 2017 está prevista la publicación de una extensión de los paquetes *MIPSPfpga Getting Started* y *MIPSPfpga Fundamentals*, en la que se incluyan, entre otras cosas: varias prácticas orientadas a estudiar diversos aspectos de la microarquitectura del procesador soft-core incluido en MIPSfpga, como la cache, la unidad de ejecución, etc.; diversas prácticas en las que se ilustre la Entrada/Salida controlada por interrupciones; alguna práctica dedicada a ilustrar cómo incluir nuevas instrucciones definidas por el usuario por medio del interfaz CoExtend UDI de MIPS; o una guía explicativa sobre cómo utilizar la infraestructura sobre un sistema operativo Linux.

IX. CONCLUSIONES

La infraestructura MIPSfpga ofrece una plataforma completamente abierta y gratuita para enseñar/aprender multitud de conceptos de arquitectura de computadores, diseño digital, diseño de SoCs y co-diseño Hardware-Software. Los paquetes *MIPSPfpga Getting Started* y *MIPSPfpga Fundamentals* [2], puestos a disposición de la comunidad académica por la empresa Imagination Technologies, incluyen una introducción al sistema MIPSfpga y un conjunto de prácticas con las que se tratan de ilustrar todos los conceptos a los que se ha hecho referencia más arriba. Los estudiantes aprenderán además cómo usar FPGAs y diversos periféricos, y cómo trabajar con un procesador comercial. Por su parte, el paquete *MIPSPfpga SoC* [2], recientemente hecho público, ilustra cómo ejecutar el Sistema Operativo Linux en el core de MIPSfpga y cómo integrar diversos Xilinx-IP en el sistema.

X. AGRADECIMIENTOS

Los autores desean agradecer las contribuciones realizadas por el Imagination University Program, por la Universidad de Nevada, Las Vegas (USA), por el Imperial College de Londres (UK), y por las siguientes personas: Yuri Panchul (IMG USA), Munir Hasan (IMG UK), Prashant Deokar (IMG India), Mahesh Firke (IMG India) Parimal Patel (Xilinx), Bruce Ableidinger (IMG USA), Kent Brinkley (IMG USA), Rick Leatherman (IMG USA), Chuck Swartley (IMG USA), Sean Raby (IMG UK), Michio Abe (IMG Japan), Bingli Wang (IMG China), Sachin Sundar (IMG USA), Alex Wong (Diligent Inc.), Matthew Fortune (IMG UK), Jeffrey Deans (IMG UK), Zubair Kakakhel (IMG UK), Laurence Keung (IMG UK), Roy Kravitz (Portland State University), Dennis Pinto (Universidad Complutense de Madrid), Tejaswini Angel (Portland State University), Christian White, Gibson Fahnestock, Jason Wong, Cathal McCabe (Xilinx), y Larissa Swanland (Diligent).

REFERENCIAS

- [1] S. Harris, R. Owen, E. Sedano, D. Chaver, "MIPSPfpga: Hands-On Learning on a Commercial Soft-Core", Proceedings of the IEEE European Workshop on Microelectronics Education, EWME (Southampton, UK), 11-13 de Mayo de 2016.
- [2] Zubair Lutfullah Kakakhel et al. "From Verilog to Linux: Coherency in the Classroom by Empowering Academics with an Industrial MIPS Processor and a System on Chip Design which Runs Linux", Proceedings of the Embedded World Conference (Nuremberg, Alemania), 23-25 de Febrero de 2016.
- [3] D. Harris and S. Harris, "Digital Design and Computer Architecture", 2nd Edition, Elsevier Science & Technology, Julio de 2012.
- [4] Imagination Technologies Ltd, "MIPSPfpga". <https://community.imgtec.com/university/resources>.
- [5] Imagination Technologies Ltd, "MIPS32® microAptiv™ UP Processor Core Family Datasheet", 31 de Julio de 2013.
- [6] ARM, "AMBA® 3 AHB-Lite Protocol Specification", 2006.
- [7] Diligent Inc., "Nexys4 DDR™ FPGA Board Reference Manual", 11 de Septiembre de 2014.
- [8] Terasic Inc, "DE2-115 User Manual", 2013.
- [9] Altera - NIOS-II Processor. <https://www.altera.com/products/processors/overview.html>.
- [10] Xilinx - MicroBlaze Soft Processor Core. <http://www.xilinx.com/products/design-tools/microblaze.html>.
- [11] ARM - Cortex M0 Design Start. <http://www.arm.com/products/designstart/index.php>.
- [12] Oracle - Open SPARC. <http://www.oracle.com/technetwork/systems/opensparc/index.html>.
- [13] Aeroflex Gaisler - LEON series Softcores. <http://www.gaisler.com/>.
- [14] Waterman, A., Lee, Y., Patterson, D.A., Asanovic, K., "The RISC-V Instruction Set Manual, Volume I: User-Level ISA", Version 2.0, 2014.
- [15] OpenCores - OpenRISC. http://opencores.org/or1k/Main_Page.

Parte II.- I Jornadas de Computación Empotrada y Regonfigurable

Diseño de Sistemas

Implementación de técnicas de estimación de la presión arterial a partir de ECG y PPG

José M. Bote ¹, Joaquín Recas ¹, Román Hermida ¹ y Marian Diaz-Vicente

Resumen—Este trabajo realiza una comparativa experimental de diferentes métodos recogidos en la literatura de estimación de presión arterial a partir de técnicas no invasivas basadas en diferencias temporales entre la señal de electrocardiograma (ECG) y la fotopleletismografía (PPG).

Para ello se ha desarrollado una aplicación móvil Android capaz de monitorizar estas señales en tiempo real, de forma continua y no invasiva, obteniendo constantes vitales como el pulso cardiaco, la oximetría y la estimación de la presión arterial. Para obtener valores fiables de presión es necesaria una calibración previa del sistema para cada sujeto frente a un esfigmomanómetro, situando a la persona en un rango amplio de presiones mediante un ejercicio aeróbico sostenido en el tiempo, iniciado y finalizado en el reposo. Esta calibración permite establecer el mejor método de estimación de la presión de un grupo representativo de 10 métodos diferentes recogidos por la literatura. Una vez calibrado el sistema, los valores de presión arterial son inferidos a partir de la señal de ECG y PPG, sin la necesidad del esfigmomanómetro.

Los resultados experimentales obtenidos con 3 sujetos muestran coeficientes de determinación R^2 cercanos a 0.8 para algunos de los modelos estudiados, indicando que existe una correlación entre la presión arterial sistólica y diastólica medidos por un esfigmomanómetro y el tiempo de llegada de la onda de pulso sanguíneo (PAT) o variables temporales de la PPG.

Palabras clave— Presión sanguínea, electrocardiograma (ECG), fotopleletismografía (PPG), tiempo real, Android

I. INTRODUCCIÓN

UNO de los retos más importantes de la medicina preventiva es la lucha contra las enfermedades cardiovasculares, principal causa de mortalidad en los países occidentales [1]. Reducir los riesgos cardiovasculares pasa por controlar diferentes parámetros fisiológicos entre los que se encuentran las cinco constantes vitales más importantes: el pulso cardiaco, la presión sanguínea, la respiración, la temperatura y la pulsioximetría [2]. Monitorizar estos signos vitales de forma continua permite registrar eventos puntuales que no se manifiestan asiduamente y puede prevenir afecciones tanto en personas sanas como con patologías. Entre dichas constantes vitales, conocer la tensión o presión arterial permite controlar enfermedades como la hipertensión, que puede conducir a infartos, o la hipotensión, que puede privar al cuerpo de oxígeno llegando a dañar órganos como el corazón o el cerebro.

Actualmente, la monitorización de la presión arterial se lleva a cabo a partir de métodos basados principalmente en auscultación y oscilometría [3]. Estas

técnicas miden de forma indirecta la tensión a partir de un esfigmomanómetro o tensiómetro, compuesto de un brazalete inflable, un manómetro y un estetoscopio. El sistema oprime la arteria braquial auscultando el intervalo de los sonidos de Korotkoff [4]. Aunque son técnicas no invasivas, pueden llegar a ser molestas para pacientes que requieran una observación más estricta debido a su naturaleza mecánica, además de obligar a respetar un periodo refractario de varios minutos para la recuperación mecánica de la arteria en el caso de una nueva medición [5].

Recientemente, se han desarrollado métodos no invasivos que permiten estimar la presión arterial a partir de la relación entre dos parámetros fisiológicos no mecánicos, que permiten una monitorización de la tensión de forma continua. El primer parámetro es el electrocardiograma (ECG), que representa la actividad eléctrica del corazón [6, p. 210-217] y puede medirse de forma sencilla mediante unos electrodos situados en el tronco superior del paciente. El segundo de los parámetros, la fotopleletismografía (PPG), representa los cambios relativos del flujo sanguíneo de los capilares más externos. Se mide mediante un pulsioxímetro, un sensor óptico que trabaja en las bandas del rojo e infrarrojo y que además de medir estas variaciones del flujo sanguíneo proporciona el nivel de saturación de oxígeno en sangre.

Este trabajo se organiza tal y como se describe a continuación. En la Sección II se recogen las diferentes técnicas de medición de la presión arterial sanguínea. En ella se describe detalladamente el principio físico que subyace en la técnica no invasiva basada en la velocidad de la onda de pulso o *Pulse Wave Velocity (PWV)* y los diferentes métodos de la literatura que lo emplean para estimar la presión. La Sección III describe la metodología empleada en los experimentos basados en 10 de los modelos más representativos, obtenidos a partir de los estudios analizados. Además se presenta una aplicación desarrollada en Android para la monitorización en tiempo real de ECG y PPG y la estimación de la presión arterial. Para finalizar, la Sección IV presenta los resultados experimentales de la evaluación de estos modelos y la Sección VI recoge las conclusiones del trabajo realizado.

II. MEDIDA DE LA PRESIÓN SANGÜNEA

La tensión o presión sanguínea es la fuerza que ejerce el flujo sanguíneo sobre las paredes de los vasos arteriales al circular por ellos. Existen dos componentes en la presión arterial: la presión sistólica (SBP) y la presión diastólica (DBP). La presión sistólica se asocia al valor máximo que toma la

¹J.M. Bote, J. Recas y R. Hermida, Dpto. de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, e-mail: jmbrosado@ucm.es, recas@ucm.es, rhermida@ucm.es.

presión y toma su nombre de la sístole, cuando el corazón se contrae y la sangre es eyectada sobre la pared arteriales. Los valores que se consideran normales para una persona sana se encuentran por debajo de 120 mmHg [7]. Por otra parte, la presión diastólica se asocia con el valor más bajo, en la diástole, cuando el corazón se relaja tras la eyección de sangre, mostrando la distensibilidad de las paredes de las arterias. Los valores normales de la presión diastólica se sitúan por debajo de 80 mmHg [7].

Los métodos más empleados en la medida de la presión pueden clasificarse en dos grandes grupos: métodos invasivos y no invasivos. Los métodos invasivos emplean una cánula en la arteria para obtener medidas directas y precisas de la presión; sin embargo, esta práctica es aplicable únicamente en entornos controlados como hospitales bajo supervisión médica, debido al riesgo que puede suponer para el paciente. Solo se recurre a este tipo de métodos en casos de cuidados intensivos, anestesiología o investigación médica. Por otra parte, los métodos no invasivos son los empleados de forma más habitual, proporcionan medidas indirectas de la presión, apenas son molestos y no suponen ningún riesgo para el paciente. Los métodos no invasivos utilizan un esfigmomanómetro para la medición indirecta de la presión basándose en diferentes técnicas como la auscultación, la oscilometría y la tonometría.

El método de la auscultación, desarrollado por Korotkoff en 1905, emplea un estetoscopio para escuchar los sonidos de Korotkoff causados por el paso de la sangre a través de la arteria al recuperarse de la constricción producida por un manguito que se infla y se desinfla. El primer sonido determina el valor de la presión sistólica y el quinto, la diastólica [8]. La oscilometría, estudiada por Marey en 1876 y desarrollada por Erlanger en 1904, es una técnica análoga a la auscultación en la que un transductor recoge las amplitudes de oscilación de las arterias mientras el brazalete se va desinflando. Es el método usado actualmente por los dispositivos automáticos o semiautomáticos [3]. La tonometría, desarrollado por Pressman y Newgard en 1963, estima la presión sanguínea a partir de alteraciones de la onda de pulso en la superficie de la arteria. Los tejidos superficiales entre la arteria y el transductor de presión son los que transmiten la onda de pulso, obteniendo una señal muy precisa, ya que no se oprime la arteria completamente [9], [10].

A. Principio físico de la estimación de la presión: velocidad de la onda de pulso (PWV)

En la actualidad, existen estudios que relacionan la presión sanguínea con la velocidad de la onda de pulso (*Pulse Wave Velocity (PWV)*). El origen de estas investigaciones [11] se remonta al matemático suizo Leonhard Euler, quien propuso las ecuaciones que rigen la conservación de la masa y cantidad de movimiento en un tubo unidimensional y distensible, sin lograr resolverlas con éxito como sí lo había hecho con los tubos rígidos. Posteriormente, el científico

inglés Thomas Young derivó una expresión para la velocidad de una onda de pulso en una arteria a partir de las leyes de Newton para la velocidad del sonido en un gas compresible; y el alemán Weber estudió la velocidad de las ondas en tubos elásticos basándose en las leyes de conservación de Euler.

No fue hasta 1878, cuando el físico holandés Moens publicó un trabajo experimental sobre la velocidad de la onda en tubos elásticos y el matemático holandés Korteweg, un estudio teórico análogo. La conclusión de sus investigaciones determinaron que la velocidad de la onda de pulso dependía de la elasticidad de la pared del tubo y de la compresibilidad del fluido. En particular, para un fluido incompresible como la sangre y un tubo elástico con una pared delgada como la arteria, Moens y Korteweg derivaron la ecuación (1) donde c es la velocidad de la pulso, E el módulo de Young, h el grosor de la pared arterial, r el radio de la arteria y ρ la densidad de la sangre [12]. Existe una relación exponencial entre el módulo de Young E y la presión P : $E = E_0 e^{\gamma P}$ donde E_0 representa el módulo de Young a presión nula y γ es un coeficiente que varía entre 0.016 y 0.018 mmHg⁻¹ en función de cada sujeto [13], [14].

$$c = PWV = \sqrt{\frac{Eh}{2r\rho}} \quad (1)$$

Finalmente, la velocidad de la onda de pulso es una medida que puede ser hallada de forma indirecta mediante una distancia conocida L y el tiempo que tarda en recorrer esa onda de pulso dicha distancia. Por tanto, a partir de medidas directas de la longitud de la arteria y el tiempo que el pulso sanguíneo tarda en recorrerla, pueden inferirse valores para la presión sanguínea. Sin embargo, ya que no es posible medir la longitud de la arteria de forma directa, el objetivo es fijar una distancia y estimar los valores de presión sanguínea a partir de variaciones en el tiempo de tránsito del pulso.

De forma general, para estimar la presión arterial latido a latido se emplea la diferencia temporal entre un punto característico en la señal de ECG recogido por un sensor en el pecho del paciente y otro punto en la señal de PPG, medido habitualmente en el dedo de la mano, el lóbulo de la oreja [15], el dedo del pie [16]. etc. El punto de referencia escogido en el ECG es el pico de la onda R ya que esta onda es la más pronunciada y, por tanto, la referencia más fácil de detectar. En el caso de la PPG, no existe un convenio claro sobre qué punto de referencia elegir, como se verá en secciones posteriores.

En cualquier caso, es conveniente aclarar que la diferencia temporal entre el pico de la onda R de ECG y un punto de la PPG es definido como el tiempo de llegada del pulso (PAT) y no es exactamente el tiempo de tránsito del pulso (PTT). Esto es debido a que una vez que ha ocurrido la despolarización del ventrículo (complejo QRS del ECG), la sangre no es eyectada hasta pasado un tiempo conocido como periodo pre-eyectivo (PEP). Este periodo pre-eyectivo

es el tiempo que tarda el miocardio en elevar suficientemente la presión para abrir la válvula aórtica y expulsar la sangre del ventrículo [17]. Por tanto, se puede afirmar que el tiempo de llegada del pulso (PAT) que se suele utilizar en este método tiene dos contribuciones: el periodo pre-eyectivo (PEP) y el tiempo de tránsito de pulso (PTT), cumpliéndose la relación (2). La Fig. 1. muestra gráficamente las definiciones anteriores.

$$PAT = PEP + PTT \quad (2)$$

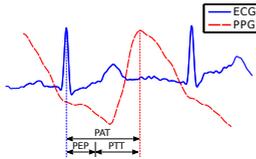


Fig. 1. Relación entre PAT y PTT.

B. Métodos basados en ECG y PPG

Los métodos basados en fotopleletismografía (PPG) y electrocardiografía (ECG) para la estimación de la tensión arterial tienen como principio físico de partida la ecuación de Moens-Korteweg (1) vista anteriormente.

Uno de los primeros estudios fue realizado por Chen *et al.* [13] donde afirmaban que no es posible determinar valores absolutos de presión sanguínea sino variaciones de ella a partir de la expresión $\Delta P = -2\Delta T/(\gamma T)$, obtenida suponiendo que las únicas variaciones de la presión provienen de variaciones pequeñas en el módulo de Young, donde γ es un coeficiente a estimar para cada persona y T es el tiempo correspondiente al PAT. Consideraron que la presión tiene dos componentes, una continua que varía muy lentamente y una componente alterna que se corresponde con la expresión anterior. Mueshlsteff *et al.* [14], partiendo de la misma ecuación de Moens-Korteweg y linealizando la expresión del módulo de Young como $E \simeq E_0(1 + \alpha P)$, obtuvieron una relación inversa cuadrática entre la presión y el tiempo de tránsito de pulso, $P \propto 1/PTT^2$, donde α y E_0 son constantes a ajustar para cada sujeto.

Por otra parte, otras investigaciones se dirigen en una dirección opuesta. Gosling *et al.* [12] afirman que la ecuación de Moens-Korteweg no es válida para la estimación de la presión ya que parte de supuestos que no son válidos como asumir que las paredes arteriales son isotropas y experimentan cambios isovolumétricos con la presión sanguínea. Según Gosling *et al.* la arteria no puede modelarse a partir del módulo de Young.

En cuanto al estudio del periodo pre-eyectivo, Payne *et al.* [18] realizaron estudios determinando que este tiempo contribuye de manera importante en el PAT y que además el PEP es variable, por lo

que el PAT no puede ser considerado como un estimador válido de la presión. Mueshlsteff *et al.* [14] concluyeron que las variaciones en el PAT provienen tanto del PEP como del PTT y, aunque estas contribuciones no tienen siempre el mismo peso, las variaciones en el PEP son siempre superiores a las del PTT, por lo que el PEP tiene una gran influencia en el PAT. En sus investigaciones encontraron una relación lineal entre el PAT y la presión sistólica para personas que realizaban actividades físicas de forma habitual. Sin embargo, dicha relación no era lineal para personas sedentarias.

Por otro lado, otros estudios miden el tiempo de tránsito de pulso (PTT) utilizando únicamente el mismo punto de referencia en dos señales de PPG medidas en dos puntos diferentes del cuerpo, como son el dedo de la mano y el dedo del pie [19]. Foo *et al.* [16], quienes también realizaron estudios sobre el PEP fijando su valor como constante (160 ms para un adulto), compararon esta técnica con valores medidos del PAT a los que se le restaba el PEP fijo de 160 ms. Como conclusión establecieron que, aún existiendo cierta correlación con la presión sanguínea, ninguno de los dos métodos anteriores son plenamente fiables debido a la complejidad del sistema circulatorio humano. Las presiones estimadas a partir de medidas recogidas en los dedos de las manos mostraban presiones parecidas a la de la aorta, ya que no está presente el efecto de la gravedad, mientras que las estimaciones de las presiones a partir de las medidas de los dedos de los pies dependían de la posición corporal en la que se encontraban los pacientes. En la misma línea, Smith *et al.* [20] establecieron que no pueden extrapolarse valores absolutos de PTT para medir la presión sanguínea. Sin embargo, pueden ser usados para predecir cambios en la presión sanguínea en un periodo corto de tiempo.

Otros trabajos se dirigen hacia la estimación de la presión a partir de varias variables. Cattivelli *et al.* [17] propusieron que la presión sanguínea se relaciona de manera lineal con PAT y con la frecuencia cardíaca. El PAT puede ser medido desde la onda R de ECG hasta el pie de la señal de PPG, el punto medio o el máximo. En sus estudios determinaron que son necesarios periodos previos de calibración de al menos 1 hora y 20 minutos para obtener resultados correctos. Chua *et al.* [21] combinaron la amplitud de la señal de PPG con el PAT, medido entre el pico de la onda R de ECG y el punto de máximo gradiente de PPG, en dos latidos consecutivos para la estimación de la tensión, obteniendo una correlación más elevada con la presión sistólica que con la diastólica.

Poon *et al.* [22] y Bin Noordin [8] propusieron unas ecuaciones con varias variables, más complejas que las anteriores, donde tanto la presión sistólica como diastólica se relacionan entre sí y de forma inversa y cuadrática con el PAT más un término logarítmico del PAT para la presión sistólica.

En cuanto al punto de referencia elegido en la PPG no existe un convenio claro, existen estudios que em-

plean, por ejemplo, el 25 % del tramo de subida [16], otros el 50 % [23] y otros tanto en el 25 % como en el 50 % [20]. Yoon *et al.* [24] estudiaron la correlación entre ambas presiones y el PAT, medido hasta el pie de la PPG y medido hasta el máximo de la primera derivada de PPG. Además estudiaron otras medidas temporales como el tiempo de ascenso en la curva de PPG (tiempo de sistólica), el tiempo de descenso o relajación (tiempo de diastólica) y el ancho temporal del pico a 2/3 de la amplitud de PPG, que ya fue propuesto anteriormente por Awad *et al.* [25] empleando 1/2 en lugar de 2/3. En sus estudios encontraron que la mejor correlación para la presión sistólica se obtenía a partir del PTT medido hasta el máximo de la primera derivada de PPG y para la presión diastólica, a partir del tiempo de diastólica. También Teng *et al.* [26] realizaron estudios sobre los tiempos de sistólica, diastólica y el ancho del pico a 1/2 y 2/3 de la amplitud de PPG.

III. METODOLOGÍA

A continuación se expone la metodología empleada en este trabajo. El primer objetivo es realizar una comparativa experimental de los métodos de estimación de la presión arterial existentes en la literatura. Para ello, estos métodos son evaluados mediante experimentos realizados a 3 personas en los que se registra el ECG, la PPG y la presión arterial dada por un esfigmomanómetro. A partir de la correlación de estas variables se determina el mejor modelo para cada sujeto y la presión arterial puede inferirse de forma continua y no invasiva, sin necesidad del esfigmomanómetro.

Como se ha expuesto en la Subsección II-B, los modelos empleados en la estimación de la presión utilizan diferentes variables y, en algunos de ellos, con dependencias funcionales diferentes. A priori, no existen indicios que determinen cuáles de los modelos son los más acertados ni una comparativa entre ellos. La Tabla I recoge 10 de los modelos propuestos como posibles estimadores de la presión y la Fig. 2, las variables utilizadas en dichos modelos.

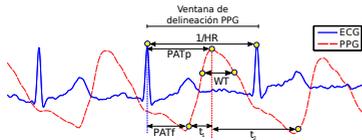


Fig. 2. Definiciones de tiempos para ECG y PPG.

El modelo 1 es propuesto por Cattivelli *et al.* [17], donde se estima la presión a partir del $PATp$, la diferencia entre el pico de la onda R de ECG y el máximo de PPG posterior, junto con la frecuencia cardiaca para cada latido. Debe ser tomado en consideración que éste es el único de los modelos multivariable considerado en este trabajo y que la adición de cualquier variable nueva a un modelo anterior siempre incre-

menta su correlación, por lo que, para correlaciones semejantes siempre es mejor considerar como válido aquél que incluya un menor número de variables de entrada.

Los modelos 2 y 3 relacionan proporcionalmente la presión sanguínea con el PAT [8]. El modelo 2 emplea el $PATp$, el máximo de PPG, mientras que el 3 utiliza el $PATf$, definido como el pie de PPG. Los modelos 4 y 5, emplean las mismas variables anteriores pero la dependencia funcional en estos casos es cuadrática inversa [14].

Los modelos 6 y 7 utilizan como variable de estimación el tiempo de tránsito de pulso (PTT) (2). En la literatura se han encontrado estudios que estiman este tiempo como variable y otros como constante. En una primera aproximación y ante la imposibilidad de medir directamente este tiempo, se tomará como una constante de 160 ms. [16]. Es conveniente aclarar que si el PEP es constante o su variabilidad es pequeña respecto del PAT, los modelos 2 y 3 presentarían los mismos resultados empleando el PAT o el PTT, ya que la adición o sustracción de una constante a estos modelos lineales no modificaría los parámetros del ajuste, salvo una traslación del modelo.

Los modelos 8, 9 y 10 han sido propuestos por Awad *et al.* [25], Teng *et al.* [26] y Yoon *et al.* [24] basados en medidas temporales de la señal de PPG exclusivamente: t_1 se define como el tiempo de ascenso en la curva de PPG o tiempo de sistólica, t_2 el tiempo de descenso o relajación o tiempo de diastólica y WT el ancho temporal del pico a 2/3 de la amplitud de PPG.

Tabla I
 MODELOS EMPLEADOS EN LA ESTIMACIÓN DE LAS PRESIONES SANGUÍNEAS.

Modelo	{SBP, DBP}
1	$a \cdot PATp + b \cdot HR + c$
2	$a \cdot PATp + b$
3	$a \cdot PATf + b$
4	$\frac{a}{(PATp)^2} + b$
5	$\frac{a}{(PATf)^2} + b$
6*	$\frac{a}{(PATp - PEP)^2} + b$
7*	$\frac{a}{(PATf - PEP)^2} + b$
8	$a \cdot WT + b$
9	$a \cdot t_1 + b$
10	$a \cdot t_2 + b$

*PEP = 160 ms.

a, b y c son constantes a ajustar.

A. Delineación de ECG y PPG

Para la comparación de los modelos recogidos en la Tabla I es necesario calcular todas las variables temporales anteriores, por lo que es esencial delinear previamente una serie de puntos característicos, tanto en el ECG como en la PPG (ver Fig. 2).

La única referencia necesaria en el ECG es el pico de la onda R, el cual constituye el inicio de lo que

se ha definido con anterioridad como PAT. Además, a partir de la diferencia entre los picos de dos ondas R consecutivas se calcula la frecuencia cardiaca (HR), que es una de las variables necesarias incluidas en el modelo 1. Para realizar esta delimitación se ha empleado un delineador de ECG basado en *wavelets* [27], capaz de realizar una delimitación completa del ECG (complejo QRS y ondas P y T) en tiempo real, de la que solo será necesaria la información correspondiente a los picos de las ondas R.

En el caso de la PPG, es necesario delinear cuatro puntos característicos para terminar de calcular cada PAT. Estos puntos son definidos como *pie*, *wt-ini*, *máximo* y *wt-fin* según la Fig. 2 y por proximidad al pico de la onda R, es decir, el *pie* será el punto más cercano al pico de la onda R mientras que *wt-fin* será el más lejano.

El proceso de delimitación de la PPG se realiza por ventanas, que es delimitadas por dos picos de ondas R consecutivos. Una vez que el intervalo de PPG queda definido, el primer punto que se busca es el *máximo*, que es el punto de mayor amplitud de la curva. Hallado este *máximo*, se localiza el *pie* de la curva como el inicio del tramo de máxima pendiente que antecede al *máximo*. Finalmente, *wt-ini* y *wt-fin* se localizan anterior y posteriormente al *máximo* a partir del valor de la amplitud calculada como la diferencia entre la señal en el *máximo* y en el *pie*.

El punto más difícil de determinar en la señal de PPG es el *pie* como ya estudiaron Kazanavicius *et al.* [28] y Posada-Quintero *et al.* [29], proponiendo cuatro métodos diferentes para localizarlo. En este trabajo, el método elegido consiste en localizar el *pie* a partir del máximo de la segunda derivada de PPG, filtrada mediante un filtro paso-bajo Butterworth de orden 10 y frecuencia de corte de 16 Hz.

Una vez delineados los puntos en el ECG y en la PPG, pueden calcularse las variables temporales de los modelos a estudiar, donde *PAT_p* es definido entre el *máximo* y el pico de la onda R y *PAT_f* entre el *pie* y el pico de la onda R. Las variables t_1 , t_2 y *WT* son las definidas anteriormente y para las cuales solo son necesarios los puntos delineados en la PPG.

B. Montaje experimental

Ya que los modelos que estiman la presión arterial anteriormente vistos se basan en regresiones lineales, es necesario adquirir pares de valores tiempo-presiones para un rango amplio de presiones. Cuanto mayor sea el rango de presiones, mayor será el rango de tiempos y, por tanto, mejor será el ajuste lineal realizado. El experimento propuesto trata de situar a la persona en un amplio rango de presiones arteriales que son medidas mediante un esfigmomanómetro, mientras tanto, el ECG y la PPG son registradas de forma continua con un sistema que se explica en la sección siguiente.

El desarrollo del experimento consistirá en una primera parte de reposo, donde se obtendrán medidas de presiones sistólica y diastólica bajas; una segunda parte, donde se le pide al sujeto que realice

algún tipo de ejercicio aeróbico progresivo en intensidad y sostenido en el tiempo, con el objetivo de lograr valores de presiones elevados; y finalmente, una tercera parte, donde se vuelve al reposo lentamente, disminuyendo los valores de presión. Las medidas realizadas por el esfigmomanómetro tienen que ser distanciados varios minutos entre cada toma de medida para dejar que las arterias se recuperen del esfuerzo al que son sometidas debido a la opresión mecánica del manguito [5].

B.1 Adquisición de datos

Como se ha visto anteriormente, en el estudio de los diferentes modelos es necesario recoger previamente información de 3 señales biológicas diferentes: ECG, PPG y presión arterial. En este trabajo se ha utilizado un prototipo experimental desarrollado por *SmartCardia*, una *spin-off* perteneciente al *Embedded Systems Laboratory (ESL)* de la *École Polytechnique Fédérale de Lausanne (EPFL)*. Este sistema empotrado portátil (*wearable*) y de bajo consumo se encarga de recoger de forma síncrona el ECG y la PPG, mientras que la presión sanguínea se obtendrá a partir de un tensiómetro digital *Omron M10-IT* de brazo validado clínicamente.

La obtención del ECG se realiza mediante el convertidor analógico digital ADS1294 [30], con 4 canales y 24 bits de resolución y con un *front-end* de ECG integrado. Este módulo recoge datos de la derivación I (definida por Einthoven como la diferencia de potencial entre brazo derecho y brazo izquierdo) mediante 2 electrodos incrustados en una cinta extensible situada alrededor del tronco superior a la altura del pecho. La frecuencia de muestreo del ECG es 250 Hz.

La PPG es obtenida a partir de un módulo *OEM III* y un sensor óptico *Nonin 8000H* [31] empleado en entornos clínicos con una frecuencia de muestreo de 83.3 Hz. Este módulo también proporciona valores de oximetría o nivel de saturación de oxígeno en sangre.

Este sistema empotrado portátil envía los datos en tiempo real a una aplicación Android que los muestra por pantalla, también en tiempo real, y los almacena para su posterior tratamiento en MATLAB. En la Fig. 3 se muestra el montaje experimental empleado.



Fig. 3. Montaje experimental.

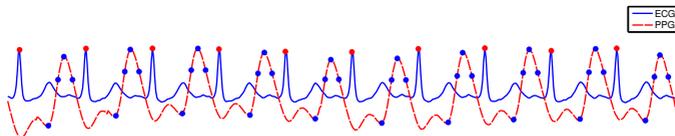


Fig. 4. Delineación de ECG y PPG.

B.2 Calibración mediante MATLAB

Una vez finalizado cada experimento, se realiza la calibración del sistema mediante MATLAB. Debido al posible ruido que puedan contener las señales de ECG y PPG, es necesario un preprocesado, por lo que ambas señales son filtradas con filtros paso-bajo eliminando toda contribución de ruido: eléctrico, muscular, recolocación de electrodos, mala sujeción del sensor de PPG, etc.

A continuación se realiza la delineación del ECG y la de la PPG como se describe en la Subsección III-A y se procede a calcular todas las variables descritas por los modelos de la Tabla I. La Fig. 4 muestra un extracto de las señales de ECG y PPG con su correspondiente delineación. Para minimizar posibles fallos en la delineación de ambas señales solo se aceptan aquellos valores temporales incluidos en el intervalo $(0.25\mu, 1.75\mu)$ donde μ representa la media de cada variable. El único criterio para descartar valores a priori es que sean claramente erróneos, por lo que el intervalo elegido cumple con el cometido de forma correcta.

Una vez que se han obtenidos las variables latido a latido y que estos valores superan un criterio de validez, a cada valor de presiones adquirido por el tensiómetro se le asocia una media de los valores temporales de los 10 latidos anteriores. Promediar para un grupo de latidos permite minimizar el error en la medida cada variable.

Las parejas de valores presiones-tiempos permiten evaluar la bondad de los ajustes lineales de los modelos de la Tabla I, obteniendo el coeficiente de determinación R^2 para cada regresión y sus parámetros a , b y c . Una vez evaluados los 10 modelos, se comparan eligiendo automáticamente el de mayor coeficiente de determinación como mejor estimador tanto para la presión sistólica como para la diastólica, junto con sus parámetros, y a partir de ese instante ya puede ser utilizado en la estimación de la presión arterial para ese sujeto de prueba.

B.3 Aplicación Android

Se ha desarrollado una aplicación original en Android para adquirir, procesar y delinear el ECG y PPG en tiempo real, obteniendo constantes vitales del sujeto tales como la frecuencia cardiaca, la oximetría y la estimación de la presión arterial. Además, esta aplicación permite la visualización de estas señales en tiempo real gracias a las li-

brerías del proyecto *Flot Android Chart* [32] con licencia LGPL. La aplicación desarrollada se usa, tanto para la adquisición de datos que permiten la calibración previa del sistema frente a un esfigmomanómetro como estándar de referencia, como la posterior monitorización continua de la persona. Una captura de la *front-end* se recoge en la Fig. 5.

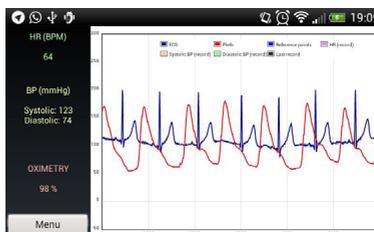


Fig. 5. Aplicación Android para delineación de ECG y PPG en tiempo real.

IV. RESULTADOS

El experimento descrito en la Sección III-B se ha llevado a cabo en 3 sujetos diferentes realizando distintos tipos de ejercicios aeróbicos. El sujeto 1 ha sido sometido a un ejercicio aeróbico sentado sobre una bicicleta, mientras que los sujetos 2 y 3 han realizado un ejercicio de elevación alterna de piernas a ritmo sostenido en la posición decúbito supino. Ambos sujetos parten y finalizan el ejercicio en reposo.

La Fig. 6 muestra la evolución temporal de las presiones durante uno de los experimentos. Puede apreciarse como los valores tanto de presión sistólica como de presión diastólica son mínimas en el reposo y aumentan con el ejercicio, al igual que la frecuencia cardiaca.

A modo de ejemplo, la Fig. 7 refleja la evolución temporal de la variable definida como $PATp$ (diferencia temporal entre el pico de la onda R de ECG y el máximo de la PPG) para el mismo experimento de la Fig. 6. Como puede observarse para esta persona, existe una relación inversa entre las presiones sistólica y diastólica y el $PATp$, ya que en el tramo de ejercicio los valores de presión arterial aumentan mientras que la variable temporal $PATp$ disminuye. De forma general, se ha observado que la relación in-

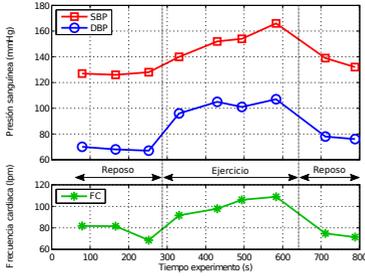


Fig. 6. Evolución temporal de las presiones durante el experimento del sujeto 1.

versa de la presión con el $PATp$ se mantiene para los 3 sujetos. Análogamente, la presión muestra una relación inversa con el $PATf$ y con el tiempo de diastólica t_2 ; sin embargo, no se ha apreciado ninguna relación clara ni con el ancho del pulso de PPG (WT) ni con el tiempo de sistólica t_1 .

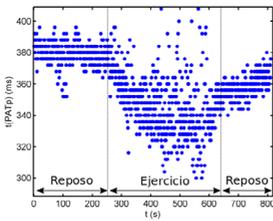


Fig. 7. Evolución temporal del $PATp$ para el experimento del sujeto 1.

La Tabla II recoge los coeficientes de determinación R^2 de los 10 modelos evaluados para cada persona. Según los resultados obtenidos, los modelos que presentan mejor coeficiente de determinación son el 1 y el 10, cuyos valores son cercanos a 0.8. Por otra parte, se observan modelos con escasa o nula correlación para los 3 sujetos, como los modelos 8 y 9, basados en la anchura del pulso de PPG (WT) y el tiempo de sistólica (t_1) respectivamente. En el caso particular del modelo 9, puede observarse cómo este modelo se ajusta bien para la presión sistólica del sujeto 1 y, sin embargo, la correlación es prácticamente nula para el sujeto 3. Esto indica la posibilidad de que no todos los modelos funcionan igual para todas personas.

Como ejemplo, la Fig. 8 y 9 muestran las rectas de regresión de los mejores modelos obtenidos para el sujeto 1 (modelo n°5 para la presión sistólica y modelo n°10 para la presión diastólica).

TABLA II

COEFICIENTES DE DETERMINACIÓN R^2 DE LOS MODELOS EVALUADOS PARA EXPERIMENTOS EN 3 SUJETOS.

Mod.	Sujeto 1		Sujeto 2		Sujeto 3	
	SBP	DBP	SBP	DBP	SBP	DBP
1	0.767	0.817	0.765	0.917	0.756	0.789
2	0.646	0.573	0.717	0.806	0.417	0.393
3	0.785	0.769	0.712	0.830	0.453	0.361
4	0.697	0.596	0.703	0.777	0.442	0.415
5	0.836	0.767	0.726	0.820	0.500	0.383
6	0.734	0.613	0.703	0.777	0.442	0.415
7	0.759	0.496	0.726	0.820	0.500	0.383
8	0.407	0.640	0.184	0.276	0.051	0.061
9	0.733	0.456	0.272	0.261	0.052	0.080
10	0.767	0.830	0.758	0.920	0.742	0.765

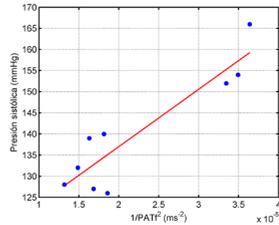


Fig. 8. Mejor ajuste de la presión sistólica para el sujeto 1 (modelo n°5).

V. DISCUSIÓN

De entre los modelos que estiman la presión a partir de ECG y PPG, existen tres líneas de investigación diferentes en función de las variables empleadas en la estimación. Por un lado, existen estudios que determinan que el tiempo de llegada del pulso (PAT) sí está correlacionado con la presión mientras que otros estudios caminan en dirección opuesta, asegurando que no existe ningún tipo de correlación por lo que no puede ser utilizado como variable en los modelos. En una situación intermedia se sitúan las investigaciones que afirman que aun

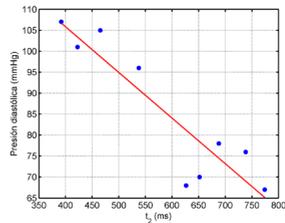


Fig. 9. Mejor ajuste de la presión diastólica para el sujeto 1 (modelo n°10).

existiendo cierta correlación entre el PAT y la presión arterial, el tiempo de tránsito del pulso (PTT) mantiene una mejor correlación con la presión, por lo que es considerado como mejor estimador.

Aunque haya estudios que aseguren que el PTT es mejor estimador que el PAT, su medición sin aparatos médicos es bastante más complicada que la del PAT, ya que requiere de un fonocardiógrafo [33]. Sin embargo un estudio de la influencia y variabilidad real del PTT en el PAT puede ser relevante para evaluar la posibilidad y los beneficios de incorporar esta variable a los modelos existentes y a una posible monitorización no invasiva en tiempo real.

A. Ventajas del sistema propuesto

El sistema portátil (*wearable*) presentado en este trabajo tiene como principal ventaja la estimación de la presión arterial de forma no invasiva y en tiempo real. Otros métodos habituales de medición de la presión requieren de, o bien métodos invasivos, como la medición directa de la presión a partir del transductor de una cánula situada directamente en la arteria, o bien métodos no invasivos pero molestos, como el esfigmomanómetro. Aunque la medida más precisa se obtiene con los métodos invasivos, esto solo es de aplicación directa en pacientes ingresados en entornos hospitalarios y bajo supervisión médica.

Mediante métodos basados en ECG y PPG pueden evaluarse tendencias o fluctuaciones de la presión sanguínea, algo que no puede ofrecer el método convencional del esfigmomanómetro. Por tanto, este pequeño dispositivo permite una monitorización ambulatoria y de forma continuada, sin ninguna molestia más que la necesidad de llevar un cinturón colocado en el pecho y un sensor en la yema de un dedo de la mano o en otras partes del cuerpo.

Esta plataforma puede ser integrada en redes de sensores corporales (*Wearable Body Sensor Networks (WBSN)*) [34] cuando estén comercialmente disponibles. Además, existe la posibilidad de añadir en un futuro, de forma sencilla, más sensores en la misma plataforma con el objetivo de realizar una monitorización más completa de los individuos.

La oportunidad de tener una buena estimación de la presión arterial en tiempo real, de manera continuada y sin grandes incomodidades para el paciente utilizando un sistema como el propuesto en este trabajo, puede abrir un gran abanico de posibilidades en la gestión de salud pública, por ejemplo, emitiendo alarmas, avisos preventivos, etc.

B. Limitaciones y futuras líneas de investigación

Entre las limitaciones de esta investigación cabe destacar: la necesidad de calibrar el sistema para cada sujeto y los pocos pacientes en los que se ha podido realizar el experimento.

Esta calibración es actualmente necesaria, puesto que se infieren variaciones en la velocidad de la onda de pulso a partir de variaciones temporales entre el ECG y la PPG, y esto solo es válido en el caso de que la longitud se suponga constante. Como trabajo

futuro se propone la posibilidad de incluir alguna variable adicional que estime la longitud recorrida por la onda de pulso desde el corazón hasta el punto de medida de la PPG tales como la longitud del brazo o la estatura del sujeto, que permitirá estudiar la relación interpersonal de los modelos.

Debido a que el sistema estima la presión a partir de medidas indirectas de ECG y PPG, es necesaria una validación de este método para un posible uso clínico. Para ello, una de las posibles mejoras puede ser la sustitución del esfigmomanómetro, considerado en este trabajo como estándar de referencia, por valores de presión arterial obtenidos directamente mediante un transductor en la arteria. De esta forma, la relación entre los valores de presión estimados mediante los métodos basados en ECG y PPG y los de la presión arterial puede ser determinada de forma más precisa.

Aunque se ha observado la existencia de modelos que se ajustan mejor para unas personas que para otras, la principal limitación en el estudio es haberlo realizado únicamente con 3 sujetos. Por tanto, se propone como trabajo futuro un estudio con una población más amplia, que permita extraer conclusiones más profundas y generales.

VI. CONCLUSIONES

En este trabajo se presenta una recopilación y evaluación experimental de 10 de los principales modelos de estimación de la presión arterial a partir de ECG y PPG que existen en la literatura. Además se presenta una aplicación desarrollada en Android de monitorización no invasiva en tiempo real que obtiene constantes vitales de la persona como la frecuencia cardiaca, el nivel de saturación de oxígeno en sangre y la estimación de la presión arterial. Para ello, el sistema debe ser calibrado previamente para cada persona utilizando en un esfigmomanómetro como estándar de referencia. El modelo que presente mejor correlación para la presión de los 10 estudiados será el elegido para esa persona.

De los resultados obtenidos en la evaluación de los modelos de estimación de presión arterial, puede apreciarse como el modelo 1 (basado en la combinación del PAT y la frecuencia cardiaca) y el modelo 10 (basado en el tiempo de diastólica) son los modelos que presentan mejor coeficiente de determinación R^2 para las 3 personas estudiadas, en torno a 0.8. Por el contrario, otros modelos basados en el ancho del pulso de PPG o en el tiempo de sistólica presentan un coeficiente de determinación cercano a 0.

Los resultados experimentales obtenidos para los 3 sujetos estudiados justifican la necesidad de continuar investigando en esta línea con una base de pacientes más amplia, para así poder determinar la posible generalización de los modelos evaluados.

El sistema portátil (*wearable*) presentado en este trabajo es capaz de registrar el ECG, la PPG y constantes vitales como la frecuencia cardiaca, la oximetría y la estimación de la presión arterial, de forma continua, no invasiva y en tiempo real, pu-

diendo utilizarse para monitorización primaria, continua y preventiva en entornos no hospitalarios.

REFERENCIAS

[1] Eurostat, "Circulatory diseases main cause of death for men and women aged 65 years and over," Tech. Rep., Eurostat, 2013.

[2] Sangesta, Bagha and Laxmi Shaw, "A Real Time Analysis of PPG Signal for Measurement of SpO2 and Pulse Rate," *International journal of computer applications*, vol. 36, no. 11, pp. 45-50, 2011.

[3] Hannu Sorvoja, "Noninvasive blood pressure pulse detection and blood pressure determination," *Acta Universitatis Ouluensis, Series C, Technica*, no. 259, 2006.

[4] Gareth Beever, Gregory Y.H. Lip, and Eoin O'Brien, "ABC of hypertension: Blood pressure measurement. Part II-conventional sphygmomanometry: technique of auscultatory blood pressure measurement," *BMJ (Clinical research ed.)*, vol. 322, no. 7293, pp. 1043-7, apr 2001.

[5] British Hypertension Society, "Blood Pressure Measurement," 2007.

[6] Arthur Guyton and John Hall, *Tratado de Fisiología Médica*, vol. 12, Elsevier Health Sciences, 2011.

[7] American Heart Association and American Stroke Association, "Understanding and Managing High Blood Pressure," 2014.

[8] Aminurrahid Bin Noordin, *Cuff-less blood pressure meter system*, Ph.D. thesis, Universiti Teknologi Malaysia, 2009.

[9] Hannu Sorvoja and Risto Myllylä, "Noninvasive blood pressure measurement methods," *Molecular and Quantum Acoustics*, vol. 27, pp. 239-264, 2006.

[10] Lysander W. J. Bogert and Johannes J. van Lieshout, "Non-invasive pulsatile arterial pressure and stroke volume changes from the human finger," *Experimental Physiology*, vol. 90, no. 4, pp. 437-446, jul 2005.

[11] Kim H. Parker, "A brief history of arterial wave mechanics," *Medical & Biological Engineering & Computing*, vol. 47, no. 2, pp. 111-118, feb 2009.

[12] Raymond G. Gosling and Marc M. Budge, "Terminology for Describing the Elastic Behavior of Arteries," *Hypertension*, vol. 41, no. 6, pp. 1180-1182, jun 2003.

[13] W. Chen, T. Kobayashi, S. Ichikawa, Y. Takeuchi, and T. Togawa, "Continuous estimation of systolic blood pressure using the pulse arrival time and intermittent calibration," *Medical & Biological Engineering & Computing*, vol. 38, no. 5, pp. 569-574, sep 2000.

[14] J. Muehlsteff, X.L. Aubert, and M. Schuett, "Cuffless Estimation of Systolic Blood Pressure for Short Effort Bicycle Tests: The Prominent Role of the Pre-Ejection Period," in *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, aug 2006, vol. 1, pp. 5088-5092, IEEE.

[15] Ming-Zher Poh, Nicholas C. Swenson, and Rosalind W. Picard, "Motion-tolerant magnetic earring sensor and wireless earpiece for wearable photoplethysmography," *IEEE transactions on information technology in biomedicine : a publication of the IEEE Engineering in Medicine and Biology Society*, vol. 14, no. 3, pp. 786-94, may 2010.

[16] J.Y.A. Foo, S.J. Wilson, G. Williams, M-A. Harris, and D. Cooper, "Pulse transit time as a derived noninvasive mean to monitor arterial distensibility changes in children," *Journal of human hypertension*, vol. 19, no. 9, pp. 723-9, sep 2005.

[17] Federico S. Cattivelli and Harinath Garudadr, "Noninvasive Cuffless Estimation of Blood Pressure from Pulse Arrival Time and Heart Rate with Adaptive Calibration," in *2009 Sixth International Workshop on Wearable and Implantable Body Sensor Networks*, jun 2009, number 1, pp. 114-119, IEEE.

[18] R.A. Payne, C.N. Symeonides, D.J. Webb, and S.R.J. Maxwell, "Pulse transit time measured from the ECG: an unreliable marker of beat-to-beat blood pressure," *Journal of Applied Physiology*, vol. 100, no. 1, pp. 136-141, jan 2006.

[19] M. Nitzan, B. Khanokh, and Y. Slovik, "The difference in pulse transit time to the toe and finger measured by photoplethysmography," *Physiological Measurement*, vol. 23, no. 1, pp. 85-93, feb 2002.

[20] Robin P. Smith, Jérôme Argod, Jean-Louis Pepin, and

Patrick A. Levy, "Pulse transit time: an appraisal of potential clinical applications," *Thorax*, vol. 54, no. 5, pp. 452-457, may 1999.

[21] C.P. Chiu and C. Heneghan, "Continuous Blood Pressure Monitoring using ECG and Finger Photoplethysmogram," in *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, aug 2006, vol. 1, pp. 5117-5120, IEEE.

[22] C.C.Y. Poon and Y.T. Zhang, "Cuff-less and Noninvasive Measurements of Arterial Blood Pressure by Pulse Transit Time," in *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, jan 2005, vol. 6, pp. 5877-5880, IEEE.

[23] Juan Fernando Franco Higueta, *Sistema para estimar la presión arterial por medio de medidas indirectas utilizando una red inalámbrica de sensores corporales*, Ph.D. thesis, Universidad de Antioquia, Medellín, Colombia, 2012.

[24] Youngzoon Yoon, Jung H. Cho, and Gilwon Yoon, "Non-constrained Blood Pressure Monitoring Using ECG and PPG for Personal Healthcare," *Journal of Medical Systems*, vol. 33, no. 4, pp. 261-266, aug 2009.

[25] Aymen A. Awad, M. Ashraf M. Ghobashi, Robert G. Stout, David G. Silverman, and Kirk H. Shelley, "How Does the Plethysmogram Derived from the Pulse Oximeter Relate to Arterial Blood Pressure in Coronary Artery Bypass Graft Patients?," *Anesthesia & Analgesia*, vol. 93, no. 6, pp. 1466-1471, dec 2001.

[26] X.F. Teng and Y.T. Zhang, "Continuous and noninvasive estimation of arterial blood pressure using a photoplethysmographic approach," in *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE Cat. No. 03CH37439)*, 2003, pp. 3153-3156, IEEE.

[27] Francisco Rincón, Joaquín Recas, Nadia Khaled, and David Atienza, "Development and evaluation of multilevel wavelet-based ECG delineation algorithms for embedded wireless sensor nodes," vol. 15, no. 6, pp. 854-863, 2011.

[28] Ejgijius Kazanavicius, Rolandas Girycs, and Arinas Vrubliauskas, "Mathematical methods for determining the foot point of the arterial pulse wave and evaluation of proposed methods," *Information Technology and Control*, vol. 34, no. 1, pp. 29-36, 2005.

[29] H.F. Posada-Quintero, D. Delisle-Rodríguez, M.B. Cuadra-Sanz, and R.R. Fernández de la Vara-Prieto, "Evaluation of pulse rate variability obtained by the pulse onsets of the photoplethysmographic signal," *Physiological measurement*, vol. 34, no. 2, pp. 179-87, feb 2013.

[30] Texas Instruments, *ADS1294*, <http://www.ti.com/product/ADS1294>.

[31] *Nonin*, <http://www.nonin.com/ReusableSensors>.

[32] *Flot Android Chart*, <https://code.google.com/archive/p/flot-android-chart/>.

[33] Abbas K. Abbas and Rasha Bassam, *Phonocardiography Signal Processing*, Morgan & Claypool Publishers, 2009.

[34] Rubén Braojos, Hossein Mamaghianian, Alair Dias Junior, Giovanni Ansaloni, David Atienza, Francisco J. Rincón, and Srinivasan Murali, "Ultra-Low Power Design of Wearable Cardiac Monitoring Systems," in *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference - DAC '14*, New York, New York, USA, 2014, pp. 1-6, ACM Press.

[35] John W. Severinghaus and Yoshiyuki Honda, "History of blood gas analysis. VII. Pulse oximetry," *Journal of Clinical Monitoring*, vol. 3, no. 2, pp. 135-138, apr 1987.

[36] G. Zhang, M. Gao, D. Xu, N. B. Olivier, and R. Mukkamala, "Pulse arrival time is not an adequate surrogate for pulse transit time as a marker of blood pressure," *Journal of Applied Physiology*, vol. 111, no. 6, pp. 1681-1686, dec 2011.

Versat, a Runtime Partially Reconfigurable Coarse-Grain Reconfigurable Array using a Programmable Controller

João D. Lopes
INESC-ID Lisboa / Técnico Lisboa
University of Lisbon
Email: joao.d.lopes@ist.utl.pt

Rui Santiago
INESC-ID Lisboa / Técnico Lisboa
University of Lisbon
Email: rui.santiago@ist.utl.pt

José T. de Sousa
INESC-ID Lisboa / Técnico Lisboa
University of Lisbon
Email: jts@inesc-id.pt

Abstract—Integrating a Coarse-Grain Reconfigurable Array (CGRA) in a System-on-Chip (SoC) is often a challenging endeavor, especially because of software integration issues. In this paper we show that a runtime partially reconfigurable CGRA with a most basic controller can be used as an accelerator by any embedded processor with minimal changes to the original software. The CGRA itself is easy to program and hides all its specificities like reconfiguration and data transfers from the host processor. Yet, it is capable of implementing rather complex kernels, which makes the integration job easier. We propose a CGRA architecture and a programming paradigm to support runtime partial reconfiguration. Experimental results are presented.

Keywords—*configurable computing, coarse-grain reconfigurable arrays, heterogeneous systemseconfigurable computing, coarse-grain reconfigurable arrays, heterogeneous systemsr*

I. INTRODUCTION

A suitable type of reconfigurable hardware for embedded devices is the Coarse Grain Reconfigurable Array (CGRA) [1], due to its compact size and low power consumption. Fine grain reconfigurable fabrics such as FPGAs can be an alternative but, compared to CGRAs, embedded FPGA cores consume significantly more silicon area and power.

A CGRA is a collection of programmable functional units and embedded memories connected by programmable interconnects. This structure is what is called the reconfigurable array. When given a set of configuration bits, the reconfigurable array forms a hardware datapath able to execute a certain task orders of magnitude faster than a conventional CPU. CGRAs are used as hardware co-processors to accelerate algorithms that are time/power consuming in regular CPUs.

Normally, the reconfigurable array is used only to accelerate program loops and the non-loop code is run on an attached processor which has a conventional architecture. For this reason, CGRAs normally include a conventional processor. For example, the Morphosys architecture [2] integrates a small Reduced Instruction Set Computer (RISC) and the ADRES architecture [3] integrates a Very Large Instruction Word (VLIW) processor.

When programming a CGRA, it is necessary to partition the code between the reconfigurable array and the processor. Such partitioning is normally done manually, as compilers that

are able to do this well have remained elusive despite two decades of research on the subject. A second problem is how the data is shared between the processor and the reconfigurable array. In Morphosys [2], a DMA block is used to transfer data between the reconfigurable array and a shared memory (loose coupling). In ADRES [3], shared registers are used between the VLIW processor and the reconfigurable array. They have mutually exclusive operation: the system is either in VLIW mode or reconfigurable array mode but both blocks have instantaneous access to the data in the shared registers. A third problem is how the reconfigurable array is configured. Both Morphosys and ADRES have a configuration memory which can store a certain number of configuration contexts. Each configuration context corresponds to a particular hardware datapath used for acceleration. If the application requires more configuration contexts than the number of configuration contexts that can fit in the configuration memory, then the extra configuration contexts have to be fetched from external memory, incurring on a performance penalty similar to a cache miss in a conventional computer architecture.

For the above reasons, it becomes quite difficult to program a CGRA. In the case of Morphosys, the programmer must separately code the context words and the main program running on the RISC processor, which includes the DMA transfers for data and configuration contexts. Also, the main program has to manage the reconfigurable array, initiating its execution and monitoring when it completes execution. These low level programming tasks can be quite distracting, as the programmer should concentrate on the application itself.

In this work we adopt a different perspective. First of all, we observe that the tasks that need to be run on a CGRA are much simpler than what is normally considered. In the DSP world many examples of these tasks can be found: Infinite Impulse Response (IIR) and Finite Impulse Response (FIR) filters, transforms such as the Fast Fourier Transform (FFT) and the Discrete Cosine Transform (DCT), and others. These well defined *kernels* have simple control code but are very intensive in terms of data processing. Therefore we propose to replace the coupled processor in a CGRA with a much simpler controller. For example, the controller should be complex enough to control a motion estimation engine, but it does not need to be as complex as to run an entire video encoder algorithm. Moreover, the controller needs low latency access to the reconfigurable array, which may be difficult to guarantee

with a more complex solution.

In our view, the CGRA is only used for the compute intensive kernels. For the rest of the code, an off-the-shelf solution such as a general purpose embedded CPU should be used. The CPU uses the CGRA as an acceleration engine and should interact with the CGRA using an application programming interface like the Open Computing Language (OpenCL) [4]. Using an off-the-shelf processor has the advantage of not having to develop a compiler for it.

In any case, a compiler is needed for the CGRA. The main difficulty in developing this compiler is supporting the reconfigurable array. Since we made the CGRA controller simple, the controller code does not need sophisticated compilation techniques and optimizations. In this way, we can focus in developing a compiler that excels in the reconfigurable array part and is very basic for the controller part. This design decision avoids the expensive integration of the compiler in a more general framework such as *gcc* or *llvm*.

In other approaches like [2], [3], the configuration contexts are prepared or compiled beforehand and are applied at runtime. One limitation of this approach is the inability to create or modify configurations during the execution of the CGRA. However, generating configurations on the fly can be interesting in the following situations: (1) supporting input parameters and (2) creating a new configuration by modifying a similar configuration already in memory.

Most CGRAs are only fully reconfigurable [3], [5], [2], [6] and do not support partial reconfiguration. The disadvantage of full reconfiguration is the amount of configuration data that must be kept and/or moved to/from external memory. Partial reconfiguration exploits the similarity between consecutive configurations, and can reduce the amount of configuration storage/bandwidth at the cost of a more complex reconfiguration hardware infrastructure. The RaPiD architecture supports partial reconfiguration but only for a subset of the configuration bitstream [7]. The PACT architecture [8] also supports self and partial reconfiguration but the reconfiguration process is significantly slow and users are recommended to avoid it and resort to full reconfiguration whenever possible.

In this paper we propose Versat, a CGRA that supports efficient runtime partial reconfiguration. Like other CGRAs, Versat features a controller module. However, unlike other approaches, the Versat controller is not meant to execute the code that cannot be run on the reconfigurable array. The purpose of the Versat controller is to efficiently deal with reconfiguration and data transfers, so that the reconfigurable array becomes a very flexible hardware accelerator that can be used by any embedded processor. The code that is not suitable for the reconfigurable array is run on a separate embedded processor for which a good set of development tools already exist. In this work, we make the reconfigurable array much more powerful and dispense with the attached processor. Compared to reconfigurable arrays that exist in other CGRAs, Versat has the following advantages:

- It can map nested loops rather than just simple loops.
- It can map sequences of nested loops rather than just single nested loops.

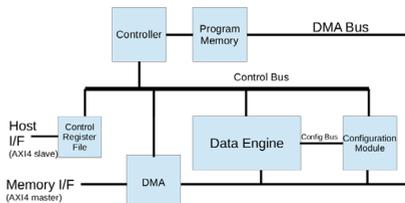


Fig. 1. Versat top-level entity

In order to achieve these distinctive features, we designed a minimal controller with just 16 instructions. In its memory map, the controller can see the configuration register file, which is organized in configuration spaces and fields to support partial reconfiguration. It can also see the DMA registers to be able to control data transfers to and from external memory. The controller has memory mapped access to the registers and memories of the reconfigurable array. Finally, the controller has access to a special register file used to communicate with host processors which are using Versat as an accelerator. Versat can be programmed in assembly language or in a higher level language such as C or C++. An assembler written in Python and a lightweight C/C++ compiler have been developed. To the best of our knowledge it is the only CGRA that can be programmed in assembly, thus permitting low-level code optimizations. The assembly programming is made easy by the simple structure of the reconfigurable array.

II. ARCHITECTURE

Some CGRAs are homogeneous [7], i.e., they consist of a collection of identical functional units, whereas others are heterogeneous and support a diversity of FUs [9]. A careful analysis published in [10] has favored heterogeneous CGRAs as having better silicon utilization and power efficiency when compared to homogeneous solutions. Thus, we have adopted a heterogeneous CGRA architecture in this project.

The top level entity of the Versat module is shown in Fig. 1 and it looks rather standard. However, each of the modules is originally designed to meet the overall objective of a minimally controlled and yet effective CGRA. Versat is designed to carry out computations on data arrays using its Data Engine (DE). To perform computations the DE needs to be configured using the Configuration Module. The data to be processed is read from an external memory using a DMA engine, also used to store the processed data back in the memory. A typical computation uses a number of DE configurations and a number of external memory transactions. The Controller executes programs stored in the Program Memory. Each program executes one or more algorithms, coordinating the reconfiguration and execution of the DE, as well as the external memory accesses. The controller accesses the modules in the system by means of the Control Bus.

The Versat top-level entity has a host interface and a memory interface. The host interface (AXI4 slave) is used by a host system to load a Versat program containing a set

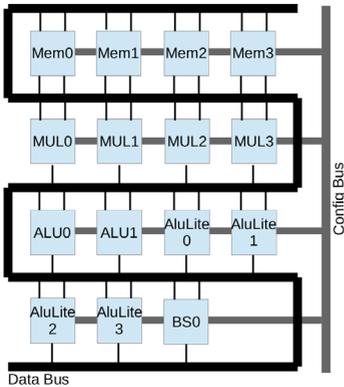


Fig. 2. Data engine

of procedures or kernels, and to command Versat to execute those kernels. Using this interface, the host issues commands to Versat, and Versat reports status information and returns little amounts of data. The memory interface (AXI4 master) is used to access data from an external memory using the DMA. Large data exchanges are reserved for the memory interface. Exceptionally, if the amount of data is small, the host interface may be used for data exchange. For example, debug data is likely to use the host interface.

A. Data engine

The Data Engine (DE) currently has a fixed topology using 15 functional units (FUs) as shown in Fig. 2. However, it is relatively easy to change its structure to better accommodate a particular application. It is a 32-bit architecture and contains the following types of FUs: embedded memories (MEM), multipliers (MUL), arithmetic and logic units (ALU), including the lightweight versions (AluLite). The Versat controller can read and write the output register of the FUs and can read and write to the embedded memories.

Each FU contributes its 32-bit output into a wide Data Bus. Additionally the data bus has two fixed entries for driving the constants 0 and 1, which are needed in many datapaths. Each FU is able to select one data bus entry for each of its inputs.

The FUs are configured by the Config Bus. Each FU is configured with a mode of operation and with its input selections. There are no global configurations – configuration data is always linked to a particular FU. The Config Bus is divided in configuration spaces, one for each FU. Each configuration space contains several configuration fields. Configuring a set of FUs results in a custom datapath for a particular computation.

Datapaths can have parallel execution lanes to exploit Data Level Parallelism (DLP) or pipelined paths to exploit Instruction Level Parallelism (ILP). An important type of FU is the dual-port embedded memory. Each port is equipped

with an address generator to enable data-flow computing. If two memories are part of the same datapath, they will have its address generators working in a synchronized fashion. Given enough resources, multiple datapaths can operate in parallel with independent address generation. This corresponds to having multiple concurrent threads in Versat – Thread Level Parallelism (TLP).

The discussion of the details of address generation in Versat falls out of the scope of this paper. We will simply state the following properties of the address generators: (1) the address generators are compact enough to be dedicated to each memory port; (2) only two levels of nested loops, where the inner loop is shorter, are supported (reconfiguration after each short inner loop would cause excessive reconfiguration overhead); (3) if the inner loop is long then Versat will work in single loop mode and rely on its fast reconfiguration time to implement the outer loops.

B. Program memory

The instruction memory is designed to contain 2048 instructions. This is deemed enough for most compute tasks (kernels) that we have in mind. In fact, the idea is to place multiple compute kernels in this memory. However, historically, memory needs always increase beyond expected. As we develop compiler tools, we notice that it is likely that the memory capacity to contain compiled programs may need to be larger. Note that compiler produced code is never as optimized as the code that results from handwritten assembly instructions.

The program memory is divided in two parts: boot ROM and execution RAM. The boot ROM contains a hardwired program which runs after hardware reset. The execution RAM is where the user program to be run is loaded. The Versat controller can write to the program memory to load it and then can read each instruction to execute it; it cannot read words from the program memory into its data register. Alternatively, the program memory can be loaded using the DMA.

The code in the boot ROM is called the boot loader. It is used for loading programs and data files into Versat, according to instructions from the host; it is also used to download data files from Versat and to start running programs previously loaded in the execution RAM. If the size of the programs increase, it is likely that the execution RAM will evolve into an instruction cache, which will share the external memory with the DMA.

C. Control register file

Host and Versat exchange information and synchronize using the Control Register File (CRF). Both can read and write to the CRF which has 16 general purpose registers.

The CRF is used by the host to pass the address of a program to be run by Versat, along with a set arguments needed by that program. Versat uses the CRF to return status information to the host and a few data words.

D. Controller

The central claim of this paper is that runtime partially reconfigurable CGRAs can operate with a very basic controller and still be able to execute compute kernels that are normally

Building a configuration of the DE for the first time requires several writes to the fields of the configuration spaces of the involved FUs. In most applications there is a high likelihood that one configuration will be reused again. It is also likely that other configurations will differ little from the current configuration. Thus, it is useful to save certain configurations in the configuration memory to later reload them. A reloaded configuration may be used as is or partially changed.

III. PROGRAMMING

Versat can be programmed in its assembly language and in its C/C++ dialect. In this section we briefly explain how to use these two languages.

A. Programming in assembly

The Versat assembler is implemented in the Python language and converts assembly code into the 16 machine code instructions supported by the Versat controller. The assembler reads a text dictionary generated by running a special Verilog testbench. The dictionary contains the names of all constants that control the hardware. In this way, one is free to change the hardware without need to change the assembler. Typically, a Versat kernel has 3 parts:

- 1) Data loading
- 2) Execution
- 3) Data saving

For efficiency reasons, data loads and data saves are normally performed using the DMA. The following code configures and runs the DMA:

```
#configure external address
    rdw R1
    wrw DMA_EXT_ADDR_ADDR
#configure internal address
    ldi 0x3000
    wrw DMA_INT_ADDR_ADDR
#configure transfer size
    ldi 256
    wrw DMA_SIZE_ADDR
#configure direction and run the DMA
    ldi 1
    wrw DMA_DIRECTION_ADDR
```

After reading the dictionary the assembler can associate labels such as R1, DMA_EXT_ADDR_ADDR, DMA_INT_ADDR_ADDR, etc. with actual memory mapped addresses. Note that the first instruction loads the DMA external memory address from the CRF register R1 into the accumulator and the second instruction writes it to the respective DMA configuration register. It might have been the host system that specified the external memory address by writing it to R1. The third instruction loads the DMA internal memory address into the accumulator as an immediate value and the fourth instruction writes it to the respective DMA configuration register. In the same way, the DMA configuration registers for the transfer size and direction are configured in the next instructions (direction=1 indicates a transfer from the external memory to the internal memory). As soon as the direction is configured the DMA starts executing.

Now, one may want to take the time while the DMA is working to configure the data engine (DE). This can be accomplished by the following code

```
ldi smem2A
wrc MEM0A_CONFIG_ADDR, MEM_CONF_SELA_OFFSET
ldi smem2B
wrc MEM0B_CONFIG_ADDR, MEM_CONF_SELA_OFFSET
ldi 1024
wrc MEM0B_CONFIG_ADDR, MEM_CONF_START_OFFSET
...
```

The first two instructions are configuring memory port A of memory 0 to take as input port A of memory 2. Note that `wrc` is an alias for `wrw`, which takes the sum of its two parameters (MEM0A_CONFIG_ADDR+MEM_CONF_SELA_OFFSET) as the store address. MEM0A_CONFIG_ADDR is the configuration base address for port A of memory 0 and MEM_CONF_SELA_OFFSET is the address offset of the input selection field.

After configuring the DE, it is time to run it. The following code does just that.

```
ldi 0xC002
ldih 0x1D
wrv ENG_CTRL_REG
```

The first two instructions load register A with a command word, which specifies which functional units to initialize and which address generators to run. The contents of the command word is not explained in detail.

While the DE runs, one may configure the DE with the next configuration. Note that the running configuration is in the configuration shadow register, which is not affected if one writes to the main configuration register. As a result, if one manages to create the next configuration before the DE completes execution, the configuration time is completely hidden.

Now it is time to check whether the DE has finished running. This can be done by the following code:

```
wait ldi 0x0002
and ENG_STATUS_REG
beqi wait
```

The first instruction loads register A with a mask to be applied to the DE status register. The second instruction performs the logical AND of the DE status register and the mask. The mask used in this example has a single non-zero bit to check if the second LSB of the status register is set. That bit indicates whether the address generator of port A of memory 0 has finished execution. The third instruction branches to the first instruction if that bit is still reset, thus implementing a loop that waits for the DE to stop.

B. Programming in C/C++

Programming Versat in C/C++ is a lot more convenient than using the assembly language. However, Versat only supports a small subset of C/C++, deemed enough for the Versat

programmer. In this section we will provide some examples of using the Versat C/C++ dialect. The Versat compiler has been implemented in C++ itself, using the `bison` and `flex` utilities to build the parser.

The first observation that needs to be made is that the Versat C/C++ dialect does not yet support object or variable declarations. All objects and variables that can be used are predefined. For example, consider the following expression in the Versat C++ language:

$$R5 = R4+R3+(2-R6);$$

The variables R3-R6 refer to CRF registers and need not be declared by the user. The assembly code generated by the above expression tree is the following:

```
ldi 2
sub R6
rw R4
rdw R4
add R3
add RB
rw R5
ldi 0
beqi 0
nop
```

This example illustrates the power of the compiler in handling arithmetic expressions. In future versions of the compiler, declaration of variables and objects and their automatic mapping to memory addresses or registers will be considered. In spite of this and other limitations, the Versat compiler has very advanced features when it comes to configuring the data engine. For example, the following code configures and runs the DMA:

```
dma.config(R1, 0x00003000, 256, 1);
dma.run();
dma.wait();
```

The predefined `dma` object has methods for configuring, running and waiting for the DMA to finish. The `dma.config` method takes as arguments the external memory address, the internal memory address, the size of the transfer in 32-bit words and the direction of the transfer, by this order. The `dma.run` method sets the DMA running and the `dma.wait` method monitors the DMA status register and exits when the DMA has completed execution. Note that one could have performed other actions in parallel with the DMA execution, and only wait for the DMA later.

The most powerful feature of our compiler is the ease in describing engine configurations. The following code configures the entire data engine to implement a product of two vectors of complex elements.

```
de.clearConfig();
for (j=0; j<R6; j++) {
    for (i=0; i<R14; i++) {
        mem2A[R7+j*R13+i] = mem0A[R7+j*R13+i]
                           +mem0B[R1+j*R13+i];
```

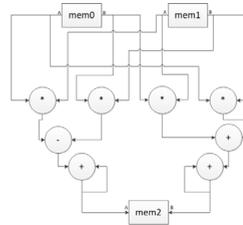


Fig. 5. Complex vector product

```
mem3A[R7+j*R13+i] = mem1A[R7+j*R13+i]
                    +mem1B[R1+j*R13+i];
mem2B[R1+j*R13+i] = mem0B[R1+j*R13+i]
                    -mem0A[R7+j*R13+i];
mem3B[R1+j*R13+i] = mem1B[R1+j*R13+i]
                    -mem1A[R7+j*R13+i];
}
}
```

The predefined `de` object, which represents the data engine, has useful methods for various purposes which will be described as needed. The `de.clearConfig` method restores the engine's the default configurations. The default configurations are very likely to be used, so starting from the default values and just adding the configurations that differ from the default configurations is the fastest way to configure the engine. In fact, the compiler does this automatically. It keeps track of the state of the configuration register and only issues assembly instructions to make the configurations that differ from the current configurations. This is one way the compiler makes use of partial reconfiguration.

The two nested `for` loops in the above code describe the DE configuration. Recall that Versat's address generators support two nested loops. Memory ports are predefined objects such as `mem0A`, `mem1B`, etc. A memory port address generator is configured by its index expression and the loop limits. For example, `mem2A[R7+j*R13+i]` configures port A of memory 2 to start with the address given by R7, to increment by 1 every cycle (1 is the coefficient of loop variable `i`), to increment by R13 every R14 cycles (R14 is the inner loop limit), and to repeat the inner loop R6 times. The body of the nested loops contains expressions where memory ports appear as variables, and create a hardware datapath that is mapped to the DE. For the above code the datapath created is depicted in Fig. 5.

More interesting is when two nested loops are placed within a third loop. The address generators do not support three nested loops but nested loops of arbitrary depths are supported via the controller. The following code shows this:

```
do { //partial reconfiguration until done
    R1=R8+R7;
    for (j=0; j<R6; j++) {
```

```

for (i=0; i<R14; i++) {
    mem2A[R7+j*R13+i] = mem0A[R7+j*R13+i]
                      +mem0B[R1+j*R13+i];
    mem3A[R7+j*R13+i] = mem1A[R7+j*R13+i]
                      +mem1B[R1+j*R13+i];
    mem2B[R1+j*R13+i] = mem0B[R1+j*R13+i]
                      -mem0A[R7+j*R13+i];
    mem3B[R1+j*R13+i] = mem1B[R1+j*R13+i]
                      -mem1A[R7+j*R13+i];
}
}
de.wait(mem2A);
de.run();

R7=R7+R6+R6;
R12=R12-1;
} while (R12!=0);
    
```

In the above code the `do while` outer loop is executed by the controller. In the body of this loop the two nested loops that run on the DE are placed. Note that the configuration of the DE changes for every iteration of the outer loop, since it depends on the values of registers `R1` and `R7`, which are updated inside the outer loop. If the two DE nested loops take a significant time to run, as we expect they do, then doing the outer loop on the controller, introduces a comparatively small overhead.

For each iteration of the outer loop, only the configurations that depend on `R1` and `R7` change. These represent the start addresses for the various memory ports. In other words, for each iteration of the outer loop one starts reading and writing to the memories at different addresses. This is another illustration of the runtime partial reconfiguration capability.

After describing the DE configuration, by means of the two nested loops, one commands the DE to wait until the address generator of port A of memory 2 has finished generating the write addresses. This is done by means of the `de.wait` method. In fact, memory 2 and memory 3 work in parallel, so one could have waited for memory 3 instead, or for both. The interesting part is why one waits for the DE to finish before we command it to run by means of the `de.run` method. The answer is that we want to wait for the DE run of the previous iteration of the outer loop to finish. In the first iteration, the `de.wait` method returns immediately as the DE has never been started. In the following iterations, the partial reconfiguration of the DE is performed by the nested loops, and only after one waits for the previous DE configuration to finish execution. Thus, DE reconfiguration occurs in parallel with DE execution. If the partial reconfiguration time is shorter than the execution of the previous configuration, the reconfiguration time is completely hidden and one will have to wait for a while before running the current configuration. However, if the partial reconfiguration takes longer than the previous configuration run, there is a reconfiguration overhead. From this explanation, the importance of partial reconfiguration in mitigating the reconfiguration overhead should be clear. If full reconfiguration were performed all the time, the chances to incur in reconfiguration overhead would be much higher.

Another way to reduce reconfiguration time is to temporarily save configurations in the configuration memory, in order to reuse them later. This will be shown in the next example. The following datapath implements simultaneous addition/subtraction of 4 vector elements.

```

de.clearConfig();
for (j=0; j<R6; j++) {
    for (i=0; i<R14; i++) {
        mem2A[i] = mem0A[i]+mem0B[i];
        mem3A[i] = mem1A[i]+mem1B[i];
        mem2B[i] = mem0B[i]-mem0A[i];
        mem3B[i] = mem1B[i]-mem1A[i];
    }
}
de.saveConfig(1);
    
```

Note that the vector indices do not depend on the `j` loop variable. That looks strange as this configuration would just repeat the inner loop `R6` times, since `R6` is the limit of the outer loop. However, this configuration is not intended to be run but rather to be saved in the configuration memory at position 1. It will be reloaded later and tweaked to implement the desired datapath. The following code illustrates that:

```

de.loadConfig(1);

mem2A.setIter(R6);
mem2A.setPer(R14);
mem2A.setDuty(R14);
mem2A.setShift(R13-R14);
...

do { //partial reconfiguration until done
    mem0A.setStart(R7);
    mem0B.setStart(R8+R7);
    mem1A.setStart(R7);
    mem1B.setStart(R8+R7);
    ...

    de.wait(mem2A);
    de.run();

    R7=R7+R6+R6;
    R12=R12-1;
} while (R12!=0);
    
```

After the configuration is reloaded via the `de.loadConfig` method, several manual changes to the configuration are effected by means of methods such as `tt mem0A.setStart`, which sets the start address of port A of memory 0. This highlights a different way of programming Versat – structural programming. Using structural programming, datapaths can be described like hardware netlists, and FUs configured individually as shown above. Below we provide a complete description of a first order IIR filter using structural programming.

```

int main() {
    de.clearConfig();
    Ralu1 = 0x73333333;
    RaluLite2 = 0x0CCCCCD;
}
    
```

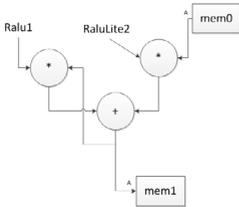


Fig. 6. First order IIR filter

```

mem0A.setIncr(1);
mem0A.setIter(512);
mem0A.setPer(5);
mem1A.connect(alu0);
mem1A.setDelay(6);
mem1A.setPer(5);
mem1A.setStart(1);
mem1A.setIncr(1);
mem1A.setIter(512);
mult0.connectPortA(alu1);
mult0.connectPortB(alu0);
mult1.connectPortA(aluLite2);
mult1.connectPortB(mem0A);
alu0.setOper('+');
alu0.connectPortA(mult0);
alu0.connectPortB(mult1);
de.run(mem0, mem1, bs0, mult0, mult1,
      mult2, mult3, aluLite3, aluLite1,
      aluLite0, alu0);
de.wait(mem1A);
}

```

In the above description one should notice the *Ralu1* and *Ralulite2* predefined variables, which represent the output registers of the *alu1* and *aluLite2* FUs, respectively. In this example these registers are used for storing filter coefficients. The connection methods such as *mem1A.connect*, *mult0.connectPortA*, etc, should also be noticed. For example *mem1A.connect(alu0)* connects portA of memory 1 to the output of *alu0*. The datapath corresponding to the above description is given in Fig. 6.

C. Packaging and using Versat kernels

Having shown how to program Versat, we will now show how can host processors use Versat kernels. In terms of hardware, Versat's standard AXI interfaces makes it ready for inclusion in a great variety of embedded systems. The Versat hardware should be integrated and host processors should take note of its base memory mapped address. In fact, running Versat kernels from a host processor is as simples as illustrated below:

```

#include "versat.h"
...
//load versat program
versat_load();

```

```

...
//call a versat kernel
versat_run(KERNEL_ID, arg1, arg2, ...);
//may insert user code here
...
//wait for versat to finish
versat_wait();
...

```

The *versat_load*, *versat_wait* and *versat_run* functions are defined in the *versat.c* module, which should be compiled along with the user program. The files *versat.h* and *versat.c* constitute the Versat driver. The *versat.h* file includes a *versat_prog.h* file provided by the Versat compiler, which contains the Versat program machine code and the definition of the kernel ID's. In fact, the *versat.h* file is as simple as given below.

```

#include "versat_prog.h"
#define VERSAT_BASE_ADDR 0x7F400000
void versat_load();
void versat_wait();
void versat_run(int kernel, ...);

```

The *versat_load* function initially loads the versat program in its program memory. Since we assume the kernels in the program will be used multiple times by the host program, then the time to initially load the Versat program should dilute into a small overhead. The *versat_wait* function, as the name indicates, blocks the host program until Versat is free to run a kernel. The *versat_run* function sets Versat running and is a non-blocking function. User code can execute in parallel with Versat. When the host needs Versat again, it should call the *versat_wait* function again.

IV. EXPERIMENTAL RESULTS

This section presents results on implementing Versat on FPGA and as an IP core for an ASIC. Performance results comparing Versat to FPGA and VLSI embedded processors are also shown.

A. FPGA results

FPGA implementation results are given in Table IV. These results show that Versat can be implemented in FPGA using a reasonable amount of resources. The frequency of operation in FPGA is somewhat low, which indicates that more work on improving timing is necessary. However, as the performance results will show, Versat can execute kernels using many fewer clock cycles than typical embedded processors.

TABLE IV. FPGA IMPLEMENTATION RESULTS

Architecture	Logic	Regs	RAM(kbit)	Mults	Fmax(MHz)
Cyclone IV	19366 LEs	4673	351	32 (9 bits)	64
Virtex V	12510 LUTs	4396	1062	16 (18 bits)	102

Now we present execution results of running a set of example kernels on Versat and on the embedded processors Microblaze and Nios II. Hardware timers have been used to measure the time in clock cycles and the results are summarized in Table V.

Whether using Versat or the embedded processors, the program has been placed in on-chip memory and the data in an external DDR memory device. Cycle counts include processing, reconfiguration and data transfer times. The results for the FPGA embedded processors have been obtained on a Xilinx ML505 board and on an Altera DE0 Nano board.

The purpose of these results is to show that Versat is capable of acceleration. However, only one example, the `fft` example, illustrates the partial self-reconfiguration capabilities of Versat. The other four examples are single configuration kernels. Execution times of C implementations running on the Microblaze and NIOS processors have been obtained and are very similar. Thus, in column *Processor* in Table V, the average cycle counts for these two processors is given. The total cycle counts for Versat are given in the *Versat* column, and the *Unhidden* column gives the unhidden controller cycles, i.e., the number of clock cycles when the Versat controller is running but the data engine is not. This happens when it is not possible to hide the execution of the controller. The *Speedup* column is simply the ratio between columns *Processor* and *Versat*.

TABLE V. EXECUTION RESULTS

Kernel	Processor	Versat	Unhidden	Speedup
<code>vec_add</code>	11356	4517	41	2.51
<code>lpf1</code>	15311	7487	59	2.05
<code>lpf2</code>	20644	10567	86	1.95
<code>cip</code>	25110	6673	106	3.76
<code>fft</code>	260226	16705	638	15.58

In Table V, `vec_add` is a vector addition, `iir1` and `iir2` are 1st and 2nd order IIR filters, `cip` is a complex vector inner product and `fft` is a Fast Fourier Transform. All kernels operate on Q1.31 fixed-point data with vector sizes of 1024.

Kernel `vec_add` is very simple and shows the lowest speedup in Versat. In Versat it produces one vector element result per cycle but the overhead of loading the data and its single configuration is significant. Moreover, in a regular processor, kernels like these, without loop carried dependencies, can undergo powerful compilation optimizations such as loop unrolling. Kernels `lpf1` and `lpf2` also show modest speedups due to the feedback loops needed to implement the filters in Versat; they produce new vector elements every 5 and 8 cycles, respectively. Kernel `cip` is more complex, using 4 multiplies in parallel followed by pipelined adders. Due to a deeper pipeline, the speedups for the `cip` kernel are greater, despite the feedback loop needed to implement the accumulation in the end; a new result vector element is accumulated every other cycle.

In Table V, the first 4 kernels use a single Versat configuration and the data transfer size dominates. For example, the `vec_add` kernel processing time is only 1090 cycles and the remaining 3427 cycles account for data transfer and control. The FFT kernel is more complex and goes through 43 Versat configurations generated on the fly by the Versat controller. The processing time is 12115 cycles and the remaining 4590 cycles is for data transfer and control. It should be noted that most of the control is done while the data engine is running.

In fact only 638 cycles are unhidden control cycles in the FFT kernel.

B. ASIC results

Versat has been designed using a UMC 130nm process. Table VI compares Versat with a state-of-the-art embedded processor and two other CGRA implementations. The Versat frequency and power results have been obtained using the Cadence IC design tools, and the node activity rate extracted from simulating an FFT kernel.

TABLE VI. IMPLEMENTATION RESULTS

Core	Node(nm)	Area(mm ²)	Freq.(MHz)	Power(mW)
ARM Cortex A9 [11]	40	4.6	800	500
Morphosys [2]	350	168	100	7000
ADRES [3]	90	4	300	91
Versat	130	4.2	170	99

Because the different designs use different technology nodes, to compare the results in Table VI, we need to use a scaling method [12]. A standard scaling method is to assume that the area scales with the square of the feature size and that the power density remains constant at constant frequency. Doing that, we conclude that Versat is the smallest and least power hungry of the CGRAs. If Versat were implemented in the 40nm technology, it would occupy about 0.4mm², and consume about 44mW running at a frequency of 800MHz. That is, Versat is 10x smaller and consumes about 11x less power compared with the ARM processor.

The ADRES architecture is about twice the size of Versat. Morphosys is the biggest one, occupying half the size of the ARM processor. These differences can be explained by the different capabilities of these cores. While Versat has a 16-instruction controller and 11 FUs (excluding the memory units), ADRES has a VLIW processor and a 4x4 FU array, and Morphosys has a RISC processor and an 8x8 FU array.

A prototype has been built using a Xilinx Zynq 7010 FPGA, which features a dual-core embedded ARM Cortex A9 system. Versat is connected as a peripheral of the ARM cores using its AXI4 slave interface. The ARM core and Versat are connected to an on-chip memory controller using their AXI master interfaces. The memory controller is connected to an off-chip DDR module.

Results on running our set of kernels on Versat and on the ARM Cortex A9 are summarized in Table VII. For both the ARM and Versat, the program has been placed in on-chip memory and the data in an external DDR memory device. Cycle counts include processing, reconfiguration and data transfer times. The speedup and energy ratio have been obtained assuming the ARM is running at 800 MHz and Versat is running at 600MHz in the 40nm technology. The energy ratio is the ratio between the energy spent by the ARM processor alone and the energy spent by an ARM/Versat combined system using the power figures in Table VI.

The results in Table VII show that, for an ASIC implementation, good performance speedups and energy savings can be obtained, even for single configuration kernels.

TABLE VII. CYCLE COUNTS, SPEEDUP AND ENERGY RATIO

Kernel	ARM Cortex A9 cycles	Versat cycles	Speedup	Energy Ratio
vec_add	14726	4517	2.45	2.29
iir1	18890	7487	1.89	1.77
iir2	24488	10567	1.74	1.62
c1p	25024	6673	2.81	2.63
fft	394334	16705	17.70	16.55

We can compare Versat with Morphosys since it is reported in [13] that the processing time for a 1024-point FFT is 2613 cycles. Compared with the 12115 cycles taken by Versat this means that Morphosys was 4.6x faster. This is not surprising since Morphosys has 64 FUs compared to 11 FUs in Versat. However, our point is whether an increased area and power consumption is justified when the CGRA is integrated in a real system. Note that, if scaled to the same technology and same frequency, Morphosys would be 5x the size of Versat and consume 16.6 times more power. In terms of energy, this means that Versat would consume 3.6x less energy compared to Morphosys.

Unfortunately, comparisons with the ADRES architecture have not been possible, since we have not found any cycle counts published, despite ADRES being one of the most published CGRA architectures.

V. CONCLUSION

In this paper we have presented Versat, a new reconfigurable architecture that can take care of the reconfiguration process itself, as well as the data movement operations to and from an external memory. Versat is programmed in its own assembly language and in its own C/C++ dialect. It is designed to handle host procedure calls using a clean interface.

Versat is a minimal CGRA with 4 embedded memories, 11 FUs and a basic 16-instruction controller. Compared with other CGRAs with larger arrays, Versat requires more configurations per kernel and a more sophisticated reconfiguration mechanism. Thus, the Versat controller can generate configurations and uses partial reconfiguration whenever possible. The controller is also in charge of data transfers and basic algorithmic flows.

The experimental results show that in general Versat can accomplish speedups even if the kernels use a single Versat configuration. For kernels that require multiple configurations, Versat can achieve speedups of one order of magnitude. This is because reconfiguration times are small and can be hidden when more than one configuration is run in a kernel.

Results on a VLSI implementation show that Versat is competitive in terms of silicon area, frequency of operation and power consumption. Performance results show that a system combining a state-of-the-art embedded processor and the Versat core can be 17x faster and more energy efficient than the embedded processor alone, in certain compute intensive kernels.

Compared to state-of-the-art CGRAs, Versat has approximately the same performance per functional unit in quantitative terms. Since Versat intentionally has a smaller number of functional units, its performance is below that of other CGRAs. However, in terms of silicon area and energy consumption,

Versat is clearly superior. In fact it has been shown that its area is half that of ADRES [3] and its energy consumption is 3.6x inferior compared to Morphosys [2]. Perhaps the biggest advantage of using Versat is the ease at which it can be programmed and used. Unlike other CGRAs which can only map one program loop, Versat can map sequences of loops (nested or not) and control code. This means that high-level functions can be mapped onto Versat, and only a minimal effort is required for a host processor to use Versat as a co-processor.

ACKNOWLEDGMENT

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

REFERENCES

- [1] Bjorn De Sutter, Praveen Raghavan, and Andy Lambrechts. Coarse-grained reconfigurable array architectures. In Shuvra S. Bhattacharyya, Ed F. Deprettere, Rainer Leupers, and Jarmo Takala, editors, *Handbook of Signal Processing Systems*, pages 449–484. Springer US, 2010.
- [2] Ming hau Lee, Hartaj Singh, Guangming Lu, Nader Bagherzadeh, and Fadi J. Kurdahi. Design and implementation of the MorphoSys reconfigurable computing processor. In *Journal of VLSI and Signal Processing-Systems for Signal, Image and Video Technology*. Kluwer Academic Publishers, 2000.
- [3] Bingfeng Mei, A. Lambrechts, J.-Y. Mignolet, D. Verkest, and R. Lauwereins. Architecture exploration for a reconfigurable architecture template. *Design & Test of Computers, IEEE*, 22(2):90–101, March 2005.
- [4] John E. Stone, David Gohara, and Guochun Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science & Engineering*, 12(3):66–73, May 2010.
- [5] G. Burns and P. Grujters. Flexibility tradeoffs in soc design for low-cost sdr. In *Proceedings of SDR Forum Technical Conference*, 2003.
- [6] R. Hartenstein, M. Herz, T. Hoffmann, and U. Nagelinger. Mapping applications onto reconfigurable Kressarrays. In Patrick Lysaght, James Irvine, and Reiner Hartenstein, editors, *Field Programmable Logic and Applications*, volume 1673 of *Lecture Notes in Computer Science*, pages 385–390. Springer Berlin Heidelberg, 1999.
- [7] Carl Ebeling, Darren C. Cronquist, and Paul Franklin. Rapid - reconfigurable pipelined datapath. In *Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers, FPL '96*, pages 126–135, London, UK, 1996. Springer-Verlag.
- [8] V. Baumgarte, G. Ehlers, F. May, A. Nckel, M. Vorbach, and M. Weinhart. PACT XPP – a self-reconfigurable data processing architecture. *The Journal of Supercomputing*, 26(2):167–184, 2003.
- [9] P.M. Heysters and G.J.M. Smit. Mapping of DSP algorithms on the MONTIUM architecture. In *Proceedings of the International Parallel and Distributed Processing Symposium, 2003*, pages 6–, April 2003.
- [10] Yongjun Park, J.J.K. Park, and S. Mahlik. Efficient performance scaling of future CGRAs for mobile applications. In *International Conference on Field-Programmable Technology (FPT)*, 2012, pages 335–342, Dec 2012.
- [11] Wei Wang and Tanima Dey. A survey on ARM Cortex A processors. <http://www.cs.virginia.edu/skadron/cs8535s11/armcortex.pdf>. Accessed 2016-04-16.
- [12] W. Huang, K. Rajamani, M. R. Stan, and K. Skadron. Scaling with design constraints: Predicting the future of big chips. *IEEE Micro*, 31(4):16–29, July 2011.
- [13] A. H. Kamalizad, C. Pan, and N. Bagherzadeh. Fast parallel FFT on a reconfigurable computation platform. In *Computer Architecture and High Performance Computing, 2003. Proceedings. 15th Symposium on*, pages 254–259, Nov 2003.

Ciberseguridad en robots autónomos: Análisis y evaluación multiplataforma del bastionado ROS

Francisco Javier Rodríguez Lera, Vicente Matellán, Jesús Balsa, Fernando Casado¹

Resumen— Los robots autónomos son cada vez más comunes, lo que hace que la ciberseguridad de los sistemas empotrados que los controlan se esté convirtiendo en una preocupación. Muchos de los sistemas robóticos que se desarrollan actualmente utilizan el *framework* distribuido ROS. Este sistema se diseñó para uso fundamentalmente investigador y prácticamente no incluye ningún mecanismo de ciberseguridad. Proponemos utilizar el cifrado 3DES para solventar los problemas de confidencialidad asociados al uso del paradigma de publicación-subscripción en claro que usa ROS. Para valorar la influencia del cifrado en el rendimiento general de las plataformas, presentamos una comparativa del desempeño de dos plataformas robóticas con diferentes CPUs. Los campos analizados fueron tres: tiempo de cifrado/descifrado, *throughput* y tiempo consumido de CPU. En los experimentos utilizamos dos sensores ampliamente extendidos en robótica: una webcam y un sensor láser. Los resultados muestran que sobre robots con poca capacidad computacional, el cifrado afecta en el rendimiento de los sensores y en el tráfico de red generado.

Palabras clave— Robótica, ciberseguridad, bastionado, ROS

I. INTRODUCCIÓN

LA línea principal de investigación de nuestro grupo es el desarrollo de software para el control de sistemas autónomos. En concreto, durante los últimos años hemos estado desarrollando una plataforma de asistencia [4] para personas mayores y dependientes.

Como muchos otros grupos de investigación, nuestro grupo ¹ utiliza ROS (Robotic Operating System) como *framework* de desarrollo. Desde su popularización por Willow Garage en 2008 este entorno distribuido se ha convertido en el estándar *de facto* para el desarrollo de software para robots. Muchos sistemas autónomos utilizan ROS en diferentes entornos, plataformas robóticas para la investigación, manipuladores como Baxter de Rethink Robotics [1] o el recién anunciado soporte de los corta-cesped robóticos del fabricante Husavarna ².

Nuestro interés en la seguridad de estos sistemas ha venido impuesta al comenzar los experimentos en entornos reales. Al desplegar nuestros robots en domicilios particulares o centros asistenciales, los responsables jurídicos nos solicitaron información

sobre la seguridad de nuestros robots puesto que accederían a información protegida: imágenes, datos médicos, etc. de personas vulnerables.

De esta forma surgió esta investigación que forma parte de una línea de trabajo más amplio de securización de sistemas autónomos, y en la que prestamos atención a sistemas robóticos que utilizan ROS como *framework* de control.

A. ROS

La arquitectura de ROS se basa en un grafo de "nodos" encargados de los distintos procesos relevantes para el sistema (procesar sensores, calcular la localización, navegar, etc.) que funcionan de forma distribuida basada en subscripción asíncrona. Para que esto tenga lugar es necesario levantar lo que se denomina *roscore*. Su labor es la de levantar un grupo de nodos y programas base del sistema: el ROS Master, el servicio de parámetros de ROS (ROS Parameter Server) y el nodo de log (*roslout*).

Para el descubrimiento de los servicios ROS mantiene el nodo ROS Master, encargado de coordinar todo el sistema y que actúa como servidor de nombres. Cuando se crea un nuevo nodo este registra su servicios en el "ROS Master" y le enumera los "topics" que ofrecerán información de forma autónoma y asíncrona. Cuando otro nodo quiere usar el servicio disponible en un topic, solicita la información al "ROS Master" que le devuelve la IP y el puerto relativo al topic. En ese momento se establece la comunicación entre nodos, se dice entonces que este segundo nodo está "suscrito" al primero.

El problema de este diseño es que todas estas comunicaciones se producen sin aplicar ningún mecanismo de seguridad, excepto en aquellos casos en los que se levanta la red ROS en un entorno VPN. De esta forma, como refleja la figura 1, se podrían generar los siguientes problemas:

Interrupción del servicio : un software malicioso suprimiría los paquetes, haciendo que los subscriptores dejen de recibir información.

Modificación : el *malware* podría modificar los mensajes, produciendo información errónea.

Suplantación : un nodo malicioso podría suplantar a uno legítimo, fabricando mensajes completamente falsos.

Estos tres problemas se asocian a los tres principios básicos que debe asegurar la seguridad de los sistemas informáticos: Confidencialidad,

¹Instituto de Ciencias Aplicadas a la Ciberseguridad. Grupo de Robótica. Universidad de León. Web: <http://robotica.unileon.es>. Autor de contacto: vicente.matellan@unileon.es.

²<http://robotica.unileon.es>

³<http://www.ros.org/news/2016/05/husqvarna-research-platform>

Integridad y Disponibilidad. Atendiendo al US-CERT [7], este es el orden de criticidad genérico ante un ataque en entornos IT. Ahora bien, el mismo documento presenta un orden diferente de criticidad cuando se consideran sobre un sistema de control industrial: Disponibilidad, Integridad y Confidencialidad.

Planteado el problema, el resto del artículo está organizado como sigue: la siguiente sección revisa el estado del arte de la seguridad en robótica y en ROS en particular. La siguiente sección analiza diferentes alternativas para bastionado de ROS y su rendimiento. Posteriormente se analiza el funcionamiento de dos robots que utilizan ROS y sobre los que se aplica un sistema de cifrado para dos campos de los menajes utilizados en dos sensores reales. A continuación se describe una propuesta para el análisis en tiempo de ejecución de las condiciones de seguridad. Finalmente se presentan unas conclusiones y las líneas de trabajo futuro de nuestro grupo en este campo.

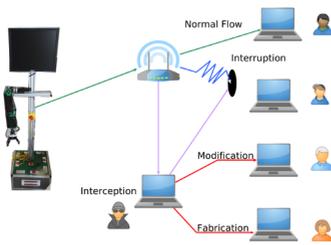


Fig. 1

TIPOS DE CIBER-ATAQUES EN UN ROBOT DE TELEASISTENCIA.

II. ESTADO DEL ARTE

Poniendo el foco en la seguridad en sistemas ciberfísicos desde un punto de vista robótica doméstica [12],[13] frente al industrial US-CERT [7] es necesario tener en cuenta dos aproximaciones. En primer lugar, la robótica asistencial doméstica es totalmente diferente a las soluciones robóticas industriales: entorno de despliegue, usuarios, rendimiento o resiliencia por enumerar alguna de ellas. Por esta razón, las cuestiones de seguridad que se venían aplicando a nivel industrial no se pueden extrapolar 1 a 1 entre entornos.

En el trabajo de Denning [10] se evalúan los puntos de fuga en plataformas de juguete como Rovio, Spykee o RoboSapien v2. Entre sus análisis, evaluaban si era posible conectarse directamente al flujo de datos de la cámara o que era posible desplazar diferentes objetos del entorno una vez situado en el mismo segmento de red.

En segundo lugar, el software de control del robot y los datos que gestiona. En el entorno doméstico,

si pensamos en un sistema aislado típico, un robot aspirador, el usuario no afrontará problemas de confidencialidad pero sí de disponibilidad, requiere que funcione cuando lo necesita. Ahora bien, si se trata de un sistema asistencial que trabaja información vital del usuario (niveles de azúcar, rutina en el hogar) el orden de importancia ante un ataque cambia y es preciso alterar el orden y las prioridades. La tabla 1 presenta nuestra propuesta para robots sociales y asistenciales junto con la propuesta para entornos críticos e industriales expuesta en [7].

Como se ha comentado, ROS es un software de control ampliamente extendido. El problema de la seguridad en ROS es un hecho conocido y se han descrito diferentes tipos de problemas. Investigadores como el Profesor Dr. Hartmut Pohl revisan ROS [11] atendiendo a los cuatro elementos básicos de la seguridad: disponibilidad, integridad, confidencialidad y autenticidad. Además, el Dr. Hartmut plantea también ataques simples sobre el nodo Master de ROS aprovechando las vulnerabilidades asociadas a XML-RPC. Dicho autor plantea el problema que nosotros abordamos en este trabajo, el cifrado de las comunicaciones más allá del uso de las VPN's o el cifrado por defecto de las WLANS.

Un ejemplo práctico de la seguridad en ROS lo plantea McClean en su trabajo [5], donde se describe el despliegue de un *honeypot* para probar la seguridad de robot. En dicho trabajo se utilizó un robot que disponía de una brújula y varios sensores de percepción del entorno y que se tele-operaba desde un nodo ROS escrito en Javascript y alojado en un servidor remoto. Los autores se centran en los problemas de utilizar comunicaciones en texto plano, el uso de puertos TCP sin proteger, o el almacenamiento de información sin cifrar. Este tipo de problemas son el reflejo en robótica de los problemas clásicos de seguridad en redes de ordenadores.

En un trabajo previo [3] presentamos un primer análisis detallado del rendimiento de las comunicaciones de ROS utilizando el sistema 3DES. En esa primera aproximación las pruebas se limitaron a mensajes de tipo `sensor_msgs/Image.msg` y de tipo `String`. El primer tipo era el utilizado por un sensor real (cámara USB) sobre la que cifrabamos el campo array de tipo `uint8` que contiene la información de la imagen. El segundo tipo se utilizó para evaluar el comportamiento global del sistema embebido del robot en condiciones de estrés. En este caso se evaluó el comportamiento del sistema de cifrado ante bloques de 256 KB, 512 KB, y 1024 KB.

III. BASTIONADO DE ROS

Atendiendo al problema de confidencialidad inherente a ROS al publicar en texto plano y teniendo en cuenta que el nivel criticidad es alta en robótica doméstica (robótica asistencial y robótica

TABLA I
 PERFILES DE SEGURIDAD ASOCIADOS A LA ROBÓTICA.

Perfil	Críticidad		
	Confidencialidad	Integridad	Disponibilidad
IT	Alta	Alta	Baja
Control Industrial	Baja	Media	Muy alta
Robótica Asistencial	Muy alta	Muy alta	Muy alta
Robótica Sociales	Muy alta	Media	Baja

social). Proponemos una solución de bastionado pasivo basado en el cifrado de las comunicaciones entre los distintos nodos del sistema de control, tanto *on-board* del robot, como en especial las comunicaciones con nodos externos. De esta forma, se plantea una solución de cifrado punto a punto. La figura 2 presenta a alto nivel el sistema propuesto.

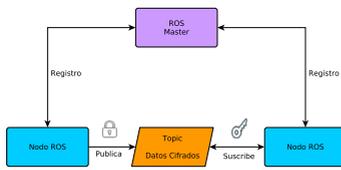


Fig. 2
 ARQUITECTURA DEL SISTEMA DE CIFRADO EN COMUNICACIONES.

De esta forma, antes de publicar la información adquirida por un sensor, realizamos un cifrado de la misma. Desde ese momento cualquier nodo que se introduzca en la red no verá la información publicada si no dispone de la llave de descifrado.

En esta primera propuesta de bastionado ROS se cifran parcialmente aquellos mensajes publicados en la red con información relevante de los usuarios: imágenes y láseres. El motivo de cifrar parcialmente es que el rendimiento empeora mucho al cifrar grandes bloques de datos, como se analizó en trabajos previos [3] al cifrar grandes bloques de datos. Como se planteó en dicho trabajo, se cifrará el bloque de datos que lleva la información del sensor, no el mensaje completo.

IV. PLANTEAMIENTO DE LA EXPERIMENTACIÓN

Uno de los problemas básicos del uso de cifrado en sistemas empotrados es la pérdida de rendimiento del sistema. En la siguiente sección realizamos un análisis del rendimiento, tanto desde el punto de vista de las comunicaciones, como desde el punto de vista del funcionamiento del sistema.

A. Escenario HW de las pruebas

El entorno que hemos diseñado para realizar las pruebas consiste en dos robots, cada uno con un sistema embarcado de control diferente y un

equipo externo. El primer robot monta un Intel(R) Atom(TM) CPU D525 @ 1.80GHz con 2GB de RAM. El segundo robot utiliza un equipo Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz con 16GB de RAM. En adelante, denominaremos al primer robot como **Robot1** o **R1** y al segundo como **Robot2** o **R2**. El equipo externo utiliza el mismo procesador Intel(R) Core(TM) i7-4790, este equipo se denominará en adelante **EE**.

Para simplificar las pruebas utilizamos dos sensores, un láser y una cámara, de los instalados en sendos robots. El modelo de láser utilizado fue en ambos casos un URG-04LX-UG01. La alimentación de dicho dispositivo se hace a través del mismo conector USB que le comunica con el sistema de control. La cámara utilizada fue del modelo QuickCam Pro 9000 Logitech, Inc. que se ha configurado para que ofrezca 15 frames por segundo (fps) con una resolución de 640x480 pixels.

Las conexiones entre equipos se han realizado por Ethernet a un switch Alcatel-Lucent OmniSwitch 6860E. En todos los casos se han utilizado cables Cat 6.

B. Escenario SW de las pruebas

Las conexiones entre el robot y equipo externo se realizan a través de ROS. Todos los equipos utilizaron GNU/Linux, concretamente Ubuntu 14.04.4 LTS aunque cada robot es completamente autónomo desde el punto de vista software. Desde el punto de vista de ROS eso quiere decir que cada robot arranca un **roscore**.

En cada robot se ha añadido un nodo ROS de cifrado que utiliza el algoritmo 3DES. El algoritmo 3DES (*Triple Data Encryption Standard* - TDES) es de tipo clave privada y cifra por bloque. Es la evolución del algoritmo DES (reconocido como estándar por el Instituto Nacional de Estándares y Tecnología NIST en 1974) pero con triple cifrado mediante claves de 64bits [8].

Hemos decidido utilizar este algoritmo ante alternativas [14], como AES o *blowfish* por tres razones. La primera por compatibilidad, puesto que ciertos dispositivos antiguos (lectores de tarjetas) con los que estamos trabajando utilizan DES. La segunda la disponibilidad de sistemas embebidos que realizan el cifrado en hardware, en lugar de a nivel software, para mejorar el rendimiento. La tercera se basa en estudiar "el peor caso", estudiamos el algoritmo más costoso en términos de coste computacional [9] para analizar la carga que se introduce sobre el sistema

respecto al comportamiento nominal o esperado.

El equipo externo realizará siempre el descifrado. Se han desarrollado dos nodos de descifrado, uno para cada sensor, ya que como se ha descrito los sensores no comparten el mismo tipo de mensaje. Los dos nodos ROS conocen la clave y son por tanto capaces de descifrar la información proporcionada por el láser o la webcam. El intercambio seguro de la clave queda fuera del objetivo de este estudio. Por el contrario, si se utiliza alguna de las herramientas de ROS como *rostopic echo* se presentará la información cifrada ya que la herramienta no dispone de la llave.

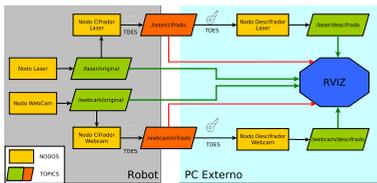


Fig. 3
 DESPLIEGUE DE NODOS ROS PLANTEADO EN LOS EXPERIMENTOS.

La figura 3 presenta el planteamiento general de los nodos utilizados en los experimentos. Se han utilizado dos nodos asociados a los sensores: el *Nodo Láser* que en el sistema de ROS es el *Hokuyo node* y el *Nodo Webcam* que corresponde en ROS al paquete *usb_cam*. Para facilitar el análisis se mantienen dos *topics* donde se puede consultar la información generada por el sensor sin cifrar (*laser/original* y *webcam/original*). Esto permite tener un patrón para comparar la información real de los sensores.

Se han incluido dos nodos de cifrado, uno para cada sensor, que están suscritos a los *topics* que publican la información sin cifrar (en texto plano). Cada uno de ellos utiliza un tipo de mensaje diferente.

Para el láser se utiliza el mensaje de tipo compuesto `sensor_msgs/LaserScan.msg`:

```
Header header
float32 angle_min
float32 angle_max
float32 angle_increment

float32 time_increment
float32 scan_time

float32 range_min
float32 range_max

float32[] ranges #array de datos del haz
float32[] intensities
```

Este tipo de mensaje está compuesto por diez campos que definen diferentes características del

sensor y del haz láser.

A la hora de cifrar la información, el planteamiento propuesto en esta investigación cambiar únicamente aquel campo que contiene la información relativa al entorno y con el que se podría realizar un seguimiento del usuario. En este caso el campo afectado es el denominado *ranges* y sobre el se realizará el cifrado conveniente para evaluar el comportamiento general del sistema. El número de elementos medio observado en dicho array durante nuestros experimentos fue de 4853.

El segundo tipo de mensaje utilizado fue `sensor_msgs/Image.msg` y también es de tipo compuesto. La matriz de información de la imagen se envía en la variable *data* que es de tipo *uint8* e igual que en el caso anterior, será el único campo que se cifre durante los experimentos. El tamaño medio en este caso del array observado en nuestros experimentos fue 39.583 elementos.

```
std_msgs/Header header #custom msg
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step

uint8[] data #matriz de datos de la imagen
```

Tas el proceso de cifrado cada nodo publica en un *topic* que sigue manteniendo el tipo de mensaje ROS `/laser/cifrado` y `/webcam/cifrado`.

Finalmente se incluyen dos nodos de descifrado que realizan los pasos inversos a los trazados en los nodos cifradores. Dichos nodos publican la información descifrada en los *topics*: `/laser/descifrado` y `/webcam/descifrado`

Por último, la figura 3 muestra un nodo denominada *rviz*. Este nodo representa la herramienta de visualización 3D que incorpora el ROS. En la figura se pueden ver dos tipos de flechas que se conectan a dicho nodo: las flechas rojas corresponden a suscripciones a *topics* cifrados y las flechas verdes a *topics* no cifrados. De esta forma la información que viene cifrada no será visualizable en *rviz* debido a que la aplicación por defecto no tiene la clave de descifrado. Por el contrario, la información de los *topics* en verde se puede visualizar perfectamente en la aplicación *rviz*.

C. Métricas de Análisis

Los elementos analizados durante la experimentación fueron tres:

- A) Tiempo de cifrado y descifrado
- B) Tiempo de proceso de CPU
- C) *Throughput* o volumen de información generado por cada nodo.

En el primer caso medimos el tiempo necesario para realizar el proceso de cifrado y descifrado. Este parámetro es clave en sistemas de mensajes por el cuello de botella que se puede generar. En particular, medimos la llamada a la función que realiza el

proceso de cifrado.

En el segundo caso medimos el consumo de CPU, en concreto medimos el tiempo total que cada nodo está usando la CPU. De este modo podríamos valorar la repercusión que hay sobre el resto de procesos del sistema.

En el tercer caso, dado que es necesario evaluar el rendimiento de red, medimos el *throughput* del sistema, es decir, el número total de bytes/segundo que publica cada nodo.

V. RESULTADOS DE LA EXPERIMENTACIÓN

A. Tiempos de ejecución del cifrado y descifrado

A.1 Robot1 - Cifrado

El *Nodo Cifrador Láser* realizó 20.584 llamadas a la función de cifrado, obteniendo un tiempo mínimo 7,812 ms y 12,461 ms de tiempo máximo. La media fue 8,190 milisegundos, con una desviación estándar de 0,211 ms.

El *Nodo Cifrador Webcam* en el robot que utilizaba el Atom(TM) obtuvo 66,394 ms de tiempo mínimo para el cifrado y un tiempo máximo de 100,157 milisegundos. La media de este proceso fue de 77,448 ms sobre 18.988 mensajes, con una desviación estándar de 0,007 ms.

A.2 Robot2 - Cifrado

En este caso el *Nodo Cifrador Láser* desplegado sobre un Core(TM) i7 generó 11.875 llamadas a la función de cifrado obteniendo un tiempo medio de 1,871 ms y una desviación típica de 0,001 ms. El valor mínimo fue de 1.064 ms y el máximo fue de 14,874 ms.

El *Nodo Cifrador Webcam* sobre el robot 2, presentó un valor mínimo de 1,309 ms y una duración máxima en el proceso de cifrado de 26,909 ms. La media en este caso fue de 10,948 ms con unas desviación estándar de 0,004 ms.

A.3 Equipo Externo - Descifrado

El comportamiento de los descifrados es similar, sea cual sea el origen de los datos. La tabla II resume los resultados de los ocho casos contemplados.

TABLA II
 RENDIMIENTO DE LA LLAMADA A LA FUNCIÓN DE DESCIFRADO.

LOS CASOS DEFINIDOS SON: 1) DESCIFRADO LÁSER EN EE CON ORIGEN R2; 2) DESCIFRADO IMÁGENES EN EE CON ORIGEN R2; 3) DESCIFRADO LÁSER EN EE CON ORIGEN R1; 4) DESCIFRADO IMÁGENES EN EE CON ORIGEN R1.
 (UNIDADES: MILLISEGUNDOS)

	1.Láser(R2-EE)	2.Webcam(R2-EE)	3.Láser(R1-EE)	4.Webcam(R1-EE)
Media	1,714	14,390	1,269	12,940
Desv. est.	0,089	2,642	0,259	3,5670
Mín	1,117	11,739	1,198	10,856
Máx	6,803	50,829	6,174	49,872

El resumen de todos los datos analizados se presenta en forma de box-plot en la figura 4

B. Tiempos de proceso de CPU

Para evaluar el tiempo real consumido de CPU por cada nodo se utilizó el comando *time* disponible en los sistemas Unix. Dicho comando ofrece el tiempo de ejecución de la aplicación (REAL), el tiempo de usuario (USER) y el tiempo del sistema (SYS). El tiempo total de CPU utilizado es la suma de los tiempos de USER y de SYS.

La tabla III muestra los resultados de la evaluación del consumo de CPU de los nodos utilizados (sin contar la aplicación *rviz*) para el experimento con el láser. Del mismo modo, la tabla IV muestra los tiempos asociados al experimento con la cámara.

TABLA III
 CONSUMO DE CPU DE LOS NODOS. CASO REVISADO:
 LÁSER.(UNIDAD:MINUTOS)

Robot 1	Nodo Láser (R1)	Nodo Cifrador Láser (R1)	Nodo Descifrador Láser (EE)
real	35.20095	31.732666667	31.451
user	0.21386666667	19.42433333333	2.08856666667
sys	0.13026666667	1.0046	0.09186666667
Robot 2	Nodo Láser (R2)	Nodo Cifrador Láser (R2)	Nodo Descifrador Láser (EE)
real	19.96713333333	19.83558333333	19.42428333333
user	0.0472	1.45013333333	1.242
sys	0.0424	0.05486666667	0.04793333333

Las duraciones de los experimentos no son fijas y superan en casi todos los casos los 30 minutos. La razón es que en la competición RoboCup@home hay una prueba denominada **Enhanced Endurance General Purpose Service Robot** en la que se busca tener el robot realizando diferentes tareas entre 30 y 45 minutos. La única prueba que no se realizó en este intervalo fue la del robot 2 con el sensor láser en la que estuvo casi 20 minutos, que corresponde a una prueba previa a utilizar el patrón RoboCup.

TABLA IV
 CONSUMO DE CPU DE LOS NODOS. CASO REVISADO:
 WEBCAM.(UNIDAD:MINUTOS)

Robot 1	Webcam (R1)	Nodo Cifrador Webcam(R1)	Nodo Descifrador Webcam(EE)
real	38.30766666667	36.4949666667	33.84076666667
user	14.01266666667	33.2512666667	4.67833333333
sys	0.73626666667	1.04213333333	0.15186666667
Robot 2	Webcam (R2)	Nodo Cifrador Webcam(R2)	Nodo Descifrador Webcam(EE)
real	48.63188333333	47.73328333333	45.09141666667
user	2.61266666667	15.65473333333	11.52933333333
sys	0.15093333333	0.38966666667	0.3404

C. Throughput

En este caso, el *Throughput* mide el flujo de datos asociado a cada nodo que publica en la red. Se han evaluado los dos flujos entre los tres nodos que corren en cada caso experimento: *Nodo Láser*, *Nodo Láser Cifrador* y *Nodo Láser Descifrador* y sus homólogos para la cámara.

En estas circunstancias los valores de *Throughput* para el láser del robot 1 son:

1. *Nodo Láser* → *Nodo Cifrador Láser*: 11 mensajes por segundo de media con un tráfico de 23199 bytes.
2. *Nodo Cifrador Láser* → *Nodo Descifrador Láser*: 28.3874 bytes por segundo con un total de 10 mensajes por segundo de media

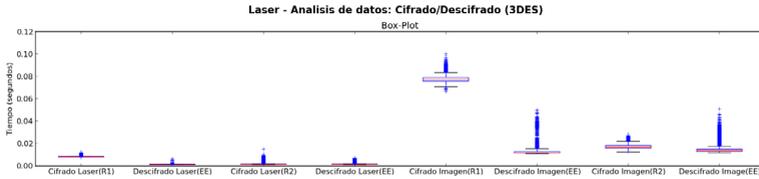


Fig. 4

BOX-PLOT GENERADO A PARTIR DE LOS DATOS OBTENIDOS DE NUESTROS PROCESOS EXPERIMENTALES.

Finalmente el flujo completo para el sensor del láser para el robot 2 mostró los siguientes datos:

1. *Nodo Láser* → *Nodo Cifrador Láser*: 11 mensajes por segundo de media con un tráfico de 21090 bytes.
2. *Nodo Cifrador Láser* → *Nodo Descifrador Láser*: 298722 bytes por segundo en un total de 10 mensajes de media por segundo.

En el caso asociado a la cámara, el flujo varía de acuerdo al formato utilizado. El *Nodo Webcam* (paquete *usb_cam* de ROS) permite dos métodos de trabajo *raw* y *Compressed*. El utilizado en el experimento fue el *Compressed*.

Si comparamos ambos en condiciones ideales se observará que la diferencia entre ambos es amplia. Para el caso *raw* tenemos un flujo de 14.285.394 bytes por segundo de media. En el segundo caso (*Compressed*) los valores arrojan un flujo de 22.438,67 bytes de media por mensaje enviado en el robot 1 y un flujo de 19.498,62 bytes de media por mensaje enviado en el robot 2.

El *Throughput* para las imágenes en el robot 1 fue:

1. *Nodo Webcam* → *Nodo Cifrador Webcam*: 8 mensajes por segundo de media con un tráfico de 425.348 bytes.
2. *Nodo Cifrador Webcam* → *Nodo Descifrador Webcam*: 431.008 bytes por segundo en un total de 8 mensajes que son enviados por segundo de media.

Los datos obtenidos en el caso del robot 2 son:

1. *Nodo Webcam* → *Nodo Cifrador Webcam*: 15 mensajes por segundo de media con un tráfico de 376.704 bytes.
2. *Nodo Cifrador Webcam* → *Nodo Descifrador Webcam*: 638.080 bytes por segundo en un total de 15 mensajes por segundo de media.

VI. DISCUSIÓN SOBRE EL RENDIMIENTO

De los resultados obtenidos anteriormente se pueden extraer las conclusiones asociadas a dos conceptos: el rendimiento del sistema y la funcionalidad del sistema.

A. Rendimiento de los sistemas

La figura 4 resume todos los casos analizados distinguiendo entre el robot 1 y el robot 2. Como era de esperar, el tiempo utilizado para el cifrado en el sistema R1 es más alto. En el caso del láser, es ligeramente mayor en media (7ms) y en el caso de la cámara es mucho mayor, siendo la diferencia de casi 60 ms.

Además del incremento en los tiempos de cifrado, es significativo el tiempo de CPU que necesita para realizar la tarea en el robot 1. Las figuras 5 y 6 muestran los tiempos (en minutos) de los experimentos con el láser y la cámara respectivamente.

Los experimentos con el láser muestran que el comportamiento del sistema es parecido en ambos robots. El número de mensajes servidos por el *Nodo Láser* es 10 msgs/segundo en todos los casos y no se produce ningún rechazo de mensajes en el nodo descifrador. A la vista de los datos presentados en la figura 5, resalta el aumento de uso de CPU que genera el *Nodo Cifrador Láser* sobre la plataforma Atom(TM), donde el cifrado requiere un 65% del tiempo (20.43 minutos) disponible de CPU.

A partir de los datos de *throughput*, se puede observar que la cantidad de datos transmitidos por el *Nodo Cifrador Láser* es mayor que el generado por *Nodo Láser*. El volumen de datos es casi el doble analizando las medias. Así el volcado del *Nodo Cifrador Láser*, es aproximadamente 14 veces mayor: 21.090 bytes frente a 298.722 bytes. Esto es debido a que es necesario acondicionar la información cifrada realizando un cambio de base para no modificar el tipo de mensaje.

Atendiendo al experimento con la cámara, observamos que el *Nodo Webcam* muestra que se requiere de un procesamiento extra en el robot 1 para conseguir los 15 fps que ofrece por defecto. Cuando el *Nodo Cifrador Webcam* corre en R1, el tiempo de CPU utilizado es superior al 90%. En el robot 2 ese consumo de CPU es inferior al 45%. En estas circunstancias, el robot 1 no cifra el 100% de los mensajes enviados (15 fps) y rechaza de media 7 mensajes, obteniendo un rendimiento de 8 fps. El elevado consumo de CPU influye en el comportamiento del resto de aplicaciones

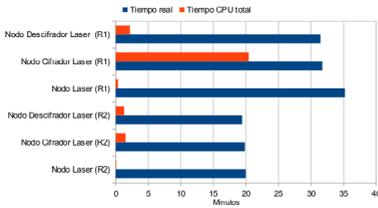


Fig. 5

EXP. LÁSER: CONSUMO DE CPU CORRIENDO UN SISTEMA DE CIFRADO EN ROBOT CON CORE(TM) I7 Y EN UN ROBOT CON UN ATOM(TM).

corriendo en el robot, pero no impide el correcto funcionamiento del mismo.

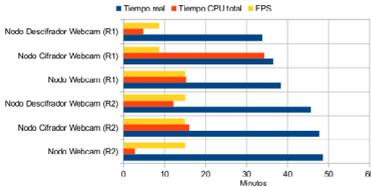


Fig. 6

EXP. WEBCAM: CONSUMO DE CPU CORRIENDO UN SISTEMA DE CIFRADO EN ROBOT CON CORE(TM) I7 Y EN UN ROBOT CON UN ATOM(TM).

B. Discusión sobre la Funcionalidad

En el caso de las imágenes el mayor problema funcional es el generado por la disminución en el **frame rate**, que baja de 15 fps a 8 fps.

En el caso del láser el problema no está ligado con el rendimiento, viene dado por el error sistemático inducido al cifrar la información en el mensaje.

La figura 7 presenta dos capturas del rviz con información del láser conectado al robot con Core(TM) i7 (superior) y al robot con un Atom(TM) (figura inferior). El haz sin cifrar se presenta en color verde en la figura, y el haz láser después de haber sido cifrado y descifrado se presenta en rojo.

En el primer caso se observa claramente que no coinciden al mismo tiempo ambos el haz de láser a partir de una distancia superior a 2,25 metros. Esto es debido a que la precisión es menor a esa distancia y el sistema de cifrado utilizado (PyCrypto) realiza un truncado de dos cifras, lo que hace que la coincidencia entre ambos no sea perfecta. En el caso del Atom (TM) se observa que el error se produce incluso a distancias menores de dos metros

y que es posible encontrar diferencias significativas (entre 2 y 10 cm). Si bien el error inferior a 5cm puede ser aceptable en entornos de interior a la hora de navegar o seguir personas, se hace necesario un análisis pormenorizado de este comportamiento ante situaciones de despliegue del robot en entornos reales.

VII. CONCLUSIONES Y TRABAJO FUTURO

ROS parece consolidarse como la plataforma estándar para el desarrollo de software para robots autónomos, pero es un entorno que no ha sido diseñado para la seguridad informática.

Resaltar que en nuestra propuesta no hemos cambiado ningún campo de los que conforman el mensaje ROS y que el resto de valores que contiene el mensaje se transmiten sin cifrar.

Tras el trabajo realizado en esta investigación hemos observado que cifrar en las comunicaciones resolvería problemas de confidencialidad. Esto supone contener ataques del tipo **man-in-the-middle** que puedan afectar al sistema. La desventaja de esta solución con respecto a las propuestas de VPN, es el empeoramiento del rendimiento de ciertos sensores e incremento de consumo de CPU. Esta situación limitará el rendimiento del resto de aplicaciones que son desplegadas en el robot.

Un segundo resultado observado en esta investigación, no por esperable menos significativo, es que la cantidad de datos vertida a la red es mucho mayor con cifrado que cuando se usa ROS sin cifrar.

Como trabajo futuro estamos trabajando en alternativas al cifrado para ofrecer un sistema de seguridad en ROS para permitir establecer políticas de autenticación a los nodos y a los equipos de la red.

AGRADECIMIENTOS

El presente trabajo ha sido financiado parcialmente por el Ministerio de Economía y Competitividad del Reino de España a través del proyecto DPI2013-40534-R y por el Instituto Nacional de Ciberseguridad (INCIBE) por la Adenda21 del convenio entre la Universidad de León e INCIBE.

REFERENCIAS

- [1] Conor Fitzgerald, Developing Baxter IEEE International Conference Technologies for Practical Robot Applications (TePRA), 2013
- [2] Huang, J., Erdogan, C., Zhang, Y., Moore, B., Luo, Q., Sundaresan, A., Rosu, G. ROSRV: Runtime verification for robots Lecture Notes in Computer Science, Vol. 8734, pp 247-254 DOI: 10.1007/978-3-319-11164-3_20
- [3] Francisco Javier Rodríguez Lera, Jesús Balsa, Fernando Casado, Camino Fernández, Francisco Martín Rico, and Vicente Matellán. *Cybersecurity in Autonomous Systems: Evaluating the performance of hardening ROS* Proceedings of the XVII Workshop of Physical Agents, pp. 47-53, 16-17 Junio 2016, Málaga (Spain).
- [4] Francisco Martín, José Mateos, Francisco Javier Rodríguez Lera, Pablo Bustos y Vicente Matellán. *A robotic platform for domestic applications*, XV Workshop of Physical Agents, 2014

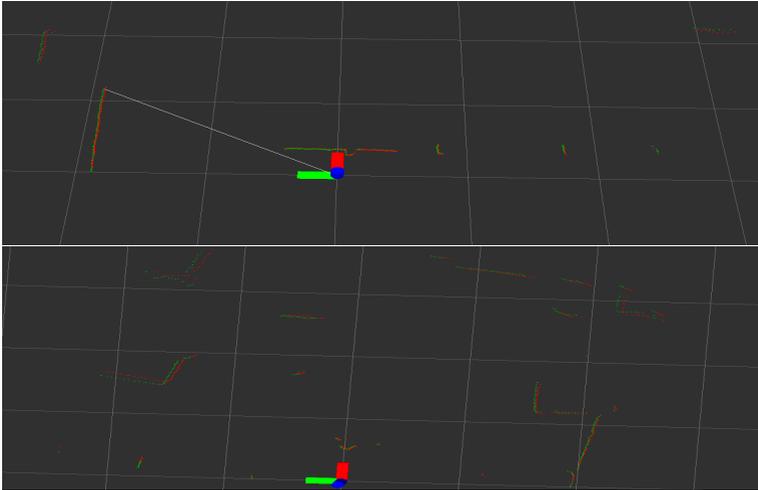


Fig. 7

VISUALIZACIÓN EN RVIZ DE UN SENSOR LÁSER (REPRESENTADO POR EL EJE DE COORDENADAS).

- [5] Jarrod McClean, Christopher Stull, Charles Farrar, David Mascareñas. *A preliminary cyber-physical security assessment of the Robot Operating System (ROS)* SPIE Defense, Security, and Sensing, Vol 8741, 2013
- [6] ROS Wiki <http://wiki.ros.org/Robots> Accedido por última vez 5 de mayo de 2016
- [7] CSSP, D. (2009). *Recommended Practice: Improving Industrial Control Systems Cybersecurity with Defense-In-Depth Strategies*. US-CERT Defense In Depth (Octubre 2009)
- [8] Merkle, R. C., and Hellman, M. E. (1981). *On the security of multiple encryption..* Communications of the ACM, 24(7), 465-467.
- [9] Elminaam, D. S. A., Abdual-Kader, H. M., Hadhoud, M. M. (2010). *Evaluating The Performance of Symmetric Encryption Algorithms*. IJ Network Security, 10(3), 216-222.
- [10] Denning, T., Matuszek, C., Koscher, K., Smith, J. R., & Kohno, T. (2009). *A spotlight on security and privacy risks with future household robots*. Proceedings of the 11th International Conference on Ubiquitous Computing - Ubicomp '09, 105. doi:10.1145/1620545.1620564
- [11] Hartmut Pohl (softScheck GmbH). (2016). *Robot Operating System (ROS): Safe & Insecure*. Automationspraxis, whitepaper. Disponible online [25/06/2016] en <http://www.automationspraxis.de/whitepaper>
- [12] Morante, S., Victores, J. G., & Balaguer, C. (2015). *Cryptobotics: Why robots need cyber safety*. Frontiers in Robotics and AI, 2(23), 1-4. doi:10.3389/frobt.2015.00023
- [13] Tadele, T. S. (2014, November 7). *Human-friendly robotic manipulators: safety and performance issues in controller design*. Tesis. Disponible en <http://dx.doi.org/10.3990/1.9789030537841>
- [14] Nadeem, A., & Javed, M. Y. (2005, August). *A performance comparison of data encryption algorithms*. In 2005 International Conference on Information and Communication Technologies (pp. 84-89).

Plataforma de Verificación para Circuitos Digitales basada en Dispositivos Reconfigurables

C. A. Torres Cerna,¹ J. J. Raygoza Panduro,² E. C. Becerra Álvarez,³ S. Ortega Cisneros⁴
y J. Rivera Domínguez⁵

Resumen— Las etapas de validación, verificación y pruebas consumen alrededor del 60% de los recursos en el proceso de producción de un circuito, por esto la importancia de desarrollar herramientas que ayuden a reducir el costo y acelerar el proceso de verificación y pruebas de circuitos. En este trabajo se desarrolló una plataforma hardware-software para generar vectores de datos que servirán para probar circuitos digitales. Esta herramienta consta de 2 bloques principales, una interfaz gráfica y un circuito generador implementado en dispositivos reconfigurables FPGAs. Se realizó la verificación de circuitos combinatoriales y con interfaz paralela UART utilizando la herramienta diseñada para validar su funcionamiento.

Palabras clave— FPGA, DUT, GUI, VLSI, ATE.

I. INTRODUCCIÓN

DESDE el comienzo de la era digital, la detección de errores y pruebas ha sido parte importante en el desarrollo de circuitos [1] [2], ya que para garantizar su correcto funcionamiento, un buen plan de pruebas y detección de errores es indispensable.

El tener un plan de verificación adecuado durante el proceso de desarrollo de un circuito digital resulta en la reducción de costos, debido a que un error encontrado en el circuito fabricado involucra, además del rediseño, la remanufactura del circuito, lo que podría representar retrasos en el tiempo de entrega y demás gastos generados.

Conforme la tecnología ha avanzado, la complejidad de los circuitos digitales ha aumentado. De acuerdo con la ley de Moore [3] el tamaño de los transistores se reduce aproximadamente a la mitad cada año. Esto resulta en una necesidad de mejorar constantemente la metodología de pruebas de los circuitos digitales, como la presentada por Wong [4] que propone una verificación jerárquica ideal para circuitos con gran escala de integración (VLSI), o como Teixeira [5] que propone una metodología de pruebas basada en la definición de una lista de fallos.

Para probar circuitos integrados se utilizan equipos especializados conocidos como Equipo de

Pruebas Automatizadas (ATE, por sus siglas en inglés, Automated Test Equipment). Los ATE son ampliamente utilizados en la industria de manufactura electrónica para probar componentes y sistemas electrónicos una vez fabricados.

Los sistemas ATE están diseñados para reducir el tiempo necesario para verificar que un Dispositivo en Prueba (DUT por sus siglas en inglés, Device Under Test) funciona correctamente o si tiene algún error antes de ser lanzado al mercado. El ATE actúa alimentando al DUT con una serie de patrones y analizando su respuesta para determinar si su funcionamiento es correcto o debe ser corregido [6].

El principal inconveniente con los ATE es su alto costo, ya que al ser sistemas muy complejos y principalmente enfocados a la industria, resulta poco viable que equipos pequeños de investigación o desarrolladores de hardware adquieran este tipo de herramientas. Una alternativa es el desarrollo de camas de pruebas utilizando hardware reconfigurable.

La capacidad de diseñar e implementar hardware en una plataforma reconfigurable permite la flexibilidad de desarrollar los diferentes instrumentos de prueba que se requieran en una herramienta de verificación. Esta flexibilidad permite el diseñar una plataforma de pruebas a medida, ideal para grupos de investigación pequeños que no tienen la capacidad de adquirir equipos de pruebas complejos y costosos. Esta ventaja se ha visto aprovechada desde el lanzamiento de las primeras FPGAs, como es el caso de Clarke, et al. [7] que desarrollaron una cama de pruebas para el estudio de técnicas de demodulación utilizando un arreglo ordenado de 10 chips XC3042 de Xilinx.

En la actualidad se siguen desarrollando herramientas de este tipo con dispositivos reconfigurables, como el trabajo de Vanitha, et. al. [8], que en el 2013 realizaron la implementación de un equipo automatizado de pruebas y generador de pruebas para circuitos digitales en FPGA. Y como S. Fransi et. al. que publicaron en 2010 el Diseño e implementación de un módulo IP basado en un ATE.

II. DISEÑO

El diseño de la herramienta propuesta se puede dividir en 2 bloques principales, una interfaz gráfica y el diseño del circuito generador, y un bloque auxiliar de interfaz de transmisión UART que dá la flexibilidad a la herramienta de trabajar con DUT que mane

¹Departamento de Electrónica y Computación, CUCEI, UDG, e-mail: agustin.torres@alumino.udg.mx

²Departamento de Electrónica y Computación, CUCEI, UDG, e-mail: juan.raygoza@cucei.udg.mx

³Departamento de Electrónica y Computación, CUCEI, UDG, e-mail: edwinbecerra@gmail.com

⁴CINVESTAV unidad Guadalajara, e-mail: susana.ortega@dl.cinvestav.mx

⁵CINVESTAV unidad Guadalajara, e-mail: riveraj@dl.cinvestav.mx

jen este protocolo. En la figura 1 se muestra el diagrama de flujo que el usuario debe seguir para configurar el circuito generador de señales utilizando la interfaz gráfica y un software sintetizador de código Verilog.

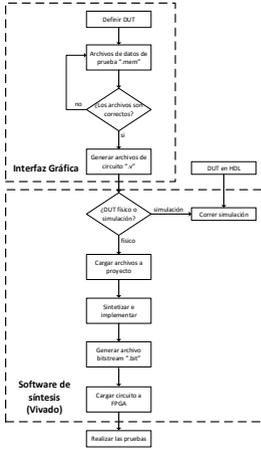


Fig. 1. Diagrama de flujo de la herramienta.

A. Interfaz Gráfica (GUI)

Para otorgar al usuario un ambiente sencillo e intuitivo a la hora de configurar el generador de señales de acuerdo a las necesidades de su DUT, se desarrolló una interfaz gráfica utilizando el ambiente de desarrollo Visual Studio versión 2015 v14.0.25123, y se programó en el lenguaje C#.

La interfaz gráfica consiste en una serie de ventanas de configuración, en donde el usuario define las características de su DUT, y los datos de entrada que lo alimentarán.

Para definir los datos de entrada del DUT, el usuario debe crear un archivo de texto plano con extensión ".mem" con los valores representados en raíz hexadecimal. Esto se puede hacer en cualquier editor de texto, procurando guardar el archivo con la extensión ".mem".

Cada línea del archivo corresponde a un vector de datos a generar, por lo que se debe considerar que la cantidad de bits necesarios para representar los valores, deben corresponder a la cantidad de entradas del DUT, de lo contrario, los bits sobrantes se ignorarán. Esto es, si en una línea del archivo se tiene el valor **FF**, pero solo se configuraron seis entradas para el DUT, el vector de datos generado para esa línea será **3F**, equivalente a **111111** en binario. Igualmente si no se declaran valores lo suficientemente grandes para cubrir todos los bits de

entrada del DUT, serán sustituidos por ceros.

Como resultado final la interfaz gráfica genera los archivos en lenguaje verilog necesarios para configurar el generador. El usuario debe sintetizarlos, generar el bitstream y programar el FPGA manualmente para utilizar la cama de pruebas con su circuito. Esto da la oportunidad al usuario de modificar alguno de los archivos si es que así lo requieren sus pruebas. También otorga la flexibilidad de sintetizar e implementar el circuito con el software que el usuario prefiera, así como utilizar la tarjeta de desarrollo y el FPGA que tenga a su disposición.

En la pantalla principal del modo manual que se muestra en la figura 2 se pueden observar 3 botones: *DUT*, *Cargar Datos* y *Generar Código*, y un cuadro de texto donde se muestran los mensajes del programa. Al presionar el botón *DUT* se abre una ventana nueva como la que se muestra en la figura 3 donde se pueden configurar las características del circuito que se quiere probar: tipo de circuito, número de entradas, número de salidas y un divisor de reloj.

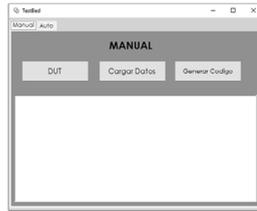


Fig. 2. Ventana principal de la GUI en modo de configuración manual.

Para seleccionar el tipo de DUT se puede escoger entre las opciones genérico y UART. Un circuito genérico es aquel que no tiene un protocolo de comunicación específico y solo recibe los datos en forma directa, como es el caso de los circuitos combinatoriales, si se selecciona esta opción, el generador de señales se configura para solamente alimentar al DUT directamente con los datos que se guardaron en el archivo de datos de prueba. Si el DUT tiene una interfaz de comunicación UART se debe elegir esta opción, en este caso el generador de señales lee los datos de prueba y alimenta al circuito con estos datos usando el protocolo UART. Es importante considerar que en modo UART el generador crea paquetes de ocho bits por cada transmisión, por lo que se deben definir datos de ocho bits en cada línea de los archivo de datos por cada entrada o salida especificada.

Para guardar la configuración se debe presionar el botón *Aceptar*, lo que guardará esta información para después generar los archivos correspondientes. Si se requiere se puede cancelar la configuración y regresar a la ventana principal presionando el botón *Cancelar*.



Fig. 3. Ventana DUT donde se configura el circuito a probar.

En la pantalla principal, al hacer clic en el botón *Cargar Datos* se abre una ventana nueva, como la que se muestra en la figura 4, en la que se pueden seleccionar los archivos de datos de entrada y salida de la prueba. En esta ventana se observan dos botones llamados *Examinar*, uno para seleccionar el archivo de datos de entrada y otro para el archivo de datos de salida. Este segundo botón se agregó pensando en extender las características de la herramienta para en un futuro verificar también la salida del DUT. Al presionar uno de estos botones se abre una ventana para buscar y seleccionar el archivo deseado. Una vez seleccionados ambos archivos y al presionar el botón *Aceptar*, se guardará esta información y cerrará esta ventana para regresar a la ventana principal.

Para continuar con la configuración del generador se deben seleccionar ambos archivos de vectores de datos, de entrada y de salida, los cuales deben ser archivos diferentes y contener la extensión ".mem", de lo contrario el sistema mostrará un mensaje de error y no permitirá continuar. Si se desea cancelar la selección de archivos se debe presionar el botón *Cancelar* para cerrar la ventana sin guardar registro de los archivos seleccionados y volver a la ventana principal.

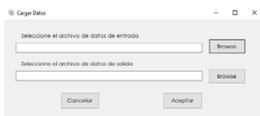


Fig. 4. Ventana Cargar Datos donde se seleccionan los archivos de datos de la prueba.

El último botón que se puede observar en la pantalla principal es *Generar Código*. Al presionarlo el sistema comienza a generar los archivos necesarios para configurar la cama de pruebas. Si se presiona el botón de *Generar Código* sin antes haber terminado la configuración o sin seleccionar los archivos de vectores de datos, el sistema genera un mensaje de error y no permite continuar.

Los archivos verilog genéricos con la configuración

del circuito generador de señales se crearon con características parametrizadas, lo que permite una fácil adaptación a las diferentes configuraciones que pueden ser requeridas. Estos se guardan en una carpeta segura a la que el sistema está direccionado.

Al presionar el botón *Generar Código* el programa selecciona los archivos verilog genéricos correspondientes y los copia en otra carpeta, modificando sus parámetros para obtener así las características que el usuario configuró utilizando la interfaz gráfica para obtener el circuito generador requerido.

B. Generador de señales genérico

En la figura 5 se muestra el diagrama a bloques del generador de señales genérico, que consiste en un bloque manejador de reloj *clk_mgr*, una memoria de datos llamada *mem* y un bloque llamado *generador*, donde el valor *n* corresponde a la cantidad de entradas del DUT definidas por el usuario, y el valor *m* es la cantidad de bits necesarios para direccionar la memoria, siendo ésta definida por la cantidad de datos en el archivo de entrada. Las entradas y salidas del generador de señales se pueden ver en la tabla I.

TABLA I
 DESCRIPCIÓN DE ENTRADAS Y SALIDAS DEL MÓDULO
 GENERADOR DE SEÑALES GENÉRICO

Señal	E/S	Descripción
en	Entrada	Señal de habilitación
rst	Entrada	Señal de reset del generador
clk.div	Entrada	Valor divisor del reloj
clk	Entrada	Reloj principal del generador
conf	Entrada	Bus de conf. del generador
enviando	Salida	En alto cuando se envían datos
cod	Salida	Código del dato de salida (contador)
data.out	Salida	Dato de salida generado

El presente trabajo se implementó en un dispositivo reconfigurable FPGA de la línea Zynq 7000, modelo XC7Z020CLG484-1, el cual contiene una serie de bloques de memorias RAM de propósito general llamadas BRAM que pueden ser configuradas como memorias RAM o ROM. Estos bloques de memoria RAM ofrecen el servicio de almacenamiento de grandes cantidades de datos con grandes velocidades de lectura y escritura [10]. El bloque *mem* es el módulo que se encarga de crear la memoria BRAM donde se almacenan los datos de prueba creados por el usuario. Los parámetros de dimensiones de la memoria se definen en la GUI; el ancho es la cantidad de entradas que el usuario defina que tendrá su circuito, y la profundidad es definida por la can-

tividad de datos que el usuario agregue al archivo de prueba. La carga de información a la memoria se hace por medio del bitstream, esto es, directamente desde el software de síntesis al momento de cargar la configuración del FPGA.

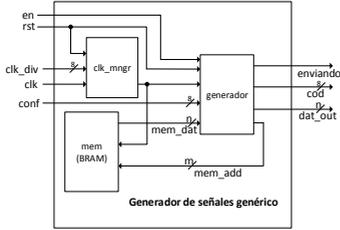


Fig. 5. Diagrama esquemático del Generador de señales.

El módulo *clk_mgr* es el encargado de dividir la frecuencia del reloj de entrada al valor que el usuario indique. Tiene como entradas el reloj *clk*, el valor del divisor de reloj *clk_div* que es un bus de 8 bits, y una señal de reset *rst*, y como salida el reloj generado *clk_out*.

El manejador de reloj además de tener un módulo divisor de reloj *clk_div*, también contiene un comparador y un multiplexor que se encargan de reordenar el divisor y arrojar como salida el valor de entrada sin dividir, esto en el caso de que el valor del divisor sea igual a uno o cero, lo que permite al generador cambiar la frecuencia del reloj durante su operación.

El encargado de direccionar la memoria para extraer los datos de prueba y generar la salida del circuito es el módulo *generador*. Éste consiste básicamente en una máquina de estados finitos (FSM por sus siglas en inglés) sincronizada con el reloj de entrada, lo que garantiza una salida de datos a la frecuencia deseada. En la figura 6 se muestra la máquina de estados finitos, que cuenta con 5 estados: *idle*, *begin*, *send_01*, *send_02* y *end*, descritos a continuación.

El primer estado es el de reposo *idle*, en donde se mantiene hasta que se habilita la señal de *en*, en este estado todas las señales de salida del módulo son cero. Una vez que se recibe la señal de habilitación, pasa al estado siguiente *begin*.

En el estado *begin* se inicializan los valores de control del generador: se tienen dos direcciones a memoria auxiliares llamadas *mem_add01* y *mem_add02* a las que se asignan valores de uno y cero respectivamente, un registro que almacena la cantidad de datos a enviar llamado *data_size* al que se asigna el valor de la señal de entrada *conf* y un contador de datos enviados llamado *data_count* que se inicializa en cero y es asignado a la salida *cod*. Si en el siguiente flanco positivo del reloj se mantiene la señal de habilitación en alto se pasa al siguiente estado que es

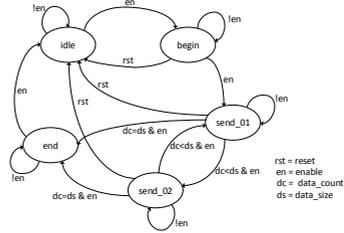


Fig. 6. Máquina de estados finitos del módulo generador.

send_01, sino se mantiene en este estado hasta que se vuelva a habilitar la señal o se active la señal de reset en cuyo caso regresará al estado *idle*.

Los estados *send_01* y *send_02* se encargan de extraer los datos de la memoria y de enviarlos por el puerto *dat_out*. Esto para lograr un correcto direccionamiento a la hora de acceder a la memoria.

En el estado *send_01*, si la señal *en* está en alto y el reset en bajo, se verifica si el contador de datos enviados *data_count* es menor a la cantidad de datos a enviar *data_size*. Si se cumple la condición, lee los datos de la memoria usando la dirección auxiliar *mem_add02* y los coloca en la salida *dat_out*, levanta la señal *enviando*, incrementa en 2 la dirección auxiliar *mem_01* y en uno el contador de datos enviados, y se asigna *send_02* como estado siguiente. Si la condición no se cumple significa que ya se leyeron y enviaron todos los datos de la memoria, la salidas *dat_out* y *enviando* se igualan a cero y se asigna *end* como siguiente estado. En caso de que la señal de habilitación *en* esté en bajo, se mantiene en este estado y no se hacen cambios a ninguna señal. Si se activa el reset se regresa al estado *idle*.

El estado *send_02* es similar al estado anterior *send_01*. La única diferencia es que la dirección auxiliar usada para leer la memoria es *mem_add01* y la dirección auxiliar incrementada es *mem_add02*.

En el último estado *end* solamente se espera al siguiente flanco positivo del reloj para volver al estado inicial de reposo *idle*. Si bien en este estado no se hace ningún procesamiento, algunos protocolos de comunicación requieren un estado final extra para agregar información a la trama de datos enviados.

C. Interfaz UART

El módulo transmisor UART consiste en un divisor de reloj que genera la frecuencia necesaria para transmitir los datos del vector de pruebas, la cual puede ser configurada por el usuario, y una FSM que se muestra en la figura 7 que consta de tres estados: *IDLE*, *SEND* y *DELAY*.

El primer estado es el de reposo, llamado *IDLE*, en el cual el transmisor está en espera de la señal *transmit* que le indica que debe empezar a transmitir.

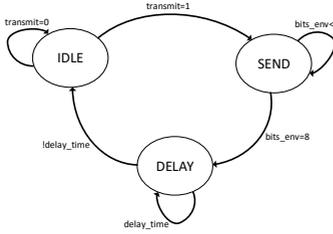


Fig. 7. Máquina de estados finitos del transmisor UART.

Una vez que ésta señal se reciba en alto almacena el valor a enviar que se recibe en el bus de entrada tx_data en un registro temporal llamado $tx_dataTemp$ y pasa al siguiente estado que es *SEND*.

En el estado *SEND* el transmisor envía los datos almacenados en el registro $tx_dataTemp$ comenzando por el bit menos significativo. Se mantiene en este estado hasta que se hayan enviado los 8 bits del registro, entonces asigna a la línea de transmisión un valor en alto y pasa al siguiente estado *DELAY*.

El último estado de la FSM llamado *DELAY* es para esperar el tiempo correspondiente a un bit para transmitir un valor en alto que indica el final de la transmisión. Una vez transcurrido este tiempo se regresa al estado *IDLE*.

Este módulo se conectó a la salida del generador de señales genérico, de tal forma que sea alimentado con el vector de datos definido por el usuario. De esta forma el transmisor UART puede utilizar estos datos para generar la señal serial que alimentará al DUT, como se muestra en la figura 8.

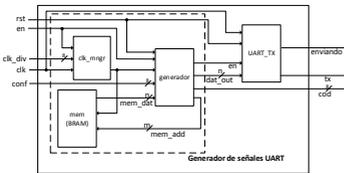


Fig. 8. Diagrama esquemático del generador de señales con transmisor UART.

III. RESULTADOS

Los diseños se crearon e implementaron para su validación utilizando el software Xilinx Vivado v.2014.5 en una tarjeta de desarrollo modelo ZC702, que contiene una FPGA de la serie Zynq 7000, modelo XC7Z020CLG484-1.

A. Validación

Para validar el circuito generador de señales genérico se verificó un circuito con algoritmo de multiplicación Vedic [11] que consta de 2 entradas de ocho bits y una salida de 16 bits, el cual se conectó al generador de señales como se muestra en la figura 9.

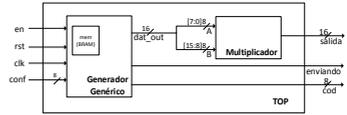


Fig. 9. Diagrama esquemático del generador de señales genérico alimentando un multiplicador comportamental.

El archivo de datos de entrada creado, que se muestra en la figura 10, consta de 31 datos de 16 bits, en los que los ocho bits menos significativos corresponden a la entrada *A* del multiplicador, y los ocho bits más significativos a la entrada *B*.

init_data.mem	x
1	1234
2	5678
3	9012
4	A12A
5	AB45
:	:
28	0C0A
29	0CAF
30	E342
31	CASA

Fig. 10. Archivo de datos generado para la simulación del generador genérico alimentando al multiplicador.

Los resultados de la simulación se muestran en la figura 13. Se puede observar en la imagen que a partir de que se levanta la señal de enviando el circuito multiplicador comienza a generar el resultado después de un pequeño retardo que corresponde a la latencia del circuito multiplicador.

Para validar el circuito generador con interfaz UART se verificó un circuito procesador de imágenes [12] que cuenta con 3 entradas, *rst*, *clk* y *rx* y una salida *tx*, y se conectó a la salida del generador de señales como se muestra en la figura 11.

Para verificar el circuito se creó un archivo con 12 datos de 8 bits que corresponde a un par de imágenes con las cuales el procesador realizará la operación de convolución como se muestra en la figura 12.

Se corrió la simulación post-implementación en la herramienta de software y se obtuvo el resultado que se presenta en la figura 14, en donde se puede apreciar que primero se envían los 12 datos del generador al procesador y después el procesador genera la señal serial correspondiente al resultado en la línea

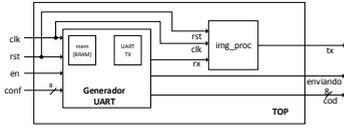


Fig. 11. Diagrama esquemático del generador de señales con interfaz UART conectado al procesador de imágenes.

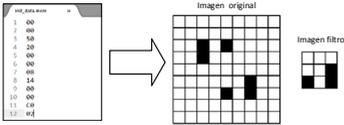


Fig. 12. Archivo de datos creado para la simulación de tiempos post-implementation del generador con interfaz UART conectado al procesador de imágenes.

ImgProcTr.

Haciendo un acercamiento a la señal *ImgProcTx* se observan los valores de salida generados, éstos se traducieron en la imagen correspondiente y el resultado fue la imagen de salida que corresponde a la convolución de las imágenes de entrada como se muestra en la figura 15.

B. Ahorro en tiempo

Para validar que la herramienta desarrollada efectivamente permite el ahorro de tiempo en el proceso de verificación, comparado con el diseño e implementación de un circuito específico para realizar una verificación manual del DUT, se realizó la comparación de tiempos en ambos casos para los circuitos verificados.

Para el multiplicador, se diseñó un circuito a la medida para generar las señales y alimentar al multiplicador. El proceso de diseño, implementación, y verificación le tomó 10 horas a un ingeniero capacitado en diseño de hardware. Generar con la herramienta, un circuito que cumpliera con estas características, fue solo cuestión de minutos y no requirió de un experto en diseño de hardware. Enseguida, teniendo los archivos verilog del circuito generador, solo se necesitó crear un sencillo testbench y realizar la simulación, en lo que se invirtieron al rededor de 30 minutos. En total, el proceso de verificación del circuito multiplicador con la herramienta, duró menos de una hora.

En el caso del procesador de imágenes, de mismo modo se diseñó un circuito que generara la señal que alimentaría al DUT, que en este caso era una señal serial con protocolo UART. El proceso de diseño y verificación tomó 14 horas. Con la herramienta, la generación del circuito y la verificación se hicieron en menos de una hora, al igual que con el multiplicador.

C. Ocupación de recursos

En la tabla II se muestra la ocupación de los recursos de la FPGA para cada uno de los circuitos implementados.

TABLA II
 OCUPACIÓN DE RECURSOS DE LOS CIRCUITOS IMPLEMENTADOS.

Elemento	Generador Genérico	%	Con UART TX	%
LUT	62/53200	0.12%	100/53200	0.19%
FF	35/106400	0.03%	66/106400	0.06%
BRAM	0.5/140	0.36%	.50/140	0.36%
IO	27/200	13.5%	24/200	12%

Se realizó un análisis para comparar la ocupación del circuito con diferentes configuraciones, variando el la capacidad de la memoria y la cantidad de entradas al DUT. Los resultados se muestran en la tabla III, en el primer columna se muestran las configuraciones con la forma *tipo de circuito entradas al DUT x localidades de memoria*.

TABLA III
 RESULTADOS DE OCUPACIÓN DE RECURSOS DEL CIRCUITO GENERADOR CON DIFERENTES CONFIGURACIONES.

	LUT	FF	BRAM	IO
	53200	106400	140	200
Gen 4x16	43	24	0.5	19
UART 8x16	78	54	0.5	16
Gen 4x256	74	35	0.5	31
UART 8x256	100	66	0.5	24
Gen 10x1024	88	41	0.5	37
UART 8x1024	113	72	0.5	28
Gen 12x4096	97	48	1.5	43
UART 8x4096	120	79	1	32
Gen 16x65K	135	60	32	55
UART 8x65K	139	91	16	40

IV. CONCLUSIONES

En este trabajo se describe el diseño de una plataforma de verificación implementada en un dispositivo reconfigurable FPGA. Una de sus principales características es su reducido consumo de recursos, debido a que al ser un circuito pequeño no requiere demasiados elementos lógicos, como se muestra en la tabla II, y que al ser escalada para DUT mas grandes, el nivel de ocupación se no se incrementa de forma crítica, como se observa en la tabla III, lo que permite que pueda ser implementado en cualquier FPGA de gama media-baja. Si bien en este trabajo se utilizó una FPGA que cuenta con elementos de memoria BRAM, estos pueden ser sustituidos por registros comunes si no se tiene un dispositivo que contenga este tipo de memorias. Lo que la hace una herramienta de verificación accesible para gru-

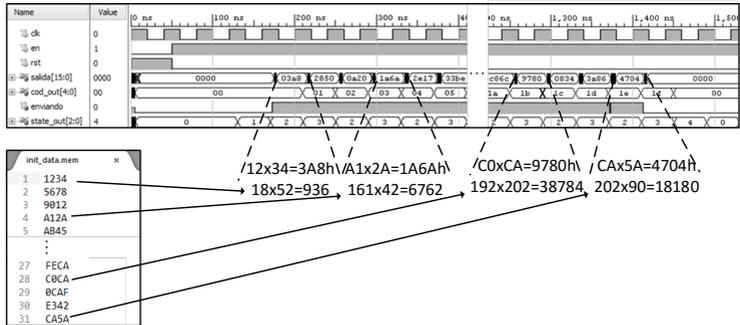


Fig. 13. Resultados de la simulación post-implantación del generador de señales genérico conectado al multiplicador.



Fig. 14. Resultados de la simulación post-implantación del generador con interfaz UART conectado al procesador de imágenes.

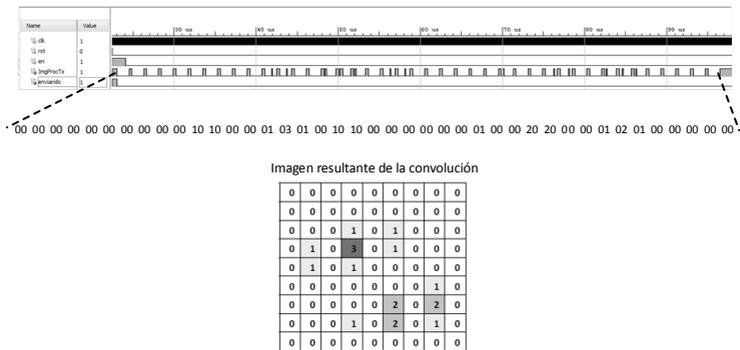


Fig. 15. Acercamiento a la simulación del resultado de la operación realizada por el procesador de imágenes.

pos de desarrollo que no cuentan con la capacidad económica para adquirir equipos especializados.

Otra de las características de la herramienta es la incorporación de una interfaz gráfica que facilita una

rápida configuración, lo que permite que el usuario no tenga que ser un experto en diseño de circuitos para utilizarla, logrando así reducir el tiempo de verificación de un circuito digital, comparado con la creación de circuitos de verificación a medida.

Los resultados obtenidos al validar esta herramienta de pruebas demuestran su validez y utilidad. El realizar este tipo de pruebas a un circuito de forma manual puede representar horas de trabajo para un ingeniero de pruebas, mientras que usando esta herramienta, en tan solo unos minutos se pueden lograr los mismos resultados sin la necesidad de tener un ingeniero experto en diseño. Esto demuestra que la herramienta propuesta satisface las necesidades de verificación de circuitos, de una forma sencilla mediante la interfaz gráfica que permite al usuario el configurar la herramienta y crear los vectores de datos con los que será alimentado su circuito.

(CCE), 2015 12th International Conference on, pages 1-5, Oct 2015.

AGRADECIMIENTOS

El presente trabajo ha sido apoyado por el Consejo Nacional de Ciencia y Tecnología (CONACYT)

REFERENCIAS

- [1] D. J. Wheeler and J. E. Robertson. *Diagnostic programs for the iliacc*. Proceedings of the IRE, 41(10):1320-1325, Oct 1953.
- [2] R. W. Hamming. *Error detecting and error correcting codes*. The Bell System Technical Journal, 29(2):147-160, April 1950.
- [3] G. E. Moore. *Progress in digital integrated electronics technical literature, copyright 1975 ieee. reprinted, with permission. technical digest. international electron devices meeting, ieee, 1975, pp. 11-13.*. IEEE Solid-State Circuits Society Newsletter, 20(3):36-37, Sept 2006.
- [4] Yiwan Wong. *Hierarchical circuit verification*. In Design Automation, 1985. 22nd Conference on, pages 695-701, June 1985.
- [5] J. P. Teixeira, C. F. B. Almeida, J. A. Gracio, P. A. Bieudo, A. L. Oliveira, and N. Rua. *Bottom-up testing methodology for vlsi*. In Custom Integrated Circuits Conference, 1988., Proceedings of the IEEE 1988, pages 16.6/1-16.6/4, May 1988.
- [6] Michawel L. Bushnell and Vishuandi D. Agrawal. *Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits*. Kluwer Academic Publishers, 2002.
- [7] K. C. Clarke, D. J. Cipolle, and R. R. Rhodes. *Development of a real-time testbed for studying demodulation techniques in a jamming environment*. In Military Communications Conference, 1991. MILCOM, Conference Record, Military Communications in a Changing World., IEEE, pages 610?616 vol.2, Nov 1991.
- [8] K. Vanitha and C. A. Sathiyamoorthy. *Implementation of an integrated FPGA based automatic test equipment and test generation for digital circuits*. in Information Communication and Embedded Systems (ICICES), 2013 International Conference on, pp. 741?746, Feb 2013.
- [9] S. Fransé, G. L. Farré, L. G. Deiros, and S. B. Manich. *Design and implementation of automatic test equipment ip module*. in Test Symposium (ETS), 2010 15th IEEE European, pp. 244?244, May 2010.
- [10] Xilinx. *Vivado Design Suite 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide*. September 2015.
- [11] U. C. S. Pavan Kumar and A. Saiprasad Goud and A. Radhika. *FPGA implementation of high speed 8-bit Vedic multiplier using barrel shifter*. Energy Efficient Technologies for Sustainability (ICEETS), 2013 International Conference on, Apr 2013.
- [12] S. Ortega Cisneros, J. Rivera D., P. Moreno Villalobos, C. A. Torres C., H. Hernández-Hector, and J. J. Raygoza P. *An image processor for convolution and correlation of binary images implemented in fpga*. In Electrical Engineering, Computing Science and Automatic Control

Arquitecturas, Diseños de Referencia
y Plataformas

Análisis y evaluación de un módulo IME hardware para el codificador HEVC usando una plataforma SoC

Estefanía Alcocer, Otoniel López-Granado, Manuel P. Malumbres¹ y Roberto Gutiérrez²

Resumen— En el estándar de codificación HEVC, la estimación de movimiento es una de las tareas más complejas del codificador de vídeo, necesitando un gran porcentaje del tiempo de codificación total. Esto se debe, principalmente, a los cambios introducidos en comparación con el estándar predecesor H264/AVC, siendo los siguientes: un conjunto más extenso de modos de partición del Coding Tree Unit, la presencia de múltiples frames de referencia y el tamaño variable de los Coding Units. Además, HEVC adopta una estimación de movimiento por bloques de tamaño variable para obtener una eficiencia de codificación avanzada.

En este trabajo, evaluamos un diseño de estimación de movimiento por enteros, IME, implementado en una plataforma SoC. En esta evaluación, mediremos el impacto en el Rate/Distortion al aplicar diferentes tamaños tanto de CTU como de áreas de búsqueda de referencia. Además, se evaluará el efecto de las transferencias del DMA requeridas en el rendimiento computacional. Esta arquitectura ha sido sintetizada e implementada en el Xilinx SoC, Zynq-7 Mini-ITX Motherboard XC7Z100 (xc7z100ffg900-2). Nuestra arquitectura IME hardware se puede configurar para trabajar con diferentes tamaños de CTU y rangos de búsqueda del frame de referencia. Los resultados muestran que no hay diferencias significativas en términos de Rate/Distortion entre las diferentes configuraciones, pero sí existe un gran impacto en la complejidad del codificador cuando tanto el tamaño del CTU como el rango de búsqueda se incrementa.

Hemos evaluado nuestro diseño IME hardware usando diferentes configuraciones de CTU, tamaños de área de búsqueda y tamaños de ráfaga del DMA, con el fin de determinar la máxima ganancia con respecto al software de referencia del HEVC. Los resultados muestran que el tiempo de codificación se podría reducir 588 veces. Además, esta evaluación revela que las transferencias del DMA son el cuello de botella del sistema completo.

Palabras clave— HEVC, codificación de vídeo, FPGA, SoC, estimación de movimiento, IME.

I. INTRODUCCIÓN

EL estándar de codificación HEVC (High Efficiency Video Coding) [1] se presentó en Enero de 2013 por el JCT-VC (Joint Collaborative Team on Video Coding). Este nuevo estándar sustituye el actual estándar H.264/AVC [2] con el fin de tratar con las tendencias multimedia del mercado actual y futuro como los contenidos de vídeo con definiciones 4K y 8K y profundidades de color de alta calidad a 10 bits. HEVC ha mejorado significativamente la eficiencia de codificación en cuanto a su predecesor H.264/AVC en un factor de casi dos veces mientras

se mantiene una calidad visual equivalente [3]. En cuanto a la complejidad, el decodificador HEVC no parece ser muy diferente al del H.264/AVC [4]. Sin embargo, se espera que el codificador HEVC sea considerablemente más complejo que el codificador de H.264/AVC [5] y será un tema candente de investigación en los próximos años, por ejemplo, codificar un segundo de un vídeo a una resolución de 1080p60 HD (High Definition) con el codificador del software de referencia puede llevar más de una hora de ejecución en un ordenador de sobremesa actual.

Como en estándares anteriores, la estimación de movimiento, ME (Motion Estimation), es la tarea más compleja del codificador, necesitando más del 90% del tiempo de codificación [6]. En HEVC, la complejidad es incluso más crítica debido a diferentes parámetros como son (a) un mayor conjunto de modos de partición del Coding Tree Unit (CTU), (b) la presencia del múltiples frames de referencia y (c) el tamaño variable de los Coding Units (CU) comparando con su predecesor H264/AVC. Además, HEVC adopta la técnica Variable Block Size Motion Estimation (VBSME) para obtener una eficiencia avanzada de codificación, lo que se logra a costa de un enorme aumento de la complejidad computacional.

Se han propuesto muchas arquitecturas hardware para acelerar el módulo del ME HEVC con el objetivo de reducir, tanto como sea posible, la complejidad total del codificador. El bloque de la estimación de movimiento por enteros, IME (Integer-pel Motion Estimation), está a cargo del ME. En la mayoría de las propuestas del estado del arte, las arquitecturas IME hardware se centran solamente en los algoritmos de búsqueda de movimiento ya que esto conlleva la mayoría del tiempo computacional del bloque IME. Generalmente, el algoritmo más popular de la búsqueda de movimiento en implementaciones hardware es el algoritmo Full Search (FS), de búsqueda completa. Este realiza una búsqueda en todos los puntos de la zona de búsqueda establecida de un frame de referencia, y como consecuencia, es capaz de proporcionar el resultado óptimo, es decir, un vector de movimiento que minimiza el error residual del CTU actual.

Las arquitecturas propuestas en [6], [7], [8], [9], [10] presentan un bloque IME hardware usando un algoritmo de búsqueda FS. En [6], se propone una unidad de Suma de Diferencias Absolutas (SAD) en una FPGA (Field-Programmable Gate Array) la cual es capaz de analizar todos los modos de partición de

¹Dpto. de Física y Arquitectura de Computadores, Univ. Miguel Hernández. Elche, e-mail: {ealcocer, otoniel, mels}@umh.es

²Dpto. de Ingeniería de Comunicaciones, Univ. Miguel Hernández. Elche, e-mail: roberto.gutierrez@umh.es

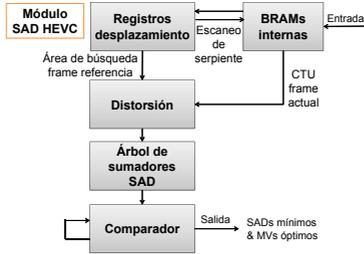


Fig. 2. Módulo SAD HEVC hardware

en HEVC consta de CUs con un tamaño máximo de 64x64 píxeles, tan grande como un CTU (Coding Tree Unit), los cuales pueden ser divididos recursivamente en bloques cuadrados hasta un tamaño de 8x8 píxeles. Cada CU consta de unidades de predicción PUs (Prediction Units) cuyo tamaño puede variar desde el tamaño máximo del CU hasta 4x8 o 8x4 en este caso de predicción Inter, soportando 8 modos de partición. En nuestra propuesta, el módulo SAD HEVC responsable del cálculo IME se puede configurar para trabajar con tamaños de CTU de 64x64 y 32x32. En el caso de un CTU de 64x64 píxeles, el PL puede trabajar a 220MHz mientras que con un tamaño de CTU de 32x32 la frecuencia de reloj del PL se fija al máximo de la placa de evaluación usada, 250 MHz. Sin embargo, dicho módulo por separado podría trabajar a una frecuencia máxima de 333 MHz. Por lo tanto, la frecuencia máxima está limitada a la frecuencia de reloj máxima del PL, la cual es 250 MHz para la plataforma de evaluación utilizada.

Nuestro módulo consta de (a) áreas de memoria interna para almacenar píxeles del CU del frame actual y los píxeles pertenecientes al área de búsqueda de los frames de referencia, (b) un bloque de distorsión donde los píxeles de ambos frames se restan, (c) un árbol de sumadores SAD y (d) un comparador acumulativo que guarda los valores mínimos de SAD y sus MVs correspondientes para todas las particiones de CU, como se muestra en la Figura 2. Para más detalles de este módulo, ver el trabajo previo de los autores [18].

En la Tabla I, se muestran los recursos usados para la implementación de nuestro módulo SAD HEVC para unos tamaños de CTU máximos de 64x64 y 32x32, respectivamente, en una Zynq-7 Mini-ITX Motherboard XC7Z100 FPGA. Como se observa, nuestro módulo SAD HEVC requiere un 63% y un 16% del total de área usada para unos tamaños de CTU de 64x64 y 32x32, respectivamente.

III. EXPERIMENTOS NUMÉRICOS

ME es una tarea integrada en la predicción Inter del HEVC. Por esta razón, se ha considerado un modo de configuración Low Delay B (LB). En LB,

TABLA I
 ÁREA UTILIZADA EN MINI-ITX

Recursos	CTU 64x64	CTU 32x32
LUTs	166383	40911
Flip-flops	190159	50028
Block-RAMs	32	16

el primer frame se codifica como una imagen intra y el resto se codifican como frames bidireccionales generalizados (tipo inter). Esta estructura de codificación es la más popular para el streaming de vídeo, diseñado principalmente para comunicaciones interactivas en tiempo real.

Dada la configuración previa, se han realizado diferentes experimentos del IME de HEVC con el fin de observar como ambos parámetros, el tamaño del CTU y el tamaño del área de búsqueda, impactan en el R/D y en la complejidad del codificador HEVC. Se han elegido unos tamaños de CTU de 64x64 y 32x32 píxeles y sus correspondientes rangos de búsqueda, SR (Search Range), cuyos tamaños son el 100% del CTU, el 80% del CTU, y/o el 50% del CTU. Además, se han seleccionado dos secuencias de vídeo del conjunto de vídeos de las *common conditions* del HEVC: *ParkScene* con una resolución de 1920x1080 (24 fps) y *Traffic* a 2560x1600 (30 fps). Para realizar estos tests, hemos usado el modelo de referencia del HEVC, HM 14 [19]. El software de referencia HEVC se ha compilado con Visual Studio 2010 y ejecutado sobre un PC con las siguientes características: Intel Core i7-3770 CPU 3.40GHz con 8GB RAM.

Se ha medido el tiempo del módulo IME software usando un algoritmo de búsqueda FS cuando se codifica una secuencia de vídeo. Nuestro trabajo se ha centrado en el diseño e implementación hardware de un algoritmo FS que es capaz de acelerar significativamente el proceso de estimación de movimiento del codificador HEVC sin perder rendimiento R/D, ya que el algoritmo FS es la estrategia más adecuada para procesos hardware, el cual busca el movimiento exhaustivamente para todos los PUs en cada uno de los puntos del área de búsqueda establecida. Por lo tanto, debido a la regularidad computacional y la excelente calidad de vídeo de salida, la estimación de movimiento con FS es comúnmente empleada en implementaciones hardware [20].

En la Figura 3, se muestra el porcentaje del tiempo total de codificación que requiere el IME en software usando el algoritmo de búsqueda FS, sin contar con el procedimiento de optimización R/D. Este porcentaje de tiempo es similar para todas las secuencias analizadas. Como se puede observar en la Figura 3, este tiempo depende de los tamaños de CTU y SR. Normalmente, el módulo IME del codificador requiere más tiempo cuanto mayor sea el tamaño tanto del CTU como del SR, como es de esperar. El tiempo que el codificador HEVC necesita para realizar la es-

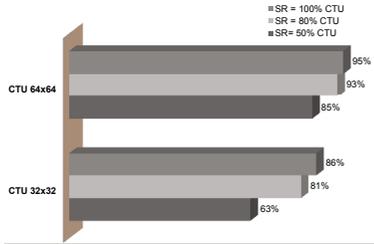


Fig. 3. Porcentaje de tiempo de codificación SW del módulo SAD con una estrategia Full-Search

timación de movimiento varía entre el 63% y el 95% del tiempo total para codificar una secuencia de vídeo completa. Por tanto, cobra sentido el diseño de una arquitectura hardware que realice el proceso IME de una manera más rápida y precisa con el fin de reducir tanto la complejidad del módulo IME como el tiempo total de codificación en la medida de lo posible.

Además, se ha analizado el impacto de los parámetros previos en el R/D mediante el cálculo de la métrica de Bjontegaard (BD-rate). La métrica de Bjontegaard calcula el ahorro en tanto por ciento del bit-rate en media entre dos curvas de Rate/Distortion. Las curvas R/D se han obtenido para los siguientes niveles de compresión (valores QP): 22, 27, 32 y 37, teniendo en cuenta una curva de referencia con la configuración típica dada por el modelo software HEVC, HM-14, con un tamaño de CTU de 64x64 píxeles y un rango de búsqueda, SR, de 64 (tamaño de área de búsqueda de 128x128).

En la Figura 4, se puede observar el porcentaje de BD-Rate obtenido con diferentes tamaños de CTU y SR para vídeos de resoluciones 1920x1080 y 2560x1600. Como se muestra, existen ligeras diferencias entre los tamaños de SR para un tamaño de CTU dado, especialmente cuando el tamaño de CTU es 64x64, donde la diferencia es, como máximo, 0.1% aproximadamente (ver Figura 4(b)). De cualquier modo, las diferencias entre las posibles configuraciones son despreciables, siendo un 2.7% el bit-rate incrementado, como máximo, para una calidad (PSNR) fija dada, cuando el tamaño del CTU es de 32x32 (ver Figura 4(a)). Aunque las diferencias del BD-Rate podrían depender del tipo de contenido de vídeo, se han obtenido resultados similares con otras secuencias de vídeo.

Por tanto, con el fin de reducir la complejidad permitiendo versiones más rápidas con un consumo reducido, los tamaños de CTU y SR pueden reducirse tanto como sea posible debido al insignificante impacto de estos parámetros en el resultado anterior en cuanto al BD-rate.

En cuanto a la arquitectura hardware del IME propuesta, se han medido los CUs por segundo que es capaz de procesar dependiendo de los tamaños de CTU, SR y ráfaga de DMA. En nuestro diseño IME

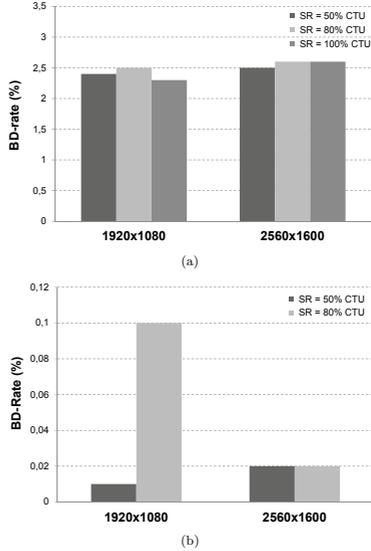


Fig. 4. BD-Rate respecto a una curva R/D de referencia con un tamaño de CTU y SR de 64: (a) Para curvas R/D con un tamaño de CTU 32x32 y SR de 16, 26 y 32; (b) Para curvas R/D con un tamaño de CTU 64x64 y SR de 32 y 56

completo descrito en la Sección II, el tamaño de palabra (o datos) del DMA se ha definido como 32 bits y el tamaño de ráfaga (palabras a transmitir en una transferencia) se puede configurar desde 16 a 256 palabras. Por ejemplo, en el caso de un tamaño de CTU de 32x32 píxeles donde la frecuencia de operación es 250 MHz y con un tamaño de ráfaga del DMA de 256, el sistema propuesto puede transferir con una tasa de 2Mbps.

En la Figura 5 se muestra la cantidad de CUs por segundo procesados para cada tamaño de ráfaga del DMA. Las Figuras 5(a) y 5(b) muestran como influye el SR en los CUs por segundo cuando el tamaño del CTU es 32x32 y 64x64, respectivamente. Como se puede observar, el número de CUs por segundo procesados aumenta cuando el tamaño de ráfaga del DMA también lo hace, siendo 10626 y 2356 los máximos CUs por segundo para los tamaños de CTU 32x32 y 64x64, respectivamente. Ese incremento es exponencial debido a que el tiempo que necesita la inicialización del módulo DMA es constante e independiente del tamaño de ráfaga del DMA. Por tanto, la configuración compuesta por un tamaño máximo de ráfaga de 256, un tamaño de CTU de 32x32 y un SR de 16 es la configuración más apropiada para conseguir el mayor número de CUs por segundo usando nuestro sistema hardware.

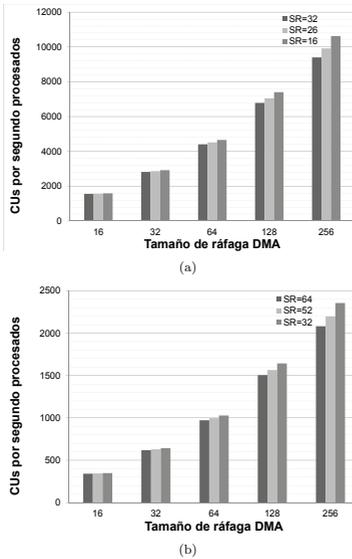


Fig. 5. CUs por segundo procesados para cada tamaño de ráfaga DMA con nuestro módulo SAD HEVC propuesto: (a) Para un tamaño de CTU de 32x32; (b) Para un tamaño de CTU de 64x64

Sin embargo, el número de CUs por segundo alcanzados previamente depende del tiempo completo de procesamiento de nuestro sistema hardware, el cual consta de las transferencias del DMA y el módulo SAD HEVC. Así, en la Figura 6 mostramos el tiempo del DMA, el tiempo del módulo SAD HEVC y el tiempo total de procesar un CU para cada tamaño de ráfaga del DMA cuando establecemos la configuración de un CTU de 32x32 y un tamaño de SR del 50% CTU (± 16). Como se esperaba, el tamaño máximo de ráfaga del DMA proporciona los mejores resultados, necesitando el menor tiempo de procesamiento de un CU. Además, dependiendo de la ráfaga de DMA elegida, las diferencias entre los tiempos de las transferencias del DMA y del módulo SAD HEVC varían, siendo el módulo SAD HEVC 21 veces más rápido para un tamaño de ráfaga de 256 y 146 veces más rápido para una ráfaga de 16 palabras.

Por lo tanto, el cuello de botella del tiempo total de nuestro sistema hardware se debe a las transferencias DMA. Habiendo escogido un tamaño de ráfaga de DMA de 256, las transferencias representan el 95% del tiempo total de procesamiento de un CU, mientras que el módulo SAD HEVC solamente necesita el 5% del tiempo. Este cuello de botella puede mitigarse en trabajos futuros, por ejemplo, transfiriendo solamente las partes de los frames de referencia que difieren de los que ya están almacenados en las BRAMs

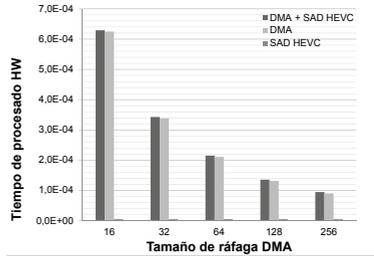


Fig. 6. Tiempo de procesamiento hardware para diferentes tamaños de ráfaga del DMA

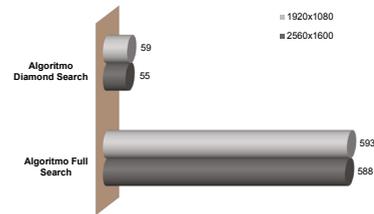


Fig. 7. Ganancia hardware frente a los algoritmos software Full Search y Diamond Search

internas. De esta manera, las transferencias DMA se podrán reducir y por consiguiente, el tiempo total de procesamiento requerido.

Tras realizar el análisis completo, se puede determinar que la configuración hardware que mejor se adapta a los requerimientos de la aplicación (bajo consumo de potencia, tiempo de codificación, calidad de vídeo comprimido) es la establecida por un tamaño de CTU de 32x32, un SR de 16 y un tamaño de ráfaga del DMA de 256.

Aunque el algoritmo de búsqueda FS es el más ampliamente usado en las implementaciones hardware del ME, el software de referencia del HEVC utiliza por defecto el algoritmo Diamond Search. Así, en la Figura 7 se muestra la ganancia que se obtiene utilizando nuestra propuesta hardware del IME, con el conjunto de configuración anteriormente establecido, en comparación con el módulo IME software utilizando tanto FS como DS. Por tanto, observando la información proporcionada por la Figura 7, para una resolución de vídeo de 2560x1600, la integración de nuestro módulo hardware acelerará la computación del IME en 55 veces en el caso de una estrategia de búsqueda DS y 588 veces con la estrategia FS.

IV. CONCLUSIONES

En este trabajo, se ha presentado una arquitectura hardware completa del IME, la cual incluye un módulo DMA. Se ha analizado cómo el tamaño

de CTU y el rango de búsqueda afectan al funcionamiento del IME del HEVC en términos de Rate/Distortion y tiempo de procesamiento. Como se ha mostrado, las diferencias en R/D son despreciables para cualquier configuración, siendo un 2.7% el bitrate incrementado cuando el tamaño del CTU es de 32x32 y 0.1% para un CTU de 64x64. En cuanto a la complejidad, tanto el tamaño de CTU como el SR impactan en el tiempo total de codificación, obteniendo un tiempo de procesamiento menor con la configuración formada por un CTU de 32x32 y un SR del 50% del tamaño de CTU. Remarcar que el módulo ME requiere entre el 63% y el 95% del tiempo total de codificación.

Respecto a la evaluación de la propuesta hardware del IME, se han medido los máximos CUs por segundo que pueden ser procesados en función de los tamaños del CTU, SR y ráfaga de DMA. Los resultados muestran que el número de CUs procesados se incrementa cuando el tamaño de ráfaga lo hace, siendo 10626 el máximo número de CUs por segundo procesados para un tamaño de CTU de 32x32, un SR del 50% del CTU y una ráfaga de DMA de 256. Analizando estos resultados, se confirma que el cuello de botella del sistema reside en las transferencias DMA, necesitando éstas el 95% del tiempo total de procesado.

Por tanto, teniendo en cuenta tanto el R/D como la complejidad de codificación, el mejor conjunto de parámetros de configuración para nuestra arquitectura hardware del IME es el conformado por un CTU de 32x32 píxeles, un SR del 50% del tamaño de CTU (± 16) y un tamaño de ráfaga del DMA de 256. En resumen, con la inclusión de nuestra propuesta hardware del IME, el tiempo de codificación se puede acelerar 588 veces.

Como posible trabajo futuro, se pretende reducir el tiempo de las transferencias DMA, reusando parte de las áreas de búsqueda de los frames de referencia para el cálculo del IME en CUs contiguos, transfiriendo solamente los nuevos píxeles de referencia requeridos en cada momento. Además, puesto que solamente se utiliza un 16% de los recursos totales de la plataforma, sería posible añadir más módulos SAD HEVC en nuestro sistema para acelerar todavía más la computación del ME.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad a través del proyecto I+D con referencia TIN2015-66972-C5-4-R y co-financiado mediante fondos FEDER.

REFERENCIAS

[1] B. Bross, W. J. Han, J. R. Ohm, G. J. Sullivan, Y.-K. Wang, and T. Wiegand, "High efficiency video coding (HEVC) text specification draft 10," *Document JCTVC-L1003 of JCT-VC, Geneva*, January 2013.
 [2] ITU-T and ISO/IEC JTC 1, "Advanced video coding for generic audiovisual services," *ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC) version 16*, 2012, 2012.
 [3] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *Circuits and systems for Video Technology*,

IEEE Transactions on, vol. 22, no. 12, pp. 1648–1667, December 2012.
 [4] J. Ohm, G. J. Sullivan, H. Schwarz, Thiew Keng Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards - including high efficiency video coding (hevc)," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1669–1684, 2012.
 [5] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC complexity and implementation analysis," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1685–1696, 2012.
 [6] Ahmed Medhat, Ahmed Shalaby, Mohammed S Sayed, and Maha Elsabrouty, "A highly parallel sad architecture for motion estimation in hevc encoder," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS'14)*, Ishigaki, Nov. 2014, pp. 280–283.
 [7] J. Byun, Y. Jung, and J. Kim, "Design of integer motion estimator of hevc for asymmetric motion-partitioning mode and 4k-uhd," *Electronics Letters*, vol. 49, no. 18, pp. 1142–1143, 2013.
 [8] Xu Yuan, Liu Jinsong, Gong Liwei, Zhang Zhi, and Robert K.F. Tse, "A high performance vlsi architecture for integer motion estimation in hevc," in *IEEE 10th International Conference on ASIC (ASICON'13)*, Shenzhen, Oct. 2013, pp. 1–4.
 [9] Thomas D'huys, "Reconfigurable data flow engine for hevc motion estimation," in *IEEE International Conference on Image Processing (ICIP'14)*, Paris, Oct. 2014, pp. 1223–1227.
 [10] Antonio Navarro Purnachand Nalluri, Luis Nero Alves, "High speed sad architectures for variable block size motion estimation in hevc video coding," in *IEEE International Conference on Image Processing (ICIP'14)*, Paris, Oct. 2014, pp. 1233–1237.
 [11] Vidyalekshmi VG, Deepa Yagain, and Ganesh Rao K, "Motion estimation block for hevc encoder on fpga," in *IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE'14)*, Jaipur, India, May 2014, pp. 1–5.
 [12] P. Davis and Marikkannan Sangeetha, "Implementation of motion estimation algorithm for h.265/hevc," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 3, no. 3, pp. 122–126, 2014.
 [13] Ching-Yeh Chen, Shao-Yi Chien, Yu-Wen Huang, Tung-Chien Chen, Tu-Chih Wang, and Liang-Gee Chen, "Analysis and architecture design of variable block-size motion estimation for h.264/avc," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 3, pp. 578–593, 2006.
 [14] Wajdi Elhamzi, Julien Dubois, and Johel Miteran, "An efficient low-cost fpga implementation of a configurable motion estimation for h.264 video coding," *Springer Journal of Real-Time Processing*, vol. 9, no. 1, pp. 19–30, 2014.
 [15] Theepan Moorthy and Andy Ye, "A scalable architecture for variable block size motion estimation on field-programmable gate arrays," in *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'08)*, Niagara Falls, May 2008, pp. 1303–1308.
 [16] M Kthiri, P Kadionik, H Levi, H Loulik, Ben Attallah, and N Masmoudi, "An fpga implementation of motion estimation algorithm for h.264/avc," in *IEEE 5th International Symposium on I/V Communications and Mobile Network (ISVC'10)*, Rabat, Sept. 2010, pp. 1–4.
 [17] Grzegorz Pastuszak and Mariusz Jakubowski, "Adaptive computationally scalable motion estimation for the hardware h.264/avc encoder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, no. 5, pp. 802–812, 2013.
 [18] Estefanía Alcocer, Roberto Gutierrez, Otoniel Lopez-Granado, and Manuel P. Malumbres, "Design and implementation of an efficient hardware integer motion estimator for an hevc video encoder," *Journal of Real-Time Image Processing*, pp. 1–11, 2016.
 [19] HEVC software repository HM-14.0 reference model, <https://hevc.hhi.fraunhofer.de/trac/hevc/browser/tags/HM-14.0>.
 [20] Youn-Long Steve Lin, Chao-Yang Kao, Hung-Chih Kuo, and Jian-Wen Chen, *VLSI Design for Video Coding - H.264/AVC Encoding from Standard Specification to Chip*, Springer, New York, NY, 2010.

Using the ChipCflow Project as a learning tool for programming Dataflow language

Jorge Luiz e Silva¹Bruno de Abreu Silva²

Resumen— The ChipCflow Project is a TOOL to convert C language in VHDL to accelerating algorithms. To develop the C compiler, an assembler tool with an assembly language was developed. The assembly language is based on Dataflow operators, that has been used to programming applications in a Dataflow model to be configured into a FPGA as a support for the C compiler. In this paper the assembler tool and the assembly language are presented as a learning tool for programming Dataflow model. Examples are discussed and results of the implementation also is described.

Palabras clave— Dataflow, Assembly, Learning tool, VHDL, FPGA.

I. INTRODUCTION

THE ChipCflow is a Project that has been developed since 2006 at University of Sao Paulo, Brazil. The Project is totally based on FPGA and VHDL language, and the objective with this project is to convert C language in VHDL using the Dataflow model to accelerating algorithms [9].

The Dataflow model was purposed at 70's and the researches were discontinued at 90's [1], [2], [3]. However it is a research interest again as an alternative for parallel systems [4].

Basically, the dataflow model is a set of operators (*nodes*) interconnected by (*arcs*) generating a *dataflow graph* with rule of operation. An example of a dataflow graph is described in Figure 1.

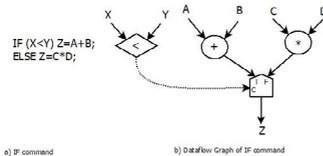


Fig. 1. An example of a Dataflow Graph.

As can be seen in Figure 1 an example of an "IF" command from C language is described as a dataflow graph.

The rule of the execution of a dataflow system is that nodes are triggered when all item of input data are present in its input arcs. When the node is triggered, all the item of input data are consumed and a new item of output data, in the output arcs of the operators, are generated. There are two kind of

dataflow model, a Static Dataflow Model where just one item of data can be in an arc, and a Dynamic Dataflow Model where more than one item of data can be in an arc, in this case, a tagged token is used to identify different instances of the data [1], [2], [3].

During the development of the ChipCflow Project, an assembler tool, to test and validate several dataflow programs, representing different commands from a C language, was generated.

The dataflow operators were developed directly in VHDL and each operator is an *entity* with ports of input and output signals followed by its architecture behaviour. The *BRANCH* operator and its VHDL *entity* are described in Figure 2.

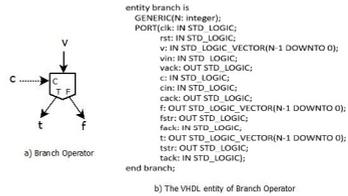


Fig. 2. Part a) the BRANCH operator and part b) the VHDL entity of BRANCH operator.

How is described in Figure 2 part a), the BRANCH operator has two input signals and two output signals. Depends on the value of the input signal *c* - (*true* or *false*) the value of input signal *v* is branched to the output signal (*t* or *f*) respectively.

In Figure 2 part b) the VHDL entity for the BRANCH operator is described. In the VHDL entity there are the signals (*v*, *c*, *t* and *f*) and for each one of those signals, there are two another signals to implement the communications process between operators. For example, in the input signal *v*, there are two other signals associated to it: *vin* and *vack*. The input signal *vin* is used to strobe a item of data, from previous operator, into the BRANCH operator associated with the input signal *v* and the output signal *vack* is used for the BRANCH operator to inform the previous operator connected to it, that it is available to receive another item of data by the input signal *v*.

Finally, to connect all the signal of the BRANCH operator with previous and next operators, it is necessary eight signals, four of them as an input and output signal and the others to control the communication.

To convert any program from C language to a

¹Dpto. of Computer Systems, Univ. of Sao Paulo, e-mail: jsilva@icmc.usp.br

²Dpto. of Computer Systems, Univ. of Sao Paulo, e-mail: brunoas@icmc.usp.br

dataflow graph, a set of operators are necessary, consequently there are several input and output signals for each operator to be connected to configure the dataflow graph. To connect all the signals in the dataflow graph a hard work need to be done. Then an assembler tool with an assembly language was generated to implement a dataflow graph, where signals like v in the BRANCH operator is associated to the corresponding signals vin and $vack$, automatically.

Initially, the assembler tool was used as a support for the C compiler, however, for several times, it was used for graduate and undergraduate students in the ChipCflow Project to programming and to understand how dataflow program is. In this paper the assembler tool and assembly language are presented as a learning tool for programming Dataflow model. Examples are discussed and results of the implementation also is described.

II. THE DATAFLOW MODEL

In the Asynchronous Dataflow Graph project developed by Teifel et al. [8], the asynchronous system is a collection of concurrent hardware processes that communicate with each other through message-passing channels. These messages consist of atomic data item called tokens. Each process can send and receive tokens to and from its environment through communication ports. In the Teifel project, asynchronous pipelines are constructed by connecting these ports to each other using channels, where each channel is allowed only one sender and one receiver. Since there is no clock in an asynchronous design, processes use handshake protocols to send and receive tokens via channels.

In Fig. 3 Teifel describes an equation converted into a dataflow graph in three different situations: (a) a pure dataflow graph, (b) a token-based asynchronous dataflow pipeline and (c) a clocked dataflow pipeline.

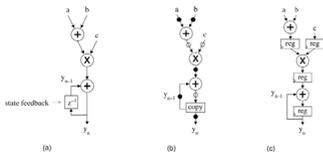


Fig. 3. Computation of $y_n = y_{n-1} + c(a+b)$: (a) pure dataflow graph, (b) token-based asynchronous dataflow pipeline (filled circles indicate tokens, empty circles indicate an absence of tokens), and (c) clocked dataflow pipeline

In our project, a collection of concurrent hardware processes that communicate with each other, but using a parallel bus with bits for data and bits to control the communication in a synchronous system of communication as described in part (c) of the Fig. 3, is also used.

A. Dataflow Computations

In the dataflow graph to the ChipCflow Project, a traditional dataflow model described in the literature, where a node is a processing element and an arc is the connection between two elements, is used [5], [6], [7]. A data bus and a control bus to execute the communication between the operators were implemented. The static dataflow graph model, where only one item of data can be in an arch was developed.

In Fig. 4, a basic operator and its data buses and control buses for communication are described. The signal data a , b and z in Fig. 4 are 16-bit data traveling through the parallel buses. The signals $stra$, $strb$, $strz$, $acka$, $ackb$ and $ackz$ are 1-bit control data to control communication between operators.

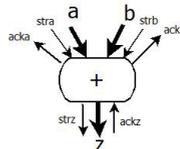


Fig. 4. The basic operator with its data buses and control buses.

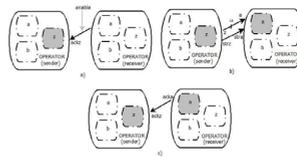


Fig. 5. The communication: a) enabling the communication, b) sending an item of data, c) Acknowledging an item of data.

The communication between operators is described in Fig. 5. As can be clearly seen in the figure, a sender operator and a receiver operator have two input data buses a and b , one output data bus z and its respective control signals $stra$, $strb$, $strz$, $acka$, $ackb$ and $ackz$. Each of the input data bus and output data bus is connected to a register to store a receiving item of data and to store a sending item of data, represented by rectangles with rounded edges denoted with the characters a , b and z in the figure. The output data bus z from the sender operator is connected to input data bus a from the receiver operator, the output control signal $strz$ from the sender operator is connected to the input control signal $stra$ from the receiver operator and the input control signal $ackz$ from the sender operator is connected to the output control signal $acka$ from the receiver operator.

A "logic-0" in the signal $ackz$ informs the sender operator that the receiver operator is ready to re-

ceive data. A "logic-1" in the signal *ackz* informs the sender operator that the receiver operator is busy. A "logic-1" in the signal *stra* informs the receiver operator that an item of data is ready to be sent to it from the sender operator. A "logic-0" in the signal *stra* informs the receiver operator that the sender have not an item of data to be sent to it.

To initiate the communication, an *enable* signal with a "logic-0" to the *ackz* connected to the sender, is set, Fig. 5a. When the receiver operator is ready to receive data, a "logic-1" in the *stra* strobes an item of data to the input data bus *a* in the receiver operator, Fig. 5b. Consequently, a "logic-0" in the *acka* acknowledges that the item of data *a* was received, Fig. 5c.

B. The Dataflow Operators

The dataflow operators were the traditional operators described by Voen in [7], which are: copy, non deterministic merge, deterministic merge, branch, conditional and primitive operators (add, sub, mul, div, and, or, not, etc.).

In order to execute the computation of an operator it is necessary that an item of data is presented in all its input buses of data. In Fig. 6, operators are described where filled circles indicate item of data and empty circles show an absence of item of data and the situation of the operator before computation and after computation [8].

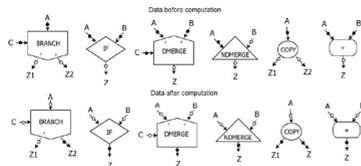


Fig. 6. The operator.

The functional execution of dataflow operators is described below:

1. *Copy*: This dataflow node duplicates an item of data to two receiver operators. It receives an item of data in its input data bus and copies the item of data to two output data buses.
2. *Primitive*: This dataflow node receives two item of data in its input data buses, computes the primitive operation with these two item of data and generates the result sending it to the output data bus. Operators such as add, sub, multiply, divide, and, or, not, if, etc., are implemented in the same way.
3. *Dmerge*: This dataflow node performs a two-way controlled data merge and allows an item of data to be conditionally read in input data buses. It receives a TRUE/FALSE item of data to decide what input data *a* or *b* respectively to send to the output data *z*

4. *NDmery*: This dataflow node performs a two-way not controlled data merge and allows an item of data to be read on input data buses. The first data to arrive into the Ndmery operator from input *a* or *b* is sent to the output data *z*.
5. *Branch*: This dataflow node performs a two-way controlled data branch and allows the item of data to be conditionally sent on to two different output buses. A control TRUE/FALSE item of data is received to decide what output data *t* or *f* respectively to transfer the input data *a*.

III. THE ASSEMBLER TOOL AND ASSEMBLY LANGUAGE

The Assembler tool was developed using C language that convert an assembly dataflow program in VHDL language to be implemented into a FPGA. The assembly dataflow program is generated from a dataflow graph.

To execute the assembler program, an assembly-file.txt is generated in .txt format, and then a command line

```
chipcflowassembler assemblyfile.txt
```

need to be executed to generate a assemblyfile.vhd that can be synthesised and implemented in an EDA (Electronic Design Automation) tool like Vivado from Xilinx or Quartus from Altera.

A. The Assembly Language

For each operator in the ChipCflow Project, the correspondent VHDL operator was implemented. Those operators are described by line in the assembly program where each line is composed by the opcode of the dataflow operator followed by its operands that correspond to each input and output signals of the dataflow operator.

A simple example of a assembly program is described in the listing below.

```
add a,b,s; Add
mul a,b,s; Multiply
ndmerge a,b,s; Non-Deterministic merge
decoder a,b,s; Decoder
copy a,s1,s2; Copy
branch a,c,s1,s2; Branch
dmerge a,b,c,s; Deterministic merge
```

How is described in the listing, the operator *add* has two input signals: *a* and *b* and one output signal *s*. The same for the operator *mul*. In the case of the *dmerge* operator, there are three input signals: *a*, *b* and *c*, and one output signal *s*. Particularly in the case of the *dmerge* operator, the input signal *c* is a control signal to decide what input data *a* or *b* respectively will be sent to the output data *z*. Then for the *dmerge* operator and all others operators, the respectively position of the operands in the assembly line need to be aligned. In Table I the basic operators and theirs respective operands with the respective positions of their operands are described.

As can be clearly seen in Table I some of operators were presented in the previous section. More-

TABLE I
 ASSEMBLY LINES

add	a	b	s		
sub	a	b	s		
mul	a	b	s		
div	a	b	s		
and	a	b	s		
or	a	b	s		
not	a	b	s		
decider	a	b	s		
ndmerge	a	b	s		
dmerge	a	b	c	s	
branch	a	c	s	s	
copy	a	s	s		

over there are a set of special operators and they are described in Table II

TABLE II
 ASSEMBLY LINES

sram	a	q	v	w	g	t		
load	b	i	j	n	d	t	r	
store	b	i	j	n	w	a	v	q
for	n	f	c	i	x			
nopop	a							
const	a	s						

How is described in Table II the *sram* operator has 6 operands (*a*, *q*, *v*, *w*, *g* and *t*) and are operands that correspond with the same signals of a dual port memory generated from an IP generator of an EDA tool from Xilinx, Altera, etc., and is defined as a component into the *sram* operator. Part of the operands from the *sram* operator are used to write data into the *sram* operator and the other part of the operands are used to read data from the *sram* operator and both actions can occur in parallel due to the characteristics of the dual port memory. The operands (*a*, *q*, *w* and *g*) are used to write data into the *sram* operator where: *a* is the address, *q* is an item of data to be written into the *sram* operator, *w* is a control signal with a "logic-1" specifying a write cycle into the *sram* operator, and *g* is not used when in write cycle. The operands (*v* and *t*) are used to read data from the *sram* operator where: *v* is the address and *t* is an item of a given read data from the *sram* operator. An example of the *sram* operator is described in Figure 7.

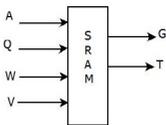


Fig. 7. The operator *sram*.

The *load* and *store* operators, also described in Ta-

ble II, are directly associated with the *sram* operator.

The *load* operator has 8 operands (*b*, *i*, *j*, *n*, *d*, *t* and *r*). The operands (*b*, *i*, *j* and *n*) are used as index to define a array position into the *sram* operator. The Expression 1 is normally used to calculate the position of each element of the array that is stored into the *sram* operator in sequential form.

$$addr = b + ((n - 1) * i) + j \quad (1)$$

The operands (*d*, *t* and *r*) are used to access the *sram* operator where: *d* is the address, *t* is an item of data coming from the *sram* operator and *r* is the data loaded from the *sram* operator to be sent to another operator. An example of the interconnection between the *load* operator and *sram* operator is described in Figure 8.

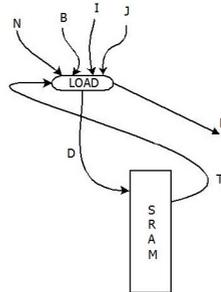


Fig. 8. The operators *load* and *sram* interconnected.

The *store* operator has 8 operands, (*b*, *i*, *j*, *n*, *w*, *a*, *v* and *q*) and the (*b*, *i*, *j* and *n*) also are used as index like in the *load* operand, however, the *store* operator need to get an item of data from previous operator and store this item of data into the *sram* operator, then the (*w*, *a*, *v* and *q*) operands are used to do this. A "logic-1" in the *w* signal is used to inform the *sram* operator that an item of data has to be stored into it; the *a* is the address, the *v* has a item of data coming from previous operator to be stored into the *sram* using the *q* signal. An example of the interconnection between the *store* operator and *sram* operator is described in Figure 9.

The *for* operator, also described in Table II, with the operands (*n*, *f*, *c*, *i* and *x*) is an operator like a "for" command in C language that control iterations of an index. In the *for* operator a initial value for iterations is defined in *f* operand and the final value for iterations is defined in *n* operand. A control signal *c* generate a "logic-1" or "logic-0" depends on the *for* operator that can still be in iterations or have finished its iteration respectively. The value of iterations is generated in *i* operand and finally, the *x* operand is a special operand that control the nested level of the *for* operator. An example of two nested *for* operators is described in Figure 10.

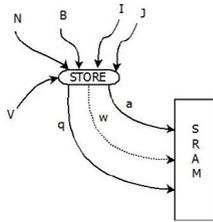


Fig. 9. The operators *store* and *sram* interconnected.

How can be seen in Figure 10, the *for* operator referred as number 2 control the iterations of the *for* operator referred as number 1, that means, while the *for* operator referred as number 2 still in iterations, the *for* operator referred as number 1 still stopped in the last iteration. When the *for* operator referred as number 2 finish its iterations, a "logic-0" is generated in its *c* operand, that is connected with *x* operand of the *for* operator referred as number 1 and then the *for* operator referred as number 1 proceeds one iteration. After this, the *for* operator referred as number 2 begins new iterations, then its *c* operand back to the "logic-1" and the *for* operator referred as number 2 stop its iteration again. The *c* operand and the *i* operand also can be used as a condition and an index respectively to others operators that are associated with that *for* operator.

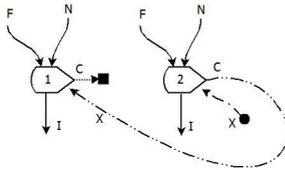


Fig. 10. The two nested *for* operators.

Finally, in Figure 10 there are other two operators, one is described as a black box, and another is described as a black circle. The black box refers to the *nopop* operator and the black circle refers to the *const* operator both described in Table II. The *nopop* operator is a terminal operator that is used when an arc in the dataflow graph is a terminal arc and then it has just one operand *a*. The *const* operator is used when a constant value is attributed to an input signal in an operator, then it has an input signal *a* that has a constant value and an output signal *s* that is used to reproduce the constant value from the signal *a* into the dataflow graph. The *const* operator is necessary because of the rule of the dataflow system, where nodes are triggered when all item of input data are present in its input arcs, then the input item of data are consumed and a new item of data are gen-

erated in the output arcs of the node, however when a constant value is consumed, like the value *n* in the *for* operator described in Figure 10, another value *n* need to be generated to test the condition to stop iterations in the *for* operator, then the *const* operator generate its constant value all the time.

IV. IMPLEMENTING DATAFLOW GRAPH USING ASSEMBLER TOOL

In a dataflow graph there are two kind of signals, the external signals and internal signals. The external signals are all input and output signals of the dataflow graph and they already are defined as operands of the dataflow operators. To define internal signals, that are related with operands of operators and also are related with arcs into the dataflow graph, many representation can be used, however, it is easier to define a sequence of signals like, *s1*, *s2*, *s3*.. and so one. The same "IF" command described in Figure 1 is used as an example to define operands into the dataflow graph, now described in the Figure 11.

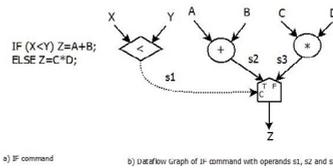


Fig. 11. An "IF" command with internal signals.

How is described in Figure 11, the signals *s1*, *s2* and *s3* are used to define all the internal signals that are operands for operators and also represent arcs into the dataflow graph for "IF" command.

After to define all the operands of the dataflow graph for "IF" command, its respective assembly program can be defined as described in the listing below.

```

ltdecider x,y,s1;
add a,b,s2;
mul c,d,s3;
dmerge s2,s3,s1,z;
    
```

How is discussed before, for each dataflow operator, there are input operands and output operands and it is necessary to use those operands in the correct place where they were defined. How is described in the listing above, the operator *add* has the input signal *a* followed by the input signal *b* followed by the output signal *s2*. The same for the others operators. Any modifications in those positions, an error will be generated and the correspondent VHDL implementation will be wrong.

To implement any program from C language like a simple "IF", as described in Figure 1, a set of operators is necessary, and all the interconnection of the operators need to be correct, to generate the correct VHDL code.

The main task of the assembler tool when the command line below is executed,

```
chipflowassembler assemblyfile.txt
```

is to identify each dataflow operator and its operands defined line by line from the assemblyfile.txt, then converting each dataflow operator to a *component* and then generate the instances of those components with respective ports. An *entity* will be generated with the name of the assemblyfile defined in the command line, and the input signals and the output signals of the entity will be generated after to analyze all the operands of the dataflow operators considering just the signals that are not internal signals in the dataflow graph.

A. Example of generating VHDL code from C language using Assembler tool

The example of a C language program to be converted into VHDL code is the *storejdualex1* dataflow graph that is composed by: two *for* operators to generate the *i* and *j* iterations; one *store* operator; one *sram* operator; eight *const* operator and one *nopop* terminal operator. The dataflow graph of the *storejdualex1* is described in Figure 12 and its function is to store *nm* item of data coming from the *da* input signal into the *sram* operator.

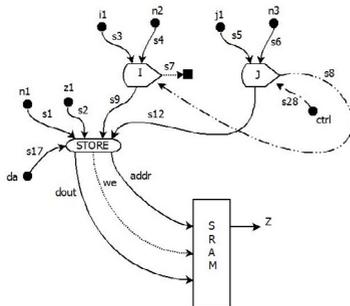


Fig. 12. The *storejdualex1* dataflow graph.

How can be seen in Figure 12, there are input signals (*da*, *n1*, *z1*, *i1*, *n2*, *j1*, *n3* and *ctrl*), output signal *z* and internal signals (*s1*, *s2*, *s3*, *s4*, *s5*, *s6*, *s7*, *s8*, *s9*, *s12*, *s17*, *s28*, *dout*, *we* and *addr*). All the input signals are connected with a *const* operator and will appear as input signals in the *storejdualex1* entity. The only one output signal *z* also will appear as output signal in the *storejdualex1* entity.

The assembly code for the *storejdualex1* dataflow graph is described in listing below. How is described in the listing, for each input signal, a *const* operator is used followed by one *store* operator, two *for* operators, a *nopop* operator and a *sram* operator. The names of the operators are different just for implementation, for example, the *constd* operator is the

const operator that operate with a 16-bit of data and *constc* operator is the *const* operator that operate with a 1-bit of data. The *dualstore* operator is the *store* operator but executing with a dual memory that is implemented in the *dualsram* operator. The *foricex* operator is the *for* operator with the control of nest and finally *nopopc* is a terminal operator that operate with a 1-bit of data.

```
constd n1,s1;
constd n2,s4;
constd z1,s2;
constd i1,s3;
constd n3,s6;
constd j1,s5;
constd da,s17;
constc ctrl,s28;
dualstore s2,s9,s12,s1,we,addr,s17,dout;
foricex s4,s3,s7,s9,s8;
foricex s6,s5,s8,s12,s28;
nopopc s7;
dualsram addr,dout,dpra,we,z,dpo;
```

After to execute the command line

```
chipflowassembler storejdualex1.txt
```

an *entity* is generated with the same name of the *storejdualex1.txt* file, now as *storejdualex1.vhd* that can be implemented using a EDA tool from Xilinx, Altera, etc. The *storejdualex1* entity is described in the listing below. The rest of the VHDL code was omitted that correspond to the component and the instances of the operators.

```
Project storejdualex1
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY storejdualex1 IS
    GENERIC(m: INTEGER := 4;n: INTEGER := 16);
    PORT(
        clk: IN STD_LOGIC;
        rst: IN STD_LOGIC;
        n1: IN STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
        n1ack2ack: OUT STD_LOGIC;
        n2: IN STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
        n2str2in: IN STD_LOGIC;
        n2ack2ack: OUT STD_LOGIC;
        s1: IN STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
        z1str2in: IN STD_LOGIC;
        z1ack2ack: OUT STD_LOGIC;
        i1: IN STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
        i1str2in: IN STD_LOGIC;
        i1ack2ack: OUT STD_LOGIC;
        n3: IN STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
        n3str2in: IN STD_LOGIC;
        n3ack2ack: OUT STD_LOGIC;
        j1: IN STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
        j1str2in: IN STD_LOGIC;
        j1ack2ack: OUT STD_LOGIC;
        da: IN STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
        dastr2in: IN STD_LOGIC;
        daack2ack: OUT STD_LOGIC;
        ctrl: IN STD_LOGIC;
        ctrlstr2in: IN STD_LOGIC;
        ctrlack2ack: OUT STD_LOGIC;
        z: OUT STD_LOGIC_VECTOR(n - 1 DOWNTO 0)
    );
END storejdualex1;
```

An implementation of *storejdualex1.vhd* was realized in the Vivado tool from Xilinx and the implementation was verified and can be seen in Figure 13.

As can be seen in Figure 13 just a piece of the image of the simulation was presented describing each input and output signals, their respective control signals and their values as a result of the simulation. As an example of the result of simulation, consider the input signal *n1* that is a input signal of a *constd* operator and their respective control signals *n1str2n* and *n1ack2ack* that are the strobe signal and acknowledge signal respectively for the input signal *n1*.

In Figure 13, the input signal *nl* has the value "0010h". The next signal, below of the *nl* input signal is the input signal *n1str2n* that is a control signal to trigger the value of *nl* into the *constd* operator and 15 lines below of the *n1str2n* signal, there is an output signal *n1ack2ack* that is the control signal to inform the previous operator connected to the *constd* operator if it can or can not receive another item of data by input *nl*.

Finally, the output signal *z* has the value "0030h" that was used to be stored in all address of the memory.

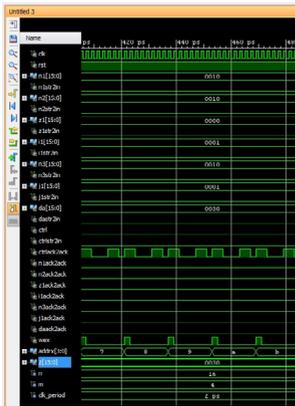


Fig. 13. The simulation of the implementation for *storeid-ualex1* dataflow graph.

V. CONSIDERATIONS ABOUT LEARNING DATAFLOW PROGRAMMING USING THE ASSEMBLER TOOL

Initially, to understand how dataflow program is, a basic concepts of the dataflow model are necessary. After this, a specific understanding of each operator and their rules of execution are fundamental to understand how to define a dataflow graph and its execution. Finally, do a lot of exercises to convert applications written in high level language into a dataflow graph, however a real implementation of the dataflow graph is fundamental, since there are details when implementing a dataflow graph that just will be understood during the implementation.

A. The Basic Concepts of Dataflow Model

The basic concepts of dataflow model begins with the understanding all kind of operators and their rules of execution. This part of the learning process is pretty simple. There is just a few dataflow operators and each of them is very simple to understand. Also simple is to understand the dataflow graph, composed by nodes interconnected by arcs,

excluding on this understanding all detail of interconnections, just experiments about how to describe a simple dataflow graphs.

It is important to explain that there is no variable and the concept of allocation. The data flow through the nodes and arcs and to understand this is the most important thing to learn, since it is totally different of the traditional computation. An example of those concepts are described in Figure 14.

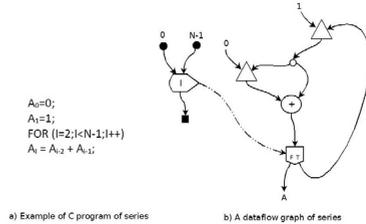


Fig. 14. The concept of variable/allocation against nodes and arcs in dataflow graph

In part a) of the Figure 14, there is an C program of series using the variable *A* where the first value of *A* is 0, the second value of *A* is 1 then the next values of *A* is a sum of the previous two values of *A*.

In part b) of the Figure 14 there is a dataflow graph of the C program of series converted into a dataflow graph. As can be clear seem in the dataflow graph, the first value of *A*, that correspond to an item of data 0, is defined as an input signal in a *ndmerge* operator and the second value of *A*, that correspond to an item of data 1, is defined as an input signal in another *ndmerge* operator.

The first value of *A* is consumed by the *ndmerge* operator then, for its output signal, a result is sent to an input signal of an *add* operator that will wait for the other item of data coming from a *copy* operator. The second value of *A* is consumed by the *ndmerge* operator then, for its output signal, the result is sent to an input signal of a *copy* operator. The *copy* operator will receive this item of data, that correspond to the second value of *A* and then, after to consume the second value of *A*, for its two output signals, a result is sent, one to an input signal of the *add* operator and the other to an input signal of the *ndmerge* operator that had the first value of *A*. Now, the *add* operator has its two item of data into its two input signal, and a result can be generated and then, for its output signal, an item of data, with the result of its operation, is sent to the *branch* operator, however, the second value of *A* was received for the *ndmerge* operator, that had the first value of *A*, now with the second value of *A*. The data flow of the values of *A* in the dataflow graph correspond to the C program of series as described in Figure 14 but to understand how to control the first value of *A* and the second value of *A* it was a fundamental part of learning about how

to program in the dataflow model.

Associated with this considerations about variable and allocation is how to use memory? In this case, just represent the memory as a black box and suppose that item of data can be loaded from the black box or stored into the black box.

B. Executing a Dataflow Graph

After to understand all kind of operators and their rules of execution and how simple is to define a dataflow graph, it is necessary to define how to execute a dataflow graph. There were several kind of dataflow machine, but recently, because of the advent of the FPGA, some machines were implemented using this kind of devices, since the dataflow is naturally parallel and the FPGA has its constitution also parallel based on cells interconnected by buses and switches. Then the next step to learn is: How nodes communicate with other nodes in the dataflow graph? To understand this question, initially is necessary to define that, for this tool, a specific kind of static dataflow machine, where just one item of data can be in an arc, will be considered. The next step is to show how is the basic implementation of an operator into a FPGA. It is interesting to show the VHDL program of several operators to understand each input and output signal of the operators and how they works. With this explanation, it will be clear that for each input or output signal of an operator, there are one data signal with several bits (in this case 16-bits), one strobe signal (to inform to the next operator that there is a item of data to it) and one acknowledge signal (to inform to the previous operator that it can or can not send another item of data). This step will show that for each arc in the dataflow graph, there are 16-bits of data, 1 bit for strobe and 1 bit for acknowledge and then for a real implementation, it will be necessary to connect all of those signals for each arc into the dataflow graph. It should be good to try to implement some examples using all the signal for a real implementation into the FPGA, directly in VHDL before to use the assembler tool. After to understand all the problems with those signal, then explain the assembly language and how to use the assembler tool.

C. Learning how to use the Assembly language and Assembler tool

Before to use the assembly language and the assembler tool, some steps need to be done. The first, obviously is to convert the application program into a dataflow graph. It is very interesting to draw the dataflow graph because it is easy to define internal sequential signals s1, s2, s3.. and so on. After to draw the dataflow graph, just generate a file.txt with the operators and their operands to be used in the assembler tool. Finally, execute the assembler tool with the file.txt that convert the assembly language into a VHDL program where operators are defined as component, and the interconnections as instances of the different components. Then the file.vhd is gener-

ated to be synthesized and implemented into a EDA tool like Xilinx or Altera.

VI. CONCLUSION

Once more, the researchers are interested in dataflow systems and with the advance in dataflow researchs, to learning how to program a dataflow language to facilitate the understanding how dataflow is, the assembler tool with an assembly language were presented. The ChipCflow Project is a tool to convert C language in VHDL to accelerating algorithms. To develop the C compiler, an assembler tool with an assembly language was developed, based on Dataflow operators, that has been used to programming applications in the Dataflow model to be configured into a FPGA as a support for the C compiler. Initially, the assembler tool was used as a support for the C compiler, however, for several times, it was used for graduate and undergraduate students in the ChipCflow Project to programming and to understand how dataflow program is. In this paper the assembler tool and assembly language were presented as a learning tool for programming Dataflow model. Some steps to teach and to learn about how to program in dataflow model was presented, and examples were discussed and results of the implementation also was described.

REFERENCIAS

- [1] A. H. Veen, *Dataflow Machine Architecture*, ACM Computing Surveys, 1986.
- [2] Jack B. Dennis, *A preliminary architecture for a basic dataflow processor*, ACM Computer Architecture News - SIGARCH'74, 1974.
- [3] C. C. Gurd, *The Manchester prototype dataflow computer*, Comm. ACM, 1985.
- [4] Michael J. Flynn, *Dataflow supercomputing*, Conference on Field-Programmable Logic and Applications (FPL), 2012.
- [5] Arvind Dataflow: Passing the token, *The 32th Annual International Symposium on Computer Architecture (ISCA Keynote)*, ACM, Madison, USA, pp. 1-42, 2005.
- [6] Swanson, S. and Schwerin, A. and Mercialdi, M. and Petersen, A. and Putnam, A. and Michelson, K. and Oskin, M. and Eggers, S. J. *The Wavescalar Architecture*, *ACM Transactions on Computer Systems*, ACM, pp. 4:1-4:54, 2007.
- [7] Veen, A. H. *Dataflow Machine Architecture*, *ACM Computing Surveys*, ACM, pp. 365-396, 1986.
- [8] Teifel, J. Rajjit, M. *An asynchronous dataflow FPGA architecture*, *IEEE Transactions on Computers*, IEEE, pp. 1376-1392, 2004.
- [9] Silva, A. C. F. ; Silva, B. A. ; LOPES, J. J. ; SILVA, J. L. *The Chipflow Project to Accelerate Algorithms using a Dataflow Graph in a Reconfigurable System*, *WSEAS Transactions on Computers*, WSEAS, pp. 265-274, 2012.

Diseño Novedoso de Neuronas Digitales de Impulsos en FPGA para procesamiento en Tiempo Real

S. Barrios-dV¹, J. J. Raygoza-Panduro², E. C. Becerra-Álvarez³, A. Salamanca-Chavarín⁴,
y E. A. Castellanos-Alvarado⁵

Resumen— En este artículo se presenta un diseño electrónico de neurona digital de impulsos implementado en FPGA. Este es capaz de replicar seis comportamientos biológicamente plausibles, responsables por el procesamiento de información en el cerebro de acuerdo al modelo de Izhikevich. Este nuevo diseño es único porque evita el uso de operaciones aritméticas, replicando las dinámicas de las neuronas mediante una máquina de estados finitos.

Palabras clave— Redes neuronales artificiales, Neuronas de impulsos, modelo de neurona, Arreglo de compuertas programable en campo (FPGA).

de impulsos propuesta es un sistema de elementos de almacenamiento lógicos (flip-flops) cuyos arreglos de cables configurables pueden generar trenes de pulsos con distintos patrones.

El diseño de la neurona consiste principalmente en N celdas de memoria V que están indexadas por

$$i \in 0, 1, \dots, N - 1 \equiv \mathbf{V} \quad (1)$$

Y M celdas de memoria U que están indexadas por

$$j \in 0, 1, \dots, M - 1 \equiv \mathbf{U} \quad (2)$$

V representa el valor del potencial de membrana y U una variable de recuperación. Cada tipo de celdas están conectadas en cascada. Cada celda guarda un dato de cero, excepto por una de ellas que guarda un uno. La neurona tiene un reloj interno periódico $C(t)$ donde $t \in [0, \infty)$. Este reloj es un tren de pulsos, cada pulso debe ser idealmente lo más pequeño posible. En este trabajo se utilizó el siguiente reloj interno donde $T_{clk} = 83.3 \pm 3$ ns. Es decir, un tren de pulsos de 12 MHz con 3.6% de ciclo de trabajo.

$$C(t) = \begin{cases} 1 & \text{si } t = 0, 1 * T_{clk}, 2 * T_{clk}, \dots, \\ 0 & \text{otros casos} \end{cases} \quad (3)$$

La neurona recibe señales para representar estímulos. Estas tienen la forma de un tren de pulsos discretos como se ve en la siguiente ecuación.

$$S(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT_s) \quad (4)$$

Donde T_s no es constante y $T_s \in (-\infty, \infty)$. δ representa la función delta de kronecker, la cual vale 1 en el momento inicial y 0 en el resto del plano. La salida de la neurona (Y) es una señal con las mismas características que tienen los estímulos. Su frecuencia pueda variar de acuerdo a la configuración de la neurona y número de celdas de memoria M y N . Su valor es descrito por la siguiente ecuación:

$$Y = \begin{cases} 1 & V = M - 1 \\ 0 & \text{otros casos} \end{cases} \quad (5)$$

Un controlador de dirección de desplazamiento decide si el dato es movido a una celda con índice mayor, menor o se mantiene en la misma. Estos elementos también se pueden ver descritos en [6], sin

I. INTRODUCCIÓN

VARIOS modelos de neuronas han sido desarrollados con el objetivo de ser usados en redes neuronales artificiales para aplicaciones de ingeniería incluyendo [1], [2], [3] y [4] entre otros. La mayoría de los modelos comúnmente usados son descritos por ecuaciones diferenciales, como el modelo de Izhikevich [5], o son incapaces de reproducir las características responsables del procesamiento en neuronas biológicas. En cambio, Tetsuya Hishiki, Hiroyuki Torikai, Takashi Matsubara y compañía han desarrollado modelos de neurona de impulsos cuyas dinámicas son descritas por autómatas celulares síncronos y asíncronos ([6],[7],[8],[9],[10], entre otros). Estos modelos tienen ventajas en aspectos de bajo consumo de energía y menor tamaño del circuito ya que pueden ser implementados sin el uso de unidades de punto flotante.

En este trabajo se presenta un diseño inspirado por la metodología expuesta en [6], llamado "Rotate-and-Fire", pero modificada para reducir aún más el tamaño del circuito. Este nuevo diseño fue implementado en un arreglo de compuertas programables en el campo (FPGA, por el inglés "Field-Programmable Gate Array") y se muestran los resultados de su funcionamiento en tiempo real. Al igual que la "rotate-and-fire" [6], la neurona digital

¹Departamento de Electrónica y Computación CUCEI, Universidad de Guadalajara, E-mail: sergio.barrios@alumnos.udg.mx.

²Departamento de Electrónica y Computación CUCEI, Universidad de Guadalajara, E-mail: juan.raygoza@cucei.udg.mx.

³Departamento de Electrónica y Computación CUCEI, Universidad de Guadalajara, E-mail: edvinbecerra@gmail.com

⁴Departamento de Electrónica y Computación CUCEI, Universidad de Guadalajara, E-mail: saleccta.ale@hotmail.com

⁵Departamento de Ciencias Computacionales CUCEI, Universidad de Guadalajara, E-mail: adrianacastellanos@hotmail.com

embargo, en esta última el controlador de dirección toma una decisión al resolver funciones de borde. Las utilizadas para obtener comportamientos similares a las neuronas biológicas en dicho artículo se pueden ver en Eq. 6 y Eq. 7.

$$f_V(V) = \alpha([k_1V^2 + k_2V + k_3]) \quad (6)$$

$$f_U(V) = \alpha([k_4V + k_5]) \quad (7)$$

donde:

α = función que limita la ecuación entre -1 y $N - 1$.
 k = parámetros configurables.

Estas tienen una complejidad similar a las ecuaciones diferenciales del modelo de Izhikevich [5]. Como se puede ver en Eqs. 8, 9 y 10.

$$v' = 0.04v^2 + 5v + 140 - u + I \quad (8)$$

$$u' = a(bv - u) \quad (9)$$

$$\text{si } v \geq +30mV \text{ entonces } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \quad (10)$$

donde:

$'$ = derivada con respecto del tiempo.
 v = potencial de membrana.
 u = variable de recuperación.
 a = escala de tiempo de variable de recuperación u .
 b = sensibilidad de variable de recuperación u a las fluctuaciones sub-umbral del potencial de membrana.
 c = valor de reinicio que adopta el potencial de membrana después del impulso.
 d = valor de reinicio después del impulso de la variable de recuperación u .

Aunque la neurona "Rotate-and-Fire" busca reducir ocupación y consumo de potencia asociado con la resolución de ecuaciones diferenciales, ésta aún necesita determinar si el diferencial del potencial membranaral y el de la variable de recuperación son positivos o negativos para determinar en qué dirección deben rotar las cadenas de memorias. Esto requiere de una menor precisión en el cálculo y no necesita usar punto flotante pero aún requiere la misma cantidad de operaciones.

II. ARQUITECTURA DE LA NEURONA

La neurona propuesta en este trabajo (fig. 1) no requiere de ninguna operación aritmética, si no que puramente operaciones lógicas. Esta también utiliza las celdas de memoria U y V pero el controlador determina la dirección del desplazamiento basado en un registro A , una señal de estado y estímulos de entrada.

El controlador de dirección envía dos bits y una señal de reloj a cada grupo de celdas V y U . El primer bit indica si se habilita el movimiento del dato mientras que el segundo indica la dirección en que se moverá este.

El estado de la neurona es determinado por el simulador de comportamiento dinámico. Este consiste

de una máquina de estados finitos. El estado puede ser 0 indicando reposo o 1 indicando cierta actividad configurable. El simulador de comportamiento dinámico también envía una señal a las celdas U y V para cargar un valor de inicio.

Los valores a los que las celdas se reinician son cargados de las salidas de los selectores de valores iniciales. Estos convierten un valor de índice i y j en una cadena de ceros y un uno a ser cargado por las celdas de memoria. Los bloques se configuran con dos índices, uno por cada estado de la neurona.

El generador de pulsos recibe el dato de la celda V_{N-1} que también representa la salida Y , y la pasa por 15 pares de LUTs (tablas de búsqueda, del inglés "Look-Up Tables"). Cada par separado por 15 posiciones. Creando una señal retrasada por $\delta t = 3.0ns$, la cual cuando es igual a uno reinicia el valor de V . Cargando un nuevo valor a V_{N-1} , si este es 0 se observará un pulso con 3 ns de ancho en la salida Y . δt no es significativamente grande en comparación con el tiempo que tarda el dato en ser reflejado en el pin de la FPGA ($T_{FPGA} = 2.24ns$). Por lo que los pulsos de salida no son señales cuadradas ideales, en cambio se observa una respuesta analógica similar a la de su contraparte biológica. Sin embargo, es importante resaltar que las neuronas de impulsos codifican la información en el tiempo en que ocurren los pulsos, no en la forma de éstos.

III. MODOS DE OPERACIÓN DE LA NEURONA DIGITAL

La neurona propuesta se implementó con $N = 8$ y $M = 8$. Esta es configurada al escribir en un registro A con ancho de palabra de 14 bits. Los bits de 0 - 5 determinan el valor de reinicio de las celdas U , del 0 - 2 para el estado de reposo y 3-5 para el activo. Bits 6 - 11 funcionan de la misma manera para las celdas V . Finalmente, los bits 12 y 13 determinan el modo de operación descrito en la tabla I.

$$A = \{\text{Modo}, V_1, V_0, U_1, U_0\} \quad (11)$$

TABLA I
 MODOS DE OPERACIÓN DE LA NEURONA DIGITAL.

Bits	Modo
00	Integrador
01	Resonador
10	Biestable
11	Muerte Neuronal

En el modo integrador la neurona dejará el estado de reposo cuando la celda V de mayor índice contenga un uno ($V = N - 1$), es decir cuando se genera un primer impulso de salida en respuesta a los estímulos. Y esta cambiará de modo activo a pasivo cuando el uno llegue a la celda U de mayor índice ($U = M - 1$). Un cambio de estado reinicia las celdas U y V .

El controlador de dirección en modo reposo, incrementará el valor de U cada ciclo del reloj interno C .

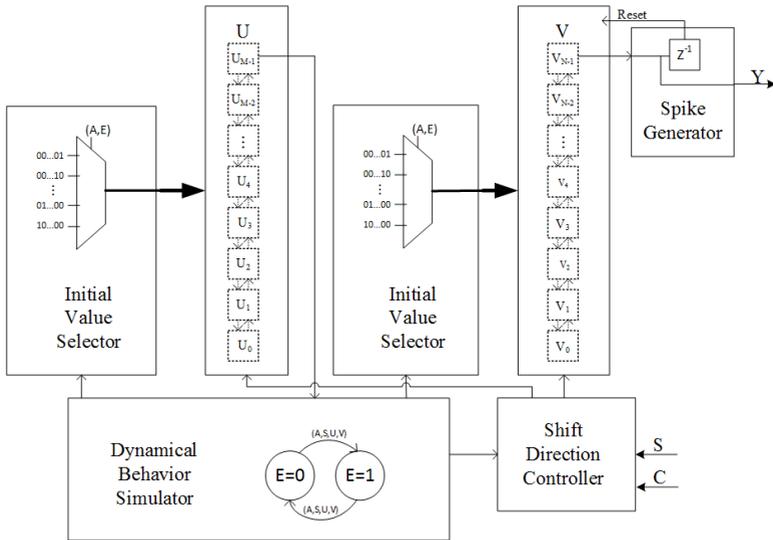


Fig. 1. Diagrama a bloques del circuito de la neurona digital.

Si este llega al máximo ($U = M - 1$) se reducirá el valor de V . El reloj interno no provoca un cambio en las celdas V , se usa una señal asíncrona que será disparada por un estímulo en la entrada o en caso de que U esté al máximo ($U = M - 1$) y V sea mayor a su valor inicial. En estado activo, el controlador de dirección incrementará el valor de las celdas U y V cada pulso del reloj interno C pero V también incrementará con los estímulos de entrada S . La dirección de V siempre estará aumentando, mientras que U se mantendrá, pero se incrementará cuando V este al máximo ($V = N - 1$).

En el modo resonador la neurona salta del estado de reposo al de actividad, cuando hay un pulso en el estímulo de entrada S al mismo tiempo que $U = M - 2$ o $U = M - 1$. Y al igual que el integrador regresa al estado de reposo cuando el valor de $U = M - 1$ y no hay pulso en S . Nótese que este valor U se reinicia cuando hay un cambio de estado, en la llegada de un estímulo o cuando $U = M - 1$. En estado de reposo el controlador de dirección mantiene incrementando el valor de U con C y constante a V . Una vez que la neurona entra al estado activo, V se comporta como en el modo integrador. En cambio U continúa comportándose como en el estado de reposo y provoca que la neurona vuelva a éste si $U = M - 1$.

Una neurona digital en el modo biestable cambia de estado de reposo a estado activo en la misma manera que en el modo integrador, pero solo regresa al modo de reposo si llega un pulso en el estímulo S mientras que $V = N - 1$ o $V = N - 2$. El contro-

lador de dirección también funciona como en el modo integrador en la mayoría de los casos. Con la única excepción de que en el estado activo, las celdas V no son afectadas directamente por los estímulos de entrada S . Esto debido a que el regreso al estado de reposo depende de S .

El último modo simplemente permite apagar la neurona. Esta no presentará respuesta en su salida y el comportamiento no cambiará debido a los estímulos de entrada.

IV. CARACTERÍSTICAS NEURO - COMPUTACIONALES EN LA FPGA

En [11], Izhikevich revisó 20 características neuro-computacionales que son prominentes en las neuronas biológicas. Y las replicó con su modelo matemático, probando así la certeza de su modelo. Se cree que estas son importantes para la forma en que el cerebro procesa información. En [3] y [6] se utilizó un enfoque similar para probar la validez de sus diseños de neuronas digitales. Igualmente, en este trabajo se replicaron 6 de estos comportamientos en la neurona artificial que se implementó en la FPGA Spartan-3E de Xilinx. A continuación, se describen las características logradas y se muestra una comparación del modelo matemático contra mediciones en osciloscopio de los resultados en chip. En las figuras, la señal verde representa la señal de salida Y y la azul el estímulo S .

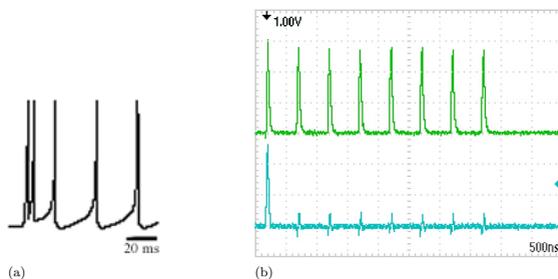


Fig. 2. Comportamiento de impulsos tónicos. a) en el modelo de Izhikevich y b) medido de la FPGA.

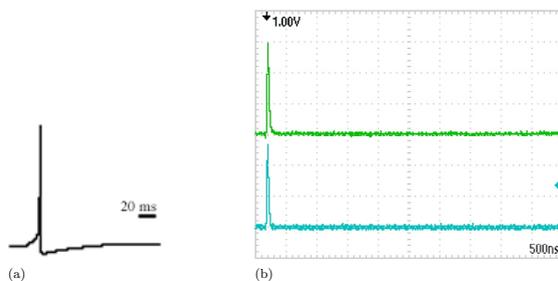


Fig. 3. Comportamiento de impulsos fásicos. a) en el modelo de Izhikevich y b) medido de la FPGA.

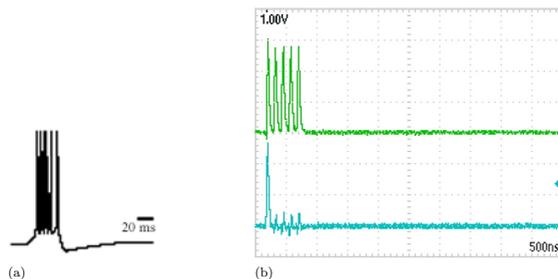


Fig. 4. Comportamiento de ráfagas fásicas. a) en el modelo de Izhikevich y b) medido de la FPGA.

A. Impulsos Tónicos

La mayoría de las neuronas son excitables, es decir, están en un estado de reposo, pero pueden disparar pulsos cuando son estimuladas. Las neuronas que exhiben este tipo de comportamiento se mantienen disparando un tren de impulsos mientras la entrada está activada. Los estudios neurofisiológicos usualmente prueban esta propiedad al inyectar corriente directa a la neurona. En sistemas biológicos el árbol dendrítico convierte los distintos impulsos de neuronas pre sinápticos en una sola entrada de corriente

directa. La neurona digital propuesta procesa impulsos directamente, por lo que cuando está configurada para un comportamiento de impulsos tónicos esta se mantiene disparando si continúa recibiendo pulsos en la entrada. Este comportamiento se puede obtener al cargar el valor hexadecimal $A = 0980$. En la fig. 2.a se ve la ilustración de esta característica reproducida por el Modelo de Izhikevich [11]. Y en la fig. 2.b se observa la medición en osciloscopio del mismo comportamiento exhibido por la neurona digital propuesta.

B. Impulsos fásicos

Una neurona puede disparar solamente un impulso al comienzo de una entrada y mantenerse en reposo después. Esta respuesta es útil para detectar el inicio de un estímulo. El comportamiento se logra al configurar la neurona con el valor hexadecimal $A = 01B8$. Funciona en modo integrador. Las celdas V son cargadas con el máximo valor antes de rebasar el umbral ($V = N - 2$), por lo que la neurona se dispara inmediatamente después de recibir un estímulo. U es inicializado con el valor más grande $U = M - 1$, por lo que al disparar un solo pulso éste vuelve al estado de reposo. Cada estímulo de entrada S es seguido casi inmediatamente por un solo pulso de salida Y , mientras que cuando no hay estímulos no se presenta actividad de salida. La comparación del modelo de Izhikevich y las mediciones en FPGA se pueden ver en la fig. 3.

C. Ráfagas fásicas

Estas neuronas disparan ráfagas de impulsos al inicio de un estímulo. Hay tres hipótesis sobre la importancia de ráfagas en el cerebro. Para compensar fallos en la transmisión sináptica y reducir ruido neuronal. También pueden transmitir la importancia de una neurona sobre otras. Esto debido a que las ráfagas tendrán un efecto más fuerte que un solo impulso sobre la neurona pos sináptica. Finalmente es posible que las ráfagas se usen para comunicación selectiva entre neuronas, donde la frecuencia entre impulsos codifica el canal de la comunicación.

La neurona digital puede reproducir este comportamiento con el valor hexadecimal $A = B98$. Funciona en modo integrador. El índice de inicio $V_0 = N - 1$ provoca que la neurona se active en cuanto llega el primer estímulo, pero esta vez el valor $V_1 = N - 4$ en estado activo causa que haya tres ciclos de reloj interno entre cada pulso. Adicionalmente el valor $U_1 = M - 6$ limita el número de pulsos a 5 por ráfaga. Cada estímulo de entrada provoca una de éstas, característica que se puede ver en la fig. 4.

D. Preferencia de una frecuencia y resonancia

Debido al fenómeno de resonancia, neuronas con potenciales oscilatorios pueden responder selectivamente a las entradas que tienen contenido en frecuencia similar a sus propias oscilaciones sub-umbral. Estas neuronas pueden implementar interacciones moduladas en frecuencia (FM) y multiplexado de señales. Estas neuronas son llamadas resonadoras. Este tipo de actividad se obtiene con la palabra de configuración hexadecimal $A = 1DB5$. Se utilizó el modo resonador. Con este modo el valor de V se mantiene constante (en el valor de inicio $V_0 = N - 2$) durante el modo de reposo siendo inmune a los estímulos de entrada.

En cambio, la variable de recuperación U , en este caso inicializada en $U_0 = M - 3$, se mantiene incrementando y una vez que llega al valor máximo ($U = M - 1$) se reinicia. Así el valor de U se mantiene oscilando con cierta frecuencia programable. Ya que

U se reinicia en cuanto llega al valor máximo, este no pasa una cantidad de tiempo significativa en la última celda. Así el periodo de oscilación es igual a $(M - 1) * T_{clk}$. Cuando un estímulo llega, este reinicia el valor de la variable de recuperación $U = U_0$, si un segundo pulso ocurre después del primero, cuando $U = M - 2$, se dispara un cambio de estado. Adicionalmente si la frecuencia de los estímulos es un múltiplo de la frecuencia de la neurona, es decir es un armónico, el segundo pulso de todas formas llegará mientras que $U = M - 2$ después de haber dado una o varias vueltas al grupo de celdas. Por lo que también disparará la neurona.

Otras frecuencias llegarán con las celdas U en un valor distinto, provocando simplemente un reinicio que se asegura que los estímulos subsiguientes y las oscilaciones de la variable de recuperación estén en la misma fase. Hay que tener cuidado ya que todo el dominio de la frecuencia es dividido en M secciones, una de ellas dispara la neurona. Por lo que un valor pequeño de M tendrá poca resolución en el dominio de la frecuencia. Mientras que uno grande consumirá muchas celdas de memoria.

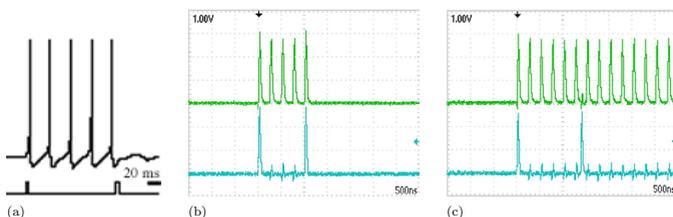
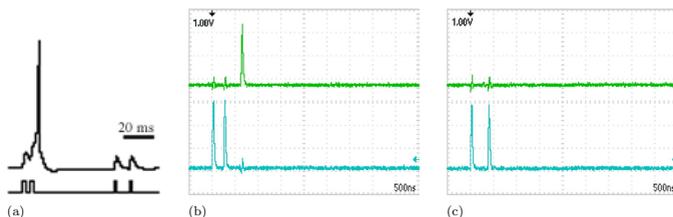
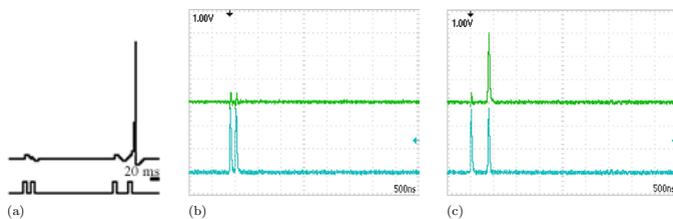
En la fig.5.a se puede ver esta característica en el modelo de Izhikevich. Dos estímulos rápidos no logran una reacción, esto también puede verse reproducido en FPGA en la fig. 5.b. En cambio, dos estímulos más lentos sí disparan la neurona como se ve en la fig. 5.c. Esto es contrario al comportamiento más común, el integrador, en el que entre más rápidos los estímulos, mayor el efecto en la neurona.

E. Integración y detección de coincidencias

Neuronas sin potenciales oscilatorios actúan como integradores. Prefieren entradas de altas frecuencias. Entre más alta la frecuencia es más probable que dispare. Esto es útil para detectar impulsos coincidentes o casi coincidentes. Este comportamiento es configurado con el modo integrador. A diferencia de los comportamientos anteriores que usan este modo, esta característica requiere de varios pulsos de entrada sucesivos antes de disparar la neurona. También es necesario que un estímulo que no active la neurona pierda su efecto a largo plazo.

Esta característica se configura con el valor de configuración hexadecimal $A = D7C$. Para obtener las imágenes se utiliza un valor de $V_0 = N - 3$ permitiendo que la neurona se dispare después de solo 2 estímulos de entrada. El valor $U = M - 5$ hace que después de 4 ciclos de reloj interno sin que haya estímulos, V disminuye. U se reinicia cada vez que hay un pulso en S , por lo que no habrá fuga de potencial de membrana si se mantienen los estímulos. En caso de que V ya esté en el valor V_0 no habrá fuga.

En la fig. 6.a se observa este comportamiento de acuerdo al modelo de Izhikevich. En el inciso b y c se observa el mismo reproducido en la FPGA.



F. Biestabilidad de estados de reposo y actividad de impulsos.

Algunas neuronas pueden exhibir dos modos estables de operación: reposo e impulsos tónicos (o inclusive ráfagas tónicas). Un pulso puede cambiar entre los dos modos, creando así una posibilidad interesante para la memoria a corto plazo. Nótese que, para cambiar de modo de impulsos tónicos a modo de reposo, el pulso de entrada debe llegar en fase con un pulso de salida.

Este comportamiento se activa configurando el valor hexadecimal ($A = 21B8$). En la fig. 7.a se muestra esta característica de acuerdo al modelo de Izhikevich. En b y c se presenta los resultados en chip. La neurona entra a estado activo con un estímulo y regresa a reposo con un segundo pulso de entrada solo si hay uno de salida al mismo tiempo.

V. CONCLUSIONES

En este trabajo se presenta una arquitectura para neuronas de impulsos en hardware reconfigurable.

Se demostró que esta implementación es capaz de reproducir 6 características que se consideran importantes para el procesamiento de la información en neuronas biológicas. En [8], el grupo de Hishiki presentó 5 comportamientos por lo que el trabajo presentado logró una característica más. Pero reduciendo la complejidad al no tener que resolver funciones cuadráticas de borde. Aunque el equipo de investigación de IBM en [3] logró 11 comportamientos usando una neurona, este requiere de varias operaciones aritméticas. El diseño propuesto en este trabajo requiere únicamente de operaciones lógicas, logrando una reducción en consumo de recursos como se puede ver en la implementación en la FPGA.

El circuito de la neurona propuesta puede cambiar su tipo de comportamiento después de ser configurada, es por lo tanto capaz de tener aprendizaje en chip. La neurona propuesta fue implementada en una FPGA Spartan-3E de Xilinx, específicamente en el chip XC3S400A. Como se muestra en la fig. 8.

En la tabla II se indica la cantidad de elementos

lógicos utilizados, nótese que los porcentajes de ocupación están entre el 1 y 2 por ciento.

TABLE II
RESUMEN DE OCUPACIÓN DE RECURSOS DE UNA SOLA NEURONA DIGITAL.

Resumen de Utilización			
Elementos lógicos	Usados	Disp.	Util.
LUTs de 4 entradas	144	7,168	2%
“Slices” ocupadas	90	3,584	2%
“Slices” con Flip-Flops	20	7,168	1%

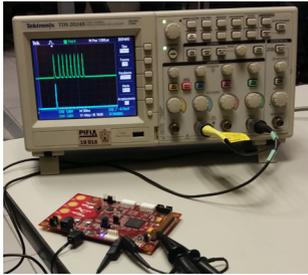


Fig. 8. Fotografía de la implementación de la neurona de impulsos en una FPGA Spartan-3E XC3S400A de Xilinx exhibiendo un comportamiento de ráfaga física.

AGRADECIMIENTOS

El presente trabajo ha sido apoyado por el Consejo Nacional de Ciencia y Tecnología (CONACYT).

REFERENCIAS

- [1] Garg, V., Shekhar, R. Garris, J. *The Time Machine: A novel spike-based computation architecture*, IEEE International Symposium of Circuits and Systems. 2011.
- [2] Grassia, F., Lévi, T., Tomas, J., Renaud, S., Saighi, S. *A Neuromimetic Spiking Neural Network for Simulating Cortical Circuits*, 45th Annual Conference on Information Sciences and Systems, IEEE. 2011.
- [3] Cassidy, A., Merolla, P., Arthur, J. *Cognitive Computing Building Block: A Versatile and Efficient Digital Neuron Model for Neurosynaptic Cores*, International Joint Conference on Neural Networks, IEEE. 2013.
- [4] Cabrera, H., Ortega, S., Raygoza, J., Rivera, J. *Implementation of a spiking digital neuron in reconfigurable hardware*, XIV Jornadas de Computación Reconfigurable y Aplicaciones. 2014.
- [5] Izhikevich, E. *Simple model of spiking neurons*, IEEE Transactions on Neural Networks. 2003.
- [6] Hishiki, T., Torikai, H. *Neural behaviors and nonlinear dynamics of a rotate-and-fire digital spiking neuron*, International Joint Conference on Neural Networks, IEEE. 2010.
- [7] Hishiki, T., Torikai, H. *A Novel Rotate-and-Fire Digital Spiking Neuron and its Neuron-Like Bifurcations and Responses* IEEE Transactions on Neural Networks. 2011.
- [8] Matsubara, T., Torikai, H., Hishiki, T. *A generalized Rotate-and-Fire Digital Spiking Neuron Model and Its On-FPGA Learning*, IEEE Transactions on Circuits and Systems. 2011.
- [9] Iguchi, T., Hirata, A., Torikai, H. *Integrate-and-Fire-Type Digital Spiking Neuron and its Learning for Spike-Pattern-Division Multiplex Communication*, International Joint Conference on Neural Networks, IEEE. 2010.

- [10] Matsubara, T., Torikai, H. *A Novel Asynchronous Digital Spiking Neuron Model and its Various Neuron-like Bifurcations and Responses*, International Joint Conference on Neural Networks, IEEE. 2011.
- [11] Izhikevich, E. *Which Model to Use for Cortical Spiking Neurons?*, IEEE Transactions on Neural Networks. 2004.

Arquitectura Paralela de Grano Fino para la Aceleración de Predicciones mediante Factorización Matricial

Arturo Durán Domínguez,¹ Juan Antonio Gómez Pulido² y David Rodríguez Lozano³

Resumen— Este artículo explica una propuesta de circuito paralelo de grano fino, implementado sobre FPGA, como base para un sistema acelerador de predicciones en los sistemas de recomendación. Específicamente, en esta propuesta se ejecutan predicciones paralelas basadas en multiplicaciones matriciales en punto flotante según modelos construidos a partir del proceso de factorización matricial y el algoritmo de gradiente en descenso. Este circuito se ha enfocado a las operaciones matriciales involucradas en un tipo de sistema de recomendación: el problema de la predicción del rendimiento del estudiante, donde se diseñaron dos circuitos de prueba, que ofrecen rendimientos según dos niveles de paralelismo. En nivel más alto, se ejecutan varias predicciones en paralelo, multiplicando al mismo tiempo distintos vectores de dos matrices. En nivel más bajo, correspondiente a la propuesta de arquitectura paralela de grano fino, cada una de estas operaciones vectoriales es ejecutada en paralelo multiplicando parejas de valores en punto flotante, sumando después los resultados correspondientes también en paralelo. De esta forma, el rendimiento global obtenido por el circuito acelerador superó al obtenido en dos microprocesadores de uso habitual, desde un factor de aceleración de $\times 20$ hasta $\times 100$, dependiendo del modelo de dispositivo FPGA, tipo de microprocesador o tamaño de matriz considerados.

Palabras clave— Acelerador, FPGA, paralelismo, aritmética punto flotante, factorización matricial, predicción, sistemas de recomendación.

I. INTRODUCCIÓN

LOS Sistemas de Recomendación o *Recommender Systems* (RS) [1] representan un dominio de investigación donde muchos esfuerzos se dirigen a elaborar recomendaciones personalizadas a los usuarios de grandes bases de datos, principalmente según su comportamiento cuando solicitan y manejan información de dichas bases de datos. Para ello, se utiliza el análisis de datos (*Data Analytic*) y técnicas de aprendizaje máquina (*Machine Learning*).

Las técnicas algorítmicas involucradas en los sistemas de recomendación se enfocan a propósitos de predicción, de aquí que puedan aplicarse también a otros ámbitos donde el conocimiento del comportamiento de los usuarios es importante, no solo para realizar recomendaciones, sino también para predecir. Este es el caso del problema de la predicción del rendimiento del estudiante, *Predicting Student Performance* (PSP), donde el rendimiento del estudiante se predice para determinadas tareas involucradas en

el proceso académico [2]. De esta forma, el problema PSP puede abordarse como un problema de predicción de rankings en los sistemas de recomendación, donde se suelen aplicar técnicas algorítmicas basadas en la factorización matricial.

La factorización matricial, *Matrix Factorization* (MF) [3] [4], es un método de predicción muy útil en los sistemas de recomendación, por lo que puede aplicarse para resolver el problema PSP. La factorización matricial realiza una aproximación a la predicción como una combinación lineal de factores, permitiendo buena escalabilidad, y puede incluir diferentes técnicas algorítmicas, como las redes neuronales [5]. A partir de esta factorización, se genera un modelo con el cual se pueden hacer predicciones. Dicho modelo de predicción se compone de dos matrices, de forma tal que, cuando se hace una predicción para un determinado estudiante y tarea, se multiplican las correspondientes fila y columna en la primera y segunda matriz, respectivamente.

Una de las características más importantes de los RS es la gran cantidad de datos involucrados, debido al elevado número de usuarios en bases de datos online o de estudiantes en un campus virtual. Manejar estos datos puede requerir grandes recursos computacionales, especialmente si se quieren satisfacer muchas peticiones de usuarios simultáneamente, en tiempo real. Para acelerar las predicciones, se puede explorar la aplicación de circuitos hardware diseñados específicamente para este propósito.

Los sistemas de recomendación pueden acelerarse utilizando la característica del paralelismo real que proporciona el hardware, siguiendo dos posibilidades. Por un lado, si se quiere ejecutar varias predicciones en paralelo, se deben multiplicar simultáneamente distintos vectores de las matrices involucradas en el modelo de factorización matricial. De esta forma, se pueden satisfacer solicitudes de predicciones en paralelo replicando los circuitos que implementan la multiplicación de vectores en el modelo de predicción. Por otro lado, a más bajo nivel se puede implementar un circuito que realice una predicción multiplicando en paralelo las correspondientes parejas de elementos en la fila y columna seleccionadas, y después sumando los resultados también en paralelo. De esta forma, si se diseña un sistema considerando estos dos niveles de paralelismo, se puede incrementar notablemente el rendimiento global del circuito hardware respecto al proporcionado por sistemas de computación basados en los microprocesadores más habituales.

¹Campus Virtual, Universidad de Extremadura, e-mail: arduran@unex.es.

²Dpto. de Tecnología de Computadores y Comunicaciones, Universidad de Extremadura, e-mail: jangomez@unex.es.

³Servicio de Informática, Universidad de Extremadura, e-mail: drlozano@unex.es.

En este artículo se expone una aproximación a la aceleración de los sistemas de recomendación utilizando un recurso hardware bien conocido para realizar implementaciones paralelas: los dispositivos *Field Programmable Gate Array* (FPGA) [6]. La tecnología FPGA favoreció el auge del área de la Computación Reconfigurable, *Reconfigurable Computing* (RC) [7], que combina la flexibilidad del software con el rendimiento del hardware cuando se aprovecha la característica del paralelismo intrínseco. De esta forma, un circuito diseñado cuidadosamente para propósitos específicos aprovechando el paralelismo intrínseco del hardware, incluso en ámbitos algorítmicos o aritméticos, puede sobrepasar el rendimiento proporcionado por un microprocesador, *Central Processing Unit* (CPU), en similares condiciones experimentales, tal como se ha demostrado en muchas aplicaciones [8]. Además, se puede apostar por las FPGA en lugar de otras tecnologías competitivas, como son las unidades de procesamiento gráfico, *Graphical Processing Unit* (GPU), puesto que las FPGAs pueden ofrecer mejor rendimiento que las GPU en niveles de paralelismo de grano fino, y menor consumo energético en muchos casos [9].

En este trabajo se aborda una primera aproximación a la aceleración hardware de los sistemas de recomendación utilizando FPGAs para diseñar un circuito paralelo de grano fino que paraleliza multiplicaciones matriciales, y así permitir predicciones paralelas al replicar estos circuitos en un nivel más alto. Específicamente, el propósito de este trabajo se enfoca a algunos aspectos del problema. Por un lado, se aborda el fuerte componente de la aritmética en punto flotante presente en la multiplicación matricial, utilizando un conjunto de operadores optimizados de suma y multiplicación en punto flotante para las operaciones aritméticas paralelas. Por otro lado, se utilizan pequeñas matrices como bancos de prueba experimentales, dejando para futuros trabajos de investigación los casos de matrices grandes que puedan abordarse por soluciones hardware más sofisticadas, siempre y cuando la presente propuesta de aceleración resulte satisfactoria. Además, esta aproximación puede ser interesante cuando los sistemas de recomendación trabajan con requisitos de tiempo real y tamaños fijos para los modelos de predicción. Finalmente, se ha buscado obtener una alta aceleración con respecto a las CPUs habituales, así como un bajo consumo energético, lo cual es muy interesante en entornos de computación intensiva.

II. TRABAJOS RELACIONADOS

Tal como se ha mencionado anteriormente, los sistemas de recomendación son muy apropiados para ser paralelizados, no solo por la repetición de elementos de procesamiento paralelo para propósitos de predicción, sino también para paralelizar las tareas de multiplicación matricial. Este segundo camino es seguido para implementar un circuito paralelo utilizando FPGAs, mientras que el primero es simulado a partir de los resultados del primero. Este

enfoque es apoyado por tendencias investigadoras en Analítica de Datos [10], un dominio donde las FPAG encuentran líneas de trabajo prometedoras para explotar al máximo sus características propias. Por ejemplo, las FPGA pueden aplicarse al procesamiento analítico on-line, análisis de texto, y procesamiento de series temporales, entre otras muchas áreas.

Respecto a la multiplicación matricial en general, se han efectuado varios trabajos con diversas aproximaciones en FPGAs, siguiendo tres objetivos: mayor velocidad, menor ocupación y consumo energético. La mayoría de estas implementaciones tratan con valores en punto fijo para las operaciones aritméticas [11] [12] [13], mientras que se han hecho menos esfuerzos en el dominio específico del punto flotante [14]. En este sentido, la presente propuesta encuentra acomodo con las necesidades habituales del cálculo en coma flotante que están presentes en la predicción de rendimientos, debido al tipo de datos y operaciones algorítmicas involucradas en los RS.

Se han efectuado algunos intentos en implementar sobre FPGAs técnicas de factorización matricial, evitando [15] o permitiendo [16] [17] usar aritmética en coma flotante. En este contexto, la propuesta que se presenta en este trabajo se restringe a la fase de predicción del sistema de recomendación, en lugar de implementar el diseño del modelo utilizando factorización matricial. Más aún, la multiplicación matricial, tal como se enfoca en este trabajo, ejecuta multiplicaciones en paralelo de vectores de matrices, siempre considerando las características de aritmética en coma flotante. En este sentido, la arquitectura paralela de grano fino propuesta trata de contribuir a la línea de investigación de otros trabajos recientes que aplican FPGAs para acelerar técnicas específicas involucradas en los sistemas de recomendación, tales como el filtrado colaborativo basado en vecindad [18].

III. PROBLEMA PSP

El problema PSP, que es un ejemplo de RS, aborda la predicción del rendimiento de un estudiante para determinadas tareas. Sean los siguientes términos:

- **S**: Conjunto de estudiantes, donde s es el índice para identificarlos desde 1 a S estudiantes.
- **I**: Conjunto de tareas, donde i es el índice para identificarlos desde 1 a I tareas.
- **P**: Conjunto de valores de rendimiento p , representado por una matriz $S \times I$.
- D^{knw} : Conjunto de rendimientos conocidos.
- D^{unk} : Conjunto de rendimientos desconocidos.
- $D^{train} \subseteq D^{knw}$: Conjunto de rendimientos de entrenamiento u observados, utilizados para entrenar el modelo matemático que permitirá predecir los rendimientos desconocidos.
- $D^{test} \subseteq D^{knw}$: Conjunto de rendimientos de prueba que se escogen para validar el modelo matemático, utilizando un determinado criterio, como *Root Mean Squared Error* (RMSE) (1). D^{test} suele ser mucho más pequeño que D^{train} .
- $\hat{P}_{|D^{test}|}$: Conjunto de predicciones de rendimiento para el conjunto de prueba.

$$RMSE = \sqrt{\frac{\sum_{s,i \in D^{test}} (p_{s,i} - \hat{p}_{s,i})^2}{|D^{test}|}} \quad (1)$$

El problema PSP se resuelve encontrando el modelo P con mínimo $RMSE$. La MF genera este modelo considerando una sola relación “estudiante - resuelve - tarea”, representada por $R = (S; I)$. En esta relación, se considera que existe un determinado número de factores latentes K , cuyo valor es a priori desconocido. No obstante, se puede hacer una estimación de este valor considerando, por ejemplo, factores como “acierto” o “fallo”.

La MF aproxima P al producto de dos matrices más pequeñas $W1$ y $W2$ (2) de tamaños $(S \times K)$ y $(I \times K)$ respectivamente.

$$P \approx W1W2^T \quad (2)$$

El rendimiento p obtenido por el estudiante s en la tarea i se predice como $\hat{p}_{s,i}$ según (3), donde k identifica el factor latente desde 1 a K .

$$\hat{p}_{s,i} = \sum_{k=1}^K (w_{1,s,k}w_{2,i,k}) = (W1W2^T)_{s,i} \quad (3)$$

El sistema se predice a partir de este modelo mediante tres pasos. Primero, se buscan los mejores parámetros para las matrices $W1$ y $W2$ en la fase de aprendizaje, utilizando D^{train} . Los parámetros óptimos se calculan midiendo las diferencias entre los valores predichos y reales, mediante un algoritmo iterativo. Una vez obtenido el modelo, se comprueba su grado de calidad prediciendo los valores de D^{test} y calculando la adiferencia con los valores reales mediante el criterio $RMSE$. Por último, se calculan los valores desconocidos de la matriz de rendimientos a partir del modelo óptimo encontrado.

IV. CIRCUITO ACELERADOR

En el sistema acelerador se pueden encontrar dos niveles de paralelismo.

A. Paralelismo en la multiplicación matricial

En el nivel más bajo, se ha diseñado un circuito de predicción que calcula la predicción $\hat{p}_{s,i}$ en dos fases consecutivas. Primero, la fila s en $W1$ se multiplica por la columna i en $W2$ por medio de K multiplicadores paralelos en coma flotante, cada uno de los cuales tiene como operandos los valores $w_{1,s,k}$ y $w_{2,i,k}$. Después, los resultados de los multiplicadores se suman en paralelo utilizando $K/2$ sumadores en coma flotante, utilizando para ello los necesarios pasos secuenciales hasta alcanzar la suma final. Por tanto, el circuito de predicción se compone de K multiplicadores y $K/2$ sumadores, y opera tanto en pasos secuenciales como paralelos.

Los elementos del circuito que operan a este nivel hacen que pueda ser considerado como un paralelismo de “grano fino”, pues no se han diseñado otros subsistemas que operen en paralelo.

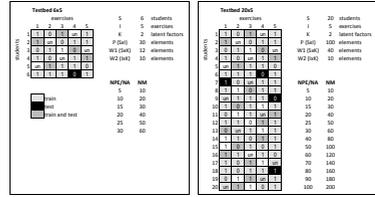


Fig. 1. Bancos de prueba para implementaciones FPGA.

B. Paralelismo para predicciones simultáneas

En un nivel más alto se puede replicar un cierto número NPE de circuitos de predicción. Este número depende principalmente del área disponible en la FPGA, fuertemente relacionado con el número de factores latentes. Estos circuitos de predicción operan en paralelo, proporcionando los resultados de la predicción de forma simultánea. Por tanto, el número de multiplicadores (NM) y sumadores (NA) en coma flotante necesarios para todo el circuito es $NPE \times K$ y $NPE \times K/2$, respectivamente.

Es importante señalar que las medidas de rendimiento de este nivel son simulaciones, ya que la única implementación hecha sobre FPGA ha sido la correspondiente al nivel de paralelismo de grano fino, que desarrolla la multiplicación matricial.

C. Bancos de prueba

La figura 1 muestra los dos bancos de prueba considerados para las arquitecturas de prueba en FPGA. Estos ejemplos corresponden con matrices de rendimiento de distintos tamaños, identificados por $S \times I$, donde el primero (6x5) permite utilizar dispositivos FPGA de distintas características, dado su menor tamaño, mientras que el segundo (20x5) solamente puede ser implementado en dispositivos de alta capacidad, si se quiere paralelizar más de 30 predicciones simultáneas. Esta figura detalla, además, los valores de rendimiento conocidos y desconocidos, así como los escogidos para propósitos de entrenamiento y predicción. Además, se muestra el valor máximo posible para el número predicciones paralelas (NPE), teniendo en cuenta los recursos hardware específicos considerados en los experimentos.

Las figuras 2 y 3 muestran el diseño del circuito acelerador para los bancos de prueba 6x5 y 20x5 respectivamente. El circuito incluye ambos niveles de paralelismo (multiplicación matricial y predicciones simultáneas) y se compone de NM multiplicadores y NA sumadores en coma flotante. Las entradas a los multiplicadores provienen de las matrices $W1$ y $W2$, mientras que las entradas a los sumadores provienen de las salidas de los multiplicadores. La multiplicación matricial está paralelizada por completo, mientras que el cálculo de las predicciones simultáneas combina etapas paralelas y secuenciales.

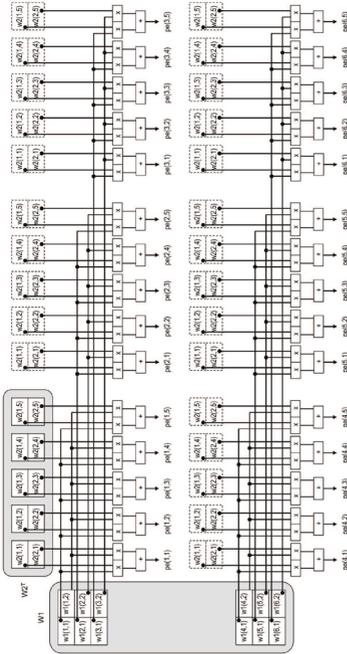


Fig. 2. Circuito de prueba 6x5.

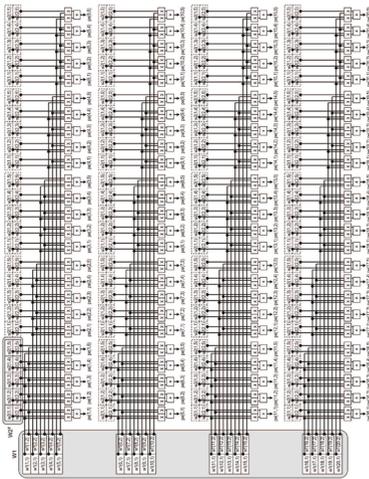


Fig. 3. Circuito de prueba 20x5.

D. Consideraciones de implementación

Se ha utilizado Xilinx ISE 14 para la simulación e implementación de los circuitos, *CORE Generator System* para los operadores aritméticos en coma flotante, y lenguajes HDL para el diseño de distintos módulos. Los circuitos fueron implementados en distintos dispositivos FPGA (tabla I), representativos de distintas características y capacidades, desde el bajo coste (Spartan6 xc6slx150) hasta el alto rendimiento (Virtex6 xc6vlx550t), pasando por prestaciones medias (Virtex5 xc5vlx330). Las características principales que los definen son: tecnología del proceso de fabricación (profundidad CMOS en nanómetros), número de celdas lógicas (relacionado con el área disponible para albergar el circuito), número de unidades DSP disponibles (para aumentar la velocidad de los operadores aritméticos en coma flotante) y número de bloques de memoria (tíles para albergar datos). Se escogieron también dos CPUs de distintas tecnologías.

TABLA I
 RECURSOS HARDWARE

Dispositivo	Datos			
FPGAs:	Tech.	Logic cells	DSP slices	RAM (kB) blocks
xc5vlx330	65nm	331,776	192	10,368
xc6vlx550t	40nm	549,888	864	22,752
xc6slx150	45nm	147,443	180	4,824
CPUs:	Tech.	GHz		
i7-2600	32nm	3.4		
Quad-Q9300	45nm	2.5		

Se diseñó el circuito de forma independiente al típico coprocesador asociado a un procesador empujado, cuestión que se deja para experimentaciones futuras. Para diseñar los operadores aritméticos, se consideraron las dos opciones de utilizar o no los DSPs internos, que elevan la velocidad de operaciones, pero disminuyen el área disponible. El compromiso velocidad/área debe ser evaluado en cada caso.

V. RESULTADOS EXPERIMENTALES

Esta sección describe los resultados experimentales, principalmente la aceleración conseguida por la FPGA respecto a la CPU, así como las estrategias de implementación seguidas.

A. Aceleración FPGA respecto a CPU

El análisis de tiempos de la implementación permite calcular el tiempo empleado por el circuito FPGA para completar las operaciones y, a partir de ahí, calcular la aceleración respecto a una CPU. El factor de aceleración o *speedup* es T_{CPU}/T_{FPGA} : la FPGA es más rápida que la CPU si el *speedup* es mayor que 1, de lo contrario la FPGA es más lenta.

Los valores T_{FPGA} y T_{CPU} miden el mismo número de predicciones. En el caso de T_{CPU} , se ha utilizado un bucle de ejecuciones secuenciales, mientras que T_{FPGA} considera *NPE* predicciones paralelas, donde el tiempo de ejecución de todas ellas es el mismo que el de una sola predicción.

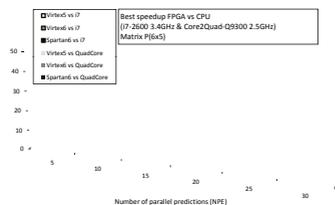


Fig. 4. FPGAs vs CPUs para la matriz 6x5.

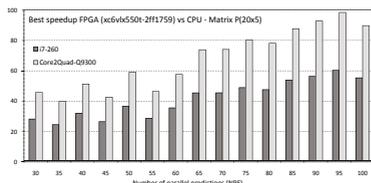


Fig. 5. FPGAs vs CPUs para la matriz 20x5.

La figura 4 muestra la aceleración de la FPGA para el caso 6x5, considerando todas las FPGAs. El rango de valores de *MPE* tiene en cuenta el máximo de predicciones que pueden operar en paralelo en la Virtex5, mientras que la Spartan6 solo puede alcanzar 15 predicciones. La Virtex6, en cambio, puede sobrepasar ampliamente esos números. Las distintas FPGAs proporcionan grandes aceleraciones en todos los casos (hasta x40), incluso contra la CPU de mayor prestación, por lo que se infiere la siguiente conclusión: cuantas más predicciones paralelas se consideren, mayor aceleración consigue la FPGA. La Virtex6 proporciona el mejor rendimiento, mientras que el dispositivo de bajo coste Spartan6 no permite albergar más de 15 predicciones paralelas.

La figura 5 muestra un análisis similar, esta vez para el caso 20x5, pero limitado al dispositivo Virtex6, porque es el único que permite hasta 100 predicciones paralelas. Tal como es esperable, se obtienen aceleraciones más altas que en el caso del primer banco de pruebas (hasta x98), debido al mayor área disponible y, por tanto, el mayor grado de paralelización para las operaciones matriciales.

No obstante, el incremento de la aceleración no es lineal ni ilimitado porque, a mayor número de operadores aritméticos en coma flotante, mayor densidad del circuito y, por tanto, menor frecuencia de reloj. Además, la figura 5 muestra el mejor resultado considerando las distintas opciones de síntesis e implementación para los operadores aritméticos, de ahí el aspecto no lineal de la tendencia del incremento. Por último, el número máximo de posibles predicciones paralelas para la Virtex6 se alcanza según el área disponible y los requisitos de memoria solicitados al computador en el proceso de síntesis.

B. Opciones de síntesis

Los informes del proceso de síntesis e implementación generados en ISE proporcionan información interesante para la escalabilidad y rendimiento de las operaciones matriciales. En primer lugar, la ocupación de área es dada por varios indicadores (*slice registers*, *slice Look-Up-Tables*, *occupied slices*) que permiten calcular el número de predicciones que se pueden ejecutar en paralelo en el mismo dispositivo. También, el rendimiento de tiempos se calcula a partir del valor de la frecuencia máxima correspondiente al periodo de reloj mínimo. Finalmente, el consumo de energía se calcula en la fase de emplazamiento y encaminado.

Cada síntesis se repitió varias veces siguiendo distintas estrategias, con el fin de obtener la frecuencia de reloj más alta posible. Por un lado, se consideraron tres perfiles de optimización para los procesos de síntesis e implementación (se descartaron otros perfiles de síntesis por sus peores resultados):

- *Default* (DEF).
- *Timing perf. with physical synthesis* (TPP).
- *Timing perf. without I/O blocks packing* (TPN).

Así, cada diseño se sintetizó hasta 6 veces (según los 3 perfiles de síntesis y las 2 opciones de utilizar DSPs en los operadores aritméticos), almacenando el mejor resultado entre estas 6 posibilidades. Por ejemplo, para la matriz 20x5 y Virtex6, se probaron 15 casos (30, 35, 40, 45 50, 55, 60, 65 70, 75, 80, 85, 90, 95 y 100 predicciones paralelas); por tanto, 15 casos x 3 perfiles x 2 optimizaciones de operadores = 90 implementaciones. Según el valor *NPE* considerado, el proceso de síntesis e implementación tardó entre 1 y 10 horas, también según el computador usado.

Las figuras 6 y 7 aportan información sobre las opciones de síntesis. La primera muestra la frecuencia de operación en Virtex6 respecto al número de predicciones paralelas, para los tres perfiles de síntesis considerados, usando DSPs o bloques lógicos en la implementación de los operadores. Las curvas siguen la tendencia de reducir la frecuencia cuando el número de predicciones aumenta, debido a la cada vez mayor densidad del circuito, aunque esta disminución es menos pronunciada a mayor número de predicciones. No obstante, frecuencias más bajas no implica peores aceleraciones, pues más predicciones paralelas ofrece mayor rendimiento (figura 7).

Ambas gráficas inducen a considerar para la síntesis el perfil por defecto, si se utilizan DSPs para los operadores aritméticos, de lo contrario la mejor opción es utilizar el perfil TPN en general.

VI. CONCLUSIONES

Se ha presentado una implementación en FPGAs de un circuito para predicción concurrente mediante multiplicación matricial, como primera aproximación a la aceleración hardware de los sistemas de recomendación. El diseño ejecuta multiplicaciones vectoriales rápidas donde están involucradas operaciones aritméticas en coma flotante.

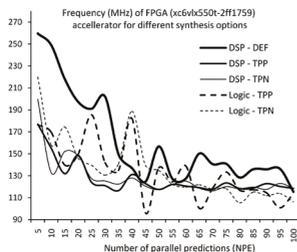


Fig. 6. Frecuencia FPGA y opciones de síntesis.

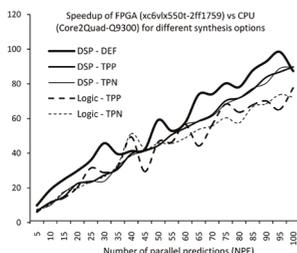


Fig. 7. Aceleración FPGA vs. CPU y opciones de síntesis.

Los resultados demuestran que las FPGAs proporcionan altas aceleraciones respecto a CPUs habituales, no solo debido a la paralelización de las operaciones matriciales, sino también gracias a las predicciones concurrentes. Además, el bajo consumo de energía de las FPGAs respecto a las CPUs facilita el diseño de soluciones de computación de bajo coste energético para escenarios de computación intensiva.

Esta aproximación puede ser interesante cuando los sistemas de recomendación abordan entornos de trabajo donde el tamaño del modelo de predicción es fijo, pero sus valores son actualizados frecuentemente, y donde se solicitan muchas predicciones simultáneas, en tiempo real. Más aún, la combinación de muchos dispositivos FPGA en la misma arquitectura permite ampliar el área disponible para considerar tamaños de matrices de rendimiento más grandes, que se corresponden con entornos más realistas. En este sentido, y gracias a los rendimientos obtenidos en esta aproximación de paralelismo de grano fino, se pueden plantear líneas de investigación futuras para construir soluciones más cercanas al mundo real, utilizando mejores dispositivos FPGA, compartiendo matrices más grandes en plataformas con múltiples FPGAs, y conectando las predicciones paralelas con aceleradores específicos para el proceso de factorización matricial y el algoritmo de gradiente en descenso, entre otras muchas posibilidades.

REFERENCIAS

- [1] D. Jannach, M. Zanker and A. Felfernig, and G. Friedrich, *Recommender Systems. An Introduction*, Cambridge University Press, 2011.
- [2] Nguyen Thai-Nghe, Lucas Drummond, Tomas Horvath, Artur Krohn-Grimberghe, Alexandros Nanopoulos, and Lars Schmidt-Thieme, "Factorization techniques for predicting student performance," in *Educational Recommender Systems and Technologies: Practices and Challenges*, pp. 129–153. IGI-Global, 2012.
- [3] Yehuda Koren, Robert Bell, and Chris Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.
- [4] Steffen Rendle and Lars Schmidt-Thieme, "Online-updating regularized kernel matrix factorization models for large-scale recommender systems," in *Proceedings of the 2008 ACM conference on Recommender systems*, 2008, pp. 251–258.
- [5] Gabor Takacs, Istva Pitasz, Botyan Nemeth, and Domonkos Tikk, "Scalable collaborative filtering approaches for large recommender systems," *The Journal of Machine Learning Research*, vol. 10, pp. 623–656, 2009.
- [6] C. Maxfield, *The Design Warrior's Guide to FPGAs. Devices, Tools and Flows*, Elsevier, 2004.
- [7] M. Gokhale and P. Graham, *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*, Springer, 2005.
- [8] D. B. Thomas, L. Howes, and W. Luk, "A comparison of cpus, gpus, fpgas and massively parallel processor arrays for random number generation," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, 2009, pp. 63–72.
- [9] S. Chey, J. Liz, J. W. Sheaffery, K. Skadrony, and J. Lach, "Accelerating compute-intensive applications with gpus and fpgas," in *Symposium on Application Specific Processors (SASP 2008)*, Anaheim, California, USA, 2008, pp. 101–107.
- [10] Rajesh Bordawekar, Bob Blainey, and Chidanand Apte, "Analyzing analytics," *SIGMOD Record*, vol. 42, no. 4, pp. 17–28, 2014.
- [11] Ju Wook Jang, Seonil Choi, and Viktor K. Prasanna, "Area and time efficient implementations of matrix multiplication on fpgas," in *Proceedings of the 2002 IEEE International Conference on Field-Programmable Technology*, 2002, pp. 93–100.
- [12] Nirav Dave, Kermin Fleming, Myron King, Michael Pel-lauer, and Muralidaran Vijayaraghavan, "Hardware acceleration of matrix multiplication on a xilinx fpga," in *5th IEEE/ACM International Conference on Formal Methods and Models for Codesign, 2007 (MEMOCODE 2007)*, 2007, pp. 97–100.
- [13] Syed Manzoor Qasim, Shuja Ahmad Abbasi, and Bandar Almashary, "A proposed fpga-based parallel architecture for matrix multiplication," in *IEEE Asia Pacific Conference on Circuits and Systems, 2008 (APCCAS 2008)*, 2008, pp. 1763–1766.
- [14] Z. Jovanovic and V. Milutinovic, "Fpga accelerator for floating-point matrix multiplication," *IET Computers and Digital Techniques*, vol. 6, no. 4, pp. 249–256, 2012.
- [15] Seonil Choi and Viktor K. Prasanna, *Field Programmable Logic and Application: 13th International Conference, FPL 2003, Lisbon, Portugal, September 1-3, 2003 Proceedings*, chapter Time and Energy Efficient Matrix Factorization Using FPGAs, pp. 507–519, Springer, Berlin, Heidelberg, 2003.
- [16] L. Zhuo and V. K. Prasanna, "High-performance and parameterized matrix factorization on fpgas," in *Proceedings of the International Conference on Field Programmable Logic and Applications 2006 (FPL '06)*, 2006, pp. 1–6.
- [17] Wei Wu, Yi Shan, Xiaoming Chen, Yu Wang, and Haazhong Yang, *Reconfigurable Computing: Architectures, Tools and Applications: 7th International Symposium, ARC 2011, Belfast, UK, March 23-25, 2011. Proceedings*, chapter FPGA Accelerated Parallel Sparse Matrix Factorization for Circuit Simulations, pp. 302–315, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
- [18] Xiang Ma, Chao Wang, Qi Yu, Xi Li, and Xuehai Zhou, "An fpga-based accelerator for neighborhood-based collaborative filtering recommendation algorithms," in *IEEE International Conference on Cluster Computing (CLUSTER 2015)*, Chicago, IL, USA, 2015, pp. 494–495.

Aplicaciones y Retos de la Sociedad

Positioning System for Recreated Reality Applications implemented on a multi-processing embedded system

Patricia Martínez Mediavilla, Eugenio Villar Bonet
TEISA department
University of Cantabria
Santander, Spain
{pmartinez, villar}@teisa.unican.es

Abstract— In recent years, there has been growing interest in systems related to the spatial location of objects or people in defined environments. The continuous evolution of manufacturing technologies has enabled the development of more complex and powerful embedded systems. This work presents the implementation of a positioning system, based on high-performance video processing, with large computational loads, in a heterogeneous, Multi-Processing System-on-Chip platform, Samsung's Odroid. In order to support specification and analysis of the system, embedded Systems implemented on complex hardware/software platforms require an increasing level of abstraction at which designers work. In order to achieve this goal, a model-driven engineering methodology was applied in which the requirements and initial functionality were captured with UML-MARTE. Video processing is one of the areas where high-level modeling and analysis based on UML have a wider impact.

Keywords— *Embedded System, MPSoC, Heterogeneous platform, UML/MARTE, Positioning System*

I. INTRODUCTION

During recent years, systems and products related to location of objects in three-dimensional environments are being deployed in a large number of sectors, such as robotics, medicine, gaming, among many others. One of the main applications is augmented and virtual reality. To obtain an individual's position, these systems use many different devices such as cameras, accelerometers, gyroscopes, GPS, etc. The positioning system implemented (see [1] for details) in this document is able to locate an individual or object in any type of environment or light conditions. The characterization of a point in the three-dimensional space requires knowledge of its coordinates (x, y, z) within the environment where it is situated, relative to a reference position. It is based on image processing of the region of interest where the individual is located using algorithms to detect reference markers. When only one reference marker is visualized, the use of stereo cameras and epipolar geometry is necessary. Otherwise, when there is more than one marker or pattern available on the image, through trigonometric algorithms, by knowing the actual distance between markers, and having a single camera, it is possible to obtain the parameters to establish the target

position in the environment. These systems require the use of a platform with sufficient computing capacity to process the images of the cameras in real-time and with the lowest possible latency.

In order to allow the analysis and optimization of the positioning system, based on complex video processing algorithms implemented in a multi-processing platform, it is necessary to raise the level of abstraction. The idea is to reduce the design gap (between complexity and design productivity) associated with the use of this high functionality applications, starting the design process from a high-level model combined with functional codes. The COMPLEX methodology (detailed in [2]) for UML/MARTE [3] modeling and design space exploration has been proposed to capture the system requirements. It follows a Model-Driven Engineering (MDE) [4], component-based, software-centric approach. The methodology can completely model the system, keeping all the functional and non-functional aspects of the system components, HW and SW resources and functional allocation to the platform resources well-defined. From the model, it is possible to generate binary files for the target platform by using a SW synthesis tool. This document goes beyond the software synthesis by carrying out a detailed profiling analysis with an evaluation in seconds or data/seconds of each system stage.

This paper is organized as follows: Section II presents a method and system for spatial location. Section III describes the design-methodology applied to the positioning system, the high-level UML/MARTE model and SW synthesis. Section IV details the profiling of timing performance for the positioning system. Section V will close with the conclusions and future working lines.

II. POSITIONING SYSTEM TECHNOLOGY

A. State-of-the-art

Positioning technology based on direct vision with cameras and reference marker detection is presented in many studies. Reference markers can be printed or lighting patterns, working in the visible or inferred spectrum [5] or working as passive or active devices. These positioning systems require or are restricted by synchronization processes, indoor

This work has been carried out as part of the UC contribution to the Artemis 332913 CopCams project and it has been partially funded by the Spanish MinEco through the TEC2014-58036-C4-3-R (REBECCA) project.

environments [6], the use of more than one camera placed on the environment walls [7], and dimensional limitations.

There are other methods that do not require video processing, such as radio-frequency-based systems for locating and tracking users [8]. This method involves recording and processing electromagnetic waves to measure distances from objects. RF techniques can detect objects at greater distances than others; however, they are quite sensitive to interference and noise. Another example is the method based on light emitting pulses, similar to LIDAR systems, which estimate the distance through the time taken for a light pulse to be reflected. This uses head-mounted HTC and Valve Corporation Vive displays. It requires IR lasers and a bank of infrared LEDs, which work as emitters and a headset and two wireless controllers with sensors as receptors. Although it is very effective in indoor environments, it is difficult to extend to larger spaces and outdoor. Other systems use the same idea with Microsoft's Kinect technology [9], which obtain a depth map of the scene with an infrared laser and a monochrome CMOS sensor under any ambient light conditions; however, they have greater distance limitations.

This paper goes beyond all the methods mentioned and tackles their restrictions; the positioning system proposed is able to locate an individual or object in any type of environment, inside or outside, whatever the lighting conditions are, with longer distances between the user and patterns.

B. System Architecture

In this section, the architecture of the spatial location and orientation system for moving objects in a scene under all environmental conditions, is presented. As shown in Fig. 1, the system is mainly composed of markers (1), a stereo camera (2), an angle measuring device (3) and a digital signal processor (4).

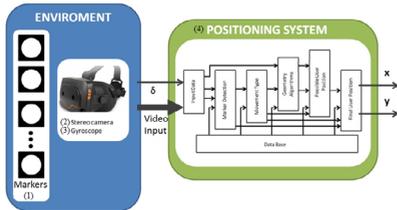


Fig. 1. Schematic of the System Architecture

The system is based on pattern detection through direct vision from a stereo camera to calculate the relative position of the individual. The reference markers are placed in the environment where the individual is located. Taking into consideration that the system works in different kinds of environments (outdoor and indoor, bright and dark light conditions and short and long distances) the markers must be designed for visibility by the camera in these situations. The proposed reference markers consist of two main elements: a

light source (such as a LED, which reaches distances greater than 50m and is considered a non-hazardous material) and a contrast surface (its dimensions and form depend on the lighting conditions, luminous flux of other light sources in the environments and the distance between the marker and the camera).

The user's view is controlled by a stereo camera, which displays markers in the image scene. The monitoring of the individual movement is completely characterized by an angle measuring device (such as a gyroscope, accelerometer or electronic compass) to provide the user's angle of rotation at each instant of time. If the positioning system is used for recreated reality applications, the stereo camera is set on the front of virtual reality glasses, such as for example, Oculus Rift, and the rotation angle is captured from its integrated gyroscope.

The input data captured from the environment are analyzed by a digital signal processor to determine the target object position in three-dimensional space. The system's functional aspects are divided into six components:

- **InputData** component: is in charge of getting the user's angle of rotation from a gyroscope and capturing images from the stereo camera. It includes an image algorithm to transform the image pair into rectified and undistorted images.
- **MarkerDetection** component: obtains the image coordinates of the reference markers and their radii. The method used to locate markers on an image is based on searching for the brightest pixels' areas and dark contours around them, thus finding higher and lower pixel values and the relations among them. In order to detect false positives, marker coordinate processing is necessary (implemented in the next stage), which is based on arithmetic mean and standard deviation operations.
- **MovementType** component: recognizes the type of movement performed by the user, whether perpendicular (from front to back or vice versa, red arrow in Fig. 2) or parallel (from right to left and vice versa, purple arrow in Fig. 2). The algorithm considers the variations of the markers' radii compared to the previous and current situations; similar values mean a parallel user movement, otherwise the user has moved perpendicularly.

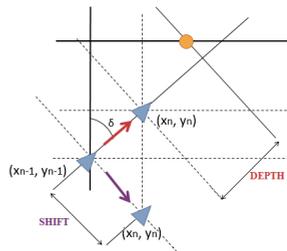


Fig. 2. User movement types

- Geometry component: estimates the user position by applying the positioning algorithms. The geometrical analysis performed depends on the movement type and the number of markers detected. In the event of a perpendicular movement, applying projective geometry (one marker on the image) or trigonometric algorithms (two or more markers), the algorithm returns the actual distance or depth between markers and user. On the other hand, the horizontal shift distance of a parallel movement is computed, from an algorithm based on stereo geometry (one marker detected) or a variation of linear triangulation (two or more markers) by using images captured consecutively with the same perspective. For better comprehension see Fig. 2.
- PossiblePosition component: computes the current possible individual position (x_n, y_n) in the environment considering the shift or depth values form the geometry component, the rotation angle and the previous user position.
- UserPosition component: obtains the target object or individual position in the three-dimensional environment. The correct position value from the preceding stage is selected taking into account movement type and whether user displacement and rotation have taken place.

III. DESIGN METHODOLOGY

In order to ensure modeling fidelity in the implementation of complex systems on multiprocessing platforms, design methodologies based on separation of concerns are required. The COMPLEX methodology for UML/MARTE modeling is implemented for this purpose. System modeling in UML/MARTE will be the first step in the design methodology workflow followed by the system implementation (see Fig. 3). The model development, with all its functionalities, implies a complete system definition, enabling automatic generation of the input code. Thus, once the complete model is implemented using eSSYN, a software synthesis tool, it is possible to create the executable files of the system by native or cross compilation – depending on whether the model will run in a host PC or on the multiprocessing platform. The system design flow includes the profiling of the timing performance for each component, in seconds or data/seconds.

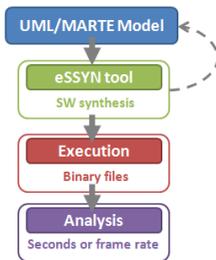


Fig. 3. Design methodology workflow

A. UML/MARTE Model

The complete model is based on graphical descriptions – views- to describe the system specifications, platform resources and resource allocation. The UML (Unified Modeling Language) collects all the required information about the system functionality, the HW/SW platform and the selected architectural mapping or resource allocation, all in a single-source. The MARTE profile captures all the specific characteristics related to the embedded system being designed, which provides UML with the required modeling features. In order to ensure the separation of concerns, the model is formed in three main parts, determined by their dependence on the target platform; each subdivision is associated with several views. A detailed explanation of all the model views is beyond the scope of this document; however, the most relevant ones are described to illustrate how the model has been developed.

1) Platform-Independent Model (PIM)

This model section describes the functional and non-functional aspects of the system, independently of where and how the system’s components will be implemented.

The positioning system functionalities are divided into six components (inputData, markerDetection, movementType, geometry, possiblePosition and userPosition) which are the basic building blocks of the system application, as was explained in the above section. The system components, their relationship and their interconnection through ports by the set of required/provided services are described in the *Application View*, as shown in Fig. 4.

These services, grouped into interfaces, are previously defined in the *Functional View* in order to specify the communication between components; moreover, this view incorporates the source code of the application components’ specifications (C/C++ code and their headers), each application component will be associated to the files (see Fig. 5). That is, these files store the implementation source-code of the application components described in the above section, there could be more than one file associated to each component.

2) Platform Description Model (PDM):

This part illustrates the hardware and software resources where the system specifications can be mapped. Thus, the PDM describes the HW/SW platform architecture.

The target platform is ODROID-XU3 octa core board, based on Exynos 5422. This board has a specialized architecture for digital imaging and multimedia embedded systems. It is a new generation computing device, an ARM big.LITTLE heterogeneous architecture, which combines two different kinds of cores: Cortex-A15 (big) for higher performance with complex tasks needing long computational time, and Cortex-A7 (LITTLE) for less intensive tasks and energy efficient computing. Thanks to the Heterogeneous Multi-Processing solution, all eight cores are freely available at the same time.

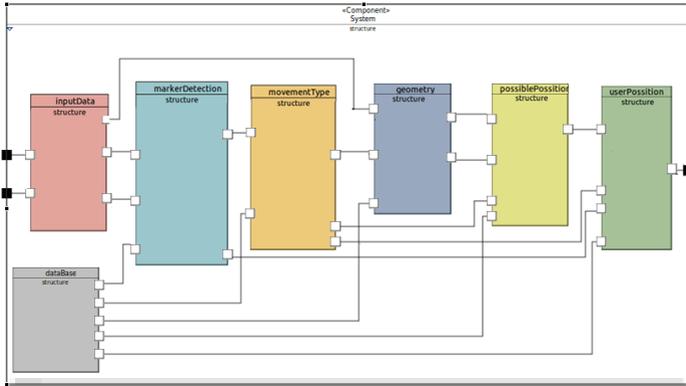


Fig. 4. Application structure

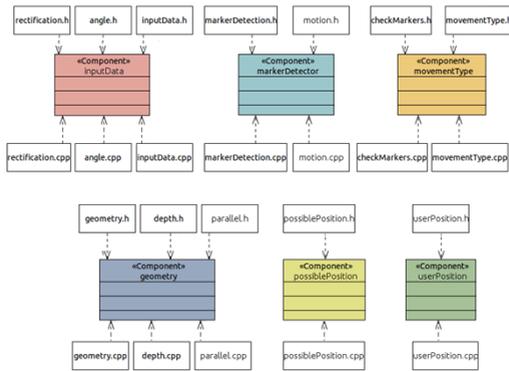


Fig. 5. File association.

The *HW Platform View* is used to describe the platform architecture; as shown in Fig 7. The target board has four Cortex-A7 processors (proc0-proc3) and four Cortex-A9 processors (proc4-proc7) connected to the 2GB LPPDD3 RAM through an AXI/AHB bus (main_bus).

The *SW Platform View* defines what is available in the HW/SW platform, a 'Debian-based' Linux operating system (OS), specifically.

3) Platform Specific Model (PSM)

This section describes the different allocations of the system. These allocations are application components to memory partitions and memory partitions to HW/SW platform resources.

The allocation of application-memory partitions consists of associating the application components defined in the Application View to the memory partitions. There will be as many memory partitions as executables in the

application system. The positioning system implemented will have just one memory partition (see Fig 6), where the six application components are associated.

The memory partitions to HW/SW platform resource allocation is the last step in the model. It is described in the Architectural View. Finally, when the SW and HW resources and the application components allocation to memory partitions are defined, the modelling methodology enable to associate this memory spaces in the selected platform resources. See Fig. 7, where the memory partition is allocated to the Linux Operating System.

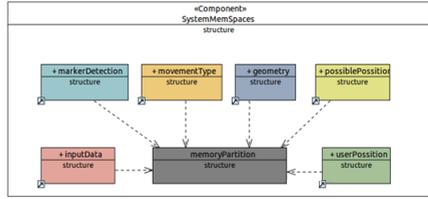


Fig. 6. Application components-memory partitions

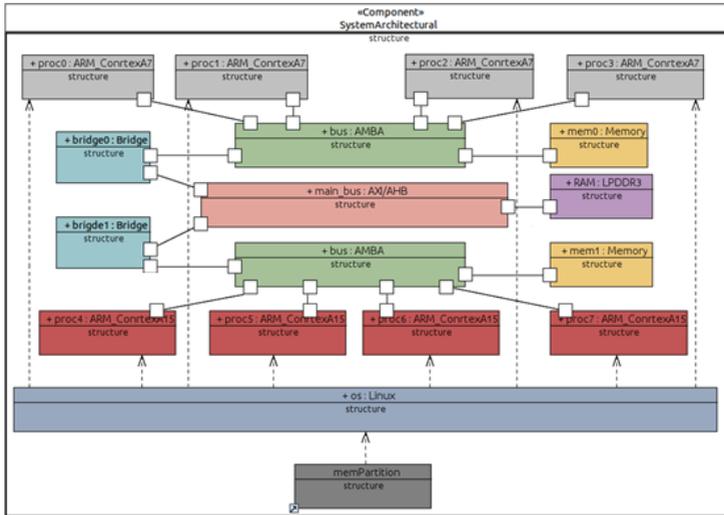


Fig. 7. Architectural View

B. SW Synthesis

Following the design methodology presented (Fig. 3), only when the whole UML/MARTE model is completely developed, can the SW synthesis be carried out.

The synthesis is performed by the eSSYN tool [10], which automatically generates platform-specific executable binaries from the component-based model of the positioning application and a simple model of the target hardware platform. The tool can create the binary files of the system

following these four steps: XML file generation, Wrapper file generation, Makefile generation and compilation (as shown in Fig. 8).

Firstly, the PC binaries files are generated by a native compilation of the model. Only when the corresponding executables work properly, the resulting binary files are ready for uploading into the host platform after cross-compilation.

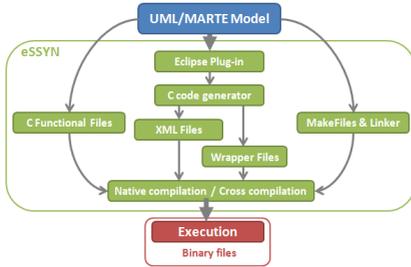


Fig. 8. Design methodology workflow

IV. RESULTS

The system design flow includes the profiling of the timing performance for each component. The goal is to achieve a correctly functioning system, in which the user does not notice the computational load. In augmented/virtual reality systems, such as re-created applications, there are three important limitations:

- The minimum frame rate required on the Virtual Reality glasses to avoid dizzy effects
 33 ms or less will provide the maximum level of latency deemed acceptable: in frequency terms this is 30 frames per second
- The minimum resolution to detect the markers in a space where the distance between the user and the marker could be at least 10 m
 640x480 pixels/image
- The maximum acceptable delay for updating user's position in the Virtual Reality engine, considering walking speed
 250 ms

These temporal restrictions have been included in the model. It can be said that the model fidelity has been satisfied by the implementation when the restrictions have been satisfied.

A first approximation to the model and a time analysis is performed, which could be compared with the ideal data. All estimated timing performance figures are shown in Table I, when processing VGA images (640x480 pixels). This time analysis is carried out on a desktop computer and on the selected platform (ODROID-XU3).

In spite of getting lower time values on the PC, its weight and size are not adequate for the positioning system, since it cannot be comfortably worn by a user. Instead of that, a portable platform is required. That is why the Odroid-XU3 is used in the final prototype.

	Component	input Data	marker Detection	movement Type	geometry
	Temporal restriction (ms)	<167	< 80	< 2	< 0,5
P C	Measured latency (ms)	81.104	14.222	0.088	0.010
B o a r d	Measured latency (ms)	137.381	59.154	0.114	0.019

TABLE I. TIMING PERFORMANCE

On the other hand, the time analysis demonstrates a good frame rate of the positioning system implemented; moreover, it reveals that the *InputData* and *MarkerDetection* components use more than 95% of the whole processing time. In order to reduce these execution times, two solutions arise. The first one, replacing the input data hardware, by a camera with USB 3.0 (this port is available on the Odroid-XU3), which has 10 times more data transfer speed than a camera with USB 2.0. The second, optimizing (i.e. parallelizing the code) using all the platform resources.

Thanks to the model implementation – just by changing interface services from sequential to concurrent UML operations (which communicates components) the system parallelization or pipeline can be easily created.

V. CONCLUSIONS AND FUTURE LINES

The advances during recent years in video display, video processing and graphic SW technologies have led to the emergence of modified-reality systems. The evolution in technologies (Moore's Law is still in place) able to support these virtual reality experiences, has brought about the growth of these systems. One of the main problems in all these applications is accurate positioning of the subject.

Many different methods have been proposed to date; some of them are valid only in small spaces, others only work indoors or outdoors and many others are light-condition dependent.

The University of Cantabria has developed a positioning system able to overcome most of the drawbacks found by other competing alternatives. The technique has been described and its UML/MARTE model outlined. At the end of this year, coinciding with the Artemis CopCams project ending, in which the UC collaborates, a prototype is expected, which will be able to assess the actual capabilities of the proposed positioning system and which could be integrated into a recreated reality application.

UML/MARTE has proven to be a powerful means to model this system based on high-performance video processing. From the UML/MARTE model the designer can decide about the system requirements, detect potential architectural problems, fully specify the different components

and based on this, start the platform-independent design. Once the whole model is developed and all the functionalities are defined, the model can be simulated and finally, synthesized by using the SW synthesis tool eSSYN. Then, based on a profiling analysis of a preliminary all-SW version, in very fast turnarounds, it is possible to verify whether the temporal restrictions imposed by the model are satisfied by the implementation. In other words, to what extent fidelity to the model has been achieved.

REFERENCES

- [1] Villar, E., Martínez, P., Alcalá, F., Sánchez, P., & Fernández, V. (2014). *Método y sistema de localización espacial mediante marcadores luminosos para cualquier ambiente*. P.N.ES-2543038-B2.
- [2] Herrera, F., Posadas, H., Peñil, P., Villar, E., Ferrero, F., Valencia, R. & Palermo, G. (2014). *The COMPLEX methodology for UML/MARTE modeling and design-space exploration of embedded systems*. Journal of Systems Architecture, V.60, N.1, Elsevier, pp.55–78.
- [3] OMG, UML Profile for MARTE: Modelling and Analysis of Real-Time Embedded Systems, Version 1.1, Dec., 2012. Available from: <<http://www.omgmarite.org>>.
- [4] Schmidt, D. C. (2006). *Model-driven Engineering*. IEEE Computer, V.39, N.2, pp. 25-31.
- [5] Naimark, L., & Foxlin, E. (2007). Fudicial Detection System. *Patent US 7,231,063 B2*.
- [6] Maeda, M., Ogawa, T., Kiyokawa, K., & Takemura, H. (2004). Tracking of user position and orientation by stereo measurement of infrared markers and orientation sensing. *IEEE, 8th International Symposium on Wearable Computers*.
- [7] Kumar, R., Samarasekera, S., & Oskiper, T. (2013). Method and apparatus for 3D spatial locazotion and traking of objects using active optical illumination and sensing. *Patent WO 2013/120041 A1*.
- [8] Bahl, P. & Padmanabhan, V. (2000). RADAR: *An In-Building RF-Based User Location and Tracking System*, *Proceedings of IEEE INFOCOM 2000*, Vol. 2
- [9] Nakano, Y. Izutsu, K., Tajistu, K., Kai, K., Tatsumi, T. (2012) *Kinect Positioning System (KPS) and its potential applications*. International Conference on Indoor Positioning and Indoor Navigation.
- [10] Peñil, P. (2014). *UML-Marte Methodology for Heterogeneous System design*. Microelectronics Engineering Group, TEISA Dpt., University of Cantabria.

Intelligent machine tool monitoring and control system

Aitor Duo Zubiaurre¹, Miren Illarramendi Rezabal², Rosa Basagoiti Astigarraga³, Pedro Arrazola Arriola⁴ Exabier Hormaetxe Fernandez⁵ Javier Aperribay Zubia⁶
Mondragon Goi Eskola Politeknikoa S.Coop
Embedded Systems Research Group
High Performance Manufacturing Research Group

Resumen— European manufacturing sector has always been a reference in delivering high quality products. With the digital transformation of Industry – or Industry 4.0, Industrial Internet – it now has the potential to lead the world in a new kind of technological transformation. The current industrial environment presents many kind of issues that must be resolved in order to achieve the objectives proposed by the Industry 4.0. One of the steps for this technological transformation is the ability to monitor and control the industrial machinery and the related machining tools. This article shows the methods in which the monitoring system can be developed in order to capture and visualize the data from both the manufacturing machines and the machine tools. A real implementation and results of a manufacturing machine monitoring and control system is shown. In this way, it is hope to improve the manufacturing processes, manufactured products and get a general vision and real information of the entire process. The Health Monitoring of the Manufacturing machines, processes and tools will be done in Real Time.

Palabras clave— Decision-making, IPC, Middleware, Automation, CNC, PLC, Monitoring software.

I. INTRODUCTION

NOWADAYS the ICTs (Information and Communication Technology) are gaining importance in people's daily life as well as in the organization of a society.

Regarding to the automation context, in last years the relation with ICTs has been getting closer. Because of that, a new concept called Industry 4.0 or Industrial Internet has been created. This concept, which is also called Intelligent Factory or Digital Factory, hopes to provide tools to adapt the needs that are created in the productive processes in real time.

Integrating the information and communication technologies on manufacturing processes and implementing tools that provide the ability to make a decision or a proposal to the operators in the factory plant, will add value to the product or

service that is providing an organization. This added value can be useful in order to improve the quality of a product or to reduce manufacturing costs.

The relation between the material and the tool used in machining processes is one of the critical concepts that the industry is interested on doing research. It is very important to select the correct tool and material in each process. If this selection is not the correct one, there can be a huge amount of cost in development, material and tools.

Thus, there is a need of using tools which allow to trace the manufacturing processes automatically. Those tools, offering a valid information to the manufacturer, provide the ability to ensure the proper operation of the machine. This, makes easier the decision-making for the plant operator and allows the detection of issues that can appear during those processes.

Among the different problems that present a manufacturing process, the life cycle of cutting tool is one of the critical issues. Once the machining tool starts with the machining process, it is impossible to know the real condition or health of it and also is very difficult to know when it is going to fail; this happens especially on large machining processes and the new trend is to include high number of sensors but in most cases it is difficult to put them in real industrial environments.

In order to monitor the signals of a machining process and bring the correct functionality of the machine, it has been detected the need of developing a monitoring system which will allow the possibility to decision-making faster and better.

The actual market on this area presents many of those tools. Each of them has its characteristics and its advantages but all of them present some disadvantages. In case there is a special need to modify the way in which the signals are processed or to customize the application logic, can have a large amount of cost.

Because of that, it has been developed a flexible and modular monitoring system. Using this system, it is possible to modify all parts of the system when it requires and adapt it to the needs of the machining processes as well as monitor large processes. In this way, it can be achieved a general vision of the process.

¹Embedded systems dept., Mondragon univ., e-mail: aitor.duo@alumni.mondragon.edu.

²Embedded systems dept., Mondragon univ., e-mail: mllarramendi@mondragon.edu.

³Embedded systems dept., Mondragon univ., e-mail: rbasagoiti@mondragon.edu.

⁴High performance manufacturing dept., Mondragon univ., e-mail: pjarrazola@mondragon.edu.

⁵High performance manufacturing dept., Mondragon univ., e-mail: ehormaetxe@mondragon.edu.

⁶High performance manufacturing dept., Mondragon univ., e-mail: japerribay@mondragon.edu.

The work done is focused on Fagor Automation machine tools. Using an API that Fagor provides in their machining environment, there is a possibility to acquire internal and external signals. Those signals are sent to the monitoring software in order to bring the correct functionality of a machine.

For this development, the research team has analyzed the technologies that make easier the communication between processes. These technologies are IPC (Inter Process Communication) as well as some middlewares that already exist in the ICT scientific community that allow on-line communication.

This paper shows how the signals are captured from the machining tool as well as the techniques used to develop the process communication. In section II the state of the technology will be shown. After that, a case study will be presented in the section III. In section IV the development of the use case will be shown and in V the results of the use case are explained. Finally, in section VI the final conclusions are presented.

II. STATE OF TECHNOLOGY

This paper hasn't the intention of establishing a theoretical basis. It hopes to achieve the objectives which have been explained on the previous chapter, starting from the technologies that are known. So it is important to review and understand the technologies and concepts that take place on the development of this system.

Nowadays, the machining tools are controlled from CNC (Computer Numerical Control) computers. The CNC computer also supplies sufficient technology to capture signals from the machining tools and facilitates the use of those signals for different purposes. The project to develop has its roots in the necessity of monitoring large machining processes and the need to know what is the state of the cutting tool in real time. For that, it is necessary to capture data from CNC and send to monitoring software in order to visualize or monitor the results. Thus, anyone can have a general vision of the manufacturing process.

In order to visualize the data captured from CNC, it must be shared between different processes. Starting from this point, this chapter describes the technologies used in order to communicate different processes. On one hand, IPC (Inter Process Communication) technologies are analyzed, which cover techniques to share data between different processes in the same PC. Two modes of communication will be shown: shared memory and message passing. On the other hand, two types of existing middleware will be shown: ICE (Internet Communication Engine) and zeroMQ (zero-em-queue). These middlewares can be interpreted as IPC techniques, but it is been decided to explain both concepts separately in order to differentiate them.

A. CNC machines

The first thing to analyze is the way the data is captured from CNC. In this point, it is important to understand the CNC and PLC concepts separately.

CNC machines are machine tools that are controlled from a computer. On these computer, it is installed a software that is called CNC and is the responsible for controlling the machine tool. [1]

On the CNC, it is executed the PLC (Programmable Logic Controller) software which has a modular structure and it's written in mnemonic language. It has the possibility to use C auxiliary functions to perform a secondary task.

On one hand there is an automatic change of information between the CNC and PLC software. The tasks that can be done by the PLC software on the CNC are the following ones:

- Control the physical I/O.
- Consult or modify the CNC-PLC interchange variables.
- Consult or modify the CNC internal variables.
- Visualize errors or messages in CNC screen.

On the other hand, the CNC can do the following functions on the CNC:

- Transfer M, H and S complementary functions.
- Consult the PLC resources from any piece program.

The M functions are used in order to do synchronized tasks and for Fagor machines it is possible to define seven M functions in the same block.

The H functions are used to perform tasks without synchronism.

The S functions are used to get the speed of the head and to establish the position of the head.

This M, H and S functions are used in the PLC program using the mnemonic .plc language. The research team has used the M functions in the PLC programs in order to detect when the caption of the data in the CNC is started and when stopped.

The PLC software has three subsections:

- CY1: Cycle which is executed on the initialization of PLC.
- PRG: It is the principal subsection of the PLC software. This is executed periodically and the minimum period that can be applied is 4ms.
- PE: This subsection is optional and is also executed periodically. It is used to perform tasks that are not evaluated in all the cycles of the PLC.

B. IPC (Inter Process Communication)

The technologies that are used to perform a communication between processes that are in the same machine are called IPC. There are two different methods to achieve the communication: shared memory or message passing. Each one is a collection of techniques that can be applied to get the desired result.

B.1 Shared Memory

As far as the shared memory is concerned, there is a case in which the processes are communicated using shared resources. This piece of memory is independent from the software that are taking place. One of the software writes the data to that memory and another one reads data from this memory. The figure 1 shows the way in which two processes are communicated. A process writes the data in shared memory and B process reads the data from this shared resource.

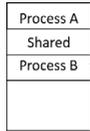


Fig. 1. Shared memory

Although it is said that the shared memory is independent from the processes that use it, this shared memory is located in the memory space of one of them.

If another process wants to write or read from this memory space, that memory has to be part of the process in question. This is because the operating system limits the access of a process to the memory of the other process. This is the reason why two processes have to reach an agreement in order to eliminate those limitations. [2]

Other difficulty that appears in this type of communications is the synchronization between processes. Consequently, the writing and reading operations must be ordered, so there are needed synchronization elements to achieve this objective.

B.2 Message passing

The message passing is a very used IPC technique. It permits to communicate different processes without use of the same memory space. It is very useful in order to communicate different machines in a network.

The message passing should have at least two operations, one to send message and another one to receive. Each operation must be blocking or non-blocking to allow synchronous or asynchronous communications between processes.

Showing figure 2 it is seen that process A has to communicate with process B. There must be a communication link between two processes and this link can be of different types. For example, a bus from the hardware or Ethernet.

One of the disadvantages of message passing is that there is the necessity of the Kernel to perform a communication, therefore the transfers are slower.

Following the communication logic there can be the following types of communication [2]:

- The communication can be synchronous or

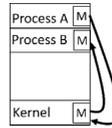


Fig. 2. Message passing

asynchronous. The operations mentioned previously are the following ones:

- Blocking send: When a message is sent the transmitter is blocked until there is a response.
- Non-blocking send: The transmitter doesn't expect any response and can continue sending messages.
- Blocking receive: The receiver is blocked until it is a new message.
- Non-blocking receive: Even if it isn't a received message the receiver continues executing.
- Queues are implemented to store messages during a period of time until the receiver process. Those can be implemented in three ways:
 - Zero-capacity: The queue has the capacity of length zero.
 - Bounded capacity: It permits fixed amount of messages and if it is not reach the maximum length, the transmitter continues sending messages.
 - Unbounded capacity: It permits infinite length of messages so the transmitter is always sending information.

C. TCP Middleware

This chapter describes different middlewares already available at the ICT domain. These middlewares are used in order to visualize/monitor the signals captured from CNC in a remote machine.

Leaving aside IPC technologies, the concept of middleware allows to separate the logic of the application from the logic of communication, making it easier to integrate communications between processes.

Although there are some more types of middlewares, only are taken into account RPC (Remote Procedure Call) and MOM (Message Oriented Middlewares) middlewares.

- RPC: With Remote Procedure Calls an application has the capacity of executing a procedure call in another process.
- MOM: Message Oriented Middlewares allow messages passing between processes and is implemented in hardware or software.

Even though there are middlewares in the market that can help in the purpose of the project, ICE, which is offered by zeroC and zeroMQ middlewares have been the analyzed ones.

On one hand, the one provided by zeroC which is called ICE [3] and it is RPC type. This framework provides a static file where the communication interfaces and the objects used during communication are defined by the developer. It makes possible the use of remote objects by calling a method which is previously defined in the interfaces of the static file.

On the other hand, zeroMQ which is message oriented middleware, is developed over the standard sockets. This middleware is written in C language and supports C++ or java languages among others. [4]

Apart from the send/receive functions, it provides tools to create complex architecture models as request-reply, publish-subscribe, etc.

Table I-C shows the results explained in Middleware trends and markets leaders article.

	Patterns	QoS	Resources	Performance	User Friendly	Community
ICE	✓	✓	✗	✓	✓	✓
zeroMQ	✓	✓	✓	✓	✓	✓

Table 1: MIDDLEWARE TRENDS AND MARKET LEADERS 2011

On the table, it is shown that ICE middleware provides less resources than zeroMQ. As it is explained in the article [5], the zeroMQ option expands less time publishing the same information to the same number of clients, therefore it has more performance.

In any case, later is explained the development with any of those options.

III. CASE STUDY

The basis of the objective is to develop an application that visualizes the signals captured from the CNC and consequently, obtain a tool that helps in the decision-making about manufacturing processes. This tool can also help controlling the manufacturing process automatically and warn to the operator about risky situations. Controlling the erosion of a machining tool is a way of achieving the best quality of the product to always maintain tools life cycle during the process. To that end, it is supposed a manufacturing process of large dimension piece. The monitoring tools used nowadays visualize little information of a manufacturing process, that's why it is impossible to show all the signals of entire manufacturing processes. If a risky situation happens in the process, it is impossible to know what is the reason of this situation and where it occurred. As explained before, the application has the responsibility to visualize the data read from the CNC. Furthermore, it must have the ability to warn about any risky situation that can happen during

the manufacturing process. In order to do that, it is necessary to develop an alarm system. On the figure 3 it is shown an example of a machined part, manufactured in three cutting conditions. C1, C2 and C3.

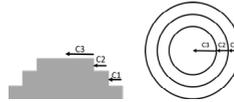


Fig. 3. Example of different cutting conditions

In order to control the manufacturing process, it is necessary to record the pattern group of manufacturing process. This pattern group should be recorded when the machining tool and material are on their best state to get the more stable signals as possible.

A pattern group is called to a group of recorded patterns. Each pattern is created from a specific cutting condition so a pattern would belong to C1 cutting condition, another pattern to C2 and so on.

The figure 4 shows a pattern group example. The PC1, PC2 and PC3 belong to signals read from CNC. As it is said, each pattern is created from a group of signals read from the CNC and a pattern group is created from different patterns.

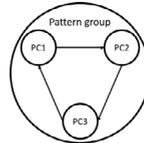


Fig. 4. Pattern group example

All the patterns are recorded in a continuous way and it is suggested to do it when the state of cutting tool and materials are in the best cutting condition possible.

Each pattern can be recorded several times and at the end, it is made an average of recorded patterns in order to have a stable pattern group.

This pattern group will be used in order to compare with the signals read in the real manufacturing process. To achieve this goal, it is established an upper and bottom limit to the recorded pattern signals.

The recorded patterns will be the reference to the future manufacturing processes. If the manufacturing process is performed in the same cutting conditions, the signals obtained will be of the same length and amplitude.

The purpose of established upper and bottom limits is to control the manufacturing process. To do that, if the area of selected signals of a pattern exceeds one of the limits established, the application shows a warning message that indicates there is a

risky situation. Thanks to this, the operator can make a decision faster.

The same thing happens if it is pretended to manufacture in different cutting conditions. The application compares the cutting conditions with the ones used in pattern recording. If there is any difference, the application shows a warning message indicating that is necessary to change the cutting conditions.

For instance, if the cutting tool is broken during the machining process, the application will show a warning message.

Leaving this aside, each pattern signals are read from CNC, so it is necessary to pull out those signals in order to visualize them. For this purpose, IPC and middleware techniques are used, which have been described in the second section of this article.

To develop a solution, two possible scenarios have been considered:

- Develop a data capturing and visualization software in the same machine where the CNC is installed in order to centralize the manufacturing process control.
- Separate the data acquisition and data visualization in order to have a data acquisition software on the CNC machine and data visualization application on a remote system.

In the next chapter, the selected developing techniques are explained.

IV. DEVELOPMENT OF THE CASE STUDY

This project has had different phases and the research team has developed two different versions of the solution during the project's life. The first solution was the centralized one, which runs the software on the same machine where data is captured. This first version has performed an evolution and the remote monitoring version has been developed. In this second option, data is captured and sent to a remote host and the visualization is also performed remotely in this remote host.

On this chapter, it is explained the development done in order to build both systems: visualization and control system.

A. Centralized monitoring application

Figure 5 shows the first option presented on this article. This machine will run the PLC software, which is executed under CNC and monitoring application. This one, is developed in Visual Basic language and consequently it will be valid only in windows environments.

The PLC software will be responsible of capturing data from CNC and the monitoring application will be visualizing the data read.

For each cycle of PLC, the data is read on PRG module and it's written to shared memory using some auxiliary C functions.

Knowing that a minimum period of PLC cycle is 4 milliseconds, the software responsible reading the information from the shared memory must be faster than this period. The logic of a shared memory used in this project is explained in part C of this chapter.

The monitoring software creates one file for each manufacturing process with all the signals read from the CNC. Furthermore, visualizing this signals gives to the person a perspective of all the process.

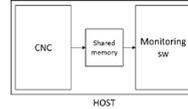


Fig. 5. Local monitoring architecture

The amount of data generated by the monitoring software is very large (each 4ms a new data caption is carried out), so it has been necessary to change the strategy and decentralize the data visualization. So this change in the strategy comes from the issue of large amount of data generated during the manufacturing process.

In the next chapter, it is explained the solution to this problem (second version of the SW) and how the data is managed in order to visualize it in a remote system.

B. Remote monitoring application

The figure 6 shows the architecture of the current remote monitoring application. In this version, it works in two separate hosts and it is a distributed system.

On one hand, in the HOST_1, two processes are executed. One of them is the PLC software, which is the responsible to read the information from CNC tool and write it in the shared memory. The other is the testPLC software, which is the bridge between the PLC and the monitoring application. This latter is the responsible of reading the information from shared memory and sending it to the monitoring application in order to visualize it graphically. In this case, the testPLC software is developed using C++ language.

On the other hand, the HOST_2, is the monitoring application. This application is developed in JAVA language and since the JAVA application runs under JVM (Java Virtual Machine) it has not dependencies with the operating system to be used.

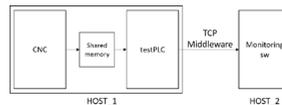


Fig. 6. Remote monitoring architecture

To communicate the testPLC and monitoring application, earlier described middlewares are used.

The first version of the solution, was developed using the ICE middleware (RPC type middleware). Then a second version has been developed and in this case zeroMQ (MOM type middleware) middleware has been used.

In any case, in the trials done, they have been read fifty signals for each pattern, and each value of a signal is 8 bytes. The signals are read with 4 milliseconds of periodicity, therefore the minimum throughput permitted is 1 Mbyte/s.

On the next chapter it is explained how the shared memory between processes works and how it is accessed through different processes.

C. Signal capture and shared memory logic

In order to capture and visualize the data, the first step is to read this data from the CNC. To that end, it is used a PLC program.

In the CY1 module of PLC is performed the initialization of the shared memory. This is created by the PLC software and always will be part of its own memory.

The PRG module of PLC is executed every 4 milliseconds and during this time the CNC has to read the values of the selected registries (signals that are going to be monitored) and write them in the shared memory. These operations are performed by the PLC software, but the read operation is done in mnemonic language because it is intended to read data from CNC. The write operation in the shared memory is done using the C auxiliary functions.

The figure 7 shows the structure of the shared memory where data is write. On one hand, it is the maximum data that can be written (length of bytes). Next variable is the number of data that has been read from the shared memory and the following one is the data that has been written. These two variables are used in order to control the write and read operations in the shared memory.

The next variables are data that has been read from the registries of the CNC and are written from N_1 to N_MaxData.

MaxData
Data_read
Data_write
N_1
N_2
...
N_MaxData

Fig. 7. Shared memory structure

In both architectures explained before, it is necessary a software that reads data from shared memory.

Since the moment that both processes need the access to the shared memory, concurrency issues appear.

In order to control the read and write operations, they are used the variables Data_read and Data_write

explained before. The figure 8 shows the execution diagram of read operation. It is shown that when Data_read is bigger than Data_write, which is the first condition of the diagram, means that PLC software has started writing from the beginning of shared memory and it only will read the last amount of data written on. Therefore, it will be read until MaxData.

The next condition, when Data_read is smaller than Data_write, the data to be read is from the last data read to the last data write.

Finally, when Data_read is equal to Data_write it is not performed any function.

With this, it is achieved a blocking receive where there is not data read until the PLC software writes on shared memory. Consequently, the PLC write operation is a non-blocking send.

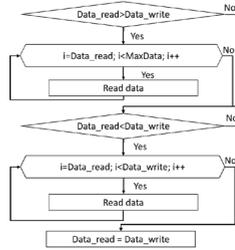


Fig. 8. Shared memory read execution diagram

With this execution diagram, it is resolved the concurrency issues mentioned earlier, and data written by the PLC is read in order.

In the next parts, it is explained how the communication between the CNC machine and monitoring software using the middlewares previously appointed have been developed.

D. testPLC and monitoring software communication using ICE middleware

This middleware is implemented in the second architecture presented.

In this case, the application is developed under ICE middleware, which is the RPC oriented middleware, explained on the second chapter.

To communicate different machines, it's necessary to define the interfaces which will use each software. To that end, it is written a slice language file where the methods that each program will implement are described. The figure 9, shows a reduced small example of how can be described interfaces very simple.

The first thing to define on the file is the module used to create the interfaces, in this case is named as utils.

The variable named sequence is of type double and it is traduced as double array on both programs.

```

module utils
{
sequence<double> Signals;

interface ClientServiceData
{
void sendData(Signals datos);
};
interface ServerService
{
void startCap(ClientServiceData *prxData);
void endCapt(ClientServiceData *prxData);
};
};
    
```

Fig. 9. Interface definition in ICE

The following definitions, belong to the services that each software will provide to the other. The first definition (sendData) is to receive data from the testPLC software, and it is configured with one method which will receive a double sequence of numbers.

The second definition is to start and end the data capture from the testPLC software. If the monitoring software sends to testPLC the proxy to communicate with startCap method, the data capture is started immediately and monitoring softwares start receiving data periodically.

The second method defined in this interface is to end the communication between the two machines.

The figure 10 shows the way in which the communication is performed between the two machines.

When the testPLC software is initialized and activates its adapter, which is the object that will implement the methods explained earlier, it is prepared to send data to any request received.

The monitoring software must create the two necessary proxies for the communication. Since the monitoring software knows the testPLC software location, it creates the proxy to communicate with testPLC and itself. On the startCap method the monitoring software sends the proxy created to the other machine and the communication can start.

The proxy can be in one way in order to create a non-blocking send, and perform a transfer much faster.

Finally, it should be said that all software provided with this simple module is able to connect with the testPLC and capture data from that.

E. testPLC and monitoring software communication using zeroMQ middleware

As explained before, in order to develop the communication between the two machines, two middlewares have been used. In this part it is explained how can be developed a monitoring system using zeroMQ middleware.

The monitoring software is also implemented

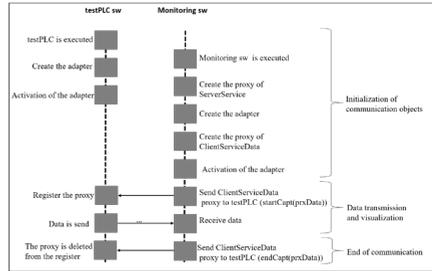


Fig. 10. ICE communication logic

under this middleware in order to communicate the testPLC software.

zeroMQ is a MOM type middleware, which is message oriented. One of the characteristics of this, is that it permits to build a lot of kinds of architectures. In this case, it is developed a publish-subscribe model in order to separate different signals with different identifiers.

The monitoring software subscribes to the signals which wants to monitor and compare to control the behaviour of the machining tool.

On the figure 11 the testPLC software will use the bind method, which is used to create an endpoint with the IP address and port of the machine.

On the other hand, the monitoring software will use the connect method to establish the communication with the endpoint created by the testPLC software.

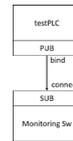


Fig. 11. Publisher subscriber architecture

Using the zeroMQ middleware the logic of communication is as shown in figure 12.

Firstly, the testPLC software will create a publisher type of socket, and with the bind operation and with the endpoint created, will be waiting a request of communication. It is the middleware which manages the communications between the different petitions creating automatically all threads necessary and linking them with the principal process of the software.

Finally, the data read from CNC machine tool is published over the network and each signal is published with a different identifier.

When the monitoring software is initialized, it creates a subscriber type of socket. When the socket

is created, it is performed a connection request to the testPLC. In this way, the connection between the two endpoints is established.

The next step consists on subscribing to those data sources wanted to receive. In order to do that, it is necessary to use the same nomenclature as in the testPLC software.

The last step is to receive data from testPLC software. Since this data is received with different identifier, it is easier to manage and organize the data came from testPLC.

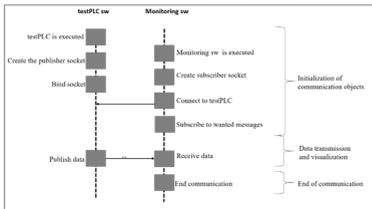


Fig. 12. ZeroMQ communication logic

As in the previous case, the communication can be done in one way.

V. RESULTS

In this chapter the results obtained with this type of systems in a real environment are explained.

The goal of the monitoring software is to make the decision-making much easier to operators working with this machine tool.

To achieve this goal, it has been performed a trial example in a real environment. This trial is based on the use case explained in section III The first step, is to record the pattern group of the process. To that end, the user must specify the patterns that will create, the pattern group and the number of repetitions that will have each pattern. The machining conditions of each pattern is distributed as follows:

- 2 turns for PC1 with cutting conditions:
 VC = 250 m/min
 Av = 0.4 mm/rev
 Ap = 1 mm
- 3 turns for PC2 with cutting conditions:
 VC = 250 m/min
 Av = 0.3 mm/rev
 Ap = 1 mm
- 2 turns for PC3 with cutting conditions:
 VC = 200 m/min
 Av = 0.2 mm/rev
 Ap = 1 mm

When all the turns are performed for all the patterns needed, it is generated a file containing all the information of the signals recorded with all the limits needed to control the behaviour of the machining.

The next step is to adjust the limits to some % of the signal obtained.

Finally, when a trial is performed over the recorded ones, if it is not any element that changes the behaviour of the machine, the alarm system doesn't show any message on the screen. When the cutting speed is increased, an increase in the power consumption and RPM signals in the manufacturing process are detected.

The figure 13 shows the result obtained during a real machining process, as it is shown, the green signal is above the blue one and it is between the limits established. A behaviour and the aspect of the monitoring system in a normal and desired status can be observed.

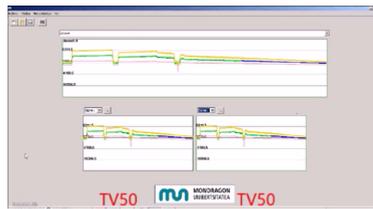


Fig. 13. Machining process without errors

On the other hand, the figure 14 shows a machining process with a deviation in power consumption and RPM signals. The SW detects the deviation of each of the signals.

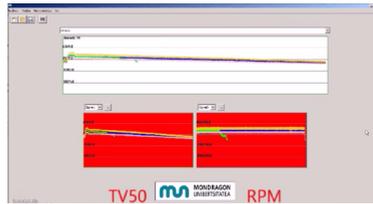


Fig. 14. Machining process with errors

Consequently, the signals that are painted in the screen exceed the bottom limit and the alarm system shows a warning message to alert that something wrong is going on.

VI. CONCLUSIONS

This final chapter describes the conclusions of the monitoring and control system and it also mentions some positive aspects of the system.

In the case of first scenario, it is shown that the large amount of data acquisition makes impossible to maintain them in the same PC.

On the second scenario, the obtained results show that it is possible to create tools that make more easy the decision making.

It makes also possible to create the concept of black box. In this black box acquired data from the machining processes will be stored. Then, this data could be analyzed off-line to improve different aspects of the manufacturing processes.

This type of systems opens new possibilities in the scenario of Industry 4.0 and make possible to save, record and analyze the captured data. Once the data is analyzed, manufacturing processes optimization, product quality optimization, flexibility of the processes, life-cycle better management and customer oriented services could be performed.

This type of systems and architectures will be very useful as basic platforms for Industry 4.0 related solutions.

VII. ACKNOWLEDGEMENT

The research leading to these results has received funding from the Basque Government (ELKARTEK program) for the FATIMA and LANA research projects. The project has been developed by the Mecanizado de Alto Rendimiento (MAR) Research group and Sistemas Embebidos Research group supported by the Department of Education, Universities and Research of the Basque Government.

REFERENCIAS

- [1] Fagor automation, *CNC8070, Manual de programación*, Fagor Automation, 2002.
- [2] A. Silberschatz, P. Baer Galvin y G. Gagne, *Operating system concepts, 8th ed.*, John Wiley & Sons inc., 2009.
- [3] Mark Spruiell, *Ice manual*, <https://doc.zeroc.com/display/Ice36/Ice+Manual>, ZeroC, Inc. 2016.
- [4] Pieter Hintjens, *zeroMQ, Messaging for many applications*, O'reilly media, Inc. 2013.
- [5] A. Dworak, P. Charrue, F. Ehm, W. Sliwinski, M. Sobczak, *MIDDLEWARE TRENDS AND MARKET LEADERS*, <http://accelconf.web.cern.ch/AccelConf/icaleps2011/papers/frhmult05.pdf> CERN, 2011.

Prototipado de sistemas sobre hardware de bajo coste para la regulación automática de glucosa en diabéticos tipo 1

César Vázquez, Ignacio Bravo, Alfredo Gardel, Jesús Berián, José Luis Lázaro¹

Resumen— En este trabajo se plantea una metodología de diseño y simulación de sistemas, para la regulación automática de la glucosa en sangre. La aparición de monitores continuos de glucosa no invasivos y bombas de infusión automática de insulina, abren la puerta al desarrollo de dispositivos que ayuden a pacientes diabéticos a controlar su enfermedad de forma autónoma (páncreas artificial), ayudando a llevar una vida lo más normal posible. El desarrollo de estos dispositivos requiere protocolos de transmisión seguros, ejecución de algoritmos de control en tiempo real y robustez frente a pérdidas de comunicación con los sensores y actuadores. Todo esto hace necesario disponer de sistemas que permitan simular las aplicaciones de la forma más realista posible. En este trabajo, esto se consigue mediante la integración de hardware de bajo coste (Raspberry PI) con la herramienta de simulación Matlab/Simulink y un modelo matemático de la dinámica glucosa-insulina en el organismo.

Palabras clave— Páncreas artificial, Raspberry PI, Monitorización continua de glucosa, Control MPC.

I. INTRODUCCIÓN

LA diabetes mellitus es una enfermedad que cada vez afecta a más personas en el mundo, sobre todo en su tipo 2. En España, según uno de los últimos estudios realizados [1] el 13.8% de los mayores de 18 años en España padecen diabetes tipo 2 y entre un 1% y un 5% de los diabéticos en España padecen diabetes tipo 1.

El control del nivel de glucosa en sangre es esencial para que las personas diabéticas puedan llevar una vida lo más normal posible. Históricamente, este control lo ha realizado el propio paciente basándose en medidas puntuales de su nivel de glucosa para calcular la insulina necesaria a la hora de compensar una determinada comida.



Fig. 1. Sensor CGM Enlite y bomba de insulina Medtronic Minimed 530g (fuente: www.diatribe.com).

¹Dpto. de Electrónica, Universidad de Alcalá, e-mail: {cesar.vazquez,ibravo,alfredo,jesus.berian,lazaro}@depeca.uah.es.

Recientemente, se ha popularizado el uso de dispositivos que permiten estimar el nivel de glucosa en el fluido intersticial, íntimamente relacionado con el nivel de glucosa en sangre. Esta información permite al paciente mantener un mejor control de la enfermedad al disponer de más información. De forma paralela, también ha aumentado la implantación de bombas de infusión subcutánea de insulina, que permiten realizar un control más preciso de la cantidad de insulina inyectada al no requerir de pinchazos continuos para su administración. Los principales modelos comerciales de sensor/bomba son los ofrecidos por las empresas Dexcom [2] y Medtronic [3]. En la figura 1 se muestra la combinación de bomba y sensor de insulina de Medtronic.

El uso conjunto de estos dos dispositivos, abre la puerta al desarrollo de aplicaciones que mejoren la vida de las personas diabéticas y ayuden con el control de la enfermedad. Por un lado, plataformas que permitan observar los datos capturados por el sensor junto con otros parámetros de relevancia como la insulina inyectada, la comida ingerida o las recomendaciones propuestas por el médico. En este grupo encontramos el proyecto Nightscout [4] o las aplicaciones proporcionadas por los fabricantes de los sensores.

Por otro lado, aplicaciones cuyo objetivo último es suplir las funciones que el páncreas del paciente no es capaz de realizar (*páncreas artificial*), esto es, un lazo cerrado de control que permita la regulación automática de los niveles de glucosa mediante la combinación del sensor y la bomba de insulina. La regulación automática de los niveles de glucosa plantea numerosos retos:

- El modelado matemático del comportamiento de la glucosa y la insulina en el cuerpo humano.
- Elección de un algoritmo de control robusto que sea capaz de lidiar con las perturbaciones (comidas) manteniendo el nivel de glucosa dentro de un intervalo dado.
- Sensado de los niveles de glucosa y transmisión segura de los datos.
- Seguridad y viabilidad clínica de los sistemas.

Para este propósito, en la literatura encontramos diferentes estrategias como control PID clásico [5], control óptimo [6, 7] o sistemas de aprendizaje automático que permitan estrategias de control más adaptadas al paciente [8, 9].

La implementación de estos sistemas debe reali-

zarse sobre dispositivos hardware empotrados, que sean capaces de lidiar con todos los problemas de seguridad y conectividad que se presenten, además de cumplir las funciones de control requeridas y ser asequibles al gran público.

Además, puesto que la regulación de glucosa es una aplicación que interactúa directamente con un paciente y puede poner en riesgo su salud en caso de fallo, los dispositivos deben pasar los controles establecidos por los diferentes organismos oficiales antes de su comercialización. El proyecto Open APS [10] intenta dar solución a algunos de estos problemas, estableciendo diferentes protocolos de seguridad en la infusión de insulina y la interacción del paciente con el sistema. Otra aproximación al problema, en este caso desde un paradigma de CPS (cyber physical systems), es la propuesta en [11].

Todo esto hace interesante disponer de plataformas de diseño en las que se puedan realizar experimentos *in-silico*, que permitan plantear situaciones extremas a las que no se podría someter a un paciente real. Por otro lado, la ejecución de los algoritmos diseñados sobre una plataforma embebida, permite integrarlos con las interfaces correspondientes al sensor de glucosa o la bomba de insulina y experimentar con la adición de sensores adicionales como pulseras cuantificadoras, cuyo uso se ha demostrado útil en el control de la glicemia [12].

En este trabajo se plantea una metodología para la evaluación de algoritmos de regulación automática de glucosa en Simulink y su implementación en una plataforma hardware de bajo coste (Raspberry PI), donde se pueden añadir fácilmente interfaces para comunicarse tanto con sensores CGM (continuous glucose monitoring) comerciales como con bombas de insulina.

Para la simulación de los controladores, es necesario contar con un modelo matemático de la dinámica glucosa-insulina suficientemente relevante, en este trabajo se ha utilizado el modelo desarrollado en [13] y [14]. De igual manera, es necesario contar con un modelo del sensor de monitorización continua. Por otro lado, la estrategia de control utilizada para el test del sistema, es un control MPC (model predictive control) basado en una versión linealizada del modelo de simulación.

El resto del artículo se organiza como sigue: en la sección II se detalla el modelo matemático del sistema glucosa-insulina, en la sección III se muestra la estrategia de control utilizada, en la sección IV se muestra el esquema de comunicación con el hardware y el flujo de trabajo propuesto y, en la sección V, se presentan algunas conclusiones.

II. MODELO MATEMÁTICO DEL SISTEMA GLUCOSA-INSULINA

Aunque hay numerosos factores que afectan a los niveles de glucosa en sangre (ejercicio, estrés, ...) las mayores variaciones vienen dadas por la ingesta de glucosa. Esto hace que disponer de un modelo matemático que relacione la concentración de glucosa

en sangre con la ingesta de glucosa sea imprescindible para el diseño de aplicaciones relacionadas con el control de la glicemia, como predictores o lazos de control. Por otro lado, también es necesario conocer como interactúa la glucosa del organismo con la insulina inyectada al mismo de forma exógena.

Para capturar la dinámica entre la glucosa y la insulina en el cuerpo humano, en [13, 14] se plantea un modelo de compartimentos cuyos parámetros fueron identificados durante un estudio clínico. Cada compartimento consiste en un sistema de ecuaciones diferenciales ordinarias cuyos parámetros pueden ser identificados de forma única a partir de datos de entrada/salida.

Este modelo tiene algunas limitaciones en comparación con la dinámica real del sistema glucosa-insulina, por ejemplo, se asume que la ingesta de glucosa es directa, no modelándose la descomposición de los alimentos reales en los diferentes nutrientes. A pesar de todo, permite realizar una estimación relativamente precisa del comportamiento del organismo durante la ingesta de glucosa, siendo este el aspecto más relevante a la hora de plantear un control en lazo cerrado.

Por otro lado, como ya se ha comentado, los dispositivos comerciales utilizados para la monitorización continua de glucosa, proporcionan una estimación de la glucosa en el fluido intersticial. Este valor está directamente relacionado con los niveles de glucosa en el plasma, siendo la diferencia principal un desfase temporal entre ambas señales. Para complementar el modelo dinámico del sistema glucosa-insulina, se ha agregado también el modelo del sensor comercial Dexom G4 Platinum a las simulaciones cuyos parámetros fueron identificados en el estudio realizado en [15].

Para los valores concretos de los parámetros se remite al lector a [13, 14] y [15]. En este caso particular se ha configurado el modelo para representar a un paciente normal con las modificaciones oportunas para reproducir los resultados en un paciente con diabetes tipo 1.

A. Absorción oral de glucosa

El primer compartimento modela la absorción oral de glucosa, esto es, el ratio de aparición de glucosa en sangre $Ra(t)$ (mg/(kg·min)) producido por la ingesta de una determinada cantidad de carbohidratos D (mg).

$$\begin{aligned} \dot{Q}_{sto1}(t) &= -k_{gr1}Q_{sto1}(t) + D\delta(t) \\ \dot{Q}_{sto2}(t) &= -k_{empt}(Q_{sto})Q_{sto2}(t) + k_{gr1}Q_{sto1} \\ \dot{Q}_{gut}(t) &= -k_{abs}Q_{gut}(t) + k_{empt}(Q_{sto})Q_{sto2} \\ Ra(t) &= \frac{f \cdot k_{abs}}{BW} \cdot Q_{gut}(t) \end{aligned} \quad (1)$$

Donde $Q_{sto1}(t)$ y $Q_{sto2}(t)$ son la cantidad de carbohidratos sólidos y disueltos en el estómago respectivamente y $Q_{gut}(t)$ es la cantidad de glucosa en el intestino. Por otro lado, k_{gr1} , k_{empt} y k_{abs} son constantes

de tiempo para la disolución de los carbohidratos, el vaciado del estómago y el paso de la glucosa al intestino y BW es el peso corporal.

El ratio de vaciado del estómago k_{empt} depende de manera no lineal de la cantidad de nutrientes en el mismo. Para modelarlo en [13] se propone la siguiente ecuación:

$$k_{empt}(Q_{sto}) = k_{min} + \frac{k_{max} - k_{min}}{2} \{ \tanh[\alpha(Q_{sto} - b \cdot D)] - \tanh[\beta(Q_{sto} - c \cdot D)] + 2 \} \quad (2)$$

Siendo Q_{sto} la cantidad total de glucosa en el estómago ($Q_{sto1} + Q_{sto2}$). Los parámetros α y β se obtienen imponiendo condiciones a k_{empt} : tal que $k_{empt} = k_{max}$ para $Q_{sto} = D$ y $Q_{sto} = 0$, esto es, haciendo que tome su valor máximo cuando en el estómago aparece la máxima cantidad de glucosa y cuando está vacío.

$$\alpha = \frac{5}{2D(1-b)} \quad \beta = \frac{5}{2Dc} \quad (3)$$

B. Dinámica de la glucosa

La cantidad de glucosa en sangre G_p (mg/kg) viene dada por las aportaciones de la glucosa ingerida y de la glucosa producida por el organismo, por otro lado, existe una utilización de glucosa por el organismo y un filtrado de la misma por los riñones.

$$\begin{aligned} \dot{G}_p(t) &= EGP(t) + Ra(t) - U_{ii} - E(t) \\ &\quad - k_1 G_p(t) + k_2 G_i(t) \\ \dot{G}_i(t) &= -U_{id}(t) + k_1 G_p(t) - k_2 G_i(t) \\ G(t) &= \frac{G_p}{V_G} \end{aligned} \quad (4)$$

Donde $EGP(t)$ es la producción endógena de glucosa, $E(t)$ es la extracción renal, U_{ii} es la utilización de glucosa independiente de la insulina (consumida por el cerebro de forma constante), U_{id} es la utilización de glucosa dependiente de la insulina (detallada más adelante), $G_p(t)$ es la cantidad de glucosa en el plasma y $G_i(t)$ es la cantidad de glucosa en los tejidos. Por otro lado, V_G es el volumen de plasma en el organismo y k_1 y $k_2 \text{ min}^{-1}$ son las constantes de tiempo del sistema.

C. Dinámica de la insulina

El subsistema que modela el comportamiento de la insulina tiene en cuenta las cantidades de la misma en el hígado I_i y el plasma sanguíneo $I_p(t)$. Por otro lado, recibe la insulina externa desde el subsistema de infusión de insulina $R_i(t)$:

$$\begin{aligned} \dot{I}_i(t) &= -(m_1 + m_3)I_i(t) + m_2 I_p(t) \\ \dot{I}_p(t) &= -(m_2 + m_4)I_p(t) + m_1 I_i(t) \\ I(t) &= \frac{I_p(t) + R_i(t)}{V_I} \end{aligned} \quad (5)$$

Siendo m_1, m_2, m_3 y m_4 las masas de insulina.

D. Producción endógena de glucosa

La producción de glucosa por el organismo depende directamente de valores pasados de la señal de insulina $I_d(t)$ y de la cantidad de glucosa en el plasma $G_p(t)$.

$$EGP(t) = k_{p1} - k_{p2} G_p(t) - k_{p3} I_d(t) \quad (6)$$

Donde k_{p1} es la producción base de glucosa por el organismo y k_{p2} y k_{p3} modelan la dependencia con la cantidad de glucosa en sangre y los valores pasados de insulina respectivamente. La dependencia de valores pasados de insulina $I_d(t)$ se obtiene mediante un sistema de dos compartimentos:

$$\begin{aligned} \dot{I}_1(t) &= -k_i(I_1(t) - I(t)) \\ \dot{I}_d(t) &= -k_i(I_d(t) - I_1(t)) \end{aligned} \quad (7)$$

Donde k_i es la sensibilidad a los cambios en $I(t)$.

E. Utilización de glucosa

El consumo de glucosa dependiente de la insulina está relacionado con la cantidad de glucosa en los tejidos ($G_i(t)$) y la cantidad de insulina que pasa al líquido intersticial $X(t)$ y sigue la ecuación de Michaelis-Menten:

$$U_{id}(t) = \frac{V_m(X(t)) \cdot G_i(t)}{K_m(X(t)) + G_i(t)} \quad (8)$$

Donde K_m y V_m son cantidades que dependen linealmente de la insulina en el líquido intersticial $X(t)$.

$$\begin{aligned} V_m(X(t)) &= V_{m0} + V_{mx} X(t) \\ K_m(X(t)) &= K_{m0} + K_{mx} X(t) \end{aligned} \quad (9)$$

$$\dot{X}(t) = -p_{2U} X(t) + p_{2U}(I(t) - I_b) X(t) = 0 \quad (10)$$

Siendo p_{2U} (min^{-1}) el ratio de acción de la insulina en el uso de glucosa.

F. Excreción renal

Cuando la glucosa en el plasma $G_p(t)$ supera un determinado umbral k_{e2} los riñones comienzan a filtrarla de forma proporcional. Este comportamiento se rige por la ecuación.

$$E(t) = \begin{cases} k_{e1}(G_p(t) - k_{e2}) & \text{si } G_p(t) > k_{e2} \\ 0 & \text{si } G_p(t) \leq k_{e2} \end{cases} \quad (11)$$

G. Infusión subcutánea de insulina

La infusión de insulina ($IIR(t)$) se puede modelar como un sistema de dos compartimentos, que representan las concentraciones subcutáneas de insulina monomérica ($I_{sc1}(t)$) y no-monomérica ($I_{sc2}(t)$). Parte de la insulina inyectada (monomérica), pasa al plasma y parte se descompone en insulina monomérica, que pasa posteriormente al plasma sanguíneo.

$$\begin{aligned} \dot{I}_{sc1}(t) &= -(k_{d1} + k_{a1})I_{sc1}(t) + IIR(t) \\ \dot{I}_{sc2}(t) &= k_{d1}I_{sc1}(t) - k_{a2}I_{sc2}(t) \\ R_i(t) &= k_{a1}I_{sc1}(t) + k_{a2}I_{sc2}(t) \end{aligned} \quad (12)$$

Donde k_d es el ratio de descomposición de la insulina y k_{a1} y k_{a2} los porcentajes de insulina que pasan al plasma.

H. Modelo del sensor CGM

Los sensores CGM utilizados en dispositivos comerciales distan de ser perfectos e introducen diversos errores en la medida. Además, puesto que la concentración de glucosa se mide en el fluido intersticial en lugar de en el plasma sanguíneo, aparece una dinámica entre ambas señales que también debe ser tenida en cuenta.

En [15] se realiza un estudio sobre los sensores empleados en los dispositivos Dexcom Seven PLUS y Dexcom G4, proponiendo un modelo que tiene en cuenta diversos aspectos del proceso de medida:

- Dinámica entre la glucosa en el plasma sanguíneo y la glucosa en el fluido intersticial.
- Ruido aditivo de medida.
- Error de ganancia y offset introducido por el sensor.

En la figura 3 se muestra un diagrama de bloques del sensor. El objetivo es obtener una relación entre la señal $CGM(t)$ proporcionada por el sensor y la glucosa en sangre $BG(t)$. En la figura 3 se muestra un diagrama de bloques del sistema propuesto en [15].

La primera etapa del modelo representa la dinámica entre la glucosa en sangre $BG(t)$ y la glucosa en el fluido intersticial $IG(t)$. La glucosa en el fluido intersticial sigue aproximadamente el mismo perfil que la glucosa en sangre, apareciendo una atenuación g y un retraso τ entre ambas señales.

Si bien el retraso entre ambas señales no es constante y se ve modificado durante el ejercicio físico o las comidas, de forma aproximada puede modelarse mediante un sistema de orden 1 tal que:

$$\dot{IG}(t) = -\frac{1}{\tau}IG(t) + \frac{1}{\tau}BG(t) \quad (13)$$

La glucosa intersticial medida por el sensor (IG_S) incorpora un error de ganancia y un error de offset representados por los parámetros $a(t)$ y $b(t)$.

$$IG_S(t) = a_i(t)IG(t) + b_i(t) \quad (14)$$

Se considera que estos errores varían en el tiempo conforme se deteriora el sensor tal que:

$$a_i(t) = \sum_{k=0}^m a_{ik}t^k \quad b_i(t) = \sum_{k=0}^l b_{ik}t^k \quad (15)$$

Siendo m y l el orden de los polinomios, y t el tiempo de vida del sensor. El valor de $a(t)$ y $b(t)$ se asume constante entre calibraciones.

El ruido de medida se describe mediante un proceso autorregresivo (AR) de orden q tal que:

$$v_i(t) = \sum_{k=1}^q \alpha_{ik}v_i(t-k) + w_i(t) \quad (16)$$

Donde $w_i(t)$ sigue una distribución normal $N(0, \sigma_w^2)$.

III. CONTROL EN LAZO CERRADO

En la figura 2 se muestra un diagrama de bloques de la estrategia de control empleada. El algoritmo de control MPC toma medidas procedentes del sensor CGM simulado $CGM(t)$ y recibe información de las comidas anunciadas por el paciente $Ra(t)$. En función de estas medidas, se genera una señal de actuación $IIR(t)$ que determina cuando es necesario aplicar insulina, para mantener la glucosa dentro de los valores recomendados. Por otro lado, se añade un valor de insulina constante a la actuación (denominado ratio basal de insulina), que es configurable por el usuario y se añade a la señal de control generada.

Para realizar el control en lazo cerrado de los niveles de glucosa, se toma como base el modelo presentado en la sección anterior, linealizado en un determinado punto de operación. Puesto que la única entrada controlable es la infusión de insulina ($IIR(t)$), de cara al diseño del controlador el ratio de aparición de glucosa ($Ra(t)$) se considera una perturbación conocida.

La linealización se realiza para $u(t) = IIR(t) = 0$ y $d(t) = Ra(t) = 0$, incluyendo únicamente un retraso de valor τ para el modelado del sensor. Lo que nos deja con el vector de estados:

$$x(t) = \begin{bmatrix} CGM \\ X \\ I_{sc1} \\ I_{sc2} \\ I_t \\ I_p \end{bmatrix} \begin{matrix} I_1 \\ I_d \\ G_p \\ G_t \\ I_p \end{matrix} \quad (17)$$

El valor recomendado a los pacientes con diabetes tipo 1 para la concentración de glucosa en sangre, se sitúa alrededor de los 130mg/dL, luego este es el valor que tomará la referencia a seguir r . Dada la complejidad del sistema, y la necesidad de imponer restricciones tanto a la entrada como a la salida, una estrategia de control popular para este propósito es el control MPC (model predictive control) [6, 7, 9].

En el control MPC, se busca encontrar la secuencia de control que minimiza una determinada función de coste J para una determinada cantidad de instantes futuros (horizonte de predicción p). Posteriormente, se aplica a la planta el primer comando de control de la secuencia calculada, repitiendo el proceso para cada instante k [16].

En este caso, se ha implementado una función de coste J con dos términos, que penalizan el error respecto a la referencia r y los cambios en la señal de control u :

$$J = \sum_{j=1}^{n_y} \sum_{i=1}^p w_y(\tau_j(k+i|k) - y_j(k+i|k))^2 + \sum_{j=1}^{n_u} \sum_{i=1}^{p-1} w_{\Delta u}(\Delta u(k+i))^2 \quad (18)$$

Donde:

- $y_j(k+i|k)$ es la predicción de la salida para el instante $k+i$ hecha en el instante k .
- Δu es el cambio en la señal de control.

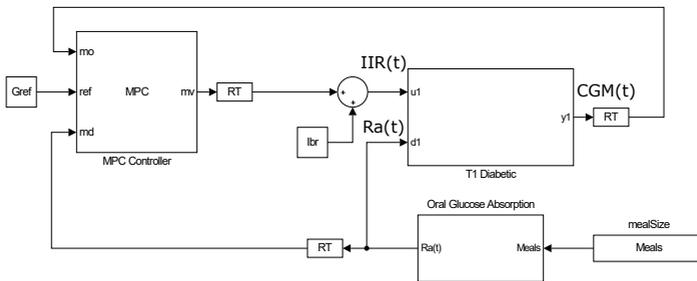


Fig. 2. Diagrama control en lazo cerrado

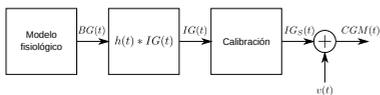


Fig. 3. Modelo del sensor CGM

- w_y y $w_{\Delta u}$ son los parámetros de ajuste del controlador que penalizan el error y los cambios en la señal de control respectivamente.

Por otro lado, para ajustarse a la situación particular del problema, la optimización se resuelve sujeta a una serie de restricciones en la entrada y la salida tal que:

$$u_{min} = 0 \leq u \leq u_{max} \quad (19)$$

$$y_{max} = 70 \leq r \leq y_{max} = 200\text{mg/dL} \quad (20)$$

Para el ajuste del controlador, puesto que la función de coste se compone únicamente de dos términos, es posible controlar el comportamiento del controlador variando la relación $w_{\Delta u}/w_y$. Para que la señal de error y la actuación no tengan influencia sobre el ajuste, es necesario elegir valores tal que $w_{\Delta u} \gg \Delta u$ y $w_y \gg (CGM - G_{ref})$.

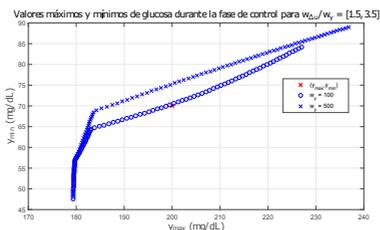


Fig. 4. Simulación paramétrica para el ajuste del controlador MPC

El ajuste se ha llevado a cabo estudiando los valores máximos y mínimos de la señal $CGM(t)$ durante los periodos de control (durante las comidas). En la

figura 4 se muestra una simulación paramétrica para valores entre 1.5 y 3.5 de la relación $w_{\Delta u}/w_y$, en la que se han representado las parejas de valores máximos y mínimos de la glucosa en sangre para cada simulación.

El objetivo es encontrar la relación en la que la salida se mantiene dentro de el intervalo establecido $[y_{min}, y_{max}] = [70, 200]\text{mg/dL}$. El valor óptimo lo encontramos en la curva correspondiente a $w_y = 500$ con una relación entre ambos parámetros de $w_{\Delta u}/w_y = 2.26$, para la cual el valor mínimo de glucosa alcanzado es de 70 y el máximo de 185, ambos dentro del intervalo elegido. Para valores de $w_y > 500$ no se obtiene ninguna mejora.

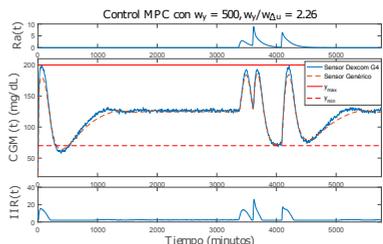


Fig. 5. Control MPC de la planta utilizando modelo mínimo de sensor y los parámetros de un sensor comercial (Dexcom G4 Platinum)

En la figura 5 se muestra la simulación del sistema para la relación $w_{\Delta u}/w_y$ seleccionada. Para estudiar el comportamiento del sistema, se ha simulado un periodo de cuatro días. Las primeras 60 horas se utilizan como periodo de establecimiento, en las siguientes 24 horas se simula un día normal con tres comidas de 45, 75 y 70 gramos de glucosa respectivamente.

En la parte superior, se puede observar el ratio de aparición de glucosa $Ra(t)$ debido a la ingesta de glucosa. Este es cero excepto durante el periodo de 24 horas durante el que se simulan las tres comidas. En la parte inferior, se muestra el ratio de infusión

de insulina $IIR(t)$ calculado por el controlador en pmol/min, esto es, la insulina que debe ser introducida de forma subcutánea por la bomba de insulina.

En la zona central se muestra la salida del sistema en dos situaciones. La línea discontinua muestra los niveles de glucosa cuando se modela el sensor como un simple retraso, mientras que la línea sólida, muestra los niveles de glucosa cuando se introduce un modelo más realista para el sensor. El arranque del controlador se produce asumiendo que el sistema se encuentra en el peor caso ($CGM(t) = 180$ mg/dL), corrigiéndose esta situación y llegando a la referencia de glucosa establecida de ($CGM(t) = G_{ref} = 130$ mg/dL). Cuando se produce el periodo con tres comidas el sistema vuelve a recuperar la referencia sin abandonar los límites de glucosa establecidos.

IV. IMPLEMENTACIÓN EN RASPBERRY PI Y SIMULACIÓN

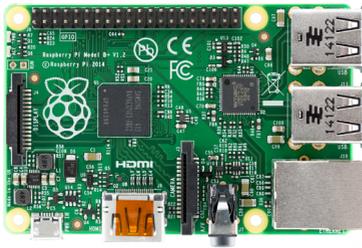


Fig. 6. Raspberry PI modelo B

Raspberry PI [17] es una plataforma hardware de bajo coste (figura 6) basada en un SoC ARM que permite ejecutar sistemas operativos basados en linux, y ofrece numerosas opciones de conectividad (I2C, UART, USB, Ethernet, WiFi,...) lo que la hacen especialmente atractiva para el prototipado de sistemas que requieran comunicación con la nube, con sensores y actuadores y con otros dispositivos de cómputo como un PC.

En este trabajo se ha optado por el uso de Raspberry PI para la implementación del algoritmo de control MPC diseñado por varios motivos: existencia de aplicaciones relacionadas con el desarrollo del páncreas artificial que hacen uso de la plataforma [10], buena integración con el entorno de simulación Matlab/Simulink y posibilidad de agregar una interfaz para comunicarse con el sensor de monitorización de glucosa y la bomba de insulina.

Para la descarga de los algoritmos de control en la plataforma hardware, se ha recurrido a la herramienta de generación automática de código del entorno Matlab/Simulink [18]. El paquete de integración de Raspberry PI con Simulink, permite trabajar con múltiples esquemas y elegir cuales se ejecutarán en el PC y cuales en la plataforma hardware.

La comunicación entre el dispositivo y el PC, se

realiza mediante protocolo UDP sobre WiFi, ejecutándose de forma concurrente el modelo del sistema glucosa-insulina en el PC y el controlador diseñado en la Raspberry PI e intercambiando datos de forma continua. En la figura 7 se muestra un diagrama de bloques del sistema.

V. CONCLUSIONES

En este trabajo se ha presentado una metodología para el diseño de sistemas de regulación automática de glucosa y su implementación en una plataforma hardware de bajo coste (Raspberry PI).

En proyectos como open APS [10] se presenta un marco de desarrollo para aplicaciones de páncreas artificial pero utilizando algoritmos de control poco elaborados que solo permiten la regulación nocturna (en periodos sin ingesta de carbohidratos) de la glucosa mientras que, trabajos como [6, 7] se centran únicamente en la simulación de los algoritmos de control.

El esquema de trabajo seguido permite una modularidad total del sistema, puesto que cada una de sus partes puede ser considerada una caja negra. Esto permite experimentar con diferentes modelos de paciente, ya sean matemáticos o identificados a partir de datos experimentales y de sensor.

La elección de un modelo fisiológico que sea suficientemente relevante de cara al diseño de controladores es un aspecto importante puesto que determinará en gran medida la robustez de los mismos y como se comportarán en una situación real. Por otro lado, disponer de un modelo realista de sensor permite tener en cuenta aspectos como su degradación o los errores de calibración cometidos.

El uso de una plataforma hardware genérica como Raspberry PI para la ejecución de los algoritmos de control, facilita en gran medida la comunicación con hardware comercial (sensores y bombas), paso importante para el prototipado de sistemas que puedan ser posteriormente aplicados a casos reales.

AGRADECIMIENTOS

Trabajo parcialmente soportado por el proyecto de la UAH CCG2015/EXP-041.

REFERENCIAS

- [1] Fundación Diabetes, "Estado de la diabetes en España," 2012. <http://www.fundaciondiabetes.org/prensa/297/1a-diabetes-en-espana>, Accessed: 2016-02-2.
- [2] Dexcom, "Dexcom g4 platinum," 2016. <https://www.dexcom.com/dexcom-g4-platinum-share>, Accessed: 2016-02-2.
- [3] Medtronic, "Minimed 530g," 2016. <http://www.medtronicdiabetes.com/products/minimed-530g-diabetes-system-with-elite>, Accessed: 2016-02-2.
- [4] Nightscout, "Nightscout project," 2016. <http://www.nightscout.info/>, Accessed: 2016-02-2.
- [5] Peter G Jacobs, Joseph El Youssef, Jessica Castle, Parvash Bakhtiani, Deborah Branigan, Matthew Breen, David Bauer, Nicholas Preiser, Gerald Leonard, Tara Stonex, et al., "Automated control of an adaptive bihormonal, dual-sensor artificial pancreas and evaluation during inpatient studies," *Biomedical Engineering, IEEE Transactions on*, vol. 61, no. 10, pp. 2569–2581, 2014.

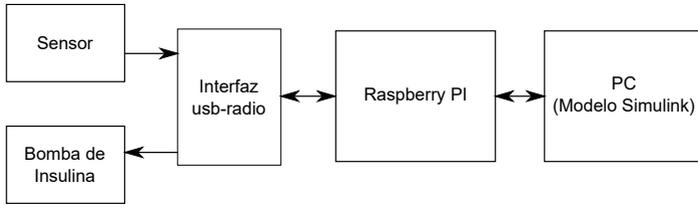


Fig. 7. Sistema de simulación con Raspberry PI como plataforma hardware in-the-loop.

- [6] Klaske van Heusden, Eyal Dassau, Howard C Zisser, Dale E Seborg, and Francis J Doyle, "Control-relevant models for glucose control using a priori patient characteristics." *Biomedical Engineering, IEEE Transactions on*, vol. 59, no. 7, pp. 1839-1849, 2012.
- [7] Lalo Magni, Davide M Raimondo, Luca Bossi, Chiara Dalla Man, Giuseppe De Nicolao, Boris Kovatchev, and Claudio Cobelli, "Model predictive control of type 1 diabetes: an in silico trial." *Journal of diabetes science and technology*, vol. 1, no. 6, pp. 804-812, 2007.
- [8] J. Fernandez de Canete, S Gonzalez-Perez, and JC Ramos-Diaz, "Artificial neural networks for closed loop control of in silico and ad hoc type 1 diabetes." *Computer methods and programs in biomedicine*, vol. 106, no. 1, pp. 55-66, 2012.
- [9] Konstantia Zarkogianni, Andriani Vazeou, Stavroula G Mougialakou, Aikaterini Prountzou, and Konstantina S Nikita, "An insulin infusion advisory system based on autotuning nonlinear model-predictive control." *Biomedical Engineering, IEEE Transactions on*, vol. 58, no. 9, pp. 2467-2477, 2011.
- [10] Dana Lewis, "openaps project," 2016, <https://openaps.org>. Accessed: 2016-02-2.
- [11] Insup Lee, Oleg Sokolsky, Sanjian Chen, John Hatcliff, Eunyoung Jee, BaekGyu Kim, Andrew King, Margaret Mullen-Fortino, Soojin Park, Alexander Roederer, et al., "Challenges and research directions in medical cyber-physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 75-90, 2012.
- [12] Meriyan Eren-Oruklu, Ali Cinar, Derrick K. Rollins, and Lauretta Quinn, "Adaptive system identification for estimating future glucose concentrations and hypoglycemia alarms." *Automatica*, vol. 48, no. 8, pp. 1892 - 1897, 2012.
- [13] C.D. Man, M. Camilleri, and C. Cobelli, "A system model of oral glucose absorption: Validation on gold standard data." *Biomedical Engineering, IEEE Transactions on*, vol. 53, no. 12, pp. 2472-2478, Dec 2006.
- [14] C.D. Man, R.A. Rizza, and C. Cobelli, "Meal simulation model of the glucose-insulin system." *Biomedical Engineering, IEEE Transactions on*, vol. 54, no. 10, pp. 1740-1749, Oct 2007.
- [15] A. Facchinetti, S. Del Favero, G. Sparacino, J.R. Castle, W.K. Ward, and C. Cobelli, "Modeling the glucose sensor error." *Biomedical Engineering, IEEE Transactions on*, vol. 61, no. 3, pp. 620-629, March 2014.
- [16] Eduardo F Camacho and Carlos Bordons Alba, *Model predictive control*, Springer Science & Business Media, 2013.
- [17] Raspberry PI Foundation, "Raspberry pi," 2016, <https://www.raspberrypi.org/>. Accessed: 2016-02-2.
- [18] MathWorks, "Raspberry pi support from simulink," 2016, <http://es.mathworks.com/hardware-support/raspberry-pi-simulink.html>, Accessed: 2016-02-2.

Monitorización de la Contaminación Ambiental Mediante Sensores Móviles

William Zamora, Oscar Alvear, Carlos T. Calafate, Juan-Carlos Cano and Pietro Manzoni¹

Resumen— La monitorización de la contaminación del aire se ha convertido en un requisito esencial para las ciudades de todo el mundo. Actualmente, la manera más habitual de supervisar la contaminación del aire es a través de estaciones de monitorización fijas, que son caras y difíciles de instalar. Para resolver este problema, hemos desarrollado EcoSensor, una solución para controlar la contaminación del aire mediante el uso de sensores móviles.

EcoSensor recoge la contaminación del aire mediante sensores integrados y transfiere los datos capturados a un dispositivo basado en Android, que muestra al usuario los niveles de contaminación del aire en tiempo real. Además, EcoSensor almacena las diferentes trazas de contaminación en un servidor basado en la nube para su posterior análisis. El servidor en la nube permite crear mapas detallados de la distribución de la contaminación utilizando técnicas de predicción espacial basada en Kriging.

Para optimizar el uso de nuestro sistema, en este trabajo analizamos el impacto de la orientación de los sensores en presencia de movilidad. Además, analizamos las mejores estrategias de muestreo en el tiempo y espacio para determinar la estrategia más eficaz de captación de datos. Los resultados experimentales demuestran que la orientación del sensor y el periodo de muestreo tienen mucho menor impacto sobre mapas creados que la trayectoria real de los recursos.

Palabras clave— Monitorización, contaminación del aire, Sensores móviles, Arquitectura, Interpolación geo-espacial.

I. INTRODUCCIÓN

La contaminación del aire es un aspecto esencial a considerar en la actualidad porque influye directamente en la vida de las personas, especialmente en ciudades muy pobladas, causando problemas de salud (principalmente en las vías respiratorias), provocando cambios climáticos y la reducción de la producción agrícola, entre otros. La contaminación se debe a la emisión de gases y partículas a la atmósfera, produciendo cambios en su composición. La contaminación del aire pueden ser de dos tipos: (i) la contaminación del aire primario, cuando los gases o partículas se emiten directamente a la atmósfera, como el monóxido de carbono (CO), el dióxido de carbono (CO₂), partículas menores de 10 micras (PM₁₀) o partículas inferiores a 2,5 micras (PM_{2,5}); y (ii) la contaminación del aire secundario, cuando los gases son producidos por una reacción química entre los contaminantes primarios y algunos elementos de entorno, como el ozono (O₃), el cual es producido por la combinación de óxidos de nitrógeno (NO_x), el oxígeno(O₂), los compuestos orgánicos volátiles (COV), y la luz del sol. La mayoría de las

ciudades europeas despliegan sensores estáticos para supervisar y controlar la evolución de los niveles de contaminación a gran escala, proporcionando niveles de contaminación de la ciudad con mayor granularidad. En toda Europa existen alrededor de mil quinientas estaciones fijas de vigilancia de la contaminación atmosférica [1].

El equipo tradicional para medir la contaminación ambiental se instala normalmente en una estación de monitorización estática. En estas estaciones, la supervisión se basa en sensores sofisticados, que son muy precisos y presentan oscilaciones mínimas para el proceso de captura de datos. Sin embargo, éstas son muy caras y difíciles de administrar. Debido a su tamaño, deben instalarse en una ubicación específica y los datos que se obtienen son representativo en una pequeña zona circundante.

Una alternativa para medir la contaminación ambiental se basa en la detección móvil. Específicamente, los pequeños dispositivos de bajo coste pueden ser instalados en vehículos para la monitorización de diferentes partes de una ciudad simultáneamente, y de forma continua. El principal problema de los sensores móviles de gama baja es que tienen menos precisión que los sensores sofisticados, y por lo tanto necesitan ser calibrados periódicamente.

En este artículo proponemos EcoSensor, un sistema para vigilar la contaminación del aire en zonas urbanas, a través de sensores de gama baja. Estos sensores pueden ser instalados en el sistema de transporte público o en bicicletas. Cada sensor recopila los datos de contaminación y los transfiere a un servidor en la nube a través de un smartphone. En el servidor, nuestro sistema analiza la distribución de la contaminación mediante técnicas de interpolación espacial para generar mapas detallados de la contaminación en una ciudad.

Este documento está organizado como sigue: En la sección II se presentan los trabajos relacionados. La sección III describe la arquitectura y el esquema de nuestra propuesta. La sección IV presenta EcoSensor, nuestra propuesta para monitorización móvil de la calidad del aire. En la sección V analizamos el momento óptimo y el muestreo espacial, así como el impacto de la orientación del sensor en la presencia de la movilidad. Por último, la sección VI presenta las conclusiones y trabajos futuros.

II. TRABAJOS RELACIONADOS

En los últimos veinte años, la monitorización de la contaminación del aire se ha convertido en algo muy importante en todo el mundo debido a la influencia de la calidad del aire en nuestras vidas. Existen nu-

¹Department of Computer Engineering, Universitat Politècnica de Valencia, Email: w1zame@posgrado.upv.es, osa1@doctor.upv.es, {calafate, jucano, manzoni}@disca.upv.es

merosos trabajos de investigación que estudian los efectos de la contaminación atmosférica sobre la salud. Entre ellas podemos encontrar la contribución de Chen et al. [2], quien analizó los efectos del ozono sobre la salud humana. Brook et al. [3] también contribuyó a este campo por estudiar la relación entre la exposición a la contaminación del aire (incluido el ozono) y los problemas cardiovasculares.

Para determinar la distribución de la contaminación en una ciudad basada en unas pocas muestras se requiere la adopción de técnicas de interpolación espacial. En este sentido, estudios como [4] y [5] se han basado en técnicas de interpolación kriging para predecir la contaminación en las ciudades de Quebec y Toronto, respectivamente.

Para tener una descripción detallada de la distribución de la contaminación, se requiere una captura de datos con elevada granularidad espacial, y a través de la monitorización móvil es la mejor opción para lograrlo. En la literatura se pueden encontrar varios trabajos que adoptan este enfoque. Por ejemplo, Brković et al. [6] propone un sistema para vigilar la contaminación del medio ambiente en la ciudad de Belgrado mediante sensores Waspote instalados en el sistema de transporte público. Hu et al. [7] utilizan una red de sensores de vehículos para el control de la contaminación del aire.

Más recientemente, Calafate y Ducourthial [8] han combinado técnicas de muestreo móvil con métodos de interpolación basados en kriging para determinar la precisión alcanzable al estimar la distribución de ozono en una ciudad, apoyándose en el sistema de transporte público para la recopilación de datos.

Cheng et al. [9] propone un sistema para vigilar las concentraciones de PM2.5 utilizando crowdsourcing, que es una alternativa al uso de los sensores móviles. Éstos se centran en el análisis mecánico del diseño de sensores para optimizar la recepción de aire, así como en técnicas de fusión y análisis de datos. La calibración del sensor se logra mediante el análisis de los datos producidos en el laboratorio usando redes neuronales.

Por último, Zheng et al. [10] muestra cómo analizar los datos obtenidos de diferentes fuentes, tales como los niveles de tráfico, condiciones meteorológicas, y la contaminación, mediante el uso de diferentes técnicas para grandes volúmenes de datos. Los autores evidencian cómo estas técnicas permiten inferir los niveles de contaminación ambiental con una mejor granularidad.

Con respecto a trabajos previos nuestra propuesta proporciona una solución integral. En particular, combinamos sensores de gama baja, teléfonos inteligentes, y servicios cloud para supervisar eficientemente los niveles de contaminación.

III. ECOSENSOR

EcoSensor es un sistema que permite la monitorización de la contaminación atmosférica de una manera sencilla y económica, siendo especialmente útil en ciudades muy concurridas. Combina datos proceden-

tes de las estaciones de monitorización de la calidad del aire existente con los datos recogidos por los sensores móviles para generar informes detallados sobre los niveles de contaminación. Los sensores móviles pueden ser instalados en bicicletas o en el sistema de transporte público para vigilar la ciudad de una forma sencilla y eficaz. Toda la información recogida se almacena en un servidor con tecnología cloud en la nube, para una posterior generación de informes detallados.

La arquitectura del sistema propuesto integra varios componentes hardware y software. Estos componentes pueden ser clasificados como elementos sensores móviles o elementos cloud, siendo este último un conjunto de servicios que se ejecutan en un servidor y en el que se analizan los datos recopilados y se presenta información detallada. Los elementos móviles están compuestos por tres componentes distintos: (i) un sensor Waspote [11] para medir la contaminación, (ii) un dispositivo empujado Raspberry Pi [12] que actúa como una pasarela entre el sensor y el dispositivo basado en Android, y (iii) un dispositivo basado en Android que muestra en tiempo real el estado de la contaminación, y que también permite el almacenamiento de los datos y transferencia al servidor en la nube cuando se dispone de conexión a la red. Esta arquitectura se muestra en la Figura 1.

El sensor Waspote, está basado en una plataforma Arduino [13], y mide la calidad del aire a través de diversos sensores (Ozono, CO2, contaminación del aire y temperatura). Además, tiene una interfaz GPS que permite determinar la ubicación exacta de cada medición. Una vez que los datos están listos, se transfieren a la Raspberry Pi mediante tecnología Zigbee [14].

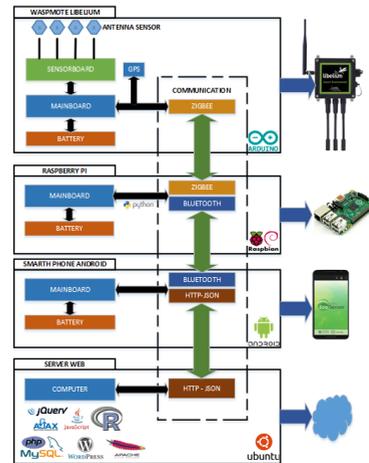


Fig. 1. Arquitectura EcoSensor.

El dispositivo basado en Android muestra, en tiempo real, el nivel de contaminación registrado en una ubicación determinada, y permite transferir los datos recopilados al servidor en la Nube. En particular, utilizamos la interfaz Bluetooth para recibir datos de la Raspberry Pi, y mediante la interfaz de red celular o Wifi transferimos datos al servidor en la Nube.

El servidor de la nube tiene un sistema web que gestiona la información recibida desde el dispositivo Android. Los datos recibidos se almacenan en una base de datos, que se procesa a través de la herramienta R [15]. Por último, se generan informes detallados disponibles para el administrador del servidor web.

IV. APLICACIONES CLIENTE Y SERVIDOR

EcoSensor es una plataforma compuesta por dos módulos integrados, (i) elementos móviles responsables de capturar los datos de contaminación, y (ii) un servidor en la nube que se encarga de almacenar y procesar los datos recopilados. El elemento móvil es controlado a través de una aplicación basada en Android, y el servidor es gestionado por una aplicación web en la Nube. A continuación se proporciona una descripción más detallada de ambas aplicaciones.

A. Aplicación Android

La aplicación Android ha sido desarrollada utilizando Android Studio IDE [16]. Esta aplicación permite iniciar o detener un trayecto, visualizar datos capturados en tiempo real, cargar datos en el servidor, y realizar diversas tareas de gestión.

Internamente, la aplicación tiene dos módulos: (i) un servicio que recibe continuamente (a través de Bluetooth) los datos enviados por el sensor, y que se almacena en una base de datos interna; y (ii) una interfaz de usuario que permite iniciar o detener una captura de datos de la traza del sensor. También proporciona información en tiempo real sobre los niveles de contaminación en la ubicación actual según la AQI INDEX [17]. Por otra parte, la traza completa puede ser representada en un mapa que muestra las variaciones de la contaminación a través de diferentes identificadores de color. Una vez que la traza se haya completado, los datos pueden ser enviados al servidor a través de un mensaje HTTP POST utilizando el formato JSON. La aplicación móvil se muestra en la Figura 2.

B. Server application

El servidor de la nube proporciona una interfaz web basada en Wordpress [18], que permite al administrador tener acceso completo a la información en términos de manejo de trazas, procesamiento y visualización.

Una vez iniciada sesión, el administrador visualiza las trazas cargadas, pudiendo realizar diferentes análisis estadísticos sobre el conjunto de datos (CO₂, ozono, contaminación del aire y temperatura). Para el análisis estadístico y la generación del informe

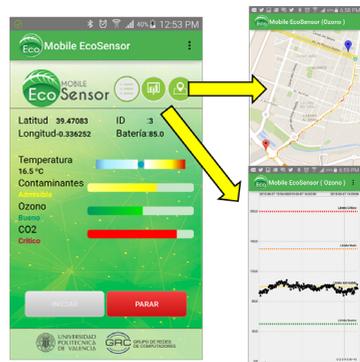


Fig. 2. EcoSensor: Aplicación móvil.

nos hemos basado en la herramienta gráfica R, para generar gráficos sobre:

- Nivel de contaminación.- Muestra los detalles de los niveles de contaminación en un área específica.
- Análisis de Kriging.- Presenta el método kriging con tres diferentes salidas: (i) Resultado de interpolación, (ii) el error del método kriging, y (iii) análisis del variograma.
- Filtrado de datos capturados.- Presenta el ajuste y el proceso de calibración, mostrando la relación entre los datos originales y los datos resultantes.
- Variación de datos.- Proporciona mediante un diagrama de caja y bigotes la distribución de los datos en general.

El sitio web está disponible en <http://www.ecosensor.net>, y su diseño se muestra en la Figura. 3.

V. BÚSQUEDA DE LA MEJOR ESTRATEGIA DE MEDICIÓN

Una vez que la arquitectura propuesta ha sido totalmente desarrollada y probada desde un punto de vista funcional, nuestro siguiente objetivo consiste en determinar la estrategia óptima para recopilar datos sobre la contaminación del aire a través de los sensores móviles.

Con este propósito se analiza, en primer lugar, el impacto de la movilidad mediante la comparación de las mediciones estáticas frente a las móviles. Además, también hemos calculado la influencia de la orientación del sensor en el proceso de detección móvil. Nuestro siguiente paso es analizar el impacto que produce la reducción de la frecuencia de muestreo en la movilidad estudiando la precisión del proceso de kriging. Del mismo modo, se analizó el impacto de la reducción del número de muestras espaciales en



Fig. 3. EcoSensor: Aplicación Cloud.

la precisión del proceso de kriging. Para realizar este experimento se paso por ciertas calles seleccionadas mientras se capturaba datos, reduciendo progresivamente el recorrido general.

A. Mejor estrategia de medición

Para analizar el impacto de la movilidad en el proceso de captura de datos se realizaron diferentes pruebas, de recolección de los niveles de ozono en un área específica tanto de forma estática, como mediante una bicicleta en movimiento (velocidad aproximada de 20 km/h). Para las pruebas de movilidad, hemos recogido las mediciones con diferentes orientaciones del sensor.

La figura 4 muestra las lecturas del sensor con y sin movilidad. Se evidencia que la movilidad, al menos a la velocidad utilizada para las pruebas, no tiene un impacto significativo en las mediciones del sensor. En particular, verificamos que los valores de la mediana y el rango intercuartil son bastante similares a los obtenidos cuando el sensor está en situación estática.

B. Impacto del tiempo de muestreo en el proceso de predicción

En esta sección se analiza el impacto del tiempo de muestreo previsto para los mapas de contaminación. En particular, queremos determinar si la reducción del número de muestras permite hacer predicciones similares o si, por el contrario, existe un error de predicción significativo al generar el mapa de contaminación. Con este propósito, se monitorizó el campus de la Universidad Politécnica de Valencia (1.5km × 0.6km) con un sensor móvil instalado en una bicicleta. El contaminante utilizado para las pruebas fue el ozono, debido a sus bien conocidos efectos negativos para la salud.

TABLA I
 RESUMEN ESTADÍSTICO - ANÁLISIS DE MUESTRAS DE TIEMPO.

Periodo de muestra	Media	Desviación estándar	Error (ϵ_i)
5 sec.	60.3125	1.1403	-
10 sec.	60.3193	1.1589	0.9734
20 sec.	60.3782	1.1315	0.0421
40 sec.	60.3612	1.1317	0.0774
80 sec.	60.4563	1.1266	0.0818

Para obtener una correcta distribución de los niveles de ozono, controlamos todo el campus, estableciendo el período de muestreo para el valor mínimo permitido por el sensor (5 segundos). A continuación, hemos reducido la frecuencia de muestreo estableciendo intervalos entre muestras de 10, 20, 30, 40, y 80 segundos. Esto se logró mediante el filtrado de la traza completa y la recuperación de los conjuntos de datos para 1/2, 1/4, 1/6, 1/8 y 1/16 de la muestra total, respectivamente.

A continuación, interpolamos cada traza mediante un proceso de kriging, obteniendo una distribución detallada de la contaminación. Hemos utilizado la traza completa (muestras cada 5 segundos) como referencia, y en comparación con los resultados obtenidos mediante los otros conjuntos de datos relacionados.

Para calcular el error relativo usando muestras de 5s como referencia, hemos utilizado la ecuación 1.

$$\epsilon_i = \frac{1}{m \cdot n} \sum_{x=0}^m \sum_{y=0}^n |k_{i_{xy}} - k_{0_{xy}}| \quad (1)$$

En esta ecuación, ϵ_i representa el error relativo del conjunto de datos i con respecto al conjunto de datos de referencia, m y n representan el ancho y la longitud de la zona de destino bajo análisis, $k_{i_{xy}}$ representa el valor calculado mediante el método de interpolación de kriging para el conjunto de datos i en la posición xy , $k_{0_{xy}}$ representa el valor calculado

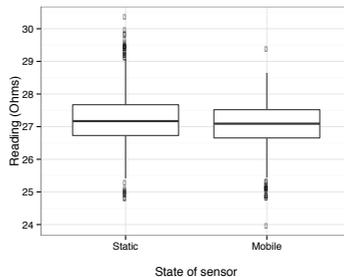


Fig. 4. Análisis de la variabilidad

mediante kriging para el conjunto de datos de referencia en la posición xy , y Δk_0 representa el total de la variación de los valores pronosticados para el conjunto de datos de referencia.

Analizando la tabla I, podemos ver que la media y la desviación estándar de los valores son casi las mismas en todos los casos, aunque el error aumenta cuando el período entre muestras aumenta. No obstante, los valores del error relativo son relativamente bajos, dado que el proceso de interpolación basado en kriging también actúa como un filtro de error, ayudando a aproximar el valor medio cuando se carece de suficientes valores de referencia.

Los mapas detallados para algunas trazas (5, y 80 segundos) se muestra en la Figura 5, junto con la fuente de datos utilizada para la interpolación kriging.

Al observar los mapas de calor, generados mediante interpolación espacial, podemos ver claramente que el nivel de detalle sufre una degradación a medida que aumentamos el período de muestreo. En particular, observamos diferencias significativas al aumentar el período de muestreo a 80 segundos, donde la distribución de ozono estimada es completamente diferente a la utilizada como referencia (5 segundos). Finalmente, en base a estos mapas, resulta bastante evidente que las pequeñas diferencias en términos de análisis estadístico básico puede representar enormes diferencias en términos de la distribución espacial real de dichos valores.

C. Impacto de muestreo espacial en predicciones geostatísticas

Finalmente hemos procedido a evaluar el impacto que tiene el muestreo espacial en el mapa de contaminación generado. Específicamente, queremos determinar hasta qué grado la elección de una ruta más corta, y por lo tanto menos exhaustiva en cuanto al área a monitorizar (reduciendo el tiempo de viaje y el número de muestras) afecta a la precisión de los pronósticos realizados.

Para encontrar la mejor estrategia de muestreo espacial producimos diferentes conjuntos de datos mediante la eliminación de fragmentos de la traza inicial. Concretamente, a partir de la traza completa (100% de los datos), se eliminan los trazados seleccionados con el fin de producir un viaje más corto, y sin embargo, válido, manteniendo ambas ubicaciones de inicio y de fin. Como resultado, hemos obtenido las trazas con el 72%, 50% y 42% de los datos. De forma similar a la sección anterior realizamos, para cada conjunto de datos, un análisis estadístico de los datos resultantes. Se generan, además, mapas de calor relativos a la contaminación, mediante interpolación kriging, y se calcula el error de predicción mediante la ecuación 1.

En la tabla II se presenta el análisis estadístico. La tabla muestra la media, la desviación estándar y el error relativo, siendo este último calculado utilizando el conjunto de datos de referencia inicial.

Según la tabla II, encontramos que el valor pro-

TABLA II
 ANÁLISIS ESPACIAL DE LAS MUESTRAS.

Tamaño muestra	Media	Desviación estándar	Error (ϵ_i)
100 %	60.3125	1.1403	-
72 %	60.4925	1.0003	0.0664
50 %	60.6652	1.1373	0.1180
46 %	60.6608	1.1373	0.8872
42 %	60.5127	1.0827	0.1470

medio está cercano al de referencia (60.31 ppm) en todos los casos, a pesar de ser en general ligeramente superior. Esto ocurre porque la primera trayectoria eliminada casualmente contenía los valores más bajos. En comparación con las variaciones temporales del muestreo realizado, (ver Tabla I), nos encontramos con que ahora el error tiene un aumento más pronunciado, lo que significa que la reducción de la ruta tomada por el área objetivo es propensa a eliminar muestras pertinentes, lo que se traduce en un mapa de contaminación menos detallado.

En la Figura 6, se muestran mapas detallados para los conjuntos de datos que representan el 100% y el 42% de los datos. Tomando como referencia estos mapas de calor, podemos observar claramente cómo el submuestreo espacial provoca una distorsión en la distribución espacial de la contaminación en toda la zona objetivo.

En general, podemos concluir que la granularidad del muestreo espacial es el factor más importante a tener en cuenta, siendo el tiempo de muestreo menos relevante, aunque importante. Por último, la orientación del sensor es el factor que menos impacto tuvo sobre los resultados obtenidos.

VI. CONCLUSIONES

En este artículo proponemos EcoSensor, una solución completa para la vigilancia del medio ambiente, que combina sensores de gama baja, smartphones y servicios cloud para medir los niveles de contaminación con una alta granularidad espacial. Nuestro enfoque se basa en el uso de un sensor móvil para proporcionar mediciones de contaminación, un smartphone para ofrecer información en tiempo real sobre las condiciones de calidad del aire, y que también actúa como un gateway para cargar datos recopilados al servidor en la nube. Además, EcoSensor proporciona un servidor para el procesamiento de datos y visualización disponible en la nube.

Para hacer frente a los desafíos asociados a la toma de mediciones móviles en un área objetivo, analizamos la influencia de la orientación de los sensores, así como el impacto que el tiempo y el muestreo espacial tienen en el proceso de detección. En particular, hemos variado el período de muestreo de 5 segundos a 80 segundos, así como la longitud del trayecto desde 100% a un 42%, pudiendo así determinar la estrategia de monitorización más eficaz. Los resul-

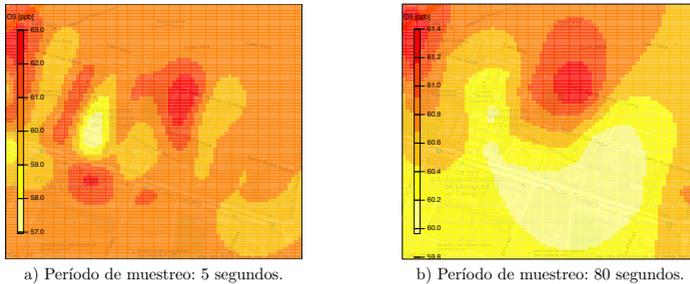


Fig. 5. Análisis de la distribución del ozono variando el período de muestreo.

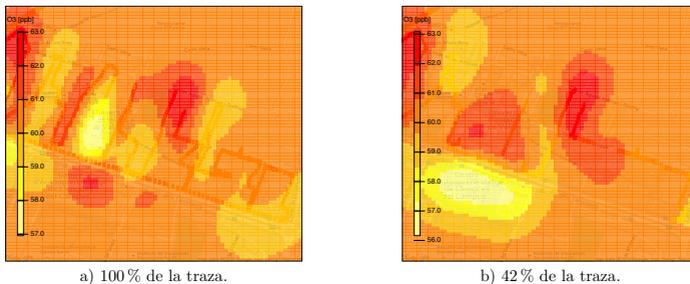


Fig. 6. Análisis de la distribución del ozono variando el tamaño del recorrido.

tados experimentales demuestran que la orientación del sensor y el período de muestreo, dentro de ciertos límites, tienen muy poca influencia en los datos capturados, mientras que el recorrido real tiene un mayor impacto en los resultados, especialmente cuando deseamos realizar una buena estimación de la distribución de los contaminantes en una área específica.

Como trabajo futuro nos planteamos mejoras en el proceso de interpolación espacial, así como el desarrollo de un sensor de contaminación móvil más pequeño.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente apoyado por el "Programa Estatal de Investigación, Desarrollo e Innovación orientada a retos de la Sociedad, Proyecto I+D+I TEC2014-52690-R", la Universidad de Cuenca", la Universidad Laica Eloy Alfaro de Manabí (ULEAM)", y el "Programa de becas SENESCYT de la República del Ecuador".

REFERENCIAS

[1] ETC/ACM, "European topic centre on air pollution and climate change mitigation," 2015. [Online]. Available: <http://acm.eionet.europa.eu/>

[2] T.-M. Chen, J. Gokhale, S. Shofer, and W. G. Kuschner, "Outdoor air pollution: ozone health effects," *The American journal of the medical sciences*, vol. 333, no. 4, pp. 244-8, 2007.

[3] R. D. Brook, J. R. Brook, B. Urch, R. Vincent, S. Rajagopalan, and F. Silverman, "Inhalation of fine particulate air pollution and ozone causes acute arterial vasoconstriction in healthy adults," *Circulation*, vol. 105, no. 13, pp. 1534-1536, 2002.

[4] A. Adam-poupard, A. Brand, M. Fournier, M. Jerrett, and A. Smargiassi, "Spatiotemporal modeling of ozone levels in quebec (canada): a comparison of kriging, land-use regression (lur), and combined bayesian maximum entropy lur approaches," *Environmental Health Perspectives*, vol. 970, no. January 2013, pp. 1-19, 2014.

[5] L. J. S. Liu and a. J. Rossini, "Use of kriging models to predict 12-hour mean ozone concentrations in metropolitan toronto - a pilot study," *Environment International*, vol. 22, no. 6, pp. 677-692, 1996.

[6] M. Brljović and V. Sretović, "Urban sensing smart solutions for monitoring environmental quality: Case studies from serbia," in *Congress Proceedings. 48th ISO-CARP International Society of City and Regional Planners World Congress: Fast Forward: Planning in a (hyper) dynamic urban context*, 2012.

[7] H. B. Deng and L. Zhang, "Design on zigbee wireless sensor network node," *Key Engineering Materials*, vol. 474-476, pp. 283-286, 2011.

[8] C. T. Calafate and B. Ducourthial, "On the use of mobile sensors for estimating city-wide pollution levels," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2015 International*. IEEE, 2015, pp. 262-267.

[9] Y. Cheng, X. Li, Z. Li, S. Jiang, Y. Li, J. Jia, and X. Jiang, "Aircloud: a cloud-based air-quality monitoring system for everyone," in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*

- *SenSys '14*. ACM, 2014, pp. 251-265.
- [10] Y. Zheng, F. Liu, and H.-p. Hsieh, "U-air: when urban air quality inference meets big data," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*. ACM, 2013, pp. 1436-1444.
- [11] Libelium, "Waspnote plug and sense smart environment available," 2015. [Online]. Available: <http://www.libelium.com/es/products/plug-sense/>
- [12] Raspberry-Pi, "Raspberry pi available," 2015. [Online]. Available: <https://www.raspberrypi.org/>
- [13] Arduino, "Arduino available," 2015. [Online]. Available: <https://www.arduino.cc/>
- [14] Zigbee, "Zigbee available," 2015. [Online]. Available: <http://www.zigbee.org/>
- [15] R-Foundation, "R project available," 2015. [Online]. Available: <https://www.r-project.org>
- [16] "Download android studio and sdk tools | android developers." [Online]. Available: <http://developer.android.com,urldate=2016-01-01>
- [17] U. S. E. P. Agency, "Air quality index available," 2015. [Online]. Available: <http://cfpub.epa.gov/airnow/index.cfm>
- [18] "Create your stunning website on wordpress.com." [Online]. Available: <https://wordpress.com/create/>

Reconstrucción de jugadas de billar en 3D

Fco. J Rodríguez-Lozano¹, Jose M. Palomares², J. Olivares³

Resumen—Este artículo presenta un sistema para la captación de jugadas de billar a partir de cámaras de bajo coste y su reconstrucción automática en 3D. El sistema cubre todos los pasos desde la adquisición y procesamiento previo de imágenes hasta la generación de un espacio virtual en un mundo 3D para representar las jugadas. El método de seguimiento de objeto puede ser utilizado para trabajar en aplicaciones de tiempo real. Se ha utilizado una mesa de billar americano en la que existen troneras dispuestas a lo largo de los laterales de la misma, esta característica no está presente en otros tipos de mesas. La existencia de troneras hace más complejo el mecanismo de captación y de seguimiento, por la posibilidad de que las bolas desaparezcan al introducirse en dichas troneras. Sin embargo, el sistema ha sido diseñado de manera robusta y flexible, y permite que pueda ser utilizado en otro tipo de mesas que dispongan o no de troneras. Además, los pasos del algoritmo que componen el sistema pueden ser utilizados en otros ámbitos, como pueden ser la detección de movimiento o la segmentación de círculos y cuadrados.

Palabras clave—Mesa de billar, Seguimiento de objetos, Detección de bolas, Detección de bordes, Filtrado de partículas, Procesamiento de imágenes

I. INTRODUCCIÓN

ES un hecho que la tecnología está evolucionando y mejorando cada día a pasos agigantados. Tal es así que en las últimas décadas hemos podido observar un gran incremento de los métodos de seguimiento automático de diferentes objetivos en movimiento. Estos métodos se aplican a sistemas los cuales cubren un amplio rango de aplicación, desde el seguimiento de misiles lanzados desde aviones hasta sistemas para ayudar a la gente a mejorar sus habilidades en ámbitos deportivos.

El sistema presentado en este artículo pretende dar un soporte para ayudar a los jugadores noveles en el juego del billar. Los jugadores podrán jugar con normalidad y una vez finalizada la partida podrán ver las jugadas recreadas en un mundo virtual en 3D, para así observar los movimientos y mejorar sus habilidades.

El uso de grandes sistemas *ad-hoc*, de gran coste y poca movilidad, limitan la expansión de este tipo de sistemas de ayuda al entrenamiento. Sin embargo, la aparición de sistemas de cámaras RGB-D y pequeños sistemas empujados de bajo coste van a permitir la incorporación de estos sistemas para jugadores noveles, con pocos recursos económicos y de disponibilidad de mesas de juego muy limitada.

Debido al hecho de que actualmente existen diferentes tipos de mesas de billar, el sistema deberá ser invariante al tipo de mesa utilizado. Sin embargo las bolas utilizadas aunque puedan ser de diferentes tamaños, deben de ser de las mismas dimensiones entre sí. Además, es necesario que las bolas tengan un color uniforme.

El presente artículo se organiza de la siguiente manera: Sec. II describe el sistema propuesto. En la Sec. III se pueden observar los métodos del sistema, que van desde

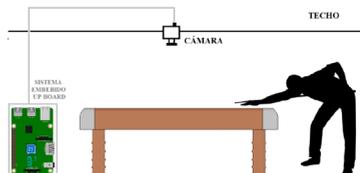


Fig. 1. Esquema del sistema.

la adquisición de imágenes hasta la reconstrucción de las jugadas en 3D. Los resultados y el análisis se muestran en la Sec. IV. Finalizando en la Sec. V con las conclusiones obtenidas de las realizaciones de los experimentos y en la Sec. VI con los posibles trabajos futuros.

II. METODOLOGÍA

El sistema realizará un seguimiento de las bolas y reconstruirá la jugada en un mundo virtual. En la Figura 1 se muestra el sistema que se encarga de realizar la grabación de los movimientos de las jugadas.

Para la grabación de las jugadas se requiere el cumplimiento de los siguientes requisitos:

- La iluminación debe de ser uniforme.
- Es necesario usar un dispositivo RGB-D para la obtención de imágenes en formato RGB y su imagen de datos de profundidad.
- El sensor RGB-D debe estar situado perpendicular al centro de la mesa de billar y a una altura mínima para que se pueda visualizar tanto el tapiz como la madera de la mesa completamente.
- La combinación de colores entre el tapiz y la madera de la mesa debe de ser de un alto contraste.
- El suelo sobre el que se encuentra la mesa debe de tener un color lo más uniforme posible.
- El taco de billar debe de ser de un color diferente al tapiz, la madera de la mesa y el color de las bolas utilizadas.
- Las bolas deben de tener todas el mismo tamaño y tener un color uniforme y diferente al del tapiz y el taco del billar.
- Para evitar oclusiones inesperadas, los jugadores deberán dejar libre la mesa de billar y esperar entre jugadas tres segundos antes de posicionarse.

Los requisitos anteriores se cumplen en la mayoría de los casos con una mínima intervención, ya que son las condiciones de juego más habituales en las partidas de billar. Para la parte de la reconstrucción el sistema debe de tener conocimiento de la siguiente información:

- Dimensiones de la mesa y del tapiz.
- Tamaño de las troneras.
- Tamaño y colores de las bolas.

¹Depto. Arquitectura de Computadores, Electrónica y Tecnología Electrónica, Universidad de Córdoba. e-mail: i02rolof@uco.es

²jmpalomares@uco.es

³olivares@uco.es

A. Revisión bibliográfica

Hay muchas propuestas de diferentes autores para la realización de un sistema con objetivos similares al que se propone en este trabajo. Jeberal *et al.* [1] presentaron un artículo sobre el diseño de un sistema de ayuda al entrenamiento de jugadores de billar basado en dispositivos de realidad aumentada de la época. La mayoría de los autores sólo se centran en la resolución de problemas similares a sólo algunas etapas del método propuesto en este artículo, como puede ser, la detección de bolas de billar. Otros autores coinciden en que la cámara no tiene porque estar colocada en posición cenital a la mesa, tal y como se mostró en la Figura 1. Por ejemplo en [2–4], la cámara no se sitúa sobre la mesa, con lo que requieren realizar transformaciones previas para simular una vista desde arriba. Sin embargo, este enfoque no cenital suele dar lugar a un mayor número de oclusiones indeseadas, cuando el jugador se sitúa en algunas posiciones para efectuar una jugada.

En lo que corresponde a la detección del tapiz, Vachaspati [3] propone un buen método para la segmentación del mismo, pero puede acarrear problemas cuando el tapiz es de un color diferente al que se utilizó en los experimentos. Otras propuestas, como pueden observarse en [2], realizan una segmentación basada en las troneras de la mesa. Esto, aunque interesante, podría dar problemas cuando la mesa de billar carezca de troneras, como es el caso de las mesas de billar francés.

Por otro lado, no existe ningún consenso en utilizar un tipo específico de bolas de billar. Diferentes experimentos [5] demuestran que la segmentación de las diferentes bolas de billar es una tarea compleja. Además, ésta se complica cuando las bolas se encuentran en movimiento, debido a que hay pares de bolas con colores similares que en movimiento pueden parecer las mismas. Una solución es la que se propone en [6], donde se utilizan bolas de *snooker* (otra modalidad de billar) las cuales tienen un color uniforme, que se mantiene invariante durante los movimientos rotatorios de las bolas.

En cuanto al seguimiento de las bolas tampoco existe una única metodología a seguir. Por ejemplo, en [7–10] se realiza una predicción de las trayectorias de las bolas basándose en parámetros físicos y estadísticos. Sin embargo, en este artículo se ha decidido realizar el seguimiento basándose en algoritmos que tienen en cuenta la posición de la bola en cada instante de tiempo sin tener en cuenta ningún tipo de modelo físico que contemple, por ejemplo, la fricción de la bola con el tapiz.

III. MÉTODO PROPUESTO

Los pasos a seguir por el sistema para la reconstrucción de las jugadas en 3D se muestran en la siguiente lista:

- A. Adquisición de imágenes.
- B. Detección de la mesa.
- C. Segmentación de las bolas.
- D. Seguimiento de las bolas.
- E. Reconstrucción en 3D.

Los siguientes apartados explican los métodos de forma más específica para cada uno de los pasos expuestos en la anterior lista.

A. Adquisición de imágenes

El objetivo de esta parte es obtener y grabar imágenes en el espacio de color RGB e imágenes de profundidad del sensor. Ambas imágenes deben de estar alineadas para que haya una correspondencia de pixel a pixel entre las dos imágenes.

Además, las imágenes en el espacio de color RGB serán transformadas al espacio de color HSV. Esta transformación hará que el análisis de las imágenes sea más simple en los pasos posteriores.

B. Detección de la mesa

Como el objetivo en el paso anterior era cambiar el espacio de color de RGB a HSV, esto asegura ahora el contraste entre el tapiz y la madera de la mesa. La segmentación del tapiz se logra siguiendo los siguientes pasos:

B.1 Algoritmo de Canny

Se aplica el *método de Canny* [11] para detectar todos los bordes de la imagen. Además, se realiza una dilatación binaria al resultado devuelto por *Canny* para obtener resultados más robustos en el siguiente paso.

B.2 Transformada de Hough

Se aplica la *transformada de Hough* [12] para la detección de líneas, con el fin de detectar todas las líneas rectas de los contornos detectados en el paso previo.

B.3 Cálculo de intersecciones

En este punto es necesario calcular las intersecciones entre todas las líneas halladas en el paso anterior. Hay que tener en cuenta que se deben descartar aquellas intersecciones que se encuentran fuera del rango de la imagen captada por el sensor, ya que es posible que dos líneas se crucen fuera del rango.

B.4 Buscar vértices

Basándose en las intersecciones encontradas, se comprueban cuales de ellas, en grupos de cuatro, cumplen la condición de formar ángulos de 90°, dos a dos. De este modo, se podrán encontrar todos los posibles rectángulos que se pueden formar.

B.5 Buscar el tapiz

El último paso de esta parte es encontrar cual de los rectángulos coincide con el tapiz de la mesa. Para lograr esto, se calcula el área del tapiz teniendo en cuenta la distancia proporcionada por el sensor RGB-D y las medidas de la mesa. La madera de la mesa será descartada y aquellos rectángulos que no cumplan dicha condición de tamaño también serán descartados. Por ello, el sistema se quedará sólo con aquel (o con una media en el caso de que haya muchos rectángulos de dimensiones similares cercanos) que corresponde con el terreno de juego que es el tapiz.

C. Segmentación de las bolas

Una vez el sistema ha logrado detectar el tapiz, se procede a la segmentación y separación de las bolas por colores. Además de dicha separación por colores, este paso

obtendrá la posición relativa en el tapiz de cada una de las bolas. Los pasos para lograr este objetivo son los siguientes:

C.1 Cálculo del gradiente

A la región del tapiz se le aplica el operador de *Sobel* [13] para obtener la derivada de la imagen. A dicha derivada se le realiza una umbralización para eliminar aquellas partes de tamaño excesivamente pequeño, ya que se puede descartar que sean bolas.

C.2 Transformada de Hough

La *transformada de Hough* [14] se aplica para obtener todos los posibles círculos con un radio similar al de las bolas, indicado previamente al sistema.

C.3 Votaciones

Por cada círculo detectado, se genera una pequeña región de interés, que se analiza píxel a píxel en el espacio de color *HSV*. Por cada bola, se crea un número específico de contadores. Habrá tantos contadores como diferentes colores de bolas. El rango de colores de las bolas utilizado en el presente artículo es:

- [0, 12] y [340, 360] en el canal H para las bolas rojas.
- [50, 66] en el canal H para las bolas amarillas.
- [0, 17] en el canal V para la bola negra.
- [88, 100] en el canal V para la bola blanca.

El rango del canal H es de [0, 360], mientras que el rango para los canales S y V es de [0, 100]. Para otros colores, los rangos se definirán modificando los valores de los canales H y V.

Para descartar falsos positivos, se hace uso de la distancia que provee el sensor RGB-D, eliminando todos aquellos círculos que no se encuentran sobre el tapiz. Los círculos no eliminados al finalizar esta etapa serán identificados como bolas.

D. Seguimiento de las bolas

El objetivo de esta parte es el seguimiento de múltiples bolas de billar en movimiento. Para ello, se propone utilizar el algoritmo de "filtrado de partículas" [15]. Sin embargo, la formulación del algoritmo de filtrado de partículas básico no es suficiente, debido a que tal y como está formulado sólo realizaría el seguimiento de un único objeto. Por ello, se modifica el filtrado de partículas para que sea capaz de realizar el seguimiento de múltiples objetos. La Figura 2 muestra las etapas del algoritmo de filtrado de partículas propuesto. El algoritmo repetirá los pasos desde el segundo para cada nueva imagen de las que se han grabado. Las partículas quedan definidas como una estructura de datos que contienen la siguiente información en un determinado instante de tiempo:

- *Coordenadas* x, y, z : el par (x, y) se corresponde con la posición donde se encuentra situada la partícula dentro del tapiz, y la coordenada z , a la profundidad a la que se encuentra, considerando como profundidad 0 la superficie del tapiz, y todo lo que esté a una altura inferior al mismo se considerarán alturas negativas.



Fig. 2. Pasos del seguimiento de las bolas.

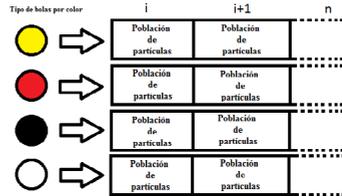


Fig. 3. Esquema de la población de partículas.

- *Peso del histograma*: medida que muestra la similitud entre el histograma de la bola en el instante inicial, antes de ejecutar el presente algoritmo, y el histograma de la partícula.
- *Peso de la distancia*: medida que muestra la distancia entre la bola estimada en la etapa anterior y la estimada en esta etapa, respecto a la partícula que estamos analizando.
- *Peso Total*: combinación de los dos pesos anteriores.
- *Radio de la partícula*: límite de la ventana que representa a la partícula.

Las partículas a diferencia de la formulación original del algoritmo no pueden cambiar su tamaño de ventana debido a que las bolas no pueden cambiar de tamaño ni pueden deformarse.

D.1 Inicializar partículas

Esta etapa del algoritmo se hace sólo la primera vez que se ejecuta el algoritmo de seguimiento. Esta parte tiene los siguientes pasos:

- A cada bola se le asigna una población de partículas. Esta población es independiente para cada bola, lo que asegura que en etapas posteriores del algoritmo cuando se analizan las bolas sólo se estará analizando una bola concreta sin que interfiera dicha bola con otras bolas de colores similares. La Figura 3 muestra un ejemplo de las poblaciones de partículas asignadas a cada bola.
- Cada partícula es inicializada con el par de coordenadas (x, y) que le corresponde a cada bola. La coordenada z toma el valor 0 (sobre la superficie del tapiz). Para cada una de estas partículas dado que es la etapa inicial y la posición inicial corresponde con la que se ha calculado en pasos anteriores, el peso seleccionado será el mejor posible, tomando como

rango $[0, 1]$, donde 0 corresponde con el mejor peso posible y 1 con el peor.

Es importante indicar que cuanto mayor sea el número de población de partículas, más realista y fiel será la representación de los movimientos de las bolas en la reconstrucción de las jugadas. Sin embargo, en dichos casos, se hace un mayor uso de memoria y se necesitará un mayor tiempo de ejecución.

D.2 Actualizar partículas

Esta etapa es la más importante del algoritmo de filtrado de partículas y una de las que mayor carga computacional tiene. Los pasos que sigue son:

- Se almacena la mejor partícula hallada en el estado anterior basándose en el peso. Para la primera ejecución del algoritmo los valores almacenados coincidirán con los de la etapa de *Inicializar partículas*.
- En este punto es necesario realizar la actualización de la coordenada z de las partículas. Cuando una bola entra en una tronera el algoritmo detendrá la búsqueda de dicha bola y a lo largo de un número de etapas posteriores la coordenada z de la mejor partícula irá disminuyendo su valor hasta dejarla completamente por debajo del tapiz. Todas las demás bolas que no se encuentran dentro de una tronera no se verán afectadas por ningún tipo de cambio en este paso.
- Se genera una nueva población de forma aleatoria, pero teniendo en cuenta los rangos de generación para cada bola. Este rango asegura que una bola no será buscada en una posición en donde la probabilidad de encontrarla sea extremadamente baja. Además, como el sistema debe de hacer el seguimiento de bolas con colores similares, las nuevas partículas generadas no podrán generarse en aquellas posiciones donde se encuentra la mejor partícula de otra bola de un color similar.
- Tras obtener la nueva población de partículas los pesos de las mismas deben de ser actualizados:
 - *Peso basado en histogramas*: Es la métrica que compara la relación existente entre el histograma de la partícula y el histograma de una bola en el instante inicial del algoritmo. Para hallar la distancia existente entre los histogramas se hace uso de la distancia propuesta por Bhattacharyya [16].
 - *Peso basado en distancias*: Es la relación existente entre la distancia del centro de una partícula en este instante con la mejor partícula del instante de tiempo anterior. Para medir la distancia se utiliza la ecuación de la distancia *Euclídea* en la que se ha propuesto que las distancias menores serán mejor porque serán las partículas cuya distancias sean menor las que serán más representativas de la bola.
 - *Peso total*: Es la combinación de los pesos anteriores. Los pesos a diferencia de la formulación básica del algoritmo de partículas no están normalizados por el número de partículas. Este peso siempre se encontrará en el rango $[0 - 1]$ dándole la misma importancia a cada uno de los pesos. Así

la forma de obtener esta medida queda como la media de los pesos anteriores.

- Para cada partícula es necesario comprobar lo siguiente:
 - *Distancia*: Si la mejor partícula no ha tenido un desplazamiento mayor que $\frac{1}{3}$ de la distancia de su radio (*este valor ha sido elegido empíricamente*), la mejor partícula de esta generación será la mejor partícula del instante de tiempo anterior.
 - *Oclusión*: Si la mejor partícula no se parece al objeto que estamos siguiendo (*está oculta o tapada*), la mejor partícula de esta generación será la mejor partícula del instante de tiempo anterior.
- El último paso del algoritmo es almacenar la posición de cada una de las bolas, o lo que es lo mismo, almacenar la posición de cada una de las mejores partículas en cada instante de tiempo.

Esta parte del algoritmo proporciona al método propuesto la capacidad de seguimiento de múltiples objetos con un paso de repulsión ante bolas del mismo color y actualizaciones de los pesos. Debido al hecho de que el algoritmo trabaja por colores, no cabe la posibilidad de que bolas de diferente color interfieran entre ellas.

D.3 Obtener mejores partículas

En este paso, se comprueba la población de partículas de cada bola y se elegirá aquella partícula representativa de la bola en función del mejor peso encontrado de entre todas ellas.

Para ordenar las partículas, se usará el algoritmo *Quick-Sort* [17] dado que es un método que se vuelve más eficiente a la hora de realizar la ordenación cuando el número de partículas empieza a ser elevado.

D.4 Remuestreo de partículas

En este paso se genera la nueva población de partículas para cada bola que será modificada nuevamente en el paso de *Actualizar Partículas*.

La población se generará por un “*enfoque orientado a la diversidad*”. Este enfoque desecha la idea de que la mejor partícula encontrada realmente es la correcta. De este modo, la nueva población de partículas será un 50% de las partículas totales de la población iguales a la mejor partícula hallada. Un 30% como copia de la segunda mejor partícula hallada. Y por último un 20% como copia de las mejores partículas halladas a partir de la segunda mejor hallada. Con esto se consigue dar una mayor tolerancia a fallos al algoritmo.

E. Reconstrucción en 3D

La reconstrucción en un mundo virtual en 3D depende de los mecanismos y la sintaxis del lenguaje utilizado. Sin embargo, existen dos módulos principales que deberán desarrollarse: la generación de la mesa de billar y las animaciones.

E.1 Generación de la mesa de billar

La generación del tapiz es un proceso complejo, pero se puede simplificar con los datos que se han especificado previamente al sistema. Dado que el diámetro de



Fig. 4. Aproximación de las troneras.

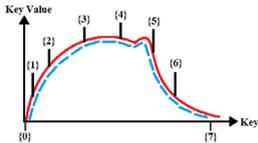


Fig. 5. Interpolación de movimientos.

las troneras es conocido y tal y como se muestra en la Figura 4, las troneras se pueden aproximar mediante puntos. Los puntos de 0° a 90° corresponden con las troneras de los vértices de la mesa y los puntos de 0° a 180° a las troneras centrales de la mesa. Para los agujeros en la madera se hace de forma análoga.

E.2 Generación de las bolas

La generación de las bolas no se limita sólo a desarrollar visualmente las bolas, sino a asignarles su propio movimiento a cada una de las bolas. Para la generación de las animaciones de dichos movimientos se propone la utilización de lo que comúnmente se denominan como nodos interpoladores. Estos nodos cambian la posición de un objeto a lo largo del tiempo.

La Figura 5 muestra un ejemplo de cómo un movimiento curvilíneo se puede aproximar mediante varios puntos a través de un nodo interpolador. Las partes más importante de este nodo tal y como se muestra en la figura son:

- *key*: instante de tiempo.
- *keyValue*: posición en el instante de tiempo, *Key*.

IV. RESULTADOS EXPERIMENTALES

La realización de los experimentos se han logrado utilizando los siguientes instrumentos:

- UP Board con Intel Atom x5-Z8350 Quad-core 2GB RAM.
- SSD Samsung 750 EVO — 250 GB.
- Kinect v1 para Xbox 360.
- Tamaño de las imágenes 640x480 (VGA).
- Taco de billar de color azul.
- Mesa de billar americano:
 - Tamaño de la mesa: 211.5 x 120.5 x 78 cm.
 - Tamaño del área de juego: 185.4 x 93.5 cm.
- Bolas con color uniforme de 57mm. de diámetro.

Los siguientes experimentos han sido seleccionados para demostrar el propósito funcional del sistema. En la Tabla 1 pueden observarse los resultados de los tiempos cuando cambia el número de partículas utilizado para realizar el seguimiento.

Experimento 1:

Este es experimento muestra varias bolas en movimiento sobre el tapiz en el momento que el

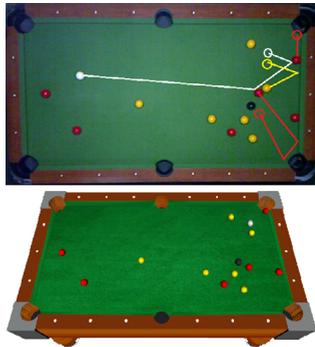


Fig. 6. Resultados del experimento 1. Arriba: Movimientos de las bolas sobre la imagen antes de que comience la jugada. Abajo: Reconstrucción de la jugada en un instante final de la misma.



Fig. 7. Resultados del experimento 2. Arriba: Movimientos de las bolas sobre la imagen antes de que comience la jugada. Abajo: Reconstrucción de la jugada en un instante intermedio de la misma.

jugador realiza su jugada. En este experimento se pueden observar diferentes colisiones y rebotes entre bolas de distintos colores. El estado inicial de la jugada y el final se muestran en la Figura 6. Además, en la misma figura, se ha añadido el resultado en un instante final de la reconstrucción de la jugada en un lenguaje 3D.

Experimento 2:

Como se muestra en la Figura 7, este experimento consiste en comprobar el comportamiento del sistema cuando hay pocas bolas y además la bola negra cae dentro de una de las troneras.

Experimento 3:

En este caso el sistema necesita seguir diferentes bolas en movimiento del mismo color. Además, una de las bolas rojas entra dentro de una tronera movida por otra bola del mismo color. A su vez, dicha bola moverá posteriormente otras bolas. Como se puede observar en



Fig. 8. Resultados del experimento 3. Arriba: Movimientos de las bolas sobre la imagen antes de que comience la jugada. Abajo: Reconstrucción de la jugada en un instante intermedio de la misma.

la Figura 8, el objetivo que persigue este experimento es el de comprobar la robustez del algoritmo frente al movimiento de bolas generados por colisiones con bolas del mismo color.

Es importante destacar que en el presente experimento, la bola roja que entra en la tronera, volverá a ser captada por el sensor en instantes de tiempo posteriores (debido al modo de retorno de bolas de la mesa). Pero no volverá a realizar el seguimiento de dicha bola ya que en instantes de tiempo anteriores entró por una tronera y se deja de realizar el seguimiento.

TABLA I
 TIEMPO DE LOS EXPERIMENTOS

Experimento	Bolas	Partículas	Tiempo (en mS)
1.1	14	14500	159.7
1.2	14	8250	96.7
2.1	4	40000	388.1
2.2	4	800	22.0
3.1	15	31000	326.7
3.2	15	83000	837.0

A. Discusión de resultados

Debido al hecho de que no existen métricas aceptadas y establecidas, se hace complicado realizar una evaluación de la precisión de la reconstrucción. Para alcanzar una evaluación de calidad subjetiva se ha aplicado *Mean Opinion Score, MOS* [18], mostrando los resultados a 7 diferentes jugadores semiprofesionales en el ámbito del billar para poder evaluar de manera empírica la similitud entre la jugada real y la reconstrucción. De acuerdo con las opiniones y coincidiendo con el número de bolas y partículas utilizadas como puede observarse en la Figura 9, las jugadas que mayor puntuación han tenido por parte de los profesionales han sido aquellas en las que el número de partículas es mayor. Además, cuando el número de partículas es muy bajo pueden observarse pequeñas desviaciones en las trayectorias del recorrido

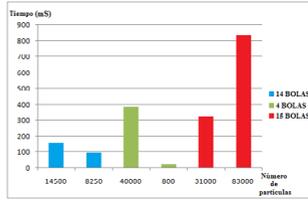


Fig. 9. Relación entre el número de partículas, bolas y tiempo que se tardan en generar los resultados de los experimentos.

de las bolas. Sin embargo, aunque se dé este pequeño fenómeno de ligeras desviaciones, el sistema es capaz de seguir las bolas en movimiento de manera efectiva. Por limitaciones de espacio no se ha incluido los datos desglados. La valoración *MOS* de todos los experimentos tomando en cuenta todos los jugadores ha sido de 6.2 (sobre 10). El experimento con mejor valoración *MOS* ha sido el Experimento 3 con 83000 partículas, con un valor de 7.3 (sobre 10). Por el contrario, el de peor valoración ha sido el Experimento 2 con 800 partículas, con un valor de 4.9 (sobre 10).

V. CONCLUSIONES

Los resultados obtenidos son en su mayoría exitosos en la reconstrucción, lo que se puede concluir que se alcanza el objetivo del sistema propuesto. Sin embargo, el sistema presenta algunas deficiencias. La mayoría de estos problemas se detectan en la fase del seguimiento de las bolas. Estos fallos están relacionados con el número de partículas utilizadas en algunos casos, pero la gran mayoría de los cuales son debidos a las limitaciones físicas del hardware utilizado (*UP Board Intel Atom x5-Z8350 y Kinect v1*). La utilización de otro sistema computacional de mayor potencia y de otro sensor con mayor calidad de imagen y mayor tasa de refresco, tanto en las imágenes RGB como en las imágenes de profundidad, permitiría realizar una reconstrucción de las jugadas mucho más precisas y realistas.

En cualquier caso, se ha mostrado que un pequeño sistema empotrado de bajo coste y limitada capacidad de cómputo puede ser capaz de realizar procesos tan complejos como la captación, seguimiento y reconstrucción de jugadas de billar con múltiples objetos en movimiento.

VI. TRABAJO FUTURO

Como mejora al sistema se propone la inclusión del taco de billar en la reconstrucción 3D, para lograr aún más realismo al sistema sacando información sobre la inclinación del taco en los tiros. Por otro lado, aunque el sistema desarrollado no provee de ningún método de seguimiento que se base en la utilización de modelos físicos teóricos, sería interesante añadir partes de mencionados modelos. De este modo se conseguiría que en caso de oclusiones parciales o totales el sistema no dependa únicamente de la información visual sino que tendría en cuenta una predicción estadística para la realización de la

reconstrucción. Las etapas del algoritmo de seguimiento, a su vez, pueden ser mejoradas añadiendo que el radio de búsqueda de las bolas que no se encuentran en movimiento o con movimientos lentos varíe en función del comportamiento de las bolas. Con esto se conseguiría que el número de partículas asignados a cada bola también fuese dinámico, pudiendo asignar menos partículas a aquellas bolas que no se encuentran en movimiento.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado mediante los proyectos P11-TIC-7462 y DPI2013-47347-C2-2-R.

REFERENCIAS

- [1] T. Jebara, C. Eyster, J. Weaver, T. Starner, and A. Pentland, "Stochastics: augmenting the billiards experience with probabilistic vision and wearable computers," *Digest of Papers. 1st Intl. Symposium on Wearable Computers*, pp. 138–145, 1997.
- [2] S. Weatherford, "Pool Cue Guide Determination of Guide Vectors Under Adverse Lighting , View Aspect and Scale," <https://www.semanticscholar.org/paper/Pool-Cue-Guide-Determination-of-Guide-Vectors-Weatherford/43997192b014739e8fe06cdf6a0607f145a0d46f/pdf>, 2013.
- [3] P. Vachaspati, "A Computer Vision System for 9-Ball Pool," Tech. Rep., MIT, 6.869 — Computer Vision., 2012, <http://pranjalv.com/poolvision.pdf>.
- [4] H. Uchiyama and H. Saito, "AR display of visual aids for supporting pool games by online markerless tracking," *Proceedings 17th International Conference on Artificial Reality and Telexistence, ICAT 2007*, pp. 172–179, 2007.
- [5] B. J. Baekdahl and S. Have, *Detection and Identification of Pool Balls using Computer Vision*, Ph.D. thesis, 2011.
- [6] M. L. Parry, P. A. Legg, D. H. S. Chung, I. W. Griffiths, and M. Chen, "Hierarchical event selection for video storyboards with a case study on snooker video visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 1747–1756, 2011.
- [7] A. Gubalkhakova and W. G. Kropatsch, "Video Analysis of a Snooker Footage Based on a Kinematic Model," *Structural, Synthetic, and Statistical Pattern Recognition*, pp. 8621, pp. 223–232, 2014.
- [8] S. Mathavan, M. R. Jackson, and R. M. Parkin, "Application of high-speed imaging to determine the dynamics of billiards," *American Journal of Physics*, vol. 77, no. 9, pp. 788, 2009.
- [9] S. Mathavan, M. R. Jackson, and R. M. Parkin, "A theoretical analysis of billiard ball dynamics under cushion impacts," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 224, no. 9, pp. 1863–1873, 2010.
- [10] M. Salzmann and R. Urtasun, "Physically-based motion models for 3D tracking: A convex formulation," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2064–2071, 2011.
- [11] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, June 1986.
- [12] J. Matas, C. Galambos, and J. Kittler, "Robust detection of lines using the progressive probabilistic hough transform," *Comput. Vis. Image Underst.*, vol. 78, no. 1, pp. 119–137, Apr. 2000.
- [13] I. Sobel and G. Feldman, "A 3x3 Isotropic Gradient Operator for Image Processing", no. JANUARY 1968, 1968.
- [14] C. Kimme, D. Ballard, and J. Sklansky, "Finding circles by an array of accumulators," *Commun. ACM*, vol. 18, no. 2, pp. 120–122, Feb. 1975.
- [15] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on Particle Filters for On-line Non-Linear/Non-Gaussian Bayesian Tracking," *Ieee*, vol. 50, no. 2, pp. 174–188, 2001.
- [16] A. Bhattacharyya, "On a measure of divergence between two multinomial populations," *Sankhya: The Indian Journal of Statistics (1933-1960)*, vol. 7, no. 4, pp. 401–406, 1946.
- [17] E. Horowitz and S. Sahni, *Fundamentals of computer algorithms*, Computer Science Press, 1978.
- [18] ITU-T, "Methods for subjective determination of transmissions quality," Recommendation P.800, 1996.

Consumo Energético de Aplicaciones Antivirus en Android

Elsa Vera, José Arteaga, Jorge Pincay,¹ Willian Zamora, Alex Santamaría²

Resumen— La presente investigación tiene por objetivo efectuar un estudio comparativo sobre el consumo de energía asociado a las aplicaciones antivirus para Smartphones que ejecuten el sistema operativo Android. Las características y atributos de los dispositivos utilizados en este estudio son detallados, juntos con los detalles de la funcionalidad ofrecida por los diferentes antivirus. Se propone una metodología que comprende el desarrollo de una aplicación que a través de un servicio realice mediciones periódicas del porcentaje restante de la batería y el voltaje demandado por las aplicaciones; permitiendo estimar las variaciones de voltaje que generan las aplicaciones antivirus y su impacto energético sobre la batería. Los resultados experimentales muestran que en general, las aplicaciones antivirus tienen un alto consumo de energía con niveles de potencia que van desde 6 a 16mW cuando la aplicación está activa, aunque igualmente se verifican las distintas soluciones antivirus estudiadas.

Palabras clave— Consumo energético, dispositivos móviles, Antivirus, Android, SmartPhone.

I. INTRODUCCIÓN

LOS teléfonos o dispositivos móviles tienen servicios atractivos y características de cómputo que compiten con el rendimiento de los ordenadores personales; sin embargo, consumen una gran cantidad de energía [1]. La mayoría de los dispositivos móviles utilizan baterías recargables electro-químicas de iones de litio (Li-ion) que son de corta duración, especialmente cuando se mantiene activa la conexión a redes de datos y se usan constantemente aplicaciones o servicios [2]. Como consecuencia, la duración de carga de la batería se ha convertido en un problema de disponibilidad para el usuario, quien tiene que preocuparse porque la batería no va a durar el tiempo suficiente; por lo tanto, el tiempo de duración de la batería es una de las principales prioridades de los fabricantes de dispositivos móviles [1].

Existen herramientas para medir el consumo eléctrico en distintos dispositivos móviles, por ejemplo, en Android se incluye la herramienta DOZE [3], que permite monitorizar el consumo porcentual, a nivel de componentes y aplicaciones. Además, en Google Play se pueden encontrar aplicaciones que modelan la potencia consumida por los componentes principales como lo son la CPU, interfaces de comunicación, pantalla, GPS y las aplicaciones utilizadas.

Al referirnos a las aplicaciones utilizadas, aparecen los antivirus [4], como medida básica para proteger al dispositivo contra los ataques de malware, pues éste constituye una amenaza constante y real

que afecta gravemente al usuario, sea emocional o financieramente. Este escenario convierte a los antivirus en aplicaciones imprescindibles en un dispositivo móvil y que, por lo tanto, merecen ser estudiados especialmente en lo que respecta al tema de consumo energético [4], más aún, si se considera que no han evolucionado hasta el punto de ser eficientes como las versiones que protegen los ordenadores personales.

La presente contribución se resume en lo siguiente:

- Se propuso una metodología que permita monitorizar el consumo energético de aplicaciones antivirus en dispositivos móviles con sistemas operativos Android.
- Se desarrolló una aplicación que se ejecuta en segundo plano y permite registrar los valores relativos a el voltaje, el porcentaje de batería y la capacidad.
- Se evaluó y comparó los resultados obtenidos a través de gráficos estadísticos.

El resto de este documento está organizado como sigue: en la sección II se presentan algunos trabajos relacionados; en la sección III, se describe los diferentes componentes que conforman la propuesta; en la sección IV, se describe la metodología de evaluación; en la sección V se muestran los resultados experimentales de la evaluación y en la sección VI se presenta las conclusiones y trabajos futuros.

II. TRABAJOS RELACIONADOS

La duración de la batería es un factor importante en el desarrollo y despliegue de las aplicaciones y servicios en dispositivos móviles, por lo que el usuario tiene que estar informado del porcentaje disponible de energía, para proporcionarle un uso conveniente, considerando que la ejecución de algunas aplicaciones demanda mayor consumo de batería y el usuario puede decidir iniciarlas o no.

Debido a la necesidad de conocer cuanta energía consume un dispositivo móvil han surgido varias investigaciones, una de ellas es la realizada de Manet et al. [5], en la cual analizan el consumo de energía por la interfaz IEEE 802.11 en diferentes modos de operación, demostrando como la tasa del tráfico de transmisión descarga la batería.

En otras investigaciones [6], [7] y [8], la mayor parte de energía consumida en dispositivos móviles se atribuye a las comunicaciones; al GPS y el display [9]; a las aplicaciones que ejecutan procesos en segundo plano [2]; al Bluetooth, WiFi, radio celular, red de datos e incluso la iluminación generada por el fondo de pantalla [6], [2], [10], [11]. En general, si las aplicaciones no utilizan el hardware de forma prudente

¹FACCI, Universidad Laica Eloy Alfaro de Manabí, Email: {elsa.verapb,jose.arteaaga,jorge.pincay}@live.uileam.edu.ec

²DISCA-DSIC, Universitat Politècnica de València, Email: e-mail: wilzame@posgrado.upv.es, asantamaria@dsic.upv.es

te aumenta el consumo de energía, por ejemplo, con la frecuencia de despertar del dispositivo en segundo plano y las transferencias simultáneas de datos a través de Internet [10].

Las investigaciones revisadas reflejan diversos tipos de problemáticas, algunos de los modelos diseñados para conseguir los datos y evaluar la potencia consumida de los diversos componentes de un smartphone son tomadas con equipos electrónicos concretos, capaces de obtener estos valores en tiempo real, pero estos sólo funcionan en dispositivos con el mismo tipo de tecnología [8].

Respecto a los desarrolladores de aplicaciones móviles, estos deben reconocer las necesidades energéticas de sus aplicaciones [12], pues en cierto modo cada aplicación que se ejecuta en un dispositivo móvil contribuye de manera diferente al consumo de la batería [13] y los programas antivirus no pueden ser una excepción.

Finalmente, en la presente revisión bibliográfica se encontraron trabajos [14] y [15], los cuales evalúan la eficiencia energética de ciertos antivirus para la plataforma Android. Específicamente miden el consumo durante varias operaciones como: el proceso de escaneo de la aplicación después de la instalación, el escaneo completo del dispositivo, y el escaneo de la tarjeta SD. Nuestra propuesta se diferencia de lo anterior ya que nosotros medimos el impacto en general que tiene diferentes antivirus sobre el consumo de energía en la batería de un smartphone y adicionalmente proponemos una metodología base para dichos estudios.

III. DESCRIPCIÓN DE LA PROPUESTA

La propuesta de la metodología consta de cuatro componentes que se muestran en la Figura 1. El componente principal es el servicio que se ejecuta en segundo plano, desarrollado en Android Studio [16]. El segundo componente es la lectura del consumo energético para el dispositivo móvil en sus diferentes estados: suspendido, activo e inactivo. En el tercer componente intervienen los diferentes antivirus que deben ser instalados e inicializados para su respectiva evaluación. Finalmente se muestra el componente que realiza la extracción y análisis estadísticos de los resultados.

A. Administración de energía en Android

El sistema operativo Android realiza una administración de energía asociada a los métodos de Linux, APM (Advanced Power Management) y ACPI (Advanced Configuration and Power Interface); sin embargo, emplea una política mucho más agresiva de gestión y ahorro de energía, con la premisa de que la CPU no consume sino hay aplicaciones ni servicios que necesiten energía, por lo tanto tiene su propia extensión de gestión de energía denominada PowerManager, que proporciona controladores de bajo nivel con el fin de gestionar los periféricos soportados.

Un wakelock [17] es una función del servicio PowerManager, que permite controlar el estado de energía

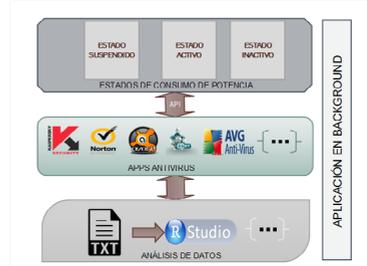


Fig. 1. Componentes de la metodología propuesta.

del dispositivo de manera dinámica [6], las aplicaciones y componentes tienen que crear y adquirir wakelocks para mantener los recursos activos, sino hay algún wakelock activo, la CPU se apaga y pasa a un estado de bajo consumo.

En la máquina de estado de PowerManager existen tres estados incorporados al modelo de administración de energía (i) SLEEP, (ii) NOTIFICATION y (iii) AWAKE. Cuando una aplicación adquiere un wakelock completo, se puede producir un evento por actividad de la pantalla o el teclado y el dispositivo va a mantenerse o cambiar al estado AWAKE; si el tiempo de espera pasa o se presiona el botón de encendido/apagado se produce la transición al estado NOTIFICATION. Mientras se adquiere un wakelock parcial el dispositivo se mantendrá en el estado NOTIFICATION si todos los wakelocks parciales se liberan, el dispositivo pasa al estado SLEEP, si en este modo todos los recursos se activan ocurre la transición al estado AWAKE.

A continuación se detallan los wakelocks que existen en Android:

- FULL_WAKE_LOCK.- Mantiene la pantalla y el teclado encendidos con brillo máximo.
- SCREEN_BRIGHT_WAKE_LOCK.- Se encarga de asegurar que la pantalla esta encendida con el máximo brillo, mientras el teclado puede tener apagada su iluminación.
- SCREEN_DIM_WAKE_LOCK.- Mantiene la pantalla encendida, además permite bajar el brillo y el teclado puede apagar su iluminación.
- PARTIAL_WAKE_LOCK.- Asegura que la CPU se mantiene activa; sin embargo, la pantalla puede apagarse.

La TABLA I muestra que wakelocks disponibles y utilizados por Android para reducir el consumo de energía en un dispositivo móvil.

B. Apps Antivirus

En este componente se encuentran los diferentes antivirus que han sido considerados para las evaluaciones. Para esto se tomó como referencia a cinco de los veinticinco productos de seguridad para Android,

TABLA I
 WAKELOCKS EN ANDROID.

FLAG VALUE	CPU	SCREEN.	KEYBOARD.
PARTIAL_WAKE_LOCK	ON	OFF	OFF
SCREEN_DIM_WAKE_LOCK	ON	DIM	OFF
SCREEN_BRIGHT_WAKE_LOCK	ON	BRIGHT	OFF
FULL_WAKE_LOCK	ON	BRIGHT	BRIGHT

en sus versiones más actuales, mejor calificados en un estudio realizado en noviembre del 2015 [18]. La Tabla II muestra las características de las aplicaciones antivirus seleccionadas.

TABLA II
 CARACTERÍSTICAS DE LAS APLICACIONES ANTIVIRUS.

Función	Avast	Eset	Kaspersky	McAfee	Norton
Detección de Malware	si	si	si	si	si
Servicio Anti Robo	si	si	si	si	si
Bloqueo de Llamadas	si	si	si	si	si
Filtrar Mensajes	si	si	si	no	no
Navegación Segura	si	si	si	si	si
Control Parental	no	no	no	no	no
Copia de Seguridad	no	no	no	si	si
Codificación de Datos	no	no	no	no	si

C. Servicio en Background

Las medidas del consumo energético del dispositivo móvil se han tomado por software, aprovechando las capacidades que brinda el sistema operativo Android, mediante una aplicación (app) desarrollada para monitorear la demanda energética. La aplicación registra en todo momento el porcentaje restante de batería, la capacidad y el voltaje, para posteriormente calcular la potencia consumida. La Tabla III muestra la estructura de estos registros.

TABLA III
 ESTRUCTURA DE LOS REGISTROS.

N	Fecha	% RESTANTE DE LA BATERIA	CAPACIDAD (mAh)	VOLTAJE (V)
1	13.39.06	100%	2600	4.375
2	13.51.16	99%	2574	4.242
3	13.51.26	98%	2548	4.214

El código fuente de la aplicación esta compuesto de dos clases Java:

- monitor.java.- Aquí se define la actividad (activity) que inicia o finaliza el servicio.
- service.java.- Define el servicio y las operaciones a realizar. Cuando el servicio se carga por primera vez, ocurre el evento ACTION_BATTERY_CHANGED, que recibe el voltaje, la capacidad y el porcentaje restante de energía en la batería.

Para obtener el valor numérico del estado de la batería se ha utilizado la clase BatteryManager [19], que permite obtener datos de la batería, como la capacidad en mAh, el voltaje, la tecnología, la temperatura, si está cargándose, entre otros. Como se mencionó, la aplicación desarrollada registra la capacidad, el voltaje y el porcentaje restante de la batería. La capacidad es registrada en la variable

EXTRA_LEVEL y el voltaje en la variable EXTRA_VOLTAGE, estas dos variables se emplean para el cálculo de la potencia consumida.

Cuando se ejecuta la aplicación por primera vez, se inicia un servicio y en adelante no se necesita de interfaz de usuario, así, mientras no se produzca variación en la descarga de la batería, no habrá consumo de CPU u otro recurso del dispositivo.

D. Análisis de datos

En este componente se extraen los datos almacenados de la memoria interna o externa del smartphone para su análisis estadístico en la herramienta gráfica R [20], generando gráficos de: (i) caja y bigotes para mostrar el voltaje, (ii) de barras, para mostrar la energía restante de la batería, y (iii) de líneas para comparar la descarga de la batería tanto con la ejecución de las distintas aplicaciones antivirus sin escanear el dispositivo, como con el smartphone en estado suspendido. Estos resultados se muestran en la sección V.

IV. METODOLOGÍA DE PRUEBAS

La evaluación del consumo de energía se ha realizado instalando la aplicación desarrollada en el dispositivo; siendo necesario obtener el consumo cuando el dispositivo está encendido y no ejecuta tarea o proceso alguno, de esta manera se consigue una medida de referencia para observar cuánto incrementa el consumo energético con el uso de la aplicación antivirus. Para obtener valores exactos se debe asegurar que todas las interfaces de redes de datos se encuentren inactivas y que la única aplicación ejecutándose sea la desarrollada. De esta forma se determinará el estado en donde el consumo energético es más bajo. Por otra parte, para el cálculo de la potencia consumida cuando se utilizan aplicaciones antivirus, se han definido dos estados:

- No Escaneo: Cuando las aplicaciones antivirus se encuentren instaladas y con sus servicios activos.
- Escaneo: Cuando las aplicaciones antivirus se encuentren constantemente en escaneo a los archivos y demás aplicaciones del dispositivo; se espera que en este estado se incremente la potencia consumida.

En la Figura 2 se muestra el diagrama de flujo de la aplicación de monitoreo.

El dispositivo móvil empleado para las diferentes evaluaciones fue un Samsung Galaxy J5 con Android 5.1.1 Lollipop, operando sobre la versión 3.10.49-787809 del kernel de Linux. Se instaló la aplicación desarrollada, y se hicieron tres muestras de cinco horas consecutivas para cada evaluación. Las muestras se registraron en intervalos de diez segundos. Para las aplicaciones antivirus en el caso de escaneo de las carpetas y aplicaciones del dispositivo móvil, se utilizó una memoria MicroSD de 32GB cargada con la información detallada en la tabla IV.

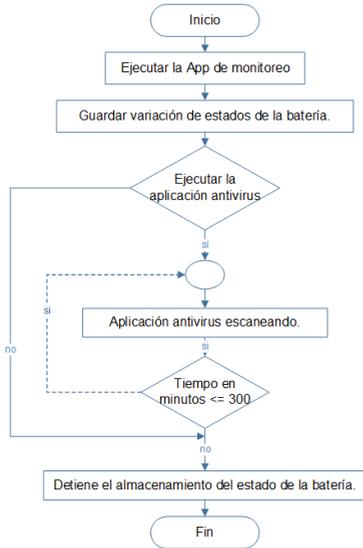


Fig. 2. Diagrama de flujo de la aplicación de monitoreo.

TABLA IV
 INFORMACIÓN CONTENIDA EN LA MEMORIA MICROSD.

Información	Tamaño (GB)
Fotografías	2 GB
Vídeos	5 GB
Documentos	15 GB
Instaladores Varios	7 GB
Música	3 GB
Total	32 GB

V. RESULTADOS

A continuación se describen los resultados según la propuesta y metodología descrita en las secciones III, IV respectivamente. Se evaluaron los estados activo, inactivo y suspendido. Debido a que la batería no es un dispositivo exactamente lineal [6], no siempre se obtienen valores exactos de potencia, por tanto para determinar el consumo medio de una aplicación antivirus se obtienen varios valores de potencia y se calcula el promedio de ellos.

A. Estudio de descarga de la batería en estado suspendido

En las pruebas realizadas, la batería del dispositivo inicialmente se encuentra cargada al 100%, y estando en esta condición, es donde se inicia la aplicación de monitorización, y por ende el registro de todas las variaciones que presente la batería hasta

cumplir las cinco horas. Se evaluó la descarga de la batería cuando el dispositivo se encuentra sin aplicaciones antivirus instaladas. Esta medida sirve como referencia para verificar en cuanto incrementó el consumo por cada aplicación antivirus. En la Figura 3 se muestra la descarga del dispositivo en estado suspendido (línea de color negro). Se puede observar que el tiempo de duración de la batería es prolongado, pues sin el uso e instalación de aplicaciones antivirus, llega a las cinco horas con una capacidad restante de alrededor del 69% de la batería.

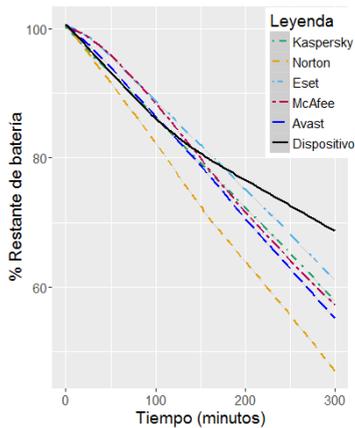


Fig. 3. Descarga de la batería al tener las aplicaciones antivirus instaladas y sin escanear el dispositivo, y con el smartphone en estado suspendido.

B. Aplicaciones antivirus en estado suspendido y no escaneo

Luego de tener el comportamiento inicial de descarga sin antivirus instalado, llegando a una carga restante del 69%, se procedió a realizar la evaluación con las aplicaciones antivirus instaladas, ejecutándose pero no escaneando archivos. En la Figura 3 se muestran los resultados de estas pruebas y refleja que los antivirus reducen rápidamente la duración de la batería del dispositivo móvil, siendo Norton la aplicación que más consume energía dejando a la batería en un 44% restante de su capacidad y Eset, la de menor consumo de energía en este estado.

C. Aplicaciones antivirus en estado activo y escaneo

Para verificar el tiempo de descarga en la batería del dispositivo con estado activo se procedió a evaluar las aplicaciones antivirus cuando se realiza el escaneo completo de las diferentes carpetas cuyo contenido se evidencia en la Tabla IV, en busca de algún tipo de virus o malware.

En la Figura 4 se observa que la aplicación antivirus de mayor consumo de energía fue McAfee, dejando la batería con un nivel restante de capacidad del 96%, en tanto que la aplicación antivirus de menor consumo fue Avast. Además, en la Figura 5 se evidencia la variación de voltaje para los diferentes antivirus, registrando rangos significativos entre Norton y Kaspersky. La Tabla V muestra la capacidad restante de la batería en función del tiempo de escaneo de cada antivirus y en la Figura 6 se muestran las variaciones de voltaje del dispositivo móvil en los estados inactivos y suspendido asociados a las aplicaciones.

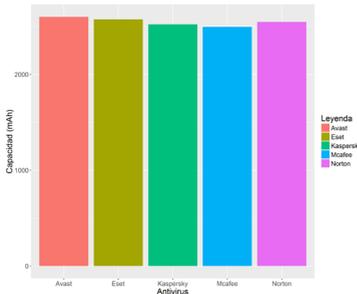


Fig. 4. Energía restante después del escaneo.

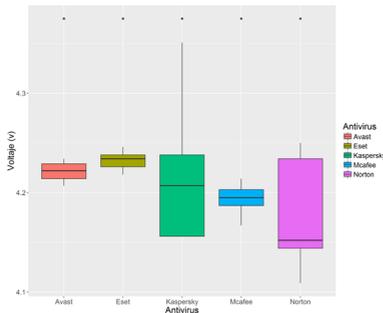


Fig. 5. Análisis de consumo energético de Apps Antivirus.

TABLA V
 RESUMEN APLICACIÓN ANTIVIRUS EN ESTADO ACTIVO.

Tiempo de Escaneo(seg)	Antivirus	% Restante de la Batería
1080 seg	McAfee	96%
1680 seg	Kaspersky	97%
450 seg	Eset	98%
120 seg	Norton	99%
300 seg	Avast	100%

D. Medida de potencia

Para evaluar el consumo del dispositivo móvil se va a registrar la potencia consumida leyendo el voltaje y la capacidad, y con esos datos calcular la potencia consumida. En las baterías de Li-ion de los dispositivos móviles, el voltaje cambia durante la descarga, permitiendo entonces considerar la potencia consumida. A medida que varía el voltaje disminuye el nivel de capacidad de la batería. Es decir que la potencia consumida se puede calcular a partir de los cambios de voltaje y el nivel de capacidad de la batería. La fórmula para calcular la potencia consumida durante un intervalo de tiempo determinado es la siguiente:

$$P = \frac{C_1 * V_1 - C_2 * V_2}{t_1 - t_2} [6] \quad (1)$$

Donde, P hace referencia a la potencia consumida, el intervalo de tiempo con las variables $(t_1 - t_2)$, los valores del nivel de capacidad restante se representan con C_1 y C_2 expresada en mAh y con referencia al valor del voltaje con V_1 y V_2 .

Se debe tener en cuenta la capacidad real disponible del dispositivo móvil debido al deterioro y con el transcurso del tiempo esta puede disminuir. Se puede verificar la capacidad con la que cuenta la batería en la variable full.bat la podemos encontrar en el directorio /sys/class/power_supply/battery, puede variar dependiendo del dispositivo móvil.

A continuación en la Tabla VI se muestran los valores de potencia consumida en cada uno de los estados explicados anteriormente y por cada una de las aplicaciones antivirus evaluadas.

TABLA VI
 POTENCIA CONSUMIDA POR LAS APLICACIONES ANTIVIRUS.

	Antivirus (mW)				
	Avast	Eset	Kaspersky	Norton	McAfee
Activo	6.45	8.33	14.94	11.45	16.33
Suspendido	15.13	11.74	14.19	18.10	13.44
Inactivo	1.01	1.20	1.08	2.07	1.30

VI. CONCLUSIÓN

En este artículo se propuso una metodología que permite monitorizar el consumo energético de dispositivos móviles con sistemas operativos Android, asociados a las aplicaciones antivirus. La aplicación desarrollada permite registrar los niveles de consumo energético producido por los antivirus estudiados. En las pruebas se comprobó que, en dos de los tres estados, el antivirus Norton es el que mayor consumo de batería demostró. Por otro lado, los antivirus que presentaron un menor consumo de energía fueron Eset y Avast. Los resultados demuestran que existe una disminución significativa de energía, en base a esto se puede deducir que los proveedores de seguridad no desarrollan aplicaciones antivirus con perfiles de consumo de energía moderada. Cabe destacar que, en esta evaluación, no se midió las prestaciones de dichos antivirus. Como trabajo futuro se

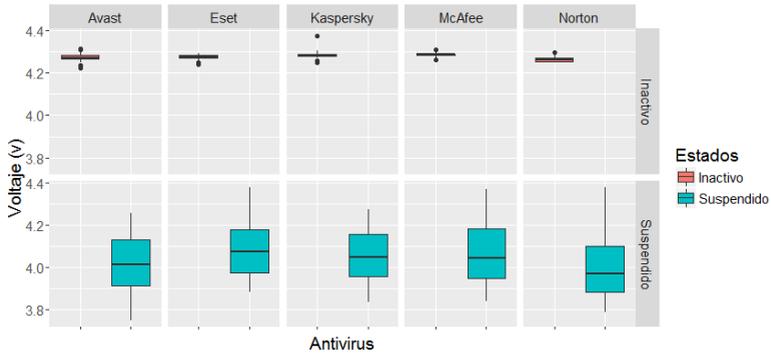


Fig. 6. Variaciones de Voltaje del dispositivo.

pretende mejorar la aplicación de recolección de datos, así como evaluar otros parámetros con respecto a las bondades de los antivirus mediante dispositivos de distintas marcas y modelos.

AGRADECIMIENTOS

Este trabajo fue parcialmente apoyado por la Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación del Ecuador (SENESCYT) mediante sus programas de becas y la Universidad Laica Eloy Alfaro de Manabí (ULEAM).

REFERENCIAS

- [1] G. P. Perrucci, F. H. Fitzek, G. Sasso, W. Kellerer, and J. Widmer, "On the impact of 2g and 3g network usage for mobile phones battery life," in *2009 European Wireless Conference*. IEEE, may 2009, pp. 255–259.
- [2] M. Martins, J. Cappos, R. Fonseca, M. Martins, J. Cappos, and R. Fonseca, "Selectively Taming Background Android Apps to Improve Battery Lifetime," 2015.
- [3] "Optimizing for Doze and App Standby." [Online]. Available: <http://developer.android.com/training/monitoring-device-state/doze-standby.html>.
- [4] H. Pieterse and M. S. Olivier, "Security steps for smartphone users," pp. 1–6, aug 2013.
- [5] E. Manet, L. Anne, F. Jean-yves, M. Serge, and F. A., "Energy Consumption Models for Ad-Hoc Mobile Terminals," 2003.
- [6] "Estudio del consumo de energía," in *Universidad Carlos III de Madrid*, 2012, pp. 54–55.
- [7] I. Bytheway, D. J. Grimwood, and D. Jayatilaka, "W-functions derived from experiment. III. Topological analysis of crystal fragments," *Acta crystallographica. Section A, Foundations of crystallography*, vol. 58, no. Pt 3, pp. 232–43, may 2002.
- [8] M. Y. Malik, "Power Consumption Analysis of a Modern Smartphone," p. 11, dec 2012.
- [9] A. Carroll and G. Heiser, "An Analysis of Power Consumption in a Smartphone," *USENIXATC'10 Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, pp. 21–21, 2010.
- [10] S. K. Datta, C. Bonnet, and N. Nikaiein, "Minimizing energy expenditure in smart devices," no. lct, pp. 712–717, 2013.
- [11] N. Vallina-Rodriguez, P. Hui, J. Crowcroft, and A. Rice, "Exhausting battery statistics," no. February, p. 9, 2010.
- [12] A. Anand, C. Manikopoulos, Q. Jones, and C. Borcea, "A Quantitative Analysis of Power Consumption

- for Location-Aware Applications on Smart Phones," pp. 1986–1991, jun 2007.
- [13] A. Shye, B. Scholbrock, and G. Menik, "Into the wild," p. 168, 2009.
- [14] J. Bickford, F. Park, A. Varshavsky, and F. Park, "Security versus Energy Tradeoffs in Host-Based Mobile Malware Detection," 2011.
- [15] I. Polakis, M. Diamantaris, and T. Petsas, "Powerslave : Analyzing the Energy Consumption of Mobile Antivirus Software," vol. 3, pp. 165–184, 2015.
- [16] "Download Android Studio and SDK Tools." [Online]. Available: <http://developer.android.com>
- [17] S. K. Datta, C. Bonnet, and N. Nikaiein, "Android power management: Current and future trends," in *2012 The First IEEE Workshop on Enabling Technologies for Smartphone and Internet of Things (ETSIoT)*, no. December 2015. IEEE, jun 2012, pp. 48–53.
- [18] "AV-TEST - The Independent IT-Security Institute," feb 2015. [Online]. Available: <https://www.av-test.org/es/antivirus/moviles/android/noviembre-2015>
- [19] "Android Developer BatteryManager." [Online]. Available: <http://developer.android.com/intl/reference/android/os/BatteryManager.html>.
- [20] R-Foundation, "R project available," 2015. [Online]. Available: <https://www.r-project.org>

SmartWaterSigfox: IoT con Sigfox para la gestión de contadores de agua

J. Jiménez-Ortega¹, M. Damas² y F. Gómez²

Resumen— El auge de Internet de las Cosas (IoT en adelante) ha provocado que la investigación e inversión en torno a este campo crezca considerablemente en los últimos años. Este artículo presenta una aplicación para el control y la gestión de contadores de agua usando la red Sigfox. Esto permite disponer de los datos en la nube para cada nodo sin necesidad de realizar conexiones GPRS o Wifi. A su vez, se reduce el consumo y permite añadir o quitar nodos, además de disminuir los costes de mantenimiento así como los de infraestructuras debido a que el diseño en torno a Sigfox sólo requiere la implementación de las motas y la interfaz de visualización de los datos. En general, estos dispositivos se pueden colocar en lugares remotos; por tanto, la durabilidad de las baterías es uno de los parámetros más importantes a tener en cuenta y, tal y como se detallará en los siguientes apartados, el compromiso conseguido entre la durabilidad y el periodo de envío de datos es bastante bueno.

Palabras clave— IoT, Sigfox, gestión de contadores de agua, bajo consumo.

I. INTRODUCCIÓN

EN el ámbito de la agricultura en general, como en otros sectores (ciudades inteligentes, seguridad, atención sanitaria, etc.), la tendencia es disponer de los datos en cualquier lugar y en cualquier momento. No hay más que ver la proliferación de los servicios *cloud* en los últimos años y que se ponga tanto empeño en la interconexión, haciendo hincapié en nuevos protocolos como MQTT y en nuevas redes de telecomunicaciones como LWPAN o 6LoWPAN que facilitan el desarrollo de nuevas aplicaciones con esta filosofía y metodología de diseño.

Con todo esto, no es de extrañar que a día de hoy *IoT* esté en lo alto de la cresta del *Ciclo de Sobre Expectación* de Gartner [1] (curva estadística que representa la viabilidad y el estado de una idea, producto o tecnología; ver Figura 1).

La gestión de suministro de agua en lo que se refiere a la lectura de contadores se está adaptando a las nuevas tecnologías progresivamente. En grandes comunidades de regantes se usan redes de sensores basadas en enlaces de radio [5] en las cuales se necesitan servidores de datos con conexión a internet para acceder a ellos. En el caso de aplicaciones para hogar, se propone el uso de *Wifi* para la conectividad de sensores [6]. Hasta ahora, existen pocas propuestas que puedan ser modulares, escalables y que ofrezcan ventajas tanto para el uso personal como para el industrial.

Además, en los últimos años los esfuerzos por diseñar nuevos tipos de redes orientadas a la comuni-



Fig. 1. Ciclo de Sobre Expectación 2015. Fuente: Gartner.

cación de objetos con una tasa de envío baja [7] han aumentado considerablemente debido a la aparición de nuevos nichos de negocio como las *Smart Cities*. Aquí, este tipo de redes ofrecen ventajas para envíos de información esporádicos [2], frente a la alternativa de usar redes mixtas para las redes de sensores (por ejemplo, el uso de Zigbee para conectar los nodos con una centralita y GPRS en la misma para el envío de la información [3], o el uso de UHF y VHF junto con GPRS [4]), donde el coste del sistema es elevado debido al uso de puertas de enlace que tienen que disponer de distintas tecnologías inalámbricas para la comunicación con los puntos finales y con el servidor donde se almacena la información, además de que la complejidad de la red aumenta considerablemente.

Es por todo ello que se plantea este trabajo en el se presta especial atención al diseño de un sistema versátil y equiparable en durabilidad a los sistemas convencionales reduciendo costes de infraestructuras e incluso en los dispositivos, utilizando una de estas nuevas redes para IoT, como es Sigfox.

En los siguientes apartados se justifican las ventajas de la red Sigfox frente a otras tecnologías, se realiza una descripción del diseño hardware y firmware del sistema y, por último, se describe cómo se accede a los datos de la plataforma para su visualización. Al final se muestran los resultados experimentales que argumentan el bajo consumo del sistema y se exponen las conclusiones del trabajo.

II. SELECCIÓN DE RED DE TELECOMUNICACIONES

Para el diseño del sistema que se propone, el primer punto a tener en cuenta son las telecomunicaciones; es decir, ¿cómo se van en enviar los datos a un servidor para su posterior procesamiento?. Ya que se pretende implementar una solución que abarque desde grandes zonas de regadío hasta pequeños

¹Universidad de Granada, e-mail: jaimenj@correo.ugr.es.

²Dpto. de Arquitectura y Tecnología de Computadores, Univ. Granada, e-mail: mdamas@ugr.es y frgomez@ugr.es

jardines particulares, las soluciones basadas en Wifi o Bluetooth quedan descartadas por su corto alcance. Por otro lado, un sistema basado en radio es ineficaz si no hay que transmitir a largas distancias además de necesitar algún tipo de puerta de enlace conectada a Internet en ambos casos para subir los datos a la nube. Otra posibilidad es el uso de GPRS, pero a causa del coste y a los requerimientos en cuanto a batería esta opción es inviable.

Centrando el foco en las tecnologías que se han desarrollado específicamente para Internet de las Cosas (Sigfox, LoraWAN, Neul, LTE-M...), dos de ellas destacan por su uso, despliegue y disponibilidad en España: LoraWAN y Sigfox.

- LoraWAN: es una red LPWAN (*Low Power Wide Area Network*) que usa las bandas ISM 433, 868 o 915 Mhz. La topología está formada por una red en estrella jerarquizada, que alberga 3 tipos de dispositivos: las motas o puntos finales, los concentradores o *gateways* y el servidor. Las motas se dividen en tres tipos (clase A, B y C) dependiendo de la latencia de descarga de datos: si se produce después de una transmisión, en periodos determinados de tiempo o si la mota está continuamente a la escucha. Esta característica está fuertemente ligada al consumo de energía necesario para cada tipo de nodo. Las tasas de transferencias de datos van desde los 0.3Kbps hasta 50Kbps.
- Sigfox: es una variación de la red que se usa en telefonía móvil, por lo que hereda sus ventajas (largo alcance, ubicuidad, facilidad de configuración respecto a redes de bajo alcance) pero añadiendo características necesarias para su uso en el marco de Internet de las Cosas como son el bajo consumo y el bajo coste. Esta empresa de origen Francés realiza sus transmisiones en UNB (*Ultra-Narrow Band*), con un ancho de banda de 100Hz. En Europa se utiliza la banda ISM de 868 MHz y la de 900 MHz en EEUU. Se pueden enviar hasta 140 mensajes al día por mota con un máximo de 12 bytes por mensaje; por tanto, no está indicada para aplicaciones en tiempo real. Además, Sigfox cuenta con un sistema web de administración (*o backend*) que permite configurar acciones del sistema al recibir datos, visualizar el estado de los dispositivos, el acceso a su sistema a través de una API, etc.

Para este caso de estudio se ha optado por usar Sigfox, ya que en LoraWan el usuario debe de encargarse de toda la estructura de la red (nodos y concentradores), y a que no se necesitan grandes tasas de envío de datos.

III. DISEÑO HARDWARE

En cuanto al diseño electrónico del sistema se refiere, se puede dividir en tres partes principales: el módulo de radio para conectarse a la red Sigfox, el microprocesador y la batería. El esquema se puede ver en la Figura 2.

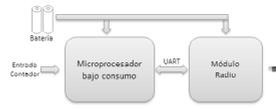


Fig. 2. Esquema que representa el diseño hardware.

El microprocesador actúa como “cerebro” del dispositivo, realizando las lecturas del contador de agua y decidiendo cuando se envían las tramas correspondientes al servidor a través del módulo de radio. Una de las principales características a la hora de seleccionarlo ha sido que tenga el menor consumo posible. En este aspecto, la gama XLP™ de Microchip, en concreto la familia **PIC18F46J50**, cumple perfectamente las expectativas. Poseen un modo de funcionamiento que denominan “*Deep Sleep*” en el que el consumo está en torno a los 15nA, además de un rango amplio de voltaje de alimentación (2.0V-3.6V) lo cual relaja el requerimiento para la selección de la batería. En la siguiente sección se detalla más el funcionamiento del microprocesador en este modo y el funcionamiento del programa.

Los contadores de agua con emisor de pulsos poseen un contacto libre de tensión que se cierra al paso de determinados litros de agua. Existen versiones de 1, 10, 100 y 1000 l/impulso. El esquema se puede ver en la Figura 3. Este contacto se conecta por una parte a la batería del dispositivo y por otra a una entrada digital del microprocesador.

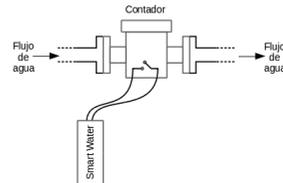


Fig. 3. Esquema del funcionamiento de un contador con salida de impulsos.

Esta entrada tiene la capacidad de sacar al microprocesador del modo de bajo consumo para así poder contabilizar los pulsos del contador.

Para la parte de radio, se ha elegido el módulo de *Telecom Design* compatible con Sigfox **TD1207**. Está indicado específicamente para uso en soluciones basadas en batería. Sus principales características son [10]:

- Sensibilidad de recepción de -126dBm
- Bajo consumo en modo ahorro de energía: 3.5µA
- Potencia de salida de +14dBm
- Amplio rango de voltaje de alimentación: 2.3-3.3V

- Tamaño: 25.4 x 12.7 x 3.81mm

Incluye un ARM Cortex M3 de 32 bits que, junto a un controlador DMA, permiten implementar protocolos seguros y altamente complejos de manera eficiente. Este procesador incluye técnicas innovadoras para sistemas de baja potencia, tiempo rápido de respuesta desde los modos de bajo consumo y una amplia selección de periféricos, permitiendo su uso en una gran variedad de aplicaciones.

Además de usarse como nodo de red de Sigfox, tiene la capacidad de crear o conectarse a una red local de banda estrecha y enviar mensajes entre distintos dispositivos al mismo tiempo que se usa para enviar y recibir datos por la red UNB de Sigfox, aunque este modo de operación no se usa en este trabajo.

El módulo se usa como dispositivo esclavo del microprocesador. Las comunicaciones son de tipo serie a través de UART (*Universal Asynchronous Receiver/Transmitter*) de bajo consumo y se utilizan comandos Hayes o AT [9] que se descodifican mediante un intérprete de comandos integrado en el módulo. Esta interfaz usa solo unos pocos μA durante la comunicación y 150nA mientras se espera a la recepción de los datos.

En lo que se refiere a la selección de la batería para el dispositivo, se propone el uso de baterías alcalinas AA. Dos de ellas en serie proporcionan 3.0V a plena carga. En concreto, el modelo de Duracell MN1500 de la serie *Plus Power*, con una capacidad de hasta 2850mAh y una durabilidad de más de 7 años son perfectas para el uso en este dispositivo.

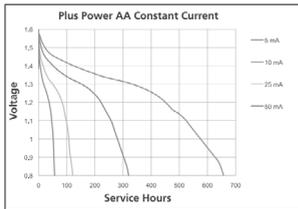


Fig. 4. Descarga de una batería Duracell MN1500 a corriente constante. Fuente: Duracell

Este tipo de baterías tienen una curva de descarga característica (ver Figura 4, donde los valores representados en el eje Y corresponden a una sola pila). Tal y como se observa, la tensión va disminuyendo cuando se va descargando. Debido a que en el diseño el dispositivo que tiene menor rango de alimentación es el módulo de radio (2.3V), por debajo de este valor no se puede enviar información al servidor, por lo que se pierde la funcionalidad del equipo. Es decir, que cada batería no debe descargarse por debajo de 1.15V para que el sistema funcione, que corresponde aproximadamente al nivel que llega la pila cuando ha suministrado el 75% de su capacidad. Este dato se usará de nuevo en el apartado VI cuando se realice el

estudio de la duración de la batería del dispositivo.

Además, por este motivo es por lo que se monitoriza el estado de la batería antes de cada envío de datos al servidor. Este proceso se volverá a discutir en el siguiente apartado.

En la figura 5 se puede ver el resultado final del prototipo¹.

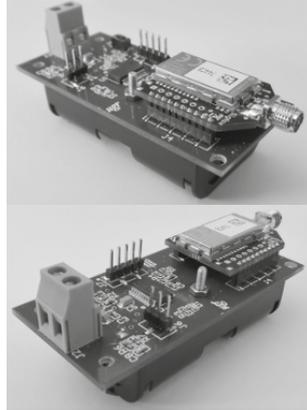


Fig. 5. Prototipo del diseño

Los costes por mota quedan resumidos en la tabla I. Tal y como se puede ver, el precio por contador es de aproximadamente 25€. A esto, habría que sumar los costes de la licencia de Sigfox que varían según el número de motas que se necesitan.

TABLA I
 COSTES POR MOTA PARA UN VOLUMEN DE 100 UNIDADES.

Item	Precio (€)
Módulo de radio	13
Microcontrolador	2
Conector antena	1.2
Antena	4
Circuito impreso	0.6
Componentes varios	4
Total	24.80

IV. DISEÑO FIRMWARE

El diseño del programa del microcontrolador gira en torno a que el sistema esté en modo de ultra bajo consumo el mayor tiempo posible. Para ello se hace uso de una característica especial de esta familia llamada *Deep Sleep*. En este modo se desconecta la alimentación del núcleo del procesador, por

¹Por motivos de disponibilidad, se ha usado el adaptador de Libelium que no es más que una pequeña placa que contiene el módulo de radio TD1207 y el conector de antena.

lo que la mayoría de los periféricos y funcionalidades del microcontrolador están deshabilitadas en este estado. Solo determinados eventos especiales pueden devolver al dispositivo a su modo de funcionamiento normal, y lo hacen provocando un reset.

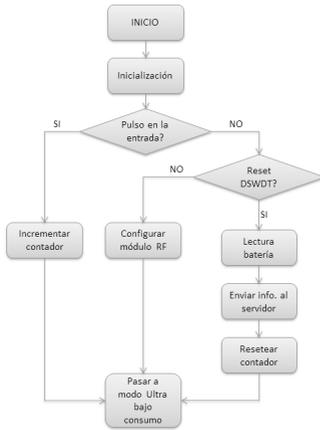


Fig. 6. Diagrama de flujo del programa del microcontrolador

En este trabajo se usan dos formas distintas de despertar al sistema: a través de la interrupción externa 0, que está conectada a la salida del contador de agua; y a través del temporizador de vigilancia específico para bajo consumo (o *Deep Sleep Watchdog Timer*, *DSWDT*), que es el encargado de señalar cuándo se ha de realizar un envío de datos al servidor.

Pero, ya que cualquiera de estos dos eventos reinician al microcontrolador, ¿cómo se puede almacenar el número de pulsos del contador entre dos envíos?. Para ello, se disponen de dos registros de 8bits que mantienen su valor mientras que se mantenga la alimentación en el dispositivo. Por tanto, entre dos envíos, se puede almacenar como máximo $2^{16} - 1 = 65535$ pulsos. Para cada instalación, será importante dimensionar correctamente el contador para que nunca se supere este número y se desborde el contador (por ejemplo, en una instalación que se consuman 10l al minuto realizando envíos cada 8 horas se ha de colocar un contador de 10l/pulso como mínimo, ya que uno de 1l/pulso superaría el valor máximo antes de realizar un envío).

Tal y como se discutía en el apartado anterior, es necesario conocer el estado de la batería para así no interrumpir el servicio de lectura del contador en ningún momento. Para esto, se usa un módulo que incluye el microcontrolador denominado *High/Low-Voltage Detect (HLVD)*. Este módulo monitoriza la alimentación del microcontrolador y la compara con

una referencia interna de 1.2V. Dependiendo de cómo se configure, puede detectar que el voltaje esta por encima o por debajo de un valor definido. Dicho voltaje de disparo se selecciona a través de un multiplexor y puede ser suministrado externamente a través de un pin de entrada (HLVDIN) o seleccionado de unos valores predefinidos que en este caso son 2.45V, 2.60V, 2.80V, 2.90V, 3.00V, 3.13V y 3.40V ([8], página 504). Debido a que el sistema deja de funcionar a partir de 2.3V, se selecciona el valor de 2.45V para que el dispositivo informe de la necesidad del cambio de batería.

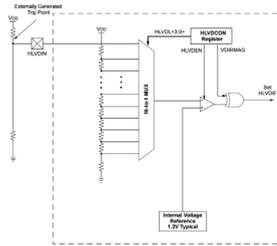


Fig. 7. Módulo HLVD del microcontrolador

En la Figura 6 se describe el flujo del programa. Como se observa, existen tres arranques posibles. El módulo de radio solo se configura la primera vez que se alimenta el dispositivo ya que no se reinicia en ningún momento.

En cuanto al envío de datos, Sigfox permite realizar envíos con un máximo de 12bytes. En nuestro sistema solo se usan 3bytes: dos para el número de pulsos y un tercero que indicará si el dispositivo tiene la batería baja (1) o si la batería está a nivel correcto (0).

En lo que se refiere a la temporización del *DSWDT*, se puede configurar para que el sistema se reinicie en los siguientes periodos: 2.1ms, 8.3ms, 33ms, 132ms, 528ms, 2.1s, 8.5s, 34s, 135s, 9m, 36m, 2.4h, 9.6h, 38.5h, 6.4d y 25.7d. En nuestra aplicación se configura para un valor de 2.4 horas para extender lo más posible la duración de la batería, tal y como se discutirá en el apartado VI.

V. DISEÑO DE LA INTERFAZ WEB DE CONTROL

Así como se mencionaba en el apartado I, Sigfox posee un *backend* [11] que permite la gestión de dispositivo, visualización de las tramas que se han enviado (en modo texto, no en modo gráfico), creaciones de grupos de dispositivos, etc. Además, incorpora la gestión de eventos de los dispositivos a través de funciones *callbacks*, que reenvían dichos eventos a través de uno de los siguientes medios: EMAIL, HTTP o mediante el acuse de recibo cuando se envía una trama. En este trabajo se usa HTTP.

Para el usuario final de la aplicación, se diseña una interfaz web para la visualización de los datos y la

gestión de los dispositivos, en la que se puede configurar, por ejemplo, el número de litros por uso de cada dispositivo. Se diseña también una base de datos para almacenar tanto la información del contador como de las motas. Para obtener el valor de cada medida, se programa una función *callback* en la interfaz de Sigfox que la reenvía. La propia web a través de un *driver* escrito en PHP se encarga de recoger el dato e introducirlo en la base de datos.

En la figura 8 se puede ver el esquema del funcionamiento del sistema completo. De nuevo, cabe destacar que Sigfox ya tiene implementada la infraestructura de puertas de enlace y su *backend*, facilitando así el diseño a nivel de ingeniería.

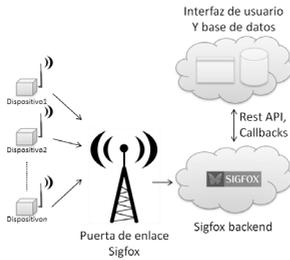


Fig. 8. Esquema de la implementación de todo el sistema

VI. PRUEBAS EXPERIMENTALES

En este apartado se van a discutir a fondo las pruebas realizadas con el prototipo en cuanto a consumo eléctrico y, por tanto, de durabilidad de la batería.

Para este cálculo del consumo medio del dispositivo, se utiliza la siguiente ecuación:

$$I_{med} = \frac{I_1 T_1 + I_2 T_2}{T_1 + T_2} \quad (1)$$

donde:

- I_1 es la corriente consumida durante el envío de datos
- T_1 es el tiempo que dura el envío de datos
- I_2 es la corriente consumida durante el modo de ultra bajo consumo
- T_2 es el tiempo que el dispositivo está en modo ultra bajo consumo

Hay dos formas de realizar envíos: con y sin acuse de recibo. En el caso de no usar acuse de recibo, el módulo envía varias veces la trama en distintas frecuencias para asegurarse su recepción por parte de la puerta de enlace, reduciendo la probabilidad de colisión. Además, todos los mensajes son enviados en *broadcast*, por lo que varias estaciones pueden recibir el mismo mensaje, reduciendo aún más el riesgo de colisión. El número de veces que se envía es configurable por software. Si se selecciona que el

módulo realice el envío con acuse de recibo, el proceso que sigue el módulo es el siguiente: se envía el paquete a la red; acto seguido, el módulo entra en estado de bajo consumo para esperar a la recepción de datos. Este proceso dura aproximadamente 40 segundos frente a los 5,4 segundos que dura el envío sin acuse de recibo, por lo que dependiendo de la aplicación, se deberá elegir entre uno u otro ya que la duración de la batería se reduce a la mitad, como se describirá a continuación.

Con un osciloscopio provisto de una sonda de corriente se han realizado las medidas para determinar empíricamente el tiempo de envío de tramas y el consumo del sistema en ese periodo. El resultado se muestra en la figura 9.

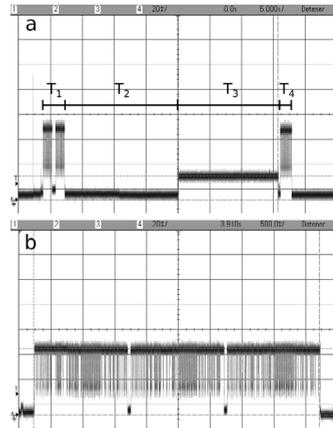


Fig. 9. Medidas de corriente del sistema durante un envío de datos: a) con acuse de recibo, b) sin acuse de recibo. Notar que la escala horizontal es distinta para cada caso: 5s/división en a) y 500ms/división en b)

Tal y como se puede observar en cada caso, el envío sigue los procesos descritos en párrafos anteriores. Para usar la ecuación 1, se calcula el consumo medio durante toda la fase de envío. A continuación se estudia cada uno de los dos casos para obtener la corriente media por envío.

En la figura 9 a) se ven claramente las cuatro partes en las que se divide el proceso de envío con acuse de recibo. La duración y el consumo de corriente de cada una de ellas es:

- Envío de la trama: $T_1 = 3.5s$. $I_1 = 55mA$
- Estado de bajo consumo: $T_2 = 17.5s$. $I_2 = 2.6\mu A$
- Recepción del acuse de recibo: $T_3 = 15.5s$. $I_3 = 18.75mA$
- Confirmación de la recepción: $T_4 = 1.7s$. $I_4 = 55mA$

Por tanto se calcula el consumo medio para envíos

con acuse de recibo:

$$I = \frac{\sum I_i T_i}{\sum T_i} = \frac{576.67mA/s}{38.2s} \approx 15.1mA \quad (2)$$

En el caso de envío sin acuse de recibo, el proceso dura aproximadamente 5.44 segundos, con un consumo máximo de unos 55mA. En la imagen 9 b) se pueden observar claramente como se producen tres envíos consecutivos (cada uno en una frecuencia distinta), con un espaciado entre cada uno de unos 50ms.

Durante el modo de ultra bajo consumo, el sistema completo consume 2.6µA. Por tanto, de la ecuación 1 solo queda por definir el parámetro T_2 , que viene determinado por los valores que se pueden configurar para el periodo del *Deep Sleep Watch Dog Timer*, descritos en el apartado IV. Con la red Sigfox se pueden enviar hasta 140 mensajes por día, o lo que es lo mismo, un mensaje cada 10.25 minutos. Los dos valores para el temporizador más cercanos son un envío cada 9 minutos (que queda descartado ya que se necesitarían 160 envíos al día) o cada 36 minutos. Por tanto, éste último valor es el mínimo periodo de envío que se puede utilizar. Teniendo en cuenta que la capacidad de la batería es 2850mAh, y que a partir de haber consumido el 75% de la carga el sistema no funciona (tal y como se ha explicado en el apartado III), se realizan los cálculos para la duración de la batería para varios valores de temporizador. Los resultados del cálculo con y sin acuse de recibo se muestran en la tabla II.

TABLA II
 RESULTADOS DEL CÁLCULO DE LA DURACIÓN DE LA BATERÍA DEL DISPOSITIVO.

Acuse de recibo	T_2	Duración bat.
No	36m	1.7 años
No	2.4h	6.6 años
No	9.6h	21.7 años
Si	36m	11 meses
Si	2.4h	3.5 años
Si	9.6h	12.7 años

En realidad, los valores cuya duración es mayor a los 10 años no se pueden considerar como reales, ya que el fabricante de las pilas no garantiza su correcto funcionamiento pasado este tiempo.

Por tanto, se considera que el periodo que representa un compromiso entre la representación de los datos en el tiempo y la durabilidad de la batería es el de 2.4h, que proporciona 10 medidas al día sin acuse de recibo, aunque este parámetro es fácilmente modificable por software.

Tal y como se comentaba en el apartado IV, cabe destacar que dependiendo de la instalación se ha de colocar un contador que no proporcione más de 65535 pulsos cada 2.4h para no desbordar el registro del que se dispone en el microprocesador.

VII. CONCLUSIONES

El foco de este artículo pretende ser el uso de redes específicas para Internet de las Cosas en el uso de la lectura de contadores de agua, implementando un sistema completo eficiente, de bajo coste y con bajo consumo. Tal y como se ha descrito en apartados anteriores, utilizando nuevas gamas de microprocesadores que están orientados a su uso con sistemas autónomos de alimentación y transceptores de radio que implementan la capa física de las comunicaciones con la red de un modo eficaz, se consiguen los objetivos de diseñar un sistema que sirve tanto para el ámbito agrícola como para el personal, debido a que su mantenimiento (cambio de batería) no requiere, como en otros casos, la intervención de un operador ni la sustitución del dispositivo, gracias al uso de pilas comunes y de fácil acceso para su sustitución.

Por otra parte, quedan demostradas las ventajas del uso de la plataforma Sigfox para el diseño de sistemas de este tipo frente a otro tipo de soluciones (por ejemplo, GPRS), ya que reducen costes de mantenimiento consiguiendo unas especificaciones similares en cuanto a alcance, ubicuidad y reduciendo drásticamente el consumo. Además, debido a que no es necesaria la implementación de las puertas de enlace entre las motas y la nube, y que dicha nube también es proporcionada por el operador, se reducen considerablemente los costes y el tiempo para realizar nuevos diseños.

REFERENCIAS

- [1] Gartner Inc., <http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp>.
- [2] Carles Gomez and Josep Paradells *Urban Automation Networks: Current and Emerging Solutions for Sensors Data Collection and Actuation in Smart Cities*. Sensors, 2015.
- [3] Luo Hongpin, Li Guanglin, Peng Weifeng, Song Jie, Bai Qiuwei *Real-time remote monitoring system for aquaculture water quality*. International Journal of Agricultural and Biological Engineering, 2015.
- [4] Parag Kulkarni and Tim Farnham *Smart City Wireless Connectivity Considerations and Cost Analysis: Lessons Learnt From Smart Water Case Studies*. IEEE Access, 2016.
- [5] Damas, M and Prados, AM and Gomez, F and Olivares, G, *HydroBus (R) system: fieldbus for integrated management of extensive areas of irrigated land*. Microprocessors and Microsystems, 2001.
- [6] Charith Perera, Arkady Zaslavsky, Peter Christen, Dimitrios Georgakopoulos *AWARE@HOME: A case study in technological design to promote environmental conservation in the american home*. Microprocessors and Microsystems, 2001.
- [7] George Margelis, Robert Piechocki, Dritan Kaleshi, Paul Thomas, *Low Throughput Networks for the IoT: Lessons Learned From Industrial Implementations*. Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on, 2015.
- [8] Microchip Technology *PIC18F46J50 Family Datasheet*.
- [9] Telecom Design *TD1207 Reference Manual*.
- [10] Telecom Design *TD1207 Datasheet*.
- [11] Sigfox Backend <https://backend.sigfox.com>.

Simulación de Tráfico Vehicular en base a Trazas Reales

Jorge L. Zambrano¹, Carlos T. Calafate¹, David Soler², Juan-Carlos Cano¹, Pietro Manzoni¹

¹Departamento de Informática de Sistemas y Computadores (DISCA)

²Intituto Universitario de Matemática Pura y Aplicada (IUMPA)

Universitat Politècnica de València, España

jorzamma@doctor.upv.es, calafate@disca.upv.es, dsoler@mat.upv.es, [jucano, pmanzoni]@disca.upv.es

Resumen—Los altos niveles de tráfico urbano son una de las principales preocupaciones en nuestras sociedades, generando problemas como consumo de combustible y emisiones de CO₂ excesivos. Recientemente, los Sistemas de Transporte Inteligente (ITS) han emergido como una tecnología capaz de mitigar estos problemas. Sin embargo, el análisis de tráfico y las correspondientes propuestas de mejora típicamente se basan en simulaciones, las cuales deberían ser lo más realistas posible. No obstante, lograr un elevado grado de realismo puede ser complejo, especialmente cuando los patrones de tráfico real, definidos a través de una matriz origen-destino (O-D) para los vehículos en una ciudad, son desconocidos. En este trabajo proponemos una nueva técnica para importar datos de tráfico a la herramienta de simulación de movilidad SUMO. Concretamente, nuestra propuesta parte de las mediciones de los anillos de inducción, típicamente disponibles por parte de las autoridades de tráfico. A continuación usando la herramienta DFROUTER conjuntamente con una heurística basada en un ajuste global del tráfico, permite generar una matriz O-D para el tráfico que se asemeje lo más posible a la distribución del tráfico real. Aplicando la técnica propuesta a la ciudad de Valencia, verificamos mediante simulación que los resultados obtenidos son muy similares a otros datos de tráfico de movilidad existentes para las ciudades de Cologne (Alemania) y Bologna (Italia), demostrando la validez de la estrategia propuesta.

Términos índice—SUMO; DFROUTER; heurística; Matriz O-D.

I. INTRODUCCIÓN

En áreas urbanas, la alta densidad poblacional genera habitualmente problemas directa o indirectamente relacionados con el tráfico, como son las emisiones de CO₂, los accidentes, ruidos y contaminación ambiental, siendo todos ellos problemas críticos para las ciudades.

En lo que respecta a soluciones para reducir la cantidad de tráfico o mejorar el flujo del tráfico en una ciudad, hay distintas alternativas que han sido propuestas, tales como el uso de transporte público o el límite en el acceso de vehículos dependiendo de su matrícula. Además, un análisis de tráfico detallado suele ser un requisito básico mejorar el flujo de tráfico. Este tipo de estudios de tráfico típicamente requieren el uso de simuladores capaces de capturar en detalle todas las particularidades y dependencias asociadas al tráfico real. Estas simulaciones dependen de diferentes factores tales como la velocidad del vehículo, la densidad vehicular, las características del entorno y el flujo del tráfico.

Para que los estudios vehiculares sean significativos, es importante contar con modelos representativos y precisos de

la carga de tráfico en el sistema (Simuladores de tráfico), los cuales usualmente requieren conocer el origen y el destino de cada trayecto (representado como una matriz O-D). Sin embargo, la obtención de estas matrices no es tarea sencilla, y muchas administraciones tratan de encontrar nuevas técnicas que permita obtener esta información.

En este trabajo proponemos un procedimiento para cuantificar el flujo del tráfico con el fin de construir modelos de movilidad realista. Nos centraremos en la ciudad de Valencia, España, importando datos de tráfico real al simulador de movilidad SUMO[1]. Concretamente, empezaremos con las medidas de los anillos de inducción, que están disponibles por parte del Ayuntamiento de Valencia (ver [2] para más detalles), y utilizaremos la herramienta DFROUTER [3], junto con una heurística basada en un ajuste global del tráfico, para generar una matriz O-D que se asemeje a la distribución real del tráfico. El resultado obtenido será comparado mediante simulación con otros datos de tráfico de movilidad existente para las ciudades de Cologne y Bologna para validar el modelo propuesto.

Este artículo está organizado de la siguiente manera: en la siguiente sección presentamos algunos trabajos relacionados con los procesos de generación de modelos de movilidad realistas. La Sección III provee detalles sobre las herramientas usadas en nuestro trabajo i.e., SUMO y DFROUTER. La sección IV describe la técnica propuesta, la cual está basada en la herramienta DFROUTER, sirviendo como un complemento que permite ajustar el número de vehículos inyectados en la red para lograr aproximarse a las trazas reales que sirven de referencia. A continuación, en la sección V, muestra los resultados obtenidos, así como un análisis comparativo con dos modelos previos generados para las ciudades de Bologna y Cologne. Finalmente, en la sección VI presentamos las principales conclusiones de este trabajo.

II. TRABAJOS RELACIONADOS

En las últimas décadas, los Sistemas de Transporte Inteligente han experimentado un excepcional crecimiento, siendo considerados como la tecnología más efectiva de mejorar el flujo de tráfico urbano.

Algunas tecnologías de posicionamiento, como el Sistema de Posicionamiento Global (GPS), junto con la creciente popularidad de teléfonos celulares, han surgido como alternativa

para monitorizar las condiciones de tráfico en tiempo real [4], proporcionando información de trayectos más realista, y permitiendo estimar la demanda de tráfico.

Diferentes métodos han sido propuestos para obtener las matrices O-D. La tecnología Bluetooth ha sido utilizada como dispositivo sensor para la medición de tráfico desde 2005. El protocolo de capa MAC que integra el estándar Bluetooth usa un identificador por cada dispositivo (dirección MAC), la cual permite determinar quien es quien en una comunicación. De esta manera, los vehículos que llevan dispositivos Bluetooth detectables pueden ser detectados por sensores Bluetooth, los cuales pueden estar instalados en múltiples localizaciones a lo largo de una ciudad. Esos sensores graban las direcciones MAC y el tiempo en que cada dirección fue detectada, permitiendo así usar dicha información para la estimación del tiempo de viaje [8]. De acuerdo a [9], los principales retos de esta tecnología son: la exactitud de la posición, y las detecciones fallidas.

Si nos centramos en la monitorización del flujo del tráfico y en la obtención de matrices O-D, es posible desplegar detectores especiales, como los anillos de inducción, en toda la red vehicular. Zijpp [10] propone un método para realizar un seguimiento de los patrones de tráfico variables en el tiempo y combinar el recuento de vehículos con observaciones de trayectoria. Este método puede ser aplicado si la información de los viajes O-D está disponible. Bugeada et al. [11] se han centrado en la estimación de las matrices O-D dependientes del tiempo y de medir las distintas variables de tráfico. Los autores asumen que la recolección de datos de tráfico mediante detecciones por anillos de inducción se complementan mediante mediciones precisas de tiempos de viaje y de velocidad entre dos sensores consecutivos, tales como un dispositivo Bluetooth a bordo de un vehículo.

En este trabajo se adopta una vía alternativa para generar matrices O-D en base a detecciones realizadas mediante anillos de inducción, la cual esencialmente se basa en realizar ingeniería inversa para estimar los puntos de origen y destino de una ruta.

III. VISIÓN GENERAL DE LAS HERRAMIENTAS ADOPTADAS PARA LA SIMULACIÓN

En esta sección presentamos detalles sobre el simulador de tráfico SUMO [1]. Además, presentaremos la herramienta DFROUTER [3] utilizadas en la generación de una matriz O-D para SUMO en base a datos de los anillos de inducción.

III-A. Visión general de SUMO

Usualmente, el modelado del tráfico consiste en obtener variables tales como tiempo de llegada, la ruta seguida por los diferentes vehículos, y en algunos casos la duración de la ruta. Notar que esto último no se puede calcular de una manera muy realista, ya que presupone un determinado vehículo, conductor, y el estado de congestión de tráfico a lo largo de la ruta. SUMO[1] aborda este reto a través de modelos microscópicos detallados de las ciudades y vehículos, ofreciendo un software de código libre para simulación del tráfico que además incluye una amplia gama de utilidades.

Siendo un simulador de código abierto, experimenta constantes mejoras, y tiene buena aceptación en la comunidad científica. Sus características incluyen: simulación multimodal del tráfico, horarios de tráfico, soporte a varios formatos como OpenStreetMap, capacidad para importar mapas con redes de vías en múltiples formatos, y la capacidad para generar rutas de múltiples orígenes. También ofrece simulación de alto rendimiento a través de la interface TraCI, permitiendo realizar simulaciones interactivas utilizando en combinación con otro simulador (e.g. OMNeT++ [14]).

La principal ventaja de SUMO es el soporte a la simulación multimodal, pues no solo incluye los movimientos de los vehículos en una ciudad, sino también el sistema de transporte público, y hasta las rutas de los peatones. Esto significa que una modalidad del tráfico puede ser descrita por múltiples rutas, las cuales pueden ser compuestas por sub-rutas.

Puesto que un flujo del tráfico es simulado microscópicamente, cada movimiento del vehículo dentro de la red es modelado individualmente, siendo localizado en una posición específica, y a una velocidad determinada. La granularidad temporal de la simulación es de un segundo por defecto, permitiendo la simulación discreta de una movilidad continua en el espacio. Además, las simulaciones respecto a los diferentes atributos de las carreteras, tales como limitaciones de velocidad y reglas de prioridad son igualmente respetadas, incluyendo modelos realistas del conductor.

III-B. Funcionamiento del DFROUTER

Entre los paquetes que se incluyen en la distribución de SUMO, nos basamos en la herramienta DFROUTER [3] para el cálculo de la matriz O-D. En particular, esta herramienta permite, partiendo de un recuento de los anillos de inducción para las diferentes vías de una ciudad, estimar las rutas actuales de los vehículos que coincidan con los datos de entrada. La Figura 1 ilustra los diferentes elementos asociados con este proceso. Se observa que DFROUTER necesita datos de entrada de los detectores de flujo, además de información sobre la propia red vial. En base a esta información, estima las rutas asociadas al patrón de tráfico, detallando el origen y el destino de los diferentes vehículos. Concretamente, los pasos seguidos por DFROUTER son los siguientes:

1. Importación de la red vial, incluyendo la posición de los detectores y medidas asociadas.
2. Clasificación de los detectores en la siguiente categoría: detectores de origen (puntos de origen de las rutas), detectores intermedios, y detectores destino (puntos finales de las rutas).
3. Cálculo del flujo de los vehículos entre detectores consecutivos.
4. Cálculo de las probabilidades de uso de las rutas. Usualmente los valores medidos son referentes a un carril específicos, y deben ser agrupados para cada sección transversal.

El principal requisito de DFROUTER es que la red vial debe contener al menos un anillo de inducción detector por cada segmento de vía. Otras fuentes de información de flujo de tráfico diferentes de los anillos de inducción no son soportados.

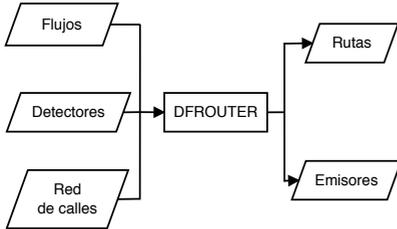


Figura 1: Diagrama de flujo de DFROUTER.

Esto significa que DFROUTER necesita una red que contenga una lista de detectores de anillos de inducción, incluyendo sus posiciones y recuento de vehículos asociados.

Un anillo de inducción es considerado un detector origen cuando no hay otro detector en la misma calle, o en alguna calle precedente. Esto significa que los vehículos comienzan nuevas rutas en esos puntos. Estos detectores son los más comunes en las ciudades, siendo imprescindibles para nuestro objetivo, que no es más que maximizar las coincidencias entre los datos de referencia y los resultados de simulación.

Después de obtener la información relativa a los puntos de entrada y salida de los vehículos en la red, el tercer paso es determinar las rutas asociadas a cada uno de los orígenes y destinos. Con este propósito DFROUTER primero genera un fichero con todas las posibles rutas, aunque no incluye información sobre el número de vehículos asociados a cada ruta. A continuación, DFROUTER combina las rutas calculadas con la información de flujos tomadas de los sensores de bucles de inducción reales para determinar el número de vehículos asociados a cada ruta. Para cumplir este cometido se requiere un archivo que contenga la siguiente información: (i) id del detector, (ii) tiempo inicial, (iii) número de vehículos que pasan sobre el detector, y (iv) la velocidad promedio de esos vehículos.

En el último paso del proceso es cuando DFROUTER realmente genera la matriz O-D, almacenando la información sobre el número de vehículos calculados junto con sus rutas. Con respecto a los vehículos, sus rutas comienzan en cualquier detector de origen localizado a lo largo de la ruta asociada.

IV. METODOLOGÍA DE GENERACIÓN DEL FLUJO DE TRÁFICO

Los datos más críticos para que los experimentos de simulación sean representativos son el número de vehículos inyectados en la red y su destino (matriz O-D). En esta sección describiremos el procedimiento seguido para lograr una matriz O-D coherente con los datos de tráfico para la ciudad de Valencia, España.

A continuación describiremos la metodología que seguimos para cumplir nuestro objetivo de producir datos realistas de

flujo de tráfico, comenzando con un análisis de la salida del DFROUTER, seguido por la propuesta realizada donde se detalla la heurística propuesta, la cual será posteriormente validada utilizando datos reales.

IV-A. Análisis de la salida del DFROUTER

Para evaluar cómo DFROUTER se comporta utilizando información real de anillos de inducción usamos los datos para la ciudad de Valencia proporcionados por los administradores del tráfico de esta ciudad. Como dato de entrada tomaremos los valores de los contadores para los distintos detectores basados en anillos de inducción existentes en las diferentes calles y avenidas. Para nuestro estudio escogemos el mes de Noviembre porque éste no tiene días festivos, y tiene valores relativamente similares a otros meses [2]. Concretamente, nos centramos en un lunes típico, durante la hora pico (entre 8:00 am y 9:00 am).

Para encontrar la descripción del tráfico correspondiente a nuestra red en términos de vehículos inyectados y su respectivas rutas, seguimos los cuatro procedimientos definidos anteriormente para la herramienta DFROUTER. No obstante, después de completar este proceso, se observaron algunas discrepancias entre el tráfico generado y los datos originales, poniendo en evidencia que los datos de salida de DFROUTER no coinciden con el tráfico real en la ciudad de Valencia para el periodo de tiempo especificado. Concretamente, el número de detecciones usando DFROUTER es 138.4% más elevado que el número total de detecciones en el dato de referencia. Esto ocurre porque DFROUTER solamente conoce el número de vehículos que pasan a través de un detector, pero no sabe el número preciso de vehículos que han sido inyectados en la red para hacerlo coincidir con el número real. Así, en la siguiente sección, proponemos una heurística para corregir este error y hacer que la caracterización del tráfico sea más realista.

IV-B. Heurística propuesta

Respecto al desajuste entre la salida del DFROUTER y los datos reales descritos anteriormente, proponemos un ajuste heurístico para compensar este error, buscando introducir un número de vehículos en nuestro entorno de simulación lo más realista posible.

Como dato de referencia tenemos la información relativa al tráfico vehicular en un intervalo de tiempo específico, disponible como recuento de vehículos para un determinado segmento de calle.

Para cumplir nuestro objetivo probamos la heurística propuesta llamada "iterativa". Esta heurística realiza un ajuste global del tráfico que afecta a todas las calles, para de esta manera ajustar el volumen del tráfico a los datos de referencia. Sin embargo, dado que hay dependencias entre los recuentos de los anillos de inducción en las diferentes calles y el tráfico inyectado, se requiere un proceso iterativo para encontrar el mejor ajuste posible. Esta heurística se ilustra en el Algoritmo 1.

Al final de ese proceso generamos dos archivos que resumen la información del tráfico asociado a los diferentes identificadores de segmentos de calle. El primer archivo está

Algoritmo 1 Heurística iterativa.

- Entrada:** Archivos de red de carreteras, flujos, y detectores
Salida: Archivo de información de Vehículos-Calle-Segmento
- 1: **repetir**
 - 2: Procesar el archivos de entrada con DFROUTER
 - 3: Archivos de salida de DFROUTER son Rutas y Emisores
 - 4: Contar el número de vehículos por identificación de la calle (β)
 - 5: Calcular el número de referencia del vehículo (α)
 - 6: Evaluar la relación con la ecuación 1
 - 7: Crear un archivo con información sobre vehículos, segmentos y calles
 - 8: Aplicar la ecuación 2 a todas las identificaciones de la calle (τ_s)
 - 9: **hasta que** $\phi \neq 1$

compuesto por una tupla de dos elementos, donde cada tupla incluye la identificación del segmento de calle y el número de vehículos por segmento. El segundo archivo une los diferentes identificadores de segmento de calle que pertenecen a la misma calle, lo cual resulta en un conjunto de tres elementos: 1) Identificación de la calle, 2) Identificación de los segmentos asociados, y 3) Identificación del segmento con el número más reducido de vehículos, donde el último corresponde al número de vehículos que van a ser inyectados (comenzando) en esa calle en particular.

IV-C. Detalles y validación de la heurística propuesta

En esta sección detallaremos los cálculos asociados a la heurística propuesta en la sección anterior. Para ello partimos de los primeros resultados que nos ofrece DFROUTER para obtener las rutas y la matriz O-D (información del vehículo) usando solo los datos de los anillos de inducción. Este dato inicial es refinado usando la heurística propuesta.

Esta heurística se basa en un proceso iterativo para determinar el número óptimo de vehículos inyectados en la red que permite igualar los valores de referencia. Después de la primera ejecución del DFROUTER, se obtiene una relación entre el número de vehículos real y el resultante (ϕ), ratio que se obtiene mediante la ecuación 1:

$$\phi = \frac{\alpha}{\beta} \tag{1}$$

donde α es el número real de vehículos que pasan a través de todos los identificadores de calle (referencia), y β es el número simulado de vehículos que pasan por esos mismos identificadores de calles. A continuación se ajusta la cantidad de vehículos inyectados en cada calle mediante la ecuación 2. Concretamente, se reduce el flujo de tráfico φ_s para cada segmento de la calle s en base al factor ϕ . Para lograrlo dividimos el identificador del segmento en la calle s que tiene el contador de vehículos más bajo (σ_s) por el número total de segmentos que pertenece a la calle s (ω_s), y multiplicamos dicho valor por el porcentaje de ajuste para cada identificador de calle (φ_s). De esta manera, se regula el número de vehículos

que realmente se inyectan en una calle concreta (τ_s). Después, se substituye el factor de reducción por calle φ_s con un valor global ϕ . El procedimiento se repite hasta que $\phi \approx 1$.

$$\tau_s = \frac{\sigma_s}{\omega_s} \cdot \varphi_s \tag{2}$$

Se verifica que, inicialmente, la variable ϕ fluctuará entre valores bajos (<1) y altos (>1), hasta que se alcance la convergencia. El enfoque iterativo es usado para encontrar la solución que logra mayor similitud con los datos de referencia. El resultado obtenido para diferentes valores del factor ϕ , incluyendo la solución óptima ($\phi = 0,45$). Al final del proceso, comparamos el número de vehículos generados mediante esta heurística con el valor de referencia, encontrando que el número de vehículos en exceso es de tan solo 7,1% superior. Destacar que este valor es claramente mejor que el inicial (exceso de 138,4%).

Como se observa, este ajuste permite lograr un comportamiento similar al deseado en términos del número de vehículos que pasan a través de los detectores ubicados en los diferentes segmentos de calle de la ciudad.

V. RESULTADOS DE LA SIMULACIÓN

En esta sección, detallamos las pruebas y simulaciones realizadas para las tres ciudades estudiadas: Valencia, Cologne, y Bologna. Para cada ciudad, evaluamos la distribución de origen del tráfico, la distribución del destino del tráfico, y la distribución de dispersión del tráfico. Destacar que, para la ciudad de Valencia, usamos la definición de flujo de tráfico obtenida conforme a la heurística iterativa definida anteriormente, mientras las demás ciudades son proporcionados por la propia herramienta SUMO, y se usan como referencia. Particularmente, queremos determinar si nuestra propuesta permite generar matrices O-D e información de rutas comparables a Cologne y Bologna, o si, por el contrario, los resultados difieren mucho de esas dos ciudades de referencia.

La Tabla 1 muestra las principales características de las tres ciudades analizadas. En términos de área, encontramos diferencias significativas. En el caso de Bologna, el área es de solamente 2.34 Km^2 . Para Cologne tenemos una situación opuesta, teniendo una área de 595.9 Km^2 . En el caso de Valencia, el área analizada cubre toda la ciudad (excluyendo áreas suburbanas), teniendo un tamaño de 77.43 Km^2 (un caso intermedio comparado con los otros dos). En términos de densidad de segmentos de calle, este valor está claramente correlacionado con todo el área.

Si nos centramos en el número promedio de segmentos por Km^2 , verificamos que la densidad de la ciudad de Valencia y la

Tabla 1: Estadísticas generales para las tres ciudades objetivo.

Ciudad	Área [Km^2]	# segmento de calles	# segmento por Km^2	Densidad del vehículo (por Km^2)
Valencia	77.43	11418	147.46	173.886991
Cologne	595.9	21953	36.84	2.31794261
Bologna	2.34	337	144.02	3751.7094

Tabla II: Estadística sobre el tráfico de origen en las ciudades bajo estudio.

Ciudad	# Posiciones origen por Km ²	% de segmento de calles usados
Valencia	3.36	2.28
Cologne	1.82	4.71
Bologna	5.13	3.56

de Bologna es bastante similar, mientras que Cologne tiene una densidad mucho más baja ya que las áreas suburbanas también son parte del mapa. Con respecto a la densidad de vehículos, esta métrica igualmente tiene una clara relación con el área objetivo. En el caso de Bologna, únicamente se analiza el área con mayor intensidad de tráfico. Cologne está en la situación opuesta, cubriendo un área muy amplia, y para Valencia se incluye toda la ciudad, pero no áreas suburbanas.

Con respecto al tráfico generado para la ciudad de Valencia, la información requerida se obtiene según el procedimiento definido en la sección IV. En el caso de Cologne, los datos fueron generados por el Centro Aeronáutico de Alemania (DLR) [15] usando el simulador de movilidad SUMO. Para ello confiaron en la herramienta DUAROUTER [16] de cara a obtener las diferentes rutas mediante el cálculo de la ruta más corta. Respecto a la ciudad de Bologna, sus datos fueron igualmente obtenidos de anillos de inducción, los cuales han sido suministrados por el ayuntamiento de Bologna. Para realizar el cálculo de rutas, obtienen nuevas rutas mediante una asignación aleatoria pero acorde a la distribución dada en [17].

Nuestro análisis estudia en tres métricas distintas: distribución de los orígenes del tráfico, distribución de los destinos del tráfico, y distribución de la dispersión del tráfico. Además, por cada uno de esos elementos, analizaremos:

- La densidad por Km².
- Porcentaje de segmento de calles afectadas.
- La distribución a lo largo del mapa.
- La función de distribución acumulativa (CDF) de vehículos por segmento.

Para que el estudio comparativo se realice en igualdad de condiciones, en los experimentos presentados a continuación, referentes a escenarios simulados, el tiempo de simulación es de 900s, y corresponde a la hora pico.

V-A. Análisis del tráfico en orígenes y destinos

Comenzaremos nuestro análisis estudiando la localización de los diferentes orígenes del tráfico, su densidad por Km², y cómo los puntos de inicio de los vehículos (i.e. orígenes) se distribuyen en el mapa de la ciudad.

Como se muestra en la Tabla II, Bologna tiene una densidad mayor que las otras ciudades; sin embargo, verificamos que solamente se estudia una pequeña y concurrida área de la ciudad. Para Cologne tenemos la situación opuesta comparado con Bologna; la Tabla II muestra que esta tiene una baja densidad en los puntos de partida con un alto número de vehículos por puntos; esta diferencia es esperada ya que el área es mucho más grande, e incluye zonas periféricas. Con respecto a Valencia, el número de puntos origen de tráfico

Tabla III: Estadísticas sobre destinos de tráfico en las ciudades bajo estudio.

Ciudad	# Destinos por Km ²	% de segmento de calles usados
Valencia	17.13	11.61
Cologne	1.74	4.71
Bologna	4.70	3.26

por kilómetro cuadrado es un valor intermedio entre ambas ciudades de referencia, y acorde a su área. Estos valores son distribuidos en toda la ciudad de una manera relativamente homogénea. En general verificamos que, en todos los casos, el número de puntos origen es relativamente pequeño.

Con respecto al número de vehículos asociados a un punto de salida concreto, la Figura 2a muestra que, en Cologne, el número de vehículos asociados a cada origen es en general muy pequeño (siempre menos de 10); por el contrario, en Bologna se llegan a inyectar hasta varios miles de vehículos por cada punto origen de tráfico, asumiendo valores realmente elevados. Para Valencia, el espectro de situaciones posibles es mucho más amplio, donde los orígenes de tráfico son heterogéneos, incluyendo desde una cantidad moderada de vehículos (entre 10 y 500 vehículos), hasta puntos de origen con tan solo 2 o 3 vehículos. En general, los resultados obtenidos para la ciudad de Valencia son más representativos de un área metropolitana, donde las carreteras de entrada a la misma pueden inyectar una alta carga vehicular en el sistema, mientras los vehículos pueden igualmente salir de otras partes de la ciudad a menor escala.

Con respecto al análisis de los destinos del tráfico, estudiaremos la localización geográfica de dichos destinos, sus densidades por Km², así como la cantidad de vehículos asociados a los diferentes puntos de finalización de ruta.

Como se muestra en la Tabla III, verificamos que Valencia es claramente la única ciudad que ofrece un conjunto más completo de destinos. Además, aunque el número de posibles destinos de tráfico para Cologne y Bologna sean ligeramente más bajo respecto al número de orígenes, para Valencia este valor se incrementa sustancialmente. Esto significa que, para Valencia, los orígenes de vehículos están a menudo vinculados a una posición específica, como puntos de acceso a la ciudad desde el exterior, pero luego los destinos de esos vehículos pueden variar sustancialmente, como ocurre en la vida real, por lo que se considera un buen resultado.

En cuanto al CDF correspondiente al destino de los vehículos, la Figura 2b muestra que Cologne mantiene una distribución similar a la de orígenes de tráfico (ver Figura 2a), mientras que para Bologna hay una tendencia a concentrar tráfico en unos pocos destinos (desplazamiento de la curva a la derecha).

Centrándonos en Valencia, verificamos que hay un claro desplazamiento a la izquierda, lo cual significa que los vehículos tienden a completar sus rutas de manera más heterogénea, de acuerdo a los índices y observaciones realizadas con anterioridad, mientras tiene lugar igualmente un efecto de concentración de vehículos en ciertos destinos. Esas localizaciones son asociadas a las principales carreteras existentes, y por lo

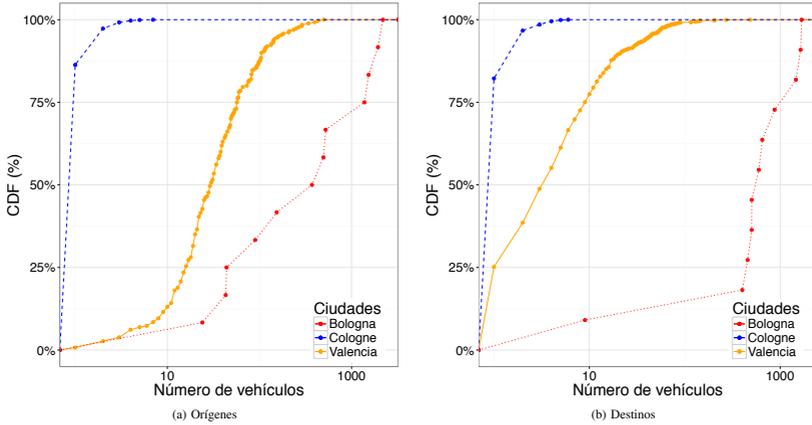


Figura 2: CDF para el número de vehículos por orígenes de tráfico (a) y CDF para el número de vehículos por destino (b).

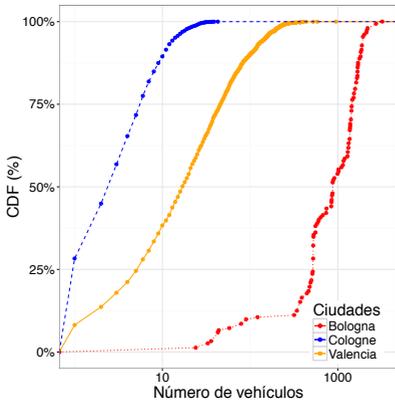


Figura 3: CDF para el número de vehículos por segmento de calle de todo el experimento.

tanto son también esperables.

V-B. Análisis de la dispersión de tráfico

Nuestro análisis concluye con el estudio de la dispersión de tráfico. Para ello determinamos cómo los diferentes segmentos de calle que conforman los mapas son ocupados por los

Tabla IV: Estadísticas sobre la dispersión del tráfico en las ciudades bajo estudio.

Ciudad	# segmentos de calles ocupadas por Km ²	% de segmento de calles usados
Valencia	95.08	64.48
Cologne	12.88	34.97
Bologna	64.96	45.10

vehículos. Para que los experimentos sean significativos, las principales calles/avenidas deberían tener un flujo de tráfico no nulo. Por esta razón, estudiamos la densidad de los segmentos de las calles ocupadas por Km² durante todo el tiempo de simulación.

Nos centramos en el número de segmentos de calle ocupados por kilómetro cuadrado, los cuales se ilustran en la Tabla IV y en la Figura 3. Esta última muestra que Valencia logra una mejor ocupación de los mismos, fenómeno que está asociado a una mejor distribución de tráfico. Para la ciudad de Cologne tenemos la situación opuesta, siendo el número de segmentos de calle ocupados por kilómetro cuadrado excesivamente bajo. Además, verificamos que la mayoría de segmentos de calle se quedan sin utilizar (65.03%). Para Valencia este valor es mucho más bajo (35.52%), valor que se acerca al porcentaje de las calles secundarias que tienen una cantidad de tráfico despreciable. Con respecto a la ciudad de Bologna, los valores obtenidos están en un rango intermedio cuando son comparados con los de Valencia y Cologne.

Con respecto a la CDF para el número de vehículos por segmento de calle en todo el experimento, la Figura 3 muestra que, de manera similar a lo que ocurre en la CDF para fuentes de tráfico (ver Figura 2a), Valencia se caracteriza por un

amplio espectro de situaciones, al contrario de Cologne (el tráfico está muy disperso) y Bologna (el tráfico es muy denso). Además, si comparamos las Figura 2a y 3 podemos observar que hay una tendencia general de ensanchamiento del rango de los valores, que tienden hacia valores centrales. En los casos de Cologne y Bologna, podemos detectar algunos efectos de concentración del tráfico (desplazamiento a la derecha de la curva), mientras que en Valencia tenemos efectos simultáneos de concentración y dispersión del tráfico.

En general, el análisis realizado muestra que el procedimiento seguido para generar matrices O-D para la ciudad de Valencia, junto con la simulación de movilidad para recuperar las rutas y la correspondiente ocupación de calles, evidencia claramente que los resultados obtenidos son adecuados, y en concordancia con el área fijada, manteniendo una relación con los resultados para las ciudades de referencia con áreas más amplias (Cologne) y áreas más reducidas (Bologna).

VI. CONCLUSIONES Y TRABAJO FUTUROS

La simulación es la principal herramienta para analizar y perfeccionar el tráfico en nuestras ciudades. Sin embargo, para que los resultados de simulación sean representativos, el flujo de tráfico inyectado debe ser realista, lo cual significa que el patrón de tráfico, definido como una matriz origen/destino (O-D) para los vehículos de una ciudad, debería ser proporcionados de antemano.

Valencia es una de las muchas ciudades donde el análisis del tráfico es de gran importancia. No obstante, no se dispone de una matriz O-D detallada para el análisis del tráfico, proporcionando a las autoridades únicamente las medidas de los anillos de inducción para las calles/avenidas más relevantes.

En este artículo combinamos la herramienta DFROUNTER, junto con una heurística iterativa, para generar una matriz O-D para el tráfico en Valencia que intenta ser una buena aproximación a la distribución de tráfico real. Comparando el resultado generado contra los datos de tráfico de movilidad existentes para la ciudad de Cologne (Alemania) y Bologna (Italia), verificamos que la definición del flujo de tráfico obtenido para Valencia ofrece resultados muy realistas. Concretamente, se observa una buena dispersión del tráfico por las diferentes calles de la ciudad, lo cual significa que el tráfico fluye a través de la mayoría de segmentos de calle, como en una situación real. Verificamos también que existe una clara asimetría entre calles/segmentos con bajo y con alto nivel de tráfico, como ocurre en situaciones reales.

En general, los resultados obtenidos nos permiten ser optimistas con la matriz O-D generada, y hace posible realizar un análisis de optimizaciones posibles de tráfico durante las horas pico para mejorar los tiempos de viaje y reducir la emisión de CO₂, aspectos que se tratarán en trabajos futuros.

AGRADECIMIENTOS

Este trabajo fue parcialmente respaldado por el Centro de Gestión de Tráfico de Valencia, y por el "Ministerio de Economía y Competitividad, Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, Proyectos I+D+I 2014", España (TEC2014-52690-R).

REFERENCIAS

- [1] Krajzewicz, D., Hertkorn, G., Rüssel, C., & Wagner, P. (2002). SUMO (Simulation of Urban MOBility)-an open-source traffic simulation. In Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM2002) (pp. 183-187).
- [2] Calafate, C. T., Soler, D., Cano, J. C., & Manzoni, P. (2015). Traffic Management as a Service: The Traffic Flow Pattern Classification Problem. *Mathematical Problems in Engineering*, 2015.
- [3] Nguyen, T. V., Krajzewicz, D., Fullerton, M., & Nicolay, E. (2015). DFROUNTER—Estimation of Vehicle Routes from Cross-Section Measurements. In *Modeling Mobility with Open Data* (pp. 3-23). Springer International Publishing.
- [4] Qiu, Z., & Cheng, P. (2007, January). State of the art and practice: cellular probe technology applied in advanced traveler information system. In 86th Annual Meeting of the Transportation Research Board, Washington, DC (No. 0223).
- [5] Caceres, N., Wideberg, J. P., & Benitez, F. G. (2007). Deriving origin destination data from a mobile phone network. *Intelligent Transport Systems, IET*, 1(1), 15-26.
- [6] Sohn, K., & Kim, D. (2008). Dynamic origin-destination flow estimation using cellular communication system. *Vehicular Technology, IEEE Transactions on*, 57(5), 2703-2713.
- [7] Pan, C., Lu, J., Di, S., & Ran, B. (2006). Cellular-based data-extracting method for trip distribution. *Transportation Research Record: Journal of the Transportation Research Board*, (1945), 33-39.
- [8] Hamed, M., Fish, R. L. W., & Haghani, A. (2010). Freeway dynamic message sign evaluation using bluetooth sensors: A case study. In 17th ITS World Congress.
- [9] Lees-Miller, J., Wilson, R. E., & Box, S. (2013). Hidden markov models for vehicle tracking with bluetooth.
- [10] Van Der Zijpp, N. (1997). Dynamic origin-destination matrix estimation from traffic counts and automated vehicle identification data. *Transportation Research Record: Journal of the Transportation Research Board*, (1607), 87-94.
- [11] Barceló Bugeda, J., Montero Mercadé, L., Ballejos, M., Serch, O., & Carmona, C. (2012). A kalman filter approach for the estimation of time dependent od matrices exploiting bluetooth traffic data collection. In *TRB 91st Annual Meeting Compendium of Papers DVD* (pp. 1-16).
- [12] Shehata, M. S., Cai, J., Badawy, W. M., Burr, T. W., Pervez, M. S., Johannesson, R. J., & Radmanesh, A. (2008). Video-based automatic incident detection for smart roads: the outdoor environmental challenges regarding false alarms. *Intelligent Transportation Systems, IEEE Transactions on*, 9(2), 349-360.
- [13] Kastriński, V., Zervakis, M., & Kalatzakis, K. (2003). A survey of video processing techniques for traffic applications. *Image and vision computing*, 21(4), 359-381.
- [14] Varga, A. (2001, June). The OMNeT++ discrete event simulation system. In Proceedings of the European simulation multiconference (ESM'2001) (Vol. 9, No. S 185, p. 65). sn.
- [15] Krajzewicz, D. (2010). Traffic simulation with SUMO—simulation of urban mobility. In *Fundamentals of traffic simulation* (pp. 269-293). Springer New York.
- [16] Behrisch, M., Bieker, L., Erdmann, J., & Krajzewicz, D. (2011, October). Sumo—simulation of urban mobility. In *The Third International Conference on Advances in System Simulation (SIMUL 2011)*, Barcelona, Spain.
- [17] Bieker, L., Krajzewicz, D., Morra, A., Michelacci, C., & Cartolano, F. (2015). Traffic simulation for all: a real world traffic scenario from the city of Bologna. In *Modeling Mobility with Open Data* (pp. 47-60). Springer International Publishing.

DAGRI: Detección Automática de Grietas en Pavimentos

A. Cubero-Fernandez¹, Jose M. Palomares², Joaquin Olivares³, R. Villatoro⁴, F. León⁵

Resumen— Cada año miles de euros se invierten en reparar y mantener las redes de carreteras de todo el mundo, siendo uno de los aspectos más importantes la detección a tiempo de estas reparaciones para minimizar los costes.

En el presente artículo se trata la detección e identificación de grietas en el pavimento mediante análisis de imágenes. Tras la captación de las imágenes estas son procesadas para extraer las características más importantes de las mismas buscando siempre resaltar las posibles grietas de la imagen en caso de que las haya. Una vez las imágenes se han procesado se aplica un algoritmo heurístico de árbol de decisión para la clasificación final de la imagen: sin grieta, grieta horizontal, grieta vertical o grieta en forma de malla.

Palabras clave— Detección de grietas, visión por computadora, retos de la sociedad

I. INTRODUCCIÓN

La seguridad vial es, sin duda alguna, un tema que en la sociedad de hoy en día resulta primordial, ya que, en mayor o menor medida, todo el mundo utiliza la red nacional de carreteras del estado. Por lo tanto, mantener en buen estado el asfalto de las vías es algo fundamental para reducir los accidentes, y que traiga como consecuencia directa una reducción en el número de fallecidos.

Hay muchas circunstancias que son imposibles de prevenir en un accidente de tráfico como pueden ser inclemencias meteorológicas, invasiones de la calzada por parte peatones, animales o algún objeto, despistes o fallos de conducción por parte del conductor, etc. Pero en cambio, hay un aspecto en el que la prevención juega un gran papel y es el mantenimiento tanto del vehículo por parte de cada conductor como sobre todo de la calzada.

Una vez están construidas las carreteras, diferentes problemas pueden mostrarse en forma de grieta en la superficie del asfalto lo cual, dependiendo de la gravedad, puede llegar a causar incluso accidentes o puede hacer que posteriormente la reparación sea muy cara por lo cual es fundamental repararlas lo antes posible.

Según la DGT, en 2012 la red de carreteras española cuenta con una cifra de 165.593 km. Dado lo dificultoso de mantener en correcto funcionamiento tantos kilómetros de vías con unos recursos tan limitados como los que puede contar una administración pública y más aún en los tiempos que corren se necesita buscar nuevas herramientas con las que poder inspeccionar las carreteras de una forma más eficaz ya que la cifra de kilómetros de vías es inabordable de cara a realizar inspecciones visuales en el terreno.

Aunque existen otros tipos, principalmente las grietas que se producen en nuestras carreteras generalmente son de tres tipos: Transversales, longitudinales y de malla o co-

codrilo.

Las grietas transversales son perpendiculares a la línea central del pavimento y normalmente son causadas por cambios térmicos. Otras posibles causas son el endurecimiento del aglutinante del asfalto o la reflexión externa de grietas bajo la superficie del asfalto.

Las grietas longitudinales tienen 2 principales causas: la fatiga y juntas débiles. Las grietas por fatiga se producen por una continua sobrecarga por el paso de vehículos pesados. Por otro lado, el asfalto es menos denso en las juntas, si una junta está localizada en una zona de alto estrés podría aparecer una grieta longitudinal.

Finalmente, las grietas en forma de malla están directamente relacionadas con la fatiga y una base para el asfalto inestable. Esto hace que sea difícil de determinar la causa exacta para ser reparadas. Estas grietas son las peores ya que hacen el asfalto altamente inestable e incluso pueden llegar a desprenderse algunos trozos de pavimento.

Además de estos tipos, otros factores pueden influir en la creación de grietas como por ejemplo raíces de árboles, desprendimientos o fuertes impactos aunque estas causas no suelen ser las más comunes.

Dada la gran cantidad de causas y las posibles consecuencias, es un aspecto fundamental del mantenimiento de carreteras el detectar a tiempo donde hay grietas y las causas de las mismas. Y es por esto que cada día no solo se gastan miles de euros en construir carreteras, sino que también se emplean en repararlas.

Antes de realizar la reparación hay que descubrir las posibles grietas. Tradicionalmente esto se ha hecho mediante chequeo visual humano, aunque esta técnica cada vez se usa menos por su coste tanto humano como económico. Teniendo esto en cuenta, las técnicas más recientes consisten en la captación de datos mediante imágenes para su posterior procesamiento. Estos métodos permiten captar grietas que a simple vista podrían pasar desapercibidas además de que aumenta la seguridad en el proceso. Por lo tanto el objetivo del presente artículo consiste en realizar un sistema automático de detección de grietas en el cual la intervención humana sea mínima para que el sistema pueda funcionar de forma autónoma.

El artículo se estructura de la siguiente manera: La Sección II hace un análisis de trabajos relacionados, la Sección III introduce la metodología utilizada en nuestro trabajo, las Sec. IV y V describen el preprocesamiento y los métodos de detección automática de grietas respectivamente y, finalmente, las Sec. VI y VII concluyen el artículo con los resultados obtenidos y las conclusiones.

II. TRABAJO RELACIONADO

Las técnicas basadas en imágenes son fundamentales en la detección de grietas en el pavimento. Estas técnicas han llamado especialmente la atención en los últimos

¹Arquitectura de Computadores, Electrónica y Tecnología Electrónica, Universidad de Córdoba. e-mail: acubero@uco.es

²jmpalomares@uco.es

³olivares@uco.es

⁴rvillatorosanchez@gmail.com

⁵fernando.leon@uco.es

años dando como resultado una serie de publicaciones. Este apartado dará una breve introducción a las publicaciones más importantes en este ámbito.

Zou *et al.* [1] muestran en su trabajo como realzar y limpiar una imagen con una grieta, dando como resultado una imagen limpia de ruido y con la grieta resaltada. Para ello, en primer lugar elimina las posibles sombras de la imagen mediante la utilización de algoritmos de eliminación de sombras geodésicos. Tras esto, mediante mapas de probabilidad, se realiza la grieta. Esto se consigue comprobando diferencias de intensidades y con algoritmos basados en umbrales para posteriormente emplear técnicas de *Tensor Voting* [2] con las que se genera el mapa de probabilidades de grietas. Finalmente construyen un árbol recubridor mínimo para realizar la grieta y eliminar el ruido.

Uno de los principales problemas de este tipo de trabajos es que se asume que la imagen será buena y con alto contraste, algo que en la realidad es difícil de conseguir. Li *et al.* [3] han diseñado una estrategia para conseguir eliminar casi por completo el ruido y la información superflua. Para ello han diseñado *FoSA - F* Seed-growing Approach* el cual elimina el requisito del algoritmo *F** el cual necesita que se establezcan previamente los puntos de inicio y fin. Además también reduce el espacio de búsqueda para incrementar la eficiencia.

Peggy *et al.* [4] buscan realzar y localizar grietas basándose en la transformada wavelet. En primer lugar, la transformada se lleva a cabo para varias escalas. Tras esto se realiza un procesado posterior el cual da como resultado una imagen binaria indicando la presencia (o no) de grietas en la imagen.

Li *et al.* [5] utilizan redes neuronales para detectar grietas. Para ello han de realizar un preprocesamiento para preparar la imagen antes del aprendizaje y clasificación. En primer lugar, corrigen el fondo haciéndolo mas uniforme para después suavizar la imagen eliminando el ruido mediante el desenfoque gaussiano y, finalmente, realizan una transformación del histograma para resaltar la grieta. Tras el preprocesamiento, se utilizan técnicas que dividen la imagen en trozos los cuales clasifican con un umbral en grietas o no y se elimina el fondo. Tras seleccionar solo las imágenes con grietas se utilizan dos redes neuronales distintas para clasificarlas. En primer lugar se hace la distinción entre grietas lineales o de malla y posteriormente, con otra red neuronal, se clasifican en longitudinales o transversales.

Finalmente, Yamaguchi *et al.* [6] intentan solventar el problema del tiempo de cómputo en este tipo de sistemas, ya que la mayoría de trabajos se centran en la eficacia detectando imperfecciones y no en el tiempo de ejecución. Muestran una serie de técnicas basadas en la percolación, empleando procesado local escalable mediante el cual consiguen llevar a cabo una reducción importante del tiempo de ejecución, a la vez que mantienen la eficacia en la detección.

III. EL MÉTODO

En esta sección se presenta DAGRI, un sistema automático de detección de grietas mediante captación y procesamiento de imágenes. Comenzaremos con una intro-

ducción al diseño seguido de un análisis de cada parte.

El trabajo presentado en este artículo se encuentra dividido en 2 partes principales: Preprocesado de la imagen y aprendizaje y clasificación.

En una situación perfecta, tras la captación de la imagen en la misma solo se vería la grieta. Sin embargo, las condiciones hacen que esto no sea así (sombras, partículas de la superficie, etc) es por esto que se ha de realizar un preprocesamiento en el cual se elimina el ruido y se realza la grieta. Esto se realiza con diferentes técnicas de procesamiento de imagen aplicando transformadas, filtros y algoritmos para posteriormente extraer características de la imagen.

Finalmente el resultado es analizado mediante árboles de decisión en los que se detecta si en la imagen hay alguna grieta y de que tipo es.

Formato de la imagen

Se parte de un desarrollo de un sistema *offline* que se pretende implementar en un sistema empujado dentro de sistemas móviles en coches o camiones. Sin embargo, se pretende comprobar la correctitud de la propuesta y por lo tanto, el sistema se desarrollará a partir de imágenes estáticas captadas ad-hoc. Para éstas, el formato de imagen de entrada del sistema es un aspecto importante del mismo, ya que una imagen muy pesada podría causar un procesamiento lento, pero una imagen muy ligera podría no tener suficiente información. Tras analizar los diferentes formatos disponibles se ha decidido utilizar imágenes en formato .JPG.

JPG es un estándar de compresión y codificación de archivos e imágenes fijas el cual puede causar un decremento de la calidad de imagen. No obstante esta pequeña pérdida es asumible para este sistema. Además, de cara a una futura implementación en un sistema empujado tanto de captación de imágenes como de detección de grietas, este formato de imagen permitiría almacenar una gran cantidad de imágenes así como un rápido procesado.

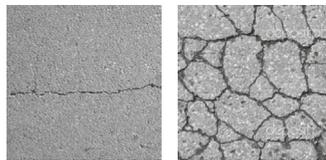


Fig. 1: Ejemplo de grieta horizontal y de malla

En la búsqueda de un compromiso entre tamaño/calidad se optó por redimensionar o dividir las imágenes a 320×320 píxeles. Este formato puede ser fácilmente procesado sin graves retardos en la ejecución del programa. Las imágenes han de ser realizadas con calidad y estar bien enfocadas. En la imagen 1 se puede observar dos ejemplos de imágenes listas para ser procesadas.

IV. PREPROCESAMIENTO DE LA IMAGEN

Para la detección de grietas en imágenes existe el problema de que la región de la imagen en la que aparece

la grieta normalmente suele ser muy pequeña en comparación con el resto de la imagen, especialmente en fotos en las que el pavimento no está en buenas condiciones. Esto quiere decir que la mayoría de la imagen pertenece al fondo innecesario. Sin embargo, este fondo, debido a las características de la captura de la imagen suele tener mucho ruido lo cual dificulta la separación de la parte útil de la imagen de la superflua.

Por lo tanto, es necesario realizar un preprocesamiento a la imagen para eliminar tanto como sea posible las interferencias ambientales. Con el fin de realizar el procesamiento de la forma más rápida y eficaz tanto en términos de implementación como de ejecución se ha decidido utilizar OpenCV [7] el cual implementa todos los algoritmos utilizados de forma nativa y C++ como lenguaje de programación para realizar todo el preprocesamiento a continuación descrito. El preprocesamiento de la imagen no consta de un único paso, sino que se realizan diferentes tratamientos hasta llegar a la imagen final.

En primer lugar, la imagen es transformada a una nueva en escala de grises y en negativo, esto facilita el posterior procesamiento y la visualización de los resultados. A continuación, se muestran los pasos que se han seguido para preprocesar cada imagen.

A. Transformada logarítmica

La transformada logarítmica se utiliza normalmente para mapear un rango pequeño de píxeles en uno mayor. Con esta transformación se busca expandir los píxeles oscuros de la imagen lo cual permite enfatizar la grieta en caso de que la haya. En la figura 2 se puede ver un ejemplo de esta transformada.

B. Suavizado de la imagen: Filtrado bilateral

El ruido aleatorio es uno de los principales problemas que podrían afectar a los resultados. Debido a las características físicas del asfalto, todas las imágenes presentarán un granulado uniforme y aleatorio por toda la imagen el cual podría alterar el contraste entre la grieta y el fondo de la imagen. Es por ello que debe ser eliminado.

El filtrado bilateral se utiliza normalmente para conseguir este objetivo siendo una de las herramientas más potentes para suavizar una imagen sin perder la precisión de los bordes, lo cual será crucial para las siguientes fases. En el filtrado lineal convencional se utiliza una máscara constante en todo el dominio teniendo en cuenta la distancia euclídea al píxel central (denominado *filtro de dominio*). La principal contribución del filtrado bilateral consiste en la utilización de una máscara adicional la cual no es lineal y que mide las variaciones de intensidad con el píxel central (en adelante *filtro de rango*).

La función del filtro de dominio es suavizar la imagen mientras que la del filtro de rango es destacar las discontinuidades de intensidades. Esto lo logra comparando las intensidades de los píxeles de la vecindad con respecto al píxel central, donde se pondera con menor peso los píxeles cuya diferencia sea mayor. La función del parámetro σ_R es proporcionar una brecha de referencia para la determinación de la existencia de un borde. El filtro resultante que se aplica a la imagen es el que se obtiene de la

multiplicación punto a punto entre ambas máscaras. En la figura 2 se puede ver el resultado de aplicar el filtrado bilateral a una imagen con grietas.

C. Detección de bordes: El algoritmo de Canny

Uno de los métodos relacionados con la detección de bordes es el uso de la primera derivada, la que es usada por que toma el valor cero en todas regiones donde no varía la intensidad y tiene un valor constante en toda la transición de intensidad. Por tanto un cambio de intensidad se manifiesta como un cambio brusco en la primera derivada, característica que es usada para detectar un borde, y en la que se basa el algoritmo de Canny. En la figura 2 se puede ver el resultado de aplicar el algoritmo de Canny a una imagen con grietas tras aplicar anteriormente el filtrado bilateral.

D. Filtrado Morfológico

Existen muchos tipos de filtrado morfológico. Sin embargo, tras analizar los resultados de los diferentes filtros se ha utilizado el filtro de cierre. El filtro de cierre es igual que el de apertura pero realizado de forma inversa. Una definición sencilla sería la dilatación seguida de una erosión.

En el filtrado de cierre de una imagen A por un elemento estructurante B se define como la dilatación de A por B seguida de la erosión por el mismo elemento estructurante. Si A no cambia con el cierre por B se dice que A es cerrada respecto a B.

La aplicación de la dilatación produce una extensión de la imagen original. Gracias a ella pueden rellenarse huecos de los objetos que forman la imagen, pero a su vez se aumenta el tamaño de éstos. La aplicación de la erosión devuelve a los objetos el tamaño inicial que tenían. El cierre morfológico binario permite rellenar estructuras sin modificar el tamaño inicial de éstas.

En la figura 2 aparece un ejemplo de aplicar el filtrado morfológico de cierre a una imagen con grietas previamente procesada con los algoritmos anteriormente descritos.

V. DETECCIÓN AUTOMÁTICA DE LAS GRIETAS

El aprendizaje y posterior clasificación se realizará mediante árboles de decisión. Los árboles de decisión son uno de los métodos de aprendizaje inductivos más usados. Consisten en aproximar una función desconocida a partir de ejemplos positivos y negativos de esa función. Estos ejemplos serán en realidad pares $[x, f(x)]$, donde x es el valor de entrada y $f(x)$ el valor de la función aplicada a x . En un árbol de decisión cada nodo del árbol es un atributo (campo), y cada rama representa un posible valor de ese atributo.

Gracias a todo el trabajo anterior dedicado al preprocesamiento se consigue una imagen final con una grieta claramente definida. No obstante, esta imagen ha de ser adaptada para posteriormente llevar a cabo el proceso de aprendizaje y clasificación siempre premiando la mínima pérdida de información de interés. Para ello se utilizarán *integrales proyectivas*.

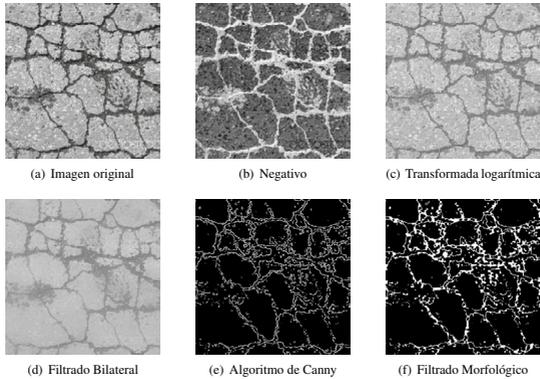


Fig. 2: Resultado del preprocesamiento paso a paso. Cada imagen es el resultado de aplicar un método a la imagen anterior. La primera fila representa la imagen original, negativa y la transformada logarítmica. En la segunda fila se puede ver el resultado de aplicar el filtrado bilateral, el algoritmo de Canny y el filtrado morfológico.

A. Simplificando las imágenes: Integrales proyectivas

Una integral proyectiva (o proyección) consiste en el cálculo de la media de los valores de gris de una imagen a lo largo de las filas o columnas de píxeles. Este método es utilizado normalmente para el reconocimiento facial. Este trabajo está basado en imágenes de 320×320 píxeles, lo cual se traduce en 102400 datos. Tras obtener las integrales proyectivas cada imagen será transformada en 2 vectores de 320 datos (uno para la integral proyectiva vertical y otro para la integral proyectiva horizontal) lo cual nos dará un total de 640 datos. Esto conlleva una importante reducción de la información a la vez que se mantiene un buen nivel caracterizador de cada tipo de imágenes. En las figuras 3 y 4 se muestra el resultado de obtener las integrales proyectivas horizontales y verticales respectivamente a una imagen con grietas. En la figura correspondiente a la integral proyectiva horizontal, el eje vertical se corresponde con las filas de la imagen y el horizontal con la suma de los valores de los píxeles de cada fila. Para la integral proyectiva vertical, los significados de los ejes son intercambiados siendo el eje horizontal el número de columna de la imagen y el vertical la suma de los valores de los píxeles para cada columna.

Alcanzado este punto, las imágenes están completamente procesadas y con la información necesaria para el aprendizaje y el procesamiento posterior.

B. Proceso de aprendizaje y clasificación

Todas las aplicaciones de aprendizaje necesitan datos de entrada y resultados. En este sistema, los datos de entrada son las integrales proyectivas de la imagen y los resultados son la existencia o no de una grieta en la imagen, así como la descripción del tipo.

Para llevar a cabo la implementación se ha utilizado el entorno Weka [8] debido a la facilidad que ofrece para trabajar con una gran colección de herramientas. Entre todas las herramientas que ofrece se ha utilizado J48, la

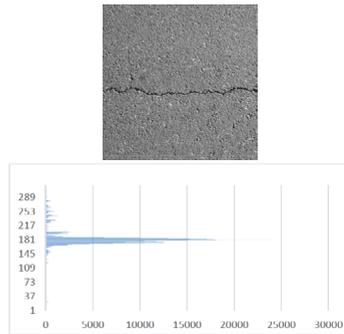


Fig. 3: Ejemplo de integral proyectiva horizontal

cual es una implementación del algoritmo C4.5.

El algoritmo C4.5 fue propuesto por Quinlan [9] en 1993, que estaba a su vez basado en el algoritmo ID3 [10], también propuesto por Quinlan. Este algoritmo se utiliza para generar árboles de decisión. Estos árboles pueden ser utilizados posteriormente para clasificación, es por esto que C4.5 se denomina un *Clasificador estadístico*.

Una vez aplicado el clasificador estadístico se obtendrán 2 modelos de clasificación, uno para las integrales proyectivas verticales y otro para las integrales proyectivas horizontales. Estos dos modelos se le aplicarán a futuras imágenes y dando como resultado la no detección de grietas (ningún modelo detecta grieta), detección de grietas horizontales o verticales (cuando solo un modelo detecta grietas) o la detección de grietas en forma de malla (ambos modelos detectan grietas).

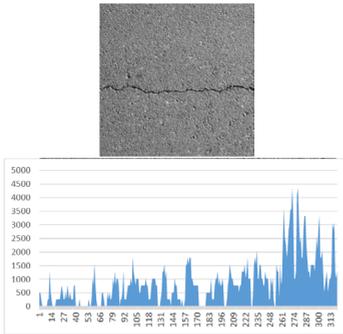


Fig. 4: Ejemplo de integral proyectiva vertical

VI. RESULTADOS

Tras realizar diferentes experimentos durante el proceso de desarrollo para descubrir los valores óptimos, se ha realizado un experimento final con 400 imágenes de diferente tipo, concretamente 100 de cada categoría (Sin grietas, Con grietas horizontales, Verticales y de Malla). Para realizar el aprendizaje se ha utilizado un aprendizaje mediante validación cruzada (*Cross-validation*). En esta técnica se repite varias veces el aprendizaje con diferentes conjuntos de aprendizaje y test para finalmente calcular la media de los resultados. Tras el proceso de aprendizaje se obtiene un modelo para la integral proyectiva vertical de 23 niveles de profundidad con 45 nodos de los cuales 23 son nodos hoja y otro modelo para la integral proyectiva horizontal de 9 niveles de profundidad con 23 nodos de los cuales 12 son nodos hoja.

En la Tabla I se pueden ver los resultados obtenidos. La columna *Éxito* indica el porcentaje de imágenes que se han clasificado correctamente. La columna *Errores de clasificación* indica los errores en los que el sistema ha clasificado una grieta incorrectamente y, finalmente, la columna *Error total* indica los casos en los que se ha clasificado una imagen como grieta cuando en realidad no la había o viceversa.

Los resultados son positivos, siendo el porcentaje de éxi-

TABLA I: Resultados del experimento

Tipo	Éxito	Error clasif.	Error total
Horizontal	72 %	19 %	9 %
Vertical	67 %	24 %	9 %
Malla	97 %	3 %	0 %
Ninguna	75 %	0 %	15 %
Total	80.25 %	11.5 %	8.25 %

to superior a dos de cada tres para todos los tipos de imágenes y, en general, por encima del 80 %. El porcentaje es menor en la imágenes horizontales y verticales, ya que son las que más confusión pueden crear al sistema. Además, en las imágenes con una sola grieta es donde más

variedad se ha usado de cara a la clasificación, por tanto es coherente que sea donde más errores se puedan encontrar. Por otro lado, en las imágenes con grietas en forma de malla y sin grietas es lógico que no haya tantos errores, ya que resulta más sencilla su clasificación.

Se había planteado la realización de comparaciones con algunos de los otros métodos descritos en la Sec. II. Sin embargo, no ha sido posible obtener el código de los respectivos autores, bien porque no han contestado a los correos que se les han enviado o bien porque dichos algoritmos están en sistemas en explotación con algún grado de protección del software. Partir de las descripciones de los métodos indicados por los autores en los respectivos artículos se consideró que excedía las posibilidades temporales de este trabajo. Por otra parte, tampoco fue posible encontrar los bancos de imágenes con las que los autores originales proporcionaron los resultados en sus artículos, por lo que tampoco se pudo recalcular los resultados de este trabajo con los datos de aquellos artículos. Por ello, únicamente se pueden proporcionar resultados de tipo general, en los que se puede observar que el sistema DAGRI proporciona resultados muy positivos. La Tabla II muestra una comparativa con los resultados ofrecidos en el artículo "Automatic Pavement Crack Recognition Based on BP Neural Network" (en adelante, Li *et al.* Method [5]). Con el fin de ajustar el sistema DAGRI de la forma más precisa al sistema con el que se realiza la comparativa, se han eliminado del experimento las imágenes que no contienen grietas, tal y como los autores realizaron en el artículo anteriormente citado. También cabe destacar que el sistema Li *et al.* Method realiza sus pruebas con un número significativamente menor de imágenes (80/278, 50/141 para el primer y el segundo experimento respectivamente) el cual se considera que puede ser insuficiente, dadas las características del sistema.

En la Tabla II se puede comprobar como los resultados son similares en el primer experimento, y en el segundo, DAGRI es considerablemente menos preciso. No obstante, aunque esto pueda parecer un fracaso, este sistema es capaz de clasificar en 4 categorías: grietas en malla, grietas horizontales, grietas verticales y sin grietas. En particular, DAGRI es capaz de realizar clasificaciones de imágenes en las que no aparezcan grietas, aspecto que el sistema Li *et al.* Method no es capaz de realizar. De hecho, en el caso del sistema Li *et al.* Method hay que realizar una clasificación previa para distinguir las imágenes en las que aparezcan grietas, lo cual ralentiza el proceso.

TABLA II: Comparativa

Método	Éxito grietas Malla	Éxito grietas lin.
DAGRI	99 %	78 %
Li <i>et al.</i> [5]	97.5 %	87.5 %

Método	Éxito grietas hor.	Éxito grietas ver.
DAGRI	77.7 %	73 %
Li <i>et al.</i> [5]	100 %	88 %

VII. CONCLUSIONES

Con esta investigación se ha conseguido realizar un sistema capaz de clasificar, con un alto índice de aciertos, un conjunto de imágenes con grietas. No obstante este trabajo puede ser continuado y mejorado. Una posible mejora sería realizar una primera clasificación distinguiendo si existe o no grieta sin tener en cuenta el tipo o probar otros sistemas de aprendizaje. Esto podría incrementar el índice de acierto.

Como futuro trabajo se plantea la integración del sistema completo en un sistema empotrado para su uso en tiempo real. Los experimentos realizados han sido llevados a cabo en un entorno de simulación partiendo de unas imágenes previamente obtenidas y sin restricciones de tiempo. Por lo tanto el siguiente paso en la investigación será la integración en un sistema real la cual creemos que se podrá realizar sin grandes dificultades ya que los cálculos necesarios para la clasificación no son excesivamente complejos.

Por otro lado, se considera que esta línea de investigación es bastante interesante y hay un gran interés público en este tipo de sistemas. A pesar de lo cual, encontrar bases de datos públicas de imágenes para poder hacer pruebas ha sido prácticamente imposible, y es por ello que se insta a la comunidad a compartir dichas imágenes en repositorios públicos.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado mediante los proyectos: P11-TIC-7462 y DPI2013-47347-C2-2-R.

REFERENCIAS

- [1] Qin Zou, Yu Cao, Qingquan Li, Qingzhou Mao, and Song Wang, "CrackTree: Automatic crack detection from pavement images," *Pattern Recognition Letters*, vol. 33, no. 3, pp. 227-238, 2012.
- [2] Gérard Medioni and Mi-Suen Lee, "Tensor Voting : Theory and Applications," *Congrès francophone sur la Reconnaissance des Formes et l'Intelligence Artificielle (RFIA)*, p. 3, 2000.
- [3] Qingquan Li, Qin Zou, Daqiang Zhang, and Qingzhou Mao, "FSA: F^o Seed-growing Approach for crack-line detection from pavement images," *Image and Vision Computing*, vol. 29, no. 12, pp. 861-872, 2011.
- [4] Peggy Subirats and Jean Dumoulin, "Automation of pavement surface crack detection using the continuous wavelet transform," *Image Processing*, vol. 1, no. 1, pp. 3037-3040, 2006.
- [5] Li Li, Lijun Sun, Guobao Ning, and Shengguang Tan, "Automatic Pavement Crack Recognition Based on Bp Neural Network," vol. 26, no. 1, pp. 11-22, 2014.
- [6] Tomoyuki Yamaguchi and Shuji Hashimoto, "Fast crack detection method for large-size concrete surface images using percolation-based image processing," pp. 797-809, 2010.
- [7] Gary Bradski et al., "The opencv library," *Doctor Dobbs Journal*, vol. 25, no. 11, pp. 120-126, 2000.
- [8] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10-18, 2009.
- [9] J. Ross Quinlan, *C4.5: Programs for Machine Learning*, 1993.
- [10] J R Quinlan, "Induction of Decision Trees," *Machine Learning*, vol. 1, no. 1, pp. 81-106, 1986.

Conectividad de Sistemas

Propagation models in single/multi-radio scenarios: overview and influence analysis

J.M. García*[‡], F.J. Estévez*[‡], G. Rebel*[‡], J.M. Castillo-Secilla[‡], J. González[‡] and P. Glösekötter*

*Lab. for Semiconductor Devices & Bussystems

University of Applied Sciences of Münster, Münster, Germany

Email: jose_m_garcia@fh-muenster.de

[†]Dept. of Computer Technology and Computation

University of Alicante, Alicante, Spain

[‡]Dept. of Computer Architecture and Technology

University of Granada, Granada, Spain

Email: jesugonzalez@ugr.es

Abstract—The evolution in low power communications and low cost electronics is pushing up the deployment of more and more devices, which traduces in an overcrowded radio electrical environment. This overcrowd requires new mechanisms to mitigate it, and therefore, Cognitive Radio and Multi Radio techniques become relevant. These techniques are gaining the attention of the research community, becoming a trendy research theme. Due to the difficulty for deploying large networks with research purposes, simulators arises as an useful alternative. Simulators are a research field itself, being based on different components, from where propagation models is one of the principals, allowing the simulation of different environment conditions. This work presents a deep analysis of the different wireless propagation models in a well-known simulator, focusing on the packet delivery ratio, error rate and energy consumption. The different models are compared under different conditions, varying the node density and sending frequency of each scenario.

Index Terms—propagation models, multi radio, WSN, AODV

I. INTRODUCTION

The evolution in low power communications and low cost hardware is pushing up the society into a world where everything is interconnected. Crowd of connected devices is based on Wireless Sensor Networks (WSN), [1] due to the worldwide legal limitations tend to use the 2.4GHz Industrial, Scientific and Medical (ISM) band. Nowadays researchers are studying how to mitigate the impact of a huge amount of devices working in the same frequency. Depending on the frequency, each standard defines a number of channels, which use different management techniques. This work focuses on the analysis of low-power and low-rate wireless connectivity using the IEEE 802.15.4-standard, which defines up to 15 channels in the 2.4GHz ISM-band.

Different applications are increasing the number of devices that coexist in the same ISM-bands and channels, which derive in overcrowded channels with a high collision-rate. The increment in the number of collisions and the downturn in the throughput lead to developing new techniques that improve the coexistence such as software defined radios or cognitive radio.

The analysis of these problems is usually difficult, as long as involve hundred or thousand devices, thus the research community turns to simulation environments, which give the opportunity to study those behaviours in detail. This peak in simulation use leads us to another question, are the wireless propagation models in the simulators reliable enough? How do they influence a study? These open questions are studied in the present article, analyzing the impact of different wireless propagation models to different scenarios based on single-/multi-radio and node density.

The rest of the paper is organized as follows: Section II provides an overview of different wireless propagation models available for simulation, as long as a short literature review. Section III presents the system design, which is mainly focused on minimizing the collisions in order to improve the throughput of the networks. Section IV presents a deep analysis in terms of Packet Delivery Ratio (PDR), Error Rate (ER) and energy consumption for the propagation models analyzed. Finally, in Section V the results are discussed and the most relevant conclusions of this paper are presented.

II. STATE OF ART

The study of wireless propagation models in simulation environment is not under the focus of the research community, regardless its importance. Lots of studies such as [2], [3], [4], [5] or [6] are based on simulation results to justify their innovation, but not as many studies analyze the impact of wireless propagation models in the results obtained. Some studies such as [7], discuss about the improvement in the propagation characteristics through the use of cognitive radio techniques. Nevertheless, and in spite of the fact that this work considers the propagation models characteristics, it does not evaluate different propagation models, remaining open the question about the impact of the wireless propagation models on the results.

Grudén et al. [8] have analyzed the radio environment in a close space, such as a jet engine, carrying out empirical tests. From those tests they simply consider the environmental factor inside the jet engine, discarding the comparison with

different wireless propagation models. In the same way goes the work of Patri et al. [9]. Their work analyses the radio frequency propagation in an underground coal mine. They focus on some physical parameters such as path loss, node distance and packet error rate (PER), but they do not consider the impact of using other propagation models.

Another interesting work has been developed by He et al. [10]. They discussed different propagation models and proposed a new one based on 3D radio propagation environments for indoor ZigBee WSNs. This new wireless propagation model considers the nearby obstacles, but it is mainly focused on indoor environments, discarding outdoor scenarios. The authors carry out a comparison between the simulated propagation model and a real use case where they validate their model, but do not consider the analysis of different propagation models for simulation purposes.

However, different propagation models already exist in the network simulators. For example, OMNeT++, a well known network simulator includes both idealistic and realistic models [11], allowing the researchers to test the behaviour under several propagation circumstances.

Propagation models can be classified into two groups: deterministic and probabilistic models. Deterministic models are conceived to represent the radio behaviour without taking into account the environment. They adjust the range of the antenna depending on the radio parameters, such as the transmitter power, the antenna gain and so on. Probabilistic models take a different approach simulating environment effects like reflections with external objects or interactions between radios. Some of the most representative models are described below.

A. Free Space Model

This model is a deterministic model that calculates the signal attenuation over a given distance d . Eq. 1 defines how the signal decreases, depending on the radio parameters:

$$Pr_{det}(d) = \frac{P_t G_t G_r \lambda}{(4\pi)^2 d^2 L} \quad (1)$$

where P_t represents the transmission power, G_t and G_r the gain of the transmitter and receiver antenna, λ is the wavelength and L is the loss factor. The main disadvantage of the model is its deterministic behaviour. Physical effects like fast fading, reflection or scattering are not taken into account, so it is an ideal approach.

B. Two Ray Ground Model

This is another deterministic model, which determines the signal attenuation similarly to the *Free Space* model, but considering the reflection on the ground too. Eq. 2 defines the path loss:

$$Pr_{det}(d) = \frac{P_t G_t G_r h_t^2 h_r^2}{d^4} \quad (2)$$

where h_t and h_r represent the height of the transceiver's and receiver's antenna. Despite it is less idealistic than the *Free Space*, it is not a realistic model.

C. Log Normal Shadowing Model

This is a probabilistic model which employs a normal distribution to hand out reception power in the logarithmic domain. The σ of the normal distribution can be modified, obtaining the following expression for the log-normal distribution:

$$Pr_{LogNormal}(d; \sigma^2) \sim LN(Pr_{det}(d), \sigma^2) \quad (3)$$

D. Rayleigh Model

This is also a probabilistic model which provides intensive variation of the reception power for non-line-of-sight communications. It hands out the reception power using a Rayleigh distribution. The deterministic model serves as a base to calculate the average power for the distribution.

$$Pr_{Rayleigh}(d) \sim Rayleigh(Pr_{det}(d)) \quad (4)$$

E. Rice Model

This model takes the Rayleigh one as a base and includes the positive effects of a line-of-sight path to calculate the reception power, as is shown in Eq 5:

$$Pr_{Rice}(d, k) \sim Rayleigh(Pr_{det}(d), k) \quad (5)$$

where k is the parameter which involves the associated effects of a line-of-sight path. It offers a more realistic alternative than the *Rayleigh* model.

F. Nakagami Model

This probabilistic model takes into account the fading effects on the signal to disseminate the reception power using a gamma distribution, as is shown in Eq. 6:

$$Pr_{Nakagami}(d; m) \sim Gamma(m, \frac{Pr_{det}(d)}{m}) \quad (6)$$

where the m parameter defines the fading effects. It has proved itself well suited to reflect some realistic conditions [12].

III. METHODOLOGY

This work has been developed over the standard IEEE 802.15.4, because it is commonly used as a base for the majority of WSN protocols.

The simulation software selected to carry out this comparative study was OMNeT++, due to its capacity to simulate a wide range of standard and non-standards protocols. However, the software itself does not provide any protocol. It is necessary to download model frameworks developed as independent projects. For WSN, the framework used was *INETMANET*, a branch of the *INET* framework that contains the standard model library of OMNeT++, adding several additional features mainly for mobile ad-hoc networks and Wireless Personal Area Networks (WPANs).

The different elements from a network are implemented in OMNeT++ through modules. These modules are single objects that contain both, the functionality of a full-functional network stack, covering from physical-layer to application layer and the battery, wireless channel control, terminal interface, etc. Therefore, the channel control module is responsible for

Parameter	Value
Effective Range (m)	65
Carrier Frequency (GHz)	2.4
Carrier sense sensitivity (dBm)	-85
Transmit power (mW)	0.2
MAC queue length	50
MAC Ack	Disabled
Routing Protocol	AODV
Message Length (Bytes)	70
Channel number Single-Radio	0
Channel number Multi-Radio	0 / 5
Path Loss Alpha	2
SNR threshold (dB)	4
Transceiver High (m)	1
Receiver High (m)	1
Sigma	1
Nakagami m	1
K (dB)	8

TABLE I: Main fixed simulation parameters

handling the information about radio interfaces of nodes at transmissions. For this reason, the main objective was to evaluate the network behaviour depending on the propagation model configured on the channel control module.

As a starting point, it was decided to compare the performance over a single-radio model and the multi-radio model, presented in [13], using a representative example in OMNeT++. Since the multi-radio model developed in the previous work was implemented using AODV, this study uses the same basic scheme allowing us to compare the results between the two radio models.

The AODV example selected exists for OMNeT++ in the framework *INETMANET*. It is called *csma802154*, and it is included in the *wpan* examples folder. Nodes present in this network example contain different modules like the referent to the User Datagram Protocol (UDP), the Transmission Control Protocol (TCP) or the Internet Protocol (IP), a routing module for mobile ad-hoc networks (MANETs) and a battery module. Furthermore, the module of the Network Interface Card (NIC), corresponding to an IEEE 802.15.4 standard interface is also present.

Modifying the propagation model of this example is a trivial task, because the module contains a parameter with several propagation models, including the *Free Space* model, the *Two Ray Ground* model, the *Rice* model, the *Rayleigh* model, the *Nakagami* model and the *Log-normal Shadowing* model.

IV. EXPERIMENT AND RESULTS

Simulations have been carried out using a static scenario. The nodes are placed applying a random spatial distribution in a two-dimensional square area. In the interest of making these experiments reproducible, we present the main parameters of physical, link and network layer in Table I.

After a testing period checking the correct behaviour of the simulation, the time was limited to 3600 seconds, due to memory restrictions.

The test context is focused on the comparative between the 6 propagation models described in Section II. In order

Parameter	Value
Number of sending nodes	5 / 9
Number of multiple destination nodes	3
Node degree	5 / 15
Time between messages	1 / 30

TABLE II: Main variable simulation parameters

to try out these models, several scenarios have been carried out. The main goal was to observe the behaviour under diverse conditions, showing an overall conception about their performance. The variable parameters modified during the simulation runs are shown in Table II.

The comparison is based on two different interval times, 1 and 30 seconds. are not arbitrary values but they have proved themselves as uncorrelated frequencies in the Kruskal-Wallis test. For these sending frequencies, two different network scenarios are tested, small scenarios and large scenarios. The network size is represented through the node density (or node degree, ND), where a large scenario corresponds to ND5 and a small scenario corresponds to ND15. Both cases are compared using a low-traffic scenario, which involves up to 15 active nodes, sending and/or receiving data to/from the network. In addition, a high-traffic scenario has also been tested for the ND15, involving up to 36 nodes, sending and/or receiving data to/from the network.

To accomplish a fair comparative between models, three primary aspects of the network performance have been analysed: packet delivery ratio (PDR), error rate (CR) and energy consumption. The name of the propagation models has been abbreviated in the graphics. The abbreviations *FS*, *TR*, *RI*, *NA*, *RA* and *LN* correspond to *Free Space*, *Two Ray*, *Rice*, *Nakagami*, *Rayleigh* and *Log Normal*, respectively.

A. Packet Delivery Rate Analysis

The PDR represents the number of messages successfully sent. This parameter gives a precise overview of the propagation model details among different scenarios with different density, sending frequency and traffic conditions.

1) *Low-Traffic Scenario*: Node density 5 and 15 were analysed for a sending frequency of 1 and 30 seconds. Results shown in Fig. 1 – 4 offer interesting details about the network performance when different models are used. Generally, deterministic models present the best delivery ratio as is expected since they are conceived to represent an idealistic transmission. On the contrary, *Nakagami* and *Rayleigh* show very poor ratios, far away of deterministic results. Their means and medians are the worst among the probabilistic models. On the other hand, *Log-Normal* model seems to be the statistic model, obtaining the best ratio results. In addition, it shows close means and medians to deterministic models in some scenarios, even improving the *Log Normal* performance under density 15 and a 1 second for the sending frequency. Multi-radio examples shows the same trend than single-radio, presenting only better delivery ratios with deterministic models.

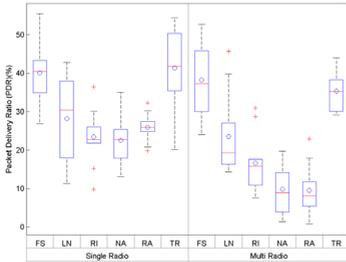


Fig. 1: PDR in low-traffic scenarios under density 5 and 1s sending frequency.

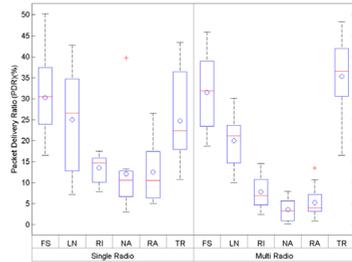


Fig. 3: PDR in low-traffic scenarios under density 15 and 1s sending frequency.

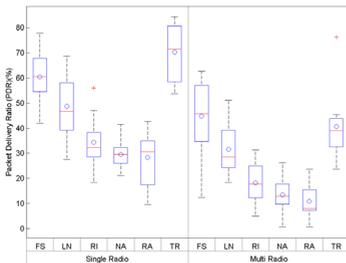


Fig. 2: PDR in low-traffic scenarios under density 5 and 30s sending frequency.

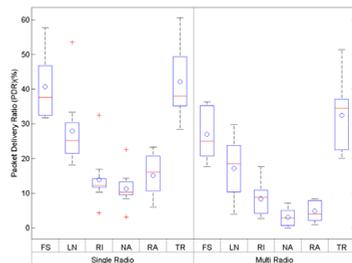


Fig. 4: PDR in low-traffic scenarios under density 15 and 30s sending frequency.

2) *High-Traffic Scenario*: In order to test the behaviour on a high traffic network, simulations were run under density 15, increasing the number of senders up to 9 nodes. Figures 5-6 show the results for these scenarios. The observed trend is similar to low-traffic. *Nakagami* and *Rayleigh* present the worst results and *Log-Normal* stands out among the probabilistic models. When probabilistic models are used, the multi-radio modules does not offer a behaviour improvement.

B. Error Rate Analysis

The collision rate or error rate indicates the amount of collisions per message sent. Lower values are associated to fewer collisions, therefore the network performance is worst for high values of this rate. Scenarios were explained in

Sections IV-A.

1) *Low-Traffic Scenario*: Node density 5 and 15 were analysed for a sending frequency of 1 and 30 seconds. Results shown in Fig. 7 – 10 does not show an overall trend. The different propagation models offer different behaviours across the scenarios. However, for most of the cases, *Two Ray* achieves the lowest error rate. Regarding on the single-radio or multi-radio performance, the error rate is fewer for the 1 second scenarios when a multi-radio device is employed. The behaviour for 30 seconds shows similar results.

2) *High-Traffic Scenario*: The high-traffic scenario included simulations under node density 15. Figures 11-12 show the results for these scenarios. The observed performance is dissimilar than low-traffic. It seems to exist a certain trend for

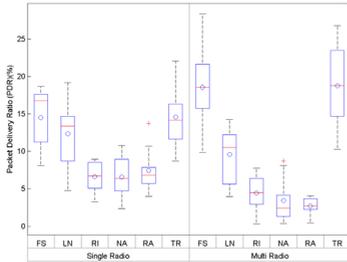


Fig. 5: PDR in high-traffic scenarios under density 15 and 1s sending frequency.

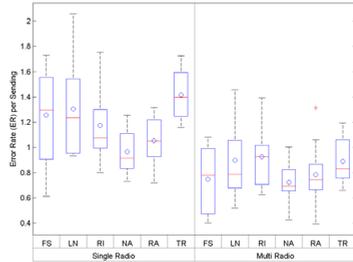


Fig. 7: ER in low-traffic scenarios under density 5 and 1s sending frequency.

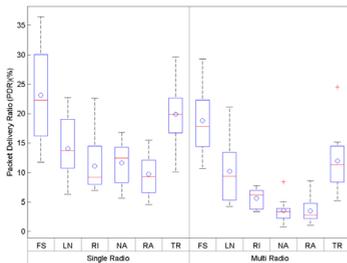


Fig. 6: PDR in high-traffic scenarios under density 15 and 30s sending frequency.

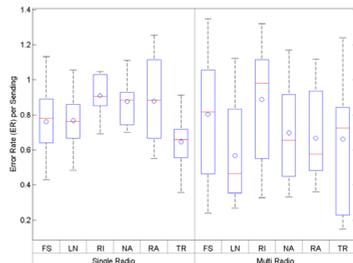


Fig. 8: ER in low-traffic scenarios under density 5 and 30s sending frequency.

the different cases. Despite of the fact that *Two Ray* shows the lowest error rate, *Log-Normal* offers the worst behaviour in all scenarios. *Nakagami* and *Rayleigh* also present a similar behaviour along the different cases. Regarding on the radio module, the error decreases when multi-radio is used. The improvement is more significant in 1 second scenario.

Rayleigh and *Rice* median are similar but *Rayleigh* mean is considerably fewer. *Log Normal* model achieves the worst results.

C. Energy Consumption Analysis

The following figures show the consumed energy during the simulations under the same scenarios explained in Sections IV-A and IV-B.

1) *Low-Traffic Scenario*: Figures 13 – 16 show the results for node densities 5, 15 and 1, 15 seconds for sending frequency. Both radio modules were analysed, offering diverse results. *Free Space* and *Two Ray* results shows the higher consumption under node density 5. On the other hand, *Nakagami* and *Rayleigh* offer the lowest consumption among the models. As was expected, multi-radio module presents a higher consumption than the single-radio.

Under scenarios with 15 density, the trend is not so clear. Single-radio scenarios with a sending frequency of 1 second keeps the ND5 behaviour whereas the other scenarios present a completely different performance. It cannot be obtained an overall performance among the models.

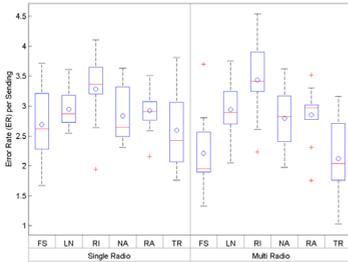


Fig. 9: ER in low-traffic scenarios under density 15 and 1s sending frequency.

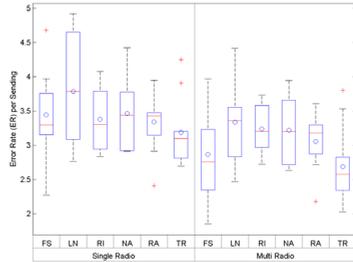


Fig. 11: ER in high-traffic scenarios under density 15 and 1s sending frequency.

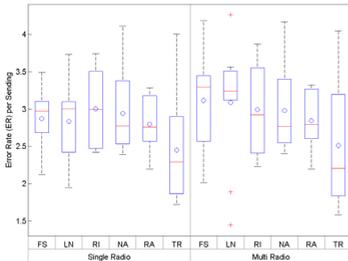


Fig. 10: ER in low-traffic scenarios under density 15 and 30s sending frequency.

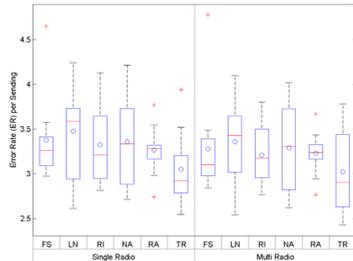


Fig. 12: ER in high-traffic scenarios under density 15 and 30s sending frequency.

2) *High-Traffic Scenario*: Scenarios under node density 15 were tested. Figures 21 – 24 show the results for these cases. As occurred in low traffic ND5 scenarios, *Nakagami* and *Rayleigh* offer the best consumption whereas *Log Normal* and *Free Space* show the worst consumption. Regarding the multi-radio behaviour, it increases the consumption of the single-radio example.

V. DISCUSSION AND CONCLUSION

This paper has carried out an analysis of the influence of propagation models in WSN. The simulations also have obtained information about their impact when a multi-radio model is used, instead a common single-radio model. Due to the amount of data provided in Section IV, we discuss the

results separately.

A. Packet Delivery Rate Discussion

Several conclusions can be obtained from Fig 1–6. Deterministic models (*Free Space* and *Two Ray*) present the best PDR in all scenarios. They barely include environmental effects on the propagation therefore it was the expected behaviour. There is also a trend for the probabilistic models. *Log Normal* stands out, showing results near deterministic models. *Rayleigh*, *Rice* and *Nakagami* models present really low rates. The reason of this behaviour could reside on the node's ability to create the routes. Since AODV needs to send messages to establish the path for packets, low delivery rates for that kind of message could prevent some routes formation. Without the

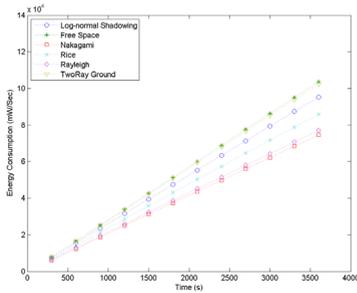


Fig. 13: Energy Consumption in low-traffic single-radio scenarios under density 5 and 1s sending frequency.

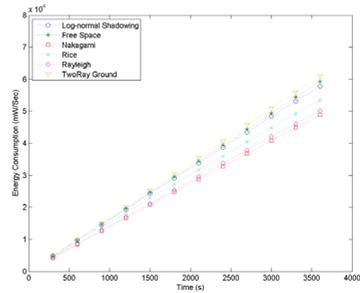


Fig. 15: Energy Consumption in low-traffic single-radio scenarios under density 5 and 30s sending frequency.

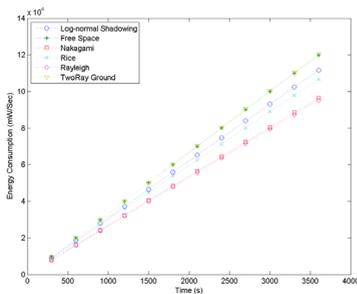


Fig. 14: Energy Consumption in low-traffic multi-radio scenarios under density 5 and 1s sending frequency.

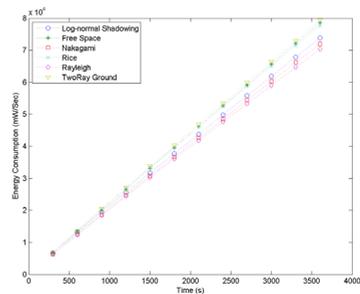


Fig. 16: Energy Consumption in low-traffic multi-radio scenarios under density 5 and 30s sending frequency.

proper routes, PDR will decrease dramatically.

Analysing the overall performance of the network, multi-radio shows better results for a sending frequency of 1 in ND15 examples. Those scenarios present a higher traffic load, therefore, obtaining a better result using a second radio working was the expected result. When time frequency is increased to 30 seconds, single-radio nodes have enough time to handle the messages and multi-radio does not improve its behaviour. The PDR's increment for 30 seconds scenarios respecting to 1 second scenarios also supports this fact. For lower density scenarios, there are not remarkable improvements or even a downturn in terms of performance.

B. Error Rate Discussion

Figures 7–12 show the error rate results. Due to the performance differences between low and high density scenarios, a separately discussion is carried out. Low density scenarios do not present a trend among the models. There is not a conclusion which could be obtained from the results respecting to the propagation models.

Regarding on the overall behaviour, multi-radio scenarios present less collision rate than single radio's. The traffic load scenario, due to the high frequency, explains the best result achieved. On the other hand, the improvement on the network performance when the sending frequency is increased to 30 seconds, can also be observed through the lower error rates for those scenarios.

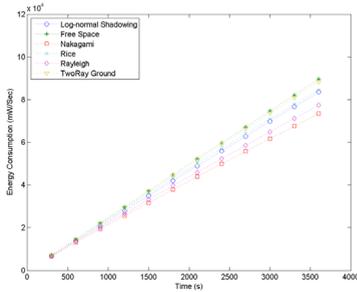


Fig. 17: Energy Consumption in low-traffic single-radio scenarios under density 15 and 1s sending frequency.

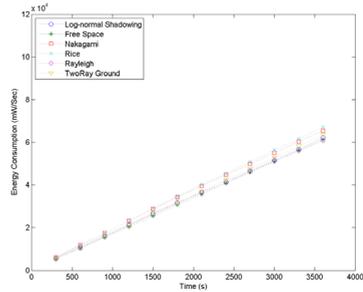


Fig. 19: Energy Consumption in low-traffic single-radio scenarios under density 15 and 30s sending frequency.

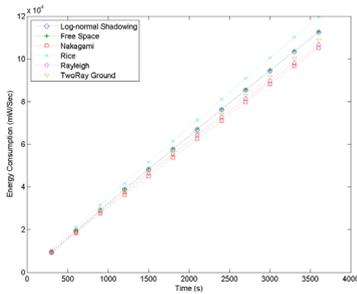


Fig. 18: Energy Consumption in low-traffic multi-radio scenarios under density 15 and 1s sending frequency.

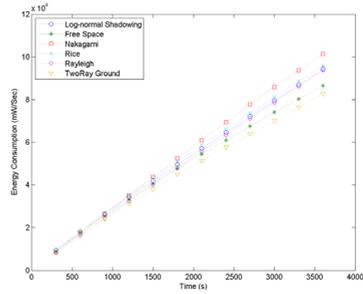


Fig. 20: Energy Consumption in low-traffic multi-radio scenarios under density 15 and 30s sending frequency.

High density cases show a trend over the different scenarios. *Two Ray* model always shows the lowest error rate. In addition, *Free Space* model also seems to be the second best in most of the cases. Therefore, we could affirm that deterministic models present a better behaviour according to the error rate. For high traffic scenarios, multi-radio model achieves a lower error rate as was expected due to the traffic load. However, difference between both radio models becomes smaller for a sending frequency of 30 seconds.

Keeping in mind the PDR results, it does not seem to be a relation between low PDR values and abnormally high error rates. The possible cause resides on the node's ability to form the routes. If they cannot, collisions related to the application messages would not present high values whereas the packet

delivery ratio will decrease significantly.

C. Energy Consumption Discussion

Battery consumption results are shown in figures 13 – 24. It does not seem to exist an overall behaviour. For most of the single-radio scenarios, deterministic models present the higher consumption, but not for all cases. In a similar way the best consumption, associated to *Nakagami* and *Rayleigh* models. The lack of a trend prevents to determine a conclusion about the influence of radio models.

Comparing individually single-radio and multi-radio results, it can be observed a higher consumption on the multi-radio scenarios. It was the expected behaviour, due to the presence of an additional radio. However, the multi-radio consumption

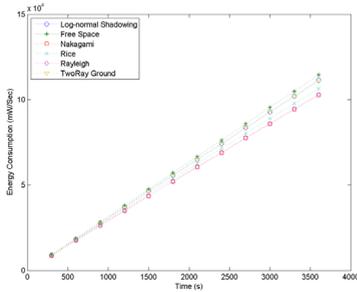


Fig. 21: Energy Consumption in high-traffic single-radio scenarios under density 15 and 1s sending frequency.

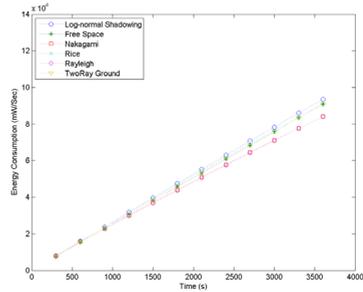


Fig. 23: Energy Consumption in high-traffic single-radio scenarios under density 15 and 30s sending frequency.

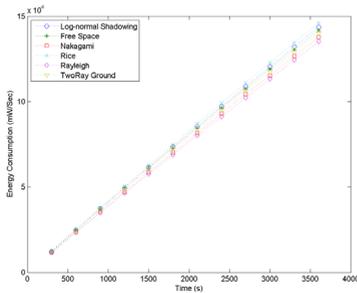


Fig. 22: Energy Consumption in high-traffic multi-radio scenarios under density 15 and 1s sending frequency.

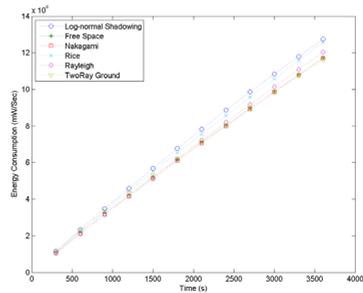


Fig. 24: Energy Consumption in high-traffic multi-radio scenarios under density 15 and 30s sending frequency.

do not duplicate single-radio consumption at any case.

D. Overall conclusion

General performance of deterministic and probabilistic models is remarkably different. Deterministic models offer a significantly better behaviour at any condition. Therefore, researchers should take into account not only idealistic environments but also realistic ones. Employing both deterministic and probabilistic models will lead to better approximations to the real performance of a WSN. Due to the dissimilar results, providing detailed information about the used model should be also a primary aspect of a research, in order to allow the replication of the experiments.

When a deterministic propagation is used, the multi-radio

model presents better results for high traffic and high node density as was presented in our previous work [13].

However, if a probabilistic propagation is used, the performance of multi-radio model is always worse than the single radio. The main reason of this behaviour could be the interactions between radios in multi-radio models.

VI. FUTURE WORK

A future work should go deeply into the misbehaviour regarding the interactions between radios. The main reason of the problem needs to be found in order to plan an achievable solution. Other research line will be analysing the behaviour under a different routing algorithm or protocol. Using a specific wireless algorithm, focused on the managing of a

multi-radio model. A multi-radio oriented policy could serve to reach or improve the single-radio behaviour for all the scenarios.

REFERENCES

- [1] D. Rudinska J. Stankunas and E. Lasauskas. Experimental Research of Wireless Sensor Network Application in Aviation. *Elektronika ir Elektrotechnika*, 111(5), 2011.
- [2] Kermajani H. and Gomez C. On the network convergence process in rpl over ieee 802.15.4 multihop networks: Improvement and trade-offs. *Sensors*, 14:11993–12022, 2014.
- [3] Chatzivasileiadis S.; Bonvini M.; Matanza J.; Zin R.X.; Noudui T.S.; Kara E.C.; Parmar R.; Lorenzetti D.; Wetter M. and Kiliccote S. Cyber-physical modeling of distributed resources for distribution system operations. In *Proceedings of the IEEE*, pages 789–806. Piscataway, NJ, USA, 2016. IEEE.
- [4] Lall S.; Maharaj B.T.J. and Van Vuuren P.A.J. Null-frequency jamming of a proactive routing protocol in wireless mesh networks. *Journal of Network and Computer Applications*, 61:133–141, 2016.
- [5] Anchora L.; Capone A.; Mainetti L.; Mighali V. and Patrono L. As2-mac: An energy-efficient mac protocol for wireless sensor networks. *Ad Hoc Sensor Wireless Networks*, 31:199–226, 2016.
- [6] Bhor D.; Angappan K. and Sivalingam K.M. Network and power-grid co-simulation framework for smart grid wide-area monitoring networks. *Journal of Network and Computer Applications*, 59:274–284, 2016.
- [7] M. Abbaspour R. Tizvar and M. Dehghani. CR-CEA: A collision- and energy-aware routing method for cognitive radio wireless sensor networks. *Wireless Networks*, (5):2037–52, May 2014.
- [8] M. Jobs M. Grudén and A. Ryydberg. Empirical Tests of Wireless Sensor Network in Jet Engine Including Characterization of Radio Wave Propagation and Fading. *IEEE Antennas and Wireless Propagation Letters*, 13:762–65, 2014.
- [9] D.S. Nimaje A. Patri and R. Odisha. Radio Frequency Propagation Model and Fading of Wireless Signal at 2.4GHz in Underground Coal Mine. *Journal South African Institute of Mining and Metallurgy*, 115:629–36, August 2015.
- [10] G. Liang J. Portilla D. He, G. Mujica and T. Riesgo. Radio propagation modeling and real test of ZigBee based indoor wireless sensor networks. *Journal of Systems Architecture*, 60:711–25, 2014.
- [11] O. Grunte H. Hartenstein A. Kuntz, F. Schmidt-Eisenlohr and M. Zitterbart. Introducing Probabilistic Radio Propagation Models in OMNeT++ Mobility Framework and Cross Validation Check with NS-2. In *SimuTools '08 Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems workshops*, number 72, Brussels, Belgium, 2008.
- [12] Vikas Taliwal, Daniel Jiang, Heiko Mangold, Chi Chen, and Raja Sengupta. Empirical determination of channel characteristics for dsrc vehicle-to-vehicle communication. In *Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks, VANET '04*, pages 88–88, New York, NY, USA, 2004. ACM.
- [13] J.M. Castillo-Secilla J. González F. Estevez, J.M. García and P. Gloeskoetter. Enabling validation of IEEE 802.15.4 performance through a new dual-radio OMNeT++ model. *Elektronika ir Elektrotechnika*, 2016.

Propagation models in single/multi-radio scenarios: overview and influence analysis

Fernando León¹, Jose M. Palomares², Joaquín Olivares³, A. Cubero-Fernández⁴

Resumen— Los sistemas basados en reglas (RBS) utilizan RETE como uno de sus principales mecanismos para reconocer patrones y disparar las reglas consecuentes. Tradicionalmente, éste se ha implementado en sistemas con altas capacidades computacionales debido a sus altos requerimientos de memoria y cómputo. Sin embargo, el actual auge de dispositivos de reducido tamaño, coste y prestaciones utilizados en las redes inalámbricas de sensores dentro del marco de la IoT, hace especialmente interesante portar este modelo al nuevo paradigma distribuido. RETE parte de unos hechos evaluados externamente, quedando fuera del modelo la gestión de los requerimientos computacionales requeridos para esta tarea. En este trabajo se propone un modelo matemático ampliado, basado en RETE, que va a expandir el control de todo el proceso de cómputo desde la introducción de las variables de entrada hasta la detección del patrón que dispararía la regla, pasando por la evaluación de los hechos. El modelo propuesto permite balancear la calidad del resultado frente al consumo de recursos computacionales y de comunicaciones a través de ciertos parámetros simples en los nodos. En este trabajo se incluyen resultados experimentales obtenidos mediante simulación que demuestran la validez y potencial utilidad de la propuesta.

Palabras clave— RETE, Sistemas Basados en Reglas, Reconocimiento de patrones para reglas, WSN, IoT

I. INTRODUCCIÓN

EL desarrollo y optimización de técnicas de reconocimiento de patrones en la década de los 80 extendió el uso de los sistemas basados en reglas. El algoritmo RETE propuesto por Forgy [1] sentó un precedente debido a la eficiencia que presentaba trabajando con grandes volúmenes de reglas. Desde entonces, se han propuesto muchas mejoras al algoritmo original, que sigue siendo la base de muchos sistemas de producción basados en reglas de la actualidad [2]. Por una parte, la computación paralela dio lugar a los *sistemas basados en reglas distribuidos*. Por otra, los lenguajes de definición de las reglas fueron evolucionando, siendo cada vez más declarativos y menos procedurales. Estas nuevas características, más el incremento de la potencia de los computadores de la época, sentaron las bases necesarias para el desarrollo de sistemas más complejos compuestos por componentes autónomos que se acabaron denominando *agentes*; concepto ampliamente utilizado en la década de los 90. La idea de módulos software autónomos capaces de colaborar y de tomar decisiones en respuesta a los estímulos de un entorno complejo y cambiando replanteó la manera en que se concebía la inteligencia artificial [3]. En la actualidad nos vemos inmersos en el desarrollo del concepto de *Internet of Things (IoT)* por sus siglas en inglés), que en esencia surge de las posibilidades que plantea la miniaturización de la tecnología de computación y comunicación, y engloba un amplio rango de disciplinas

¹Arquitectura de Computadores, Electrónica y Tecnología Electrónica, Universidad de Córdoba (España). e-mail: fernando.leon@uco.es

²jmpalomares@uco.es

³olivares@uco.es

⁴acubero@uco.es

[4][5]. También surge como respuesta necesaria al mundo tecnológico que viene, donde los dispositivos conectados a redes globales se contarán por miles de millones y la mayor parte del tráfico no será generado por seres humanos [6]. Tecnologías relativamente recientes, como la *identificación por radiofrecuencia (RFID)* o las *redes inalámbricas de sensores (WSN)* están siendo claves en el desarrollo de la IoT, ya que han hecho posible el inicio de la computación ubicua[7].

La tecnología WSN utiliza dispositivos pequeños denominados *motas* que colaboran para extraer datos del entorno mediante enlaces inalámbricos. La principal característica de las motas es que están muy restringidas debido principalmente a su reducido coste, consumo y tamaño. Esta tecnología se está empezando a utilizar en sectores clave como entornos industriales, aplicaciones militares, aplicaciones ambientales, entornos urbanos, infraestructuras o medicina[8].

Este trabajo se enmarca dentro de una línea de investigación de usos no convencionales de la tecnología WSN, y la principal motivación es el desarrollo de técnicas específicas que permitan no solo extraer y acumular simples datos del entorno sino extraer conocimiento verdaderamente útil según el contexto de la aplicación, y para cumplir este objetivo planteamos utilizar las WSN como sistemas basados en reglas distribuidos. En esta publicación proponemos un modelo alternativo al clásico RETE que permita minimizar los recursos de cómputo y de red necesarios para su ejecución, ya que pretende detectar las condiciones de las reglas en cada mota.

II. REVISIÓN CIENTÍFICO-TÉCNICA

A. RETE original

Forgy [1] introdujo RETE en 1982. El algoritmo original plantea cada regla como un grafo acíclico compuesto por nodos de dos tipos: α y β . Un *hecho* es una afirmación lógica realizada sobre datos o entradas del sistema, y se representan por tuplas ordenadas de datos puntuales. RETE modela los antecedentes de cada regla como un árbol cuyos nodos son las premisas que dan lugar al disparo de la misma. Los nodos α son el conjunto de nodos hoja del árbol y en una primera fase modelan hechos simples que vienen dados como entradas al sistema. La segunda fase es la red β , donde los nodos encadenan operaciones lógicas sobre las afirmaciones de α hasta un resultado final, que supone la satisfacción del predicado de la regla. Por último, cada ejecución del algoritmo RETE recoge una lista de predicados que se satisfacen, resuelve posibles conflictos y asigna un orden de ejecución de las reglas.

B. Limitaciones de RETE y mejoras propuestas

RETE sentó las bases del reconocimiento de patrones al ser el primer algoritmo que permitió hacer frente a volúmenes de reglas inmanejables hasta el momento. Esencialmente, el método sacrifica memoria a cambio de velocidad de cómputo, aunque para grandes volúmenes de datos este consumo de memoria supone un problema serio. Como es de suponer, muchas mejoras se han venido proponiendo hasta la fecha.

TREAT [9] fue el primer gran avance en los sistemas de reconocimiento de patrones tras RETE. No es exactamente una mejora, ya que aunque parte de TREAT es similar a RETE, el primero presenta un método diferente para el reconocimiento de patrones y no utiliza memoria en los nodos beta. Gracias a esta propuesta, se reduce el número de comparaciones y de uso de memoria. Todo lo cual se traduce en una mayor velocidad de respuesta.

El campo de la Inteligencia Artificial en la industria de los Videojuegos ha sido uno de los grandes impulsores de las mejoras de RETE debido al gran uso de patrones y reglas en este tipo de aplicaciones. Un mecanismo más eficiente en términos de velocidad de procesamiento fue propuesto por Wright y Marshall [10] denominado RETE*. Este mecanismo se centra en optimizar los nodos beta, dotándoles de cierta flexibilidad en la gestión de la memoria de dichos nodos.

Recientemente se han propuesto [11][12][13] mejoras encaminadas a reducir el espacio de búsqueda de patrones, desactivando todas aquellas reglas que no son susceptibles de estar disponibles según el contexto de ejecución actual. Para ello, se aplican funciones hash sobre los nodos alfa para aumentar la velocidad de *matching*.

C. RETE distribuido y con restricciones de cómputo

Existe una clara tendencia hacia un despliegue máximo de sistemas de toma de decisión con múltiples elementos, bien sea a través del soporte de Cloud Computing, de las WSN, IoT u otros sistemas distribuidos. Esta tendencia también ha tenido influencia en RETE, proponiéndose mecanismos que lo dotan de un carácter distribuido. A su vez, los nodos que componen los sistemas distribuidos se presentan con elevadas restricciones de cómputo y memoria, que difieren en mucho de las especificaciones de los nodos de los sistemas RETE clásicos.

Ligado al ámbito de la Inteligencia Artificial para Videojuegos, Madden [14] realizó una propuesta para optimizar RETE en sistemas multiagentes con baja memoria. En ella se introducen una serie de optimizaciones buscando minimizar el uso de memoria manteniendo un alto porcentaje de recursos compartidos, lo cual hace que la reducción del rendimiento se mantenga a un nivel acotado. Este trabajo hace hincapié en la eficiencia del uso de memoria y de los requisitos de cómputo.

Para lograr la distribución de RETE, el sistema RUNES-II [15] se apoya en las plataformas Cloud Computing para construir un motor de distribuido de reglas basado en RETE mediante el modelo distribuido por paso de mensajes. Este trabajo se centra principalmente en aumentar el rendimiento de los sistemas de comunicaciones.

III. REPLANTEAMIENTO DEL MODELO RETE

En esta sección se describirá un modelo matemático basado en RETE. La principal motivación es hacer compatible el reconocimiento de patrones en redes de dispositivos muy restringidos, como suelen ser las motas que conforman las WSN. El modelo matemático pretende sentar una base sólida para su implementación, e introduce una serie de optimizaciones que reducirán las necesidades de recursos de cómputo y de red para llevar a cabo su tarea. Como solución de compromiso entre el consumo y la eficacia del método, las optimizaciones son controlables de manera dinámica, con lo que se puede lograr una adaptación en tiempo real a la carga del sistema en función de las necesidades.

Partiendo de RETE, el modelo propuesto divide el árbol de las reglas en tres partes: árboles α , interfaces $\alpha\beta$ y árbol β . El proceso de reconocimiento de patrones utiliza valores de entrada, los somete a una serie de operaciones y ofrece como resultado un valor booleano, que representa si el patrón ha sido reconocido. Las entradas se obtienen de sensores o de variables del sistema, y éstas son operadas por operadores aritméticos dentro de los árboles α . Los resultados de los árboles α , almacenados en las raíces de los mismos, se combinan entre sí mediante operadores relacionales en las interfaces $\alpha\beta$, produciendo así *hechos* evaluados. Así, hemos introducido el concepto de nodo $\alpha\beta$ en sustitución de los antiguos nodos α , que ahora se modelan mediante un árbol. La razón de este cambio es tener bajo control del modelo todo el cálculo previo de los hechos, para poder someterlo a restricciones controladas y, por lo tanto, minimizar la carga del sistema cuando sea conveniente. Por último, el patrón será reconocido mediante la combinación lógica de estos hechos en el árbol β , tal y como se hace en RETE.

Teniendo en cuenta esta breve descripción previa, se puede observar un criterio de división de las partes. Cada nivel tiene un dominio de entrada, un conjunto de operadores y un dominio de salida. Los árboles α operan variables del dominio de los datos del mundo real utilizando operadores aritméticos, su salida es el resultado de una expresión matemática definida en el mismo dominio de entrada. La interfaz $\alpha\beta$ relaciona valores del dominio del mundo real para producir un resultado en el dominio booleano (verdadero, falso e indefinido) utilizando operadores relacionales. Finalmente el árbol β combina los hechos resultantes de las interfaces $\alpha\beta$ mediante operadores lógicos para componer la expresión del patrón a detectar, cuyo resultado es un valor también booleano. La Figura 1 ilustra la división del árbol en las tres zonas descritas.

La sección se estructura de la siguiente manera: la parte III-A se centra en el planteamiento matemático de la propuesta. En la parte III-B se expone el árbol de las reglas desde el punto de vista de la arquitectura, donde se introducen consideraciones importantes para comprender los mecanismos de optimización. Las partes III-C, III-D y III-E explican la aportación del trabajo en cuanto a los mecanismos de restricción introducidos.

A. Fundamentos matemáticos

Los sistemas basados en reglas utilizan reconocimiento de patrones para disparar reglas. Éstas pueden ser

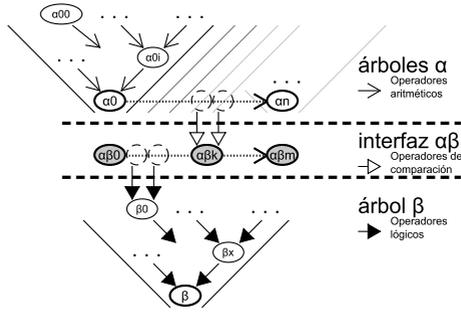


Fig. 1. Árbol α - $\alpha\beta$ - β

definidas como estructuras condicionales formadas por antecedente y consecuente; causa y efecto, respectivamente. Así, ejecutar una regla es la consecuencia de que su antecedente se ha cumplido.

Sea un antecedente una combinación lógica de hechos. Y sea un hecho una relación entre valores cuyo resultado solo puede ser verdadero, falso o indefinido (en el caso en el que falten datos para poder realizar operaciones y proporcionar un resultado). La estructura de árbol descrita en el apartado anterior satisface la definición de combinación de hechos; desde la entrada de datos hasta una compleja expresión lógica. El enfoque matemático debe empezar por proponer definiciones apropiadas para toda esta estructura, incluyendo expresiones y variables involucradas.

Definición 1. Sea Ω el dominio de los datos de entrada del sistema.

Definición 2. Sea Θ_α el conjunto de operadores cuyos operandos y resultado están definidos en Ω :

$$\Theta_\alpha = \{f : \Omega^n \rightarrow \Omega\} \quad (1)$$

Definición 3. Sea un árbol α una estructura compuesta por nodos α a los que denominaremos nodos α . Cada nodo α representa un valor definido en Ω ; y la relación de hijos a padre representa una operación definida en Θ_α , siendo los hijos, operandos y el padre, el resultado. Este árbol se puede expresar mediante una definición recursiva de nodo α , dada la posibilidad de anidamiento que permiten los operadores Θ_α al ofrecer una salida en el mismo dominio que las entradas.

$$\alpha = \omega \quad | \quad \omega \in \Omega \quad (2)$$

o

$$\alpha = \theta_\alpha(\omega_0, \omega_1, \dots, \omega_n) \quad | \quad \omega_0, \omega_1, \dots, \omega_n \in \Omega, \theta_\alpha \in \Theta_\alpha \quad (3)$$

Definición 4. Sea $\mathbb{B} = \{\text{Verdadero, Falso, Indefinido}\}$ una extensión del dominio booleano que representa los diferentes estados de una *hecho*.

Definición 5. Sea $\Theta_{\alpha\beta}$ el conjunto de operadores cuyos operandos están definidos en Ω y su resultado en \mathbb{B} :

$$\Theta_{\alpha\beta} = \{f : \Omega^n \rightarrow \mathbb{B}\} \quad (4)$$

Definición 6. Sea una interfaz $\alpha\beta$ una transición de diferentes nodos raíz de árboles α a un nodo que denominaremos β , el cual representa un hecho cuya evaluación se define en \mathbb{B} .

$$\beta = \theta_{\alpha\beta}(\omega_0, \omega_1, \dots, \omega_n) \quad | \quad \omega_0, \omega_1, \dots, \omega_n \in \Omega, \theta_{\alpha\beta} \in \Theta_{\alpha\beta} \quad (5)$$

Definición 7. Sea Θ_β el conjunto de operadores cuyos operandos y resultado están definidos en \mathbb{B} :

$$\Theta_\beta = \{f : \mathbb{B}^n \rightarrow \mathbb{B}\} \quad (6)$$

Definición 8. Dados un conjunto de árboles α evaluados en hechos mediante un conjunto de interfaces $\alpha\beta$ para dar lugar a un conjunto de nodos β , se define la estructura de árbol β mediante la aplicación de un conjunto de operadores definidos en Θ_β sobre los mismos. Al igual que en los árboles α , aparece una definición recursiva al disponer de una segunda expresión para los nodos β .

$$\beta = \theta_\beta(b_0, b_1, \dots, b_n) \quad | \quad b_0, b_1, \dots, b_n \in \mathbb{B}, \theta_\beta \in \Theta_\beta \quad (7)$$

Introducidos estos dominios y conjuntos de operadores, cabe destacar que la mínima expresión de antecedente para una regla comparte la definición de interfaz $\alpha\beta$. La definición extendida que pretende este modelo consiste en N árboles α relacionados en M interfaces $\alpha\beta$ cuyos resultados son combinados en un árbol β . En esta estructura, los datos de entrada del mundo real se modelan en los árboles α , los hechos que los relacionan se modelan

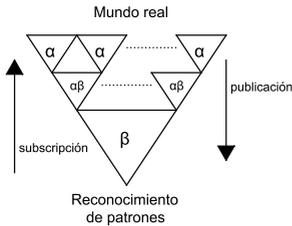


Fig. 2. Flujos de publicación / suscripción

mediante interfaces $\alpha\beta$ y el árbol β modela la expresión lógica que relaciona los hechos. Un antecedente mínimo evalúa únicamente un hecho, y un hecho mínimo depende únicamente de valores directos definidos en Ω . Por esto, la interfaz $\alpha\beta$ soporta la mínima expresión condicional posible.

Esta división conceptual del modelo se introduce para permitir la aplicación de optimizaciones en las diferentes partes del mismo. Estas optimizaciones serán descritas en secciones posteriores.

B. Distribución de los datos

A modo de resumen, podemos decir que detectar un patrón es la síntesis de datos del mundo real a un hecho simple o complejo. Entre tanto se ven involucrados dos dominios de datos y tres conjuntos de operadores. La división lógica del árbol descrita anteriormente corresponde con las etapas en las cuales el dominio de la información se transforma, dando lugar a dos tipos de nodo: nodos α y β , organizados en estructuras de árbol que reciben el mismo nombre. Estas divisiones de la estructura completa van a permitir cierto control sobre la información a lo largo de todo el proceso.

Los datos entran en el sistema en las hojas de la estructura árbol $\alpha - \alpha\beta - \beta$ y se propagan hacia la raíz. En nuestra propuesta, este tránsito responde a un mecanismo de suscripción-publicación unificado en toda la estructura, de tal manera que cualquier nodo del árbol, independientemente de su tipo, es suscriptor de sus entradas y publicador de su salida. Aplicar este enfoque permite percibir la aparición de dos procesos que van a requerir capacidad de cómputo y que surgen en direcciones opuestas en el árbol: el flujo de publicaciones desde las hojas hacia la raíz, y el flujo de suscripciones, de la raíz a las hojas.

Al hilo de este concepto se van a introducir dos políticas para la restricción de estos flujos contrapuestos. Estas políticas son máximas de comportamiento de cualquier nodo de la red, y también nos van a permitir introducir métricas del nivel de carga del sistema de manera global y relativamente simples de obtener, como se describe más adelante. Estas políticas son:

1. Cuando un nodo recibe una publicación como en-

trada, su resultado debe ser recalculado. A este proceso se le conoce como *activación*. Una vez obtenido el resultado, éste será publicado únicamente si su valor actual difiere del anterior. El enunciado de esta política presupone que el nodo hoja es nutrido con un flujo continuo de datos externos, cuyas características temporales escapan al alcance de este trabajo.

2. Si un nodo queda sin subscriptores, procederá asimismo a la desuscripción de cada una de sus entradas y detendrá el proceso de cálculo que le caracteriza. Esta operación será revertida al recibir la petición de suscripción cualquier nodo solicitante. Esta política introduce dos regímenes de funcionamiento: activo e inactivo.

El enfoque publicador/subscriptor en combinación con estas dos políticas va a aportar tres ventajas importantes. En primer lugar, distribuir en red el reconocimiento de patrones es trivial, únicamente es necesario un mecanismo de transparencia que haga equivalente el publicar datos a otro elemento de la red y consumirlos en el mismo nodo. Por otra parte, las políticas descritas permiten modificar dinámicamente la carga computacional del árbol, mediante ciertas estrategias para reducir publicaciones o desactivando partes del árbol a conveniencia. Por último, los nuevos conceptos que han surgido al hilo de esta propuesta permiten medir el estado actual del sistema; un parámetro tan simple como el tanto por ciento de nodos activos puede ser una métrica adecuada para estimar la carga de suscripción con respecto a su máximo. Por otra parte, el concepto de *activación* es interesante para controlar la actividad del flujo de publicación, considerando el número de activaciones por unidad de tiempo de un nodo concreto, parte del árbol o del sistema completo.

Abrir la puerta a este tipo de métricas nos va a permitir trabajar en estrategias más complejas y adaptadas a la situación en un futuro. Como conclusión, el mecanismo de distribución de datos en el árbol introduce dos flujos que se oponen: una demanda de datos mediante suscripción, que viene desde la raíz hacia las hojas; y un suministro de datos por publicación, desde las hojas hacia la raíz. Ambos flujos tienen una influencia clara en el número de activaciones en el árbol, lo que a su vez influye directamente en la carga del sistema (procesamiento, paquetes de red, memoria, etc). La carga de estos flujos es susceptible de ser medida y controlada local y globalmente en la propia red.

A continuación, se describen en mayor profundidad las diferentes regiones del árbol y se introducen mecanismos para tomar cierto control sobre los flujos de datos anteriormente descritos.

C. Operaciones aritmético-lógicas: árboles α

Este modelo de detección de patrones se concibe para la detección de patrones en redes de dispositivos muy limitados. En este contexto, *memoria, capacidad de cómputo, energía y ancho de banda* son parámetros que se tienen que tener en cuenta. Modelar la evaluación de los hechos -tarea externa al algoritmo RETE original- es importante para gestionar los recursos que requiere. La evaluación de un hecho es una tarea cíclica que analiza

continuamente las entradas del sistema, haciendo uso del dispositivo con mayor o menor intensidad. Por lo tanto, como parte del reconocimiento de patrones, es particularmente importante incorporarlo en el modelo.

Con este propósito, los nodos α del clásico RETE van a ser desglosados en árboles α . De acuerdo con el modelo matemático descrito en III-A, cada árbol α implementa una expresión cuyo resultado se define en el dominio Ω . De esta manera, todos los nodos α pertenecientes al mismo árbol α son los responsables de ejecutar secuencialmente los cálculos subyacentes; siendo el nodo raíz del árbol α el que calcula el resultado final, y los nodos hoja los puntos de entrada de los datos. Cada nodo de la secuencia depende de los nodos previos que necesite, operando sus resultados parciales para obtener el suyo propio. En los nodos α , tanto las entradas como el resultado están definidos en el dominio Ω , y la función a realizar es una combinación de operadores definidos en $\Theta_\alpha (\Omega^n \rightarrow \Omega)$.

Según el esquema de distribución de datos descrito en III-B, los nodos α reciben los resultados de las etapas previas del cálculo mediante subscripción. Si un nodo tiene múltiples subscripciones, su proceso se mantendrá a la espera de recibir la actualización de alguna de sus entradas. Entonces, el proceso se activará, realizará el cálculo con el nuevo valor recibido y, si el resultado difiere del último obtenido, lo publicará para que sus subscriptores hagan lo propio. Como se ha introducido en el apartado anterior, cada publicación involucra la activación en cadena de parte de la red, lo que hace muy interesante el poder restringir el flujo de activaciones convenientemente. Un mecanismo sencillo para esto se describe a continuación.

C.1 Restricción de publicaciones en nodos α

De manera externa al sistema, el mundo real se modela mediante variables que evolucionan con el tiempo y que representan información a lo que llamamos *datos*. Típicamente, los *datos* suelen provenir de un sensor, cuya medida es digitalizada en un convertidor analógico digital para obtener valores. En este proceso de obtención de información se ven involucrados algunos parámetros que influyen en la salida del convertidor. Proponemos aquí desglosar el dominio Ω para introducir algunos parámetros que caractericen la información del mundo real y que puedan ser utilizados para reducir el número de publicaciones del sistema.

Definición 9. Sea Φ el dominio de los valores de los *datos*. Y sea Δ el dominio de los parámetros que caracterizan a los *datos*

$$\Omega = \{\Phi \times \Delta\} \quad (8)$$

Definición 10. De esta manera, cada *dato* que entra en el sistema es definido por el par (valor, parámetros).

$$\omega = (\phi, \delta) \quad | \quad \omega \in \Omega, \phi \in \Phi, \delta \in \Delta \quad (9)$$

Definición 11. Dado $\omega_i = (\phi_i, \delta_i)$, sea $\text{value}(\omega_i) = \phi_i$ y $\text{param}(\omega_i) = \delta_i \forall \omega_i \in \Omega$

Definición 12. Sea $\text{evaluation}(\omega)$ la función que proporciona ϕ interpretado según los parámetros δ .

Como se ha descrito, todos los parámetros deben ser agrupados en Δ para permitir posibles extensiones al modelo. Por el momento, consideremos tres modificadores que nos van a permitir filtrar la información: *resolución* (r) and *rango*, este último definido mediante *límite inferior* y *límite superior* (ll y ul por sus siglas en inglés, respectivamente); todos ellos definidos en \mathbb{R} .

Definición 13.

$$\Delta = \mathbb{R}^3 \quad (10)$$

El método *evaluación* conlleva una limitación de la variabilidad del valor final de cada dato en consecuencia de la aplicación de estos modificadores. El algoritmo 1 es una propuesta de implementación del método con los parámetros propuestos.

Algoritmo 1 Método evaluación

```

1: procedure EVALUATION( $\phi, \delta$ ) ▷  $\phi \in \Phi, \delta \in \Delta$ 
2:   if ( $\phi < \delta[ll]$ ) then
3:     result  $\leftarrow$   $\delta[ll]$ 
4:   else if ( $\phi > \delta[ul]$ ) then
5:     result  $\leftarrow$   $\delta[ul]$ 
6:   else
7:     result  $\leftarrow$  truncate( $\frac{\phi}{\delta[r]}) \times \delta[r]$ 
8:   end if
9:   return result
10: end procedure
    
```

Como es lógico, el mecanismo de filtrado tiene una desventaja: involucra una pérdida de información por pérdida de precisión. Como en muchas situaciones, el uso del filtro responderá a una solución de compromiso entre consumo y eficiencia.

La estructura de los árboles α queda bajo el criterio del usuario. En este modelo no hay limitación para el número de subscripciones por nodo, o la complejidad de la operación. El mecanismo de filtrado por resolución y rango es una potente herramienta para aprovechar la partición del árbol α . El sentido común sugiere que es mejor tratar de evitar cálculos complejos, partiendo el árbol en fases tratando de cortar la propagación de activaciones en algún punto de la rama. De la misma manera, es lógico evitar tener que enviar paquetes de red frecuentemente, aislando cálculos remotos en una rama. Todos estos criterios sientan interesantes puntos de partida para investigaciones futuras.

D. Evaluación de hechos: transiciones $\alpha\beta$

Los datos provenientes de los nodos raíz de los árboles α se combinan en los nodos $\alpha\beta$ mediante operadores relacionales ($>, <, \leq, \geq, \dots$). Los resultados son *hechos*, de acuerdo con la definición 6. Cada nodo $\alpha\beta$ está suscrito a nodos α raíz para obtener datos definidos en Ω ; éstos son operados para obtener un valor definido en \mathbb{B} , de acuerdo con 4. Al ser estos nodos los responsables de un cambio en el dominio (Ω^n a \mathbb{B}), un nodo $\alpha\beta$ satisface por sí mismo la definición de patrón, por lo que es la mínima expresión de una regla, e.g $A > B$ donde A y B son constantes (no hay árboles α ni β). El conjunto de todos los nodos $\alpha\beta$ se denomina *interfaz* $\alpha\beta$, y engloba todos los nodos hoja del árbol β (nodos α en RETE).

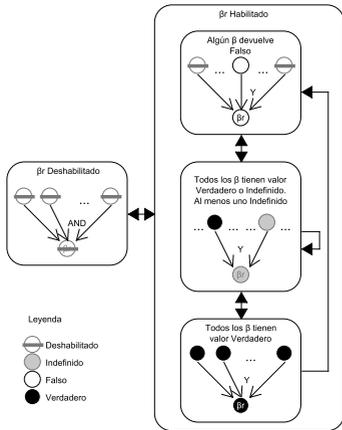


Fig. 3. Operación Y en el árbol β

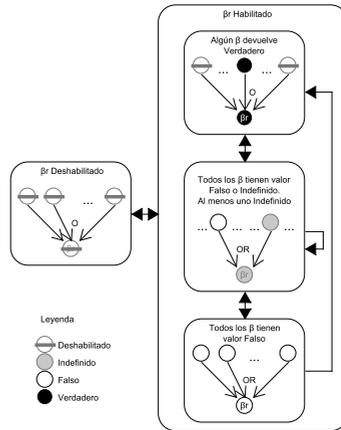


Fig. 4. Operación O en el árbol β

E. Detección de condiciones: árbol β

Toda regla tiene, al menos, un hecho. Estos es, al menos, un nodo $\alpha\beta$ comparando constantes. Si la regla tiene más de un hecho, se necesitan operadores lógicos para combinar ambos hechos en el árbol β , al igual que en RETE clásico. En este modelo, el árbol β consiste en un conjunto de nodos β , correspondiendo cada nodo β a un único operador lógico aplicado sobre los hechos de entrada. El resultado de un nodo β es definido en el dominio B , con lo que define otro hecho, tal y como el antecedente de la regla en sí mismo. A diferencia de los árboles α , los nodos en β no pueden llevar a cabo una expresión compuesta. Esto se debe a que se requiere esta limitación para poder introducir el mecanismo de optimización que se describe a continuación. Aprovechando el mecanismo de publicación/subscripción, utilizado en todo el árbol $\alpha - \alpha\beta - \beta$, los nodos β llevarán a cabo la desuscripción de aquellas de sus entradas que no tengan efecto en el resultado. La desuscripción en el árbol β tiene un efecto en cadena hacia arriba, deshabilitando árboles α completos cuyo resultado deja de ser necesario, y reduciendo drásticamente las activaciones del sistema. El modelo soporta dos operadores en los nodos β : Y y O. La negación (NO) puede ser considerada una propiedad de la subscripción, cuyas publicaciones adoptan el valor negado del resultado si se aplica. En cualquier caso no tiene efecto alguno en términos de optimización.

El dominio booleano involucra tres valores: *verdadero*, *falso* e *indefinido*. En referencia a los nodos β se debe considerar un estado más, que representa que el nodo está deshabilitado porque no tiene subscriptores. Las figuras 3 y 4 ilustran los estados y las transiciones que un nodo β puede experimentar. Ambos operadores tienen un com-

portamiento similar al compartir dos características:

1. Existe una combinación de entradas en la que únicamente una de ellas determina el resultado. Este es el mejor escenario para la optimización.
2. Cualquier otro resultado conlleva activar todas las entradas.

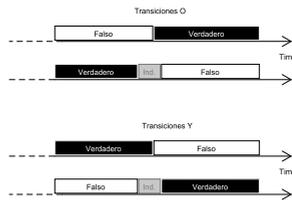


Fig. 5. Transiciones Y / Transiciones O

Como en los nodos α , limitar recursos en β produce cierta pérdida de información; en este caso en términos de tiempo. La explicación tiene que ver con el estado *indefinido*: cuando una desuscripción tiene lugar, el valor asociado a este enlace con un nodo previo pasa a ser indefinido cuando la subscripción vuelve a activarse. No así si la subscripción se hubiese mantenido activa, en cuyo caso solo habría un periodo de indefinición en el arranque del sistema. Esto genera una situación de transición ineficiente que se describe en la figura 5.

Una estrategia para mitigar este efecto podría consistir en aprovechar el procedimiento de subscripción para solicitar al nodo publicador el último valor conocido de su re-

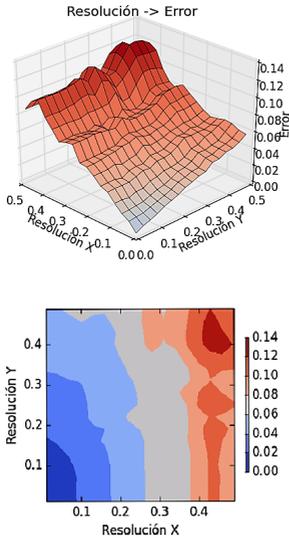


Fig. 6. Resolución vs Error

sultado. Con este sistema el tiempo de indefinición entre estados persistiría, aunque se vería reducido.

IV. EXPERIMENTACIÓN Y RESULTADOS

En este trabajo la experimentación se ha centrado en estimar el impacto del filtrado por resolución de los valores de Ω sobre la carga del sistema. Para ello se van a utilizar dos variables de entrada (x e y) que determinarán el valor de dos expresiones matemáticas: $Z(x, y)$ y $Th(x, y)$, que van a ser comparadas para evaluar un hecho: $Z(x, y) > Th(x, y)$.

$$Z(x, y) = \sin \left(2\pi \cdot \sqrt{(x+2)^2 + (y+1)^2} \right) \quad (11)$$

$$Th(x, y) = -y + 2x \quad (12)$$

En la Figura 7 se puede observar la representación de las expresiones utilizadas para todo el dominio de las señales de entrada. La imagen de la derecha corresponde a la representación de aquellos valores de Z que hacen que el hecho sea cierto (*Verdadero* en el dominio booleano).

A continuación, se van a variar los valores de resolución para las entradas x e y . Es importante tener en cuenta que la resolución en este estudio corresponde a la definición de resolución de sensor, y será expresada mediante la cuantización del mínimo incremento posible en el valor. Asimismo, es importante indicar que el plano umbral no ha sido sometido a la variación de resolución, para centrar el estudio en una única expresión y, por lo tanto, un

único árbol α .

Para cada valor de resolución de x e y se va a obtener un corte de Z con el plano Th , y éste va a ser comparado con el corte a resolución máxima, considerado libre de errores. Aquellos valores del dominio Z que, como consecuencia de la pérdida de información en el filtrado por resolución de x e y , devengan en una evaluación incorrecta del hecho, serán considerados erróneos. Al disponer de un dominio acotado y completo, podemos introducir la medición del error total calculando cuántas combinaciones de x e y , de todas las posibles, introducen un error en el hecho evaluado.

La Figura 6 presenta los datos obtenidos en el estudio descrito. El resultado tiene una lectura interesante: como es deducible por la expresión matemática y, en menor medida, por el ángulo de incidencia del plano de corte sobre la superficie Z ; la variación en las variables de entrada no tienen el mismo impacto sobre el valor de salida. En las cuatro Figuras se puede observar cómo el valor de x tiene una mayor incidencia sobre el error.

Independientemente de este detalle, lo que demuestra este estudio es que el mecanismo de filtrado por resolución permite diseñar una estrategia que combine criterios de consumo y eficacia. Tomando, por ejemplo, como error asumible uno inferior al 4 %, podríamos trabajar con una resolución de 0,1 para x y de 0,25 para y , lo que produciría una respuesta que se ilustra en la Figura 8.

Una vez relacionada la resolución con el error cometido, es necesario poder hacer lo mismo con la carga del sistema para poder cuantificar el ahorro que supone la aplicación de la técnica. Como se introdujo en la sección III-B, proponemos utilizar el número de activaciones como un parámetro proporcional a la carga del sistema.

Medir las activaciones requiere la inyección de señales de entrada reales durante un periodo concreto. Para esta parte se han tomado los valores tomados por un sensor de temperatura y otro de humedad relativa, normalizando los datos para el rango $[-1, 1]$. Para cada valor de resolución se ha contado cuántas activaciones producen ambas señales en el cálculo del hecho objeto de experimentación. Los resultados se ilustran en la Figura 9.

Combinando los resultados de ambos experimentos, podemos afirmar que sin utilizar el mecanismo de resolución vamos a disponer de un resultado de la máxima calidad posible (0 % de error según nuestro criterio) y los nodos de cálculo del árbol α sumarán aproximadamente 2400 activaciones. Sin embargo, a modo de ejemplo, utilizando una resolución de 0,1 para x y de 0,25 para y vamos a obtener una respuesta con 3,75 % de los puntos del dominio con valor incorrecto (estos errores se distribuyen en la frontera según el patrón que se muestra en la Figura 8), y los nodos del árbol sumarán entre 300 y 600 activaciones. Asumiendo que a máxima resolución, máxima carga del sistema, la reducción conseguida siguiendo este criterio asciende a un 75 %.

Debido a la complejidad y amplitud de los requisitos experimentales necesarios para una demostración empírica de la optimización por desactivación de ramas del árbol β , no se ha podido abarcar en este trabajo, quedando pendiente para futuras investigaciones.

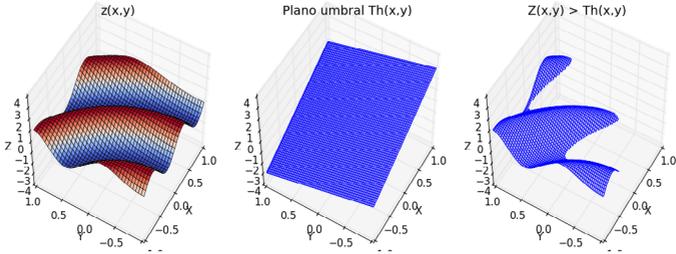


Fig. 7. Superficie Z, plano de corte Th y comparación

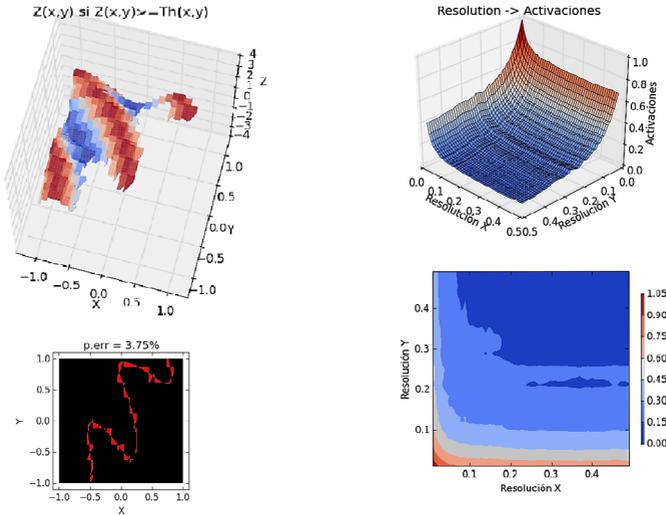


Fig. 8. Rx = 0.1, Ry = 0.25

Fig. 9. Resolución vs Activaciones

V. CONCLUSIONES

Los experimentos realizados demuestran que el cambio de resolución tiene una afectación en el resultado susceptible de ser medida y analizada. El análisis del dominio utilizado hace evidente la posibilidad de balancear la resolución de las entradas en función de un criterio de compromiso entre el ahorro de recursos y precisión en el resultado.

El modelo propuesto permite adaptar la carga de un sistema distribuido de nodos gracias al ajuste de los parámetros modificadores de las variables de entrada. Estos parámetros permiten acotar el error con el que se evalúan los hechos y, por otro lado, limitar el número de comunicaciones entre los nodos de la red y activaciones de pro-

cesos de evaluación y cómputo.

Por las características descritas, este trabajo sienta las bases científicas y técnicas que van permitir la implementación de algoritmos de reconocimiento de patrones y disparo de reglas en redes de dispositivos que presenten altas restricciones computacionales.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado mediante los proyectos P11-TIC-7462 de la Junta de Andalucía (Monitorización Inalámbrica de Pacientes en Instalaciones Hospitalarias) y DPI2013-47347-C2-2-R del Ministerio de Economía y Competitividad (Redes de Comunicación Optimizadas para el Sensado y Control en Entornos Inteligentes).

REFERENCIAS

- [1] Charles L. Forgy. "Rete: A fast algorithm for the many pattern/many object pattern match problem." 1982.
- [2] Costin Bdic, Lars Braubach, and Adrian Paschke. "Rule-based distributed and agent systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6826 LNCS, pp. 3–28, 2011.
- [3] Jörg P. Müller. "Architectures and applications of intelligent agents: A survey," *The Knowledge Engineering Review*, vol. 13, no. 4, pp. 353–380, 1999.
- [4] Meng Ma, Ping Wang, and Chao-Hsien Chu. "Data Management for Internet of Things: Challenges, Approaches and Opportunities," *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pp. 1144–1151, 2013.
- [5] S M Riazul Islam, Daehan Kwak, and Humaan Kabir. "The Internet of Things for Health Care : A Comprehensive Survey," vol. 3, 2015.
- [6] I. Strategy and P. Unit. "The Internet of Things," Tech. Rep., International Telecommunication Union, 2005.
- [7] Mark Weiser. "Ubiquitous computing," *Computer*, vol. 26, no. 10, pp. 71–72, 1993.
- [8] Priyanka Rawat, Kamal Deep Singh, Hakima Chaouchi, and Jean Marie Bonnin. "Wireless sensor networks: a survey on recent developments and potential synergies," *The Journal of Supercomputing*, vol. 68, no. 1, pp. 1–48, oct 2013.
- [9] Daniel I Miranker. "TREAT: A Better Match Algorithm for AI Production," in *AAAI-87*, 1987, pp. 42–47.
- [10] Ian Wright and James Marshall. "The execution kernel of RC++: RETE", a faster RETE with TREAT as a special case," *International Journal of Intelligent Games and Simulation*, vol. 2(1), pp. 36–48, 2003.
- [11] Di Liu, Tao Gu, and Jiang Ping Xue. "Rule engine based on improvement rete algorithm," *2010 International Conference on Perceiving Computing and Intelligence Analysis, ICACIA 2010 - Proceeding*, pp. 346–349, 2010.
- [12] Ding Xiao and Xiaon Zhong. "Improving rete algorithm to enhance performance of rule engine systems," *2010 International Conference on Computer Design and Applications, ICCDA 2010*, vol. 3, no. Iccda, pp. 572–575, 2010.
- [13] Milhan Kim, Kiseong Lee, Youngmin Kim, Taejin Kim, Yunseong Lee, Sungrae Cho, and Chan Gun Lee. "RETE-ADH: An improvement to RETE for composite context-aware service," *International Journal of Distributed Sensor Networks*, vol. 2014, 2014.
- [14] Neil Madden. "Optimising RETE for Low-Memory Multiagent Systems," *Proceedings of Game-On*, 2003.
- [15] Rui Zhou, Guowei Wang, Jinghan Wang, and Jing Li. "RUNES II: A Distributed Rule Engine Based on Rete Network in Cloud Computing," *International Journal of Grid and Distributed Computing*, vol. 7, no. 6, pp. 91–110, dec 2014.

Merging WSN and IoT Technologies towards Smart Urban Networking

Gabriel Mujica, Jorge Portilla, Teresa Riesgo

Abstract—The progressive integration of Wireless Sensor Network technologies into the ongoing evolution of IoT-based system implementations has envisioned the development of efficient hardware and software techniques to improve the performance of low-power embedded platforms into real smart distributed systems. This is particularly noticeable when addressing wireless communication strategies for data dissemination and high-level networking interfaces to efficiently exchange application content among the participatory objects. Moreover, although during the last years there has been an important maturity process of HW-SW platforms suitable for WSN-based systems, actually such a broad spectrum of possibilities contrasts with the intrinsic increase of complexity in terms of integration strategies that come out with the development of a wireless distributed system. In this work, current trends regarding efficient communication techniques for IoT and smart urban scenarios together with middleware technologies for heterogeneous and cross-connectivity among different platforms are explored, so that the main baselines to guaranty a proper generation of suitable frameworks for the future IoT can be enhanced. This analysis also allows introducing in this work a novel approach towards an urban and home automation management system, in which the combination of a modular and flexible HW-SW platform with a dynamic IoT middleware architecture is addressed. Such a proposal aims at producing a comprehensive framework for the integration of WSN technologies and smart automation capabilities.

Keywords— Wireless Sensor Networks, IoT Middleware, Smart Objects, HW-SW co-design

I. INTRODUCTION

THE progressive adoption and integration of Wireless Sensor Network technologies into ongoing IoT-based system implementations has also envisioned the evolution of novel hardware and software techniques to improve their actual performance in real smart city application scenarios. In this way, three main technological lines can be distinguished within the innovation wave of smart city objects and urban pervasive-participatory sensing. Firstly, connectivity optimization strategies and communication protocols represent the main target to produce suitable mechanisms to establish the wireless cooperation among the deployed smart objects. Secondly, hardware platforms to provide low-power highly integrated devices upon which the embedded software technologies are to be implemented, considering low-level

controllers for efficient management of the platform processing/peripheral components. Third, software and specially middleware technologies for both high-level implementation of IoT functionalities and cross-correlation among heterogeneous platforms, so as to provide a proper cohesion and interoperability of various types of technologies.

Although during the last years there has been an important maturity process of HW-SW platforms suitable for WSN-based systems, actually such a broad spectrum of possibilities contrasts with the intrinsic increase of complexity in terms of integration strategies that come out with the development of a wireless distributed system, particularly taking into account the evolved requirements associated with the IoT principles to be addressed. This is the main reason why different attempts to establish de-facto standards have been carried out by the research and industrial community, mainly allowing for wireless connectivity technologies. To this extent, it is noticeable to remark the evolution of communication methods around one of the main reference protocols within the WSNs: the IEEE 802.15.4 MAC and Physical specification [1].

Even though this standard has been introduced years ago when the WSN paradigm was in its first stage of growing, what distinguishes its continuous suitability for today's era of smart sensor devices derives not only in the low-power and low-complexity nature of the radio technology but also in the sense that it defines the underlying layers that serves as the main baseline to build upper communication protocols to fit the challenges of power-profile modes, low data transmission rate and packet dissemination efficiency. This promoted the creation of several initiatives to implement upper technologies from the basis of this PHY-MAC definition, such as the well-known ZigBee Alliance, which proposed the network and application layers on top of IEEE 802.15.4 to establish and maintain low-power wireless personal area networks [2].

However, the tendency to move WSN technologies into a global networking scenario produces the adaptation of already available platforms to the concept of IP-based IoT systems, either by strategically including smart gateways among heterogeneous networks for content processing, provision and dissemination bridging, or by directly integrating optimized internet-like protocols into low-rate communication platforms [3]. While the former solution tries to concentrate the complexity of the interoperability mechanisms in specific enhanced elements so that already established deployment platforms can be included much smoothly, the latter approach indeed implies further new challenges that the research

Centro de Electrónica Industrial, Universidad Politécnica de Madrid, José Gutiérrez Abascal 2, 28006, Madrid, Spain (tf.: +34913363191; e-mail: gabriel.mujica@upm.es, jorge.portilla@upm.es, teresa.riesgo@upm.es).

community has been trying to tackle seeking a proper WSN-IoT coupling.

In this work different approaches to face the evolution of communication techniques for IoT and smart urban scenarios are explored in combination with middleware technologies for heterogeneous and cross-connectivity capabilities among different platforms, so that the main baselines to guaranty a proper generation of suitable frameworks for the future IoT can be highlighted. This also allows introducing a novel approach towards urban and home automation management system, in which a modular and flexible HW-SW platform is merged with a dynamic IoT middleware architecture attempting to produce a more comprehensive framework to integrate WSN technologies and IoT smart automation capabilities.

II. WSN TECHNOLOGIES TOWARDS IOT-BASED IMPLEMENTATIONS

Within the context of the aforementioned technological movements, the 6LoWPAN proposal attempts to answer the question regarding how resource and bandwidth constrained devices can operate over the IPv6-based wireless infrastructures [3], taking into account that IPv6 has been selected as the main standard protocol to address the huge amount of devices that will take part in this IoT global connectivity scenario [4]. This is certainly not a trivial task since traditional communication protocols and particularly IPv6 make use of important address spaces as well as control packets that are not well-suited for typical WSN communications. Moreover, overheads also introduce more incompatibility of the capabilities of sensor nodes with respect to the associated data flow on internet communication. The main idea behind 6LoWPAN relies on a header compression and encapsulation of IPv6 packets to be transferred over low-rate constrained platforms. It is particularly intended to be integrated with IEEE 802.15.4, thus special attention are paid to comply with maximum available MTU (Maximum Transmission Unit). In fact, since the maximum frame size available in IEEE 802.15.4 is 127 bytes (which is in practice reduced by the subsequent headers included in upper implementation layers) and the minimum MTU in IPv6 is 1280 bytes, one of the main features of 6LoWPAN is the adaptation of the packet frames by implementing a segmentation layer on top of the MAC layer. By applying a stateless header compression for IPv6 datagrams [5] headers can be reduced to 4 bytes.

The main element to interface the internet applications to the LoWPAN is the so-called border routers [6], which are in charge of translating the IPv6 traffic from the unconstrained network to the low-rate resource-constrained devices, by providing the compression and neighbour discovery mechanisms to optimize the data exchange throughout the distributed system. Since every sensor node has an IPv6 ID for sending and receiving internet packets, the border routers are also responsible of converting the full IPv6 addresses to the

6LoWPAN format, whose transaction is completely transparent to both sides of the communication areas.

On the other hand, since the urban dynamic communication among the smart devices is certainly focused on a mesh networking array, the multi-hop nature of the WSNs is to be taken into account, so that nodes that participate in the LoWPAN can collaborate in transferring the information by using optimized routing strategies. For this, the Routing over Low-Power and Lossy Links Working Group (RoLL), proposed the creation of a Routing Protocol specification for Low power and lossy networks (RPL) within the context of 6LoWPAN networks [7]. The main target was to provide an alternative routing mechanism to the available protocols in order to adapt the multi-hop capabilities to the requirements of the 6LoWPAN specifications. RPL aims at providing efficient multi-hop paths between the sensor nodes and the root points of the network (e.g., the border routers of the 6LoWPAN), in which three main traffic patterns are provided: point-to-point communication, point-to-multi-point, and multi-point-to-point. RPL protocol relies on a distance-vector strategy with which a destination-oriented acyclic graph is created, so that the most optimized path is selected in accordance with constrained metrics such as hop count or link quality indicators.

As a result of such an adaptation of the low-level layers allowing for the physical and medium access control protocols for low-power sensor networks to be integrated into internet communications, open IP-based standards, protocols and implementations can be used in urban mesh networks so that an interrelation between services and smart objects is properly carried out. This means that protocols such as UDP, MQTT and HTTP-like mechanisms may be adopted for the high-level data exchange throughout the networks. However, current trends are certainly focused on the operability and usability of Web Service technologies within the IoT systems, considering object-oriented specifications such as SOAP [8] (Simple Object Access Protocol), which defines the rules for the data exchange based on XML, or REST (Representational State Transfer) [9], which has become a clear alternative to SOAP due to its simplicity, flexibility and ease of implementation based on HTTP for managing, accessing and exchanging system resources.

Actually, REST is getting an important reception within the IoT paradigm as a suitable service technology to facilitate the interoperation of heterogeneous applications. However, in order to map these technologies into the resource-constrained devices, important efforts have been made to create mirrored protocol stacks for smart objects to be integrated with the Web. In this context, CoAP (Constrained Application Protocol) [10] has been conceived to take advantage of REST attributes in order to integrate Web Services capabilities into low-power devices, though overcoming the limitations in terms of network resource usage and format space that HTTP/REST implies. In order to do that, CoAP uses a very simple binary message layer over UDP, which defines the typical request methods as GET, PUT, POST and DELETE. This allows a better interaction with Web Services by using intermediary HTTP-CoAP cross-proxies, which are in charge

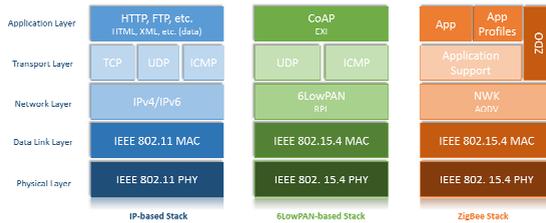


Fig.1. Comparison of IP-based LowPAN, ZigBee stack and Internet technologies.

of performing the correspondence between Web requests/responses and CoAP services. Since methods, response codes and options are present in both mechanisms, the mapping through these protocols can be seamlessly achieved [10], also considering that both CoAP and HTTP use resource identifiers (URIs) (though in case of CoAP they are deployed as *soap://*, so the reverse proxy performs the corresponding conversion). Being aware of this fast evolution of WSN IoT suitable technologies targeting IP-based strategies, ZigBee Alliance also adapted its specification in order to evolve around such changes by introducing a ZigBee IP profile where a 6LoWPAN adaptation layer and related features are used, adding network and security capabilities apart from an application framework [11].

Besides this technological movement towards the integration of low-rate communication techniques with traditional internet protocols and open web approaches by creating novel specifications and standards to adapt the resource and power constraints of autonomous sensory systems, another important trend that has been progressively growing lies on the adaptation of the smart objects to a new evolution of the wireless local area networks, i.e., through IEEE 802.11. In this direction, two main approaches that contribute to IoT implementations can be highlighted: on one hand ultra-low power Wi-Fi modules come out as a clear alternative to produce a good trade-off between communication performance and power consumption profile, especially aimed at its integration in embedded platforms. Manufacturers offer chipsets with less than 4uA of power consumption in sleep modes, programmable awake configurations and power management capabilities, which represents a very attractive proposal to integrate communication modules into hardware platforms for IoT, although high picks in transmission and reception modes are to be pondered within the overall consumption characterization as well.

In fact, Wi-Fi Alliance recently released Wi-Fi HaLow [12], which is the designation for products that integrate IEEE 802.11ah, specially intended to IoT and smart city environments. The main contributions of the IEEE 802.11ah relies on the extension of the connectivity ranges by using sub-1GHz frequency bands, in addition to defining saving power modes and wake time configuration schemes.

On the other hand, following the approach of mesh networking strategies for urban distributed systems, the IEEE 802.11s standard was created to tackle the lack of support for mesh networks in the current implementations of the specification, particularly to provide capabilities for ad-hoc and WLAN mesh network interconnections [13]. The main objective focuses on the interoperation of BSS (Basic Service Set, which is commonly integrated by access point and base stations in infrastructure mode) which means mesh connectivity among them, also enabling the integration of a new type of BSS, a mesh BSS (MBSS). Efforts such as open80211s [14] attempt to provide open implementations of the IEEE 802.11s to be integrated in hardware platforms that support, in that case, Linux kernel, or actual implementations into HW-SW platforms such as in [15]. Figure 1 presents a brief comparison of the different communication stacks that are considered in traditional internet, IoT and WSN technologies.

III. MIDDLEWARE-BASED IOT SOLUTIONS FOR HETEROGENEOUS PLATFORM INTEGRATION

Apart from the wireless communication technologies and associated protocols that are being adopted to foster IoT implementations into urban networks, a key element to be also taken into account is the diversity in terms of HW-SW platforms as well as the applications and services to be offered. WSNs and IoT pose important challenges related to technology heterogeneity, particularly considering embedded systems for the provision of smart objects and their participatory interaction with the target application. In order to create a proper synergy between hardware platforms and service/application developments, middlewares for IoT take place as a fundamental role in order to bring interoperable and reusable solutions throughout different IoT systems, in addition to the reference objective related to provide an efficient abstraction layer for low-level dependencies and platform internal architectures. The importance of middleware technologies goes beyond the integration of network communication capabilities, since operating systems and software layers for system development on top of resource constrained architectures are to be also addressed [16].

However, the inherent challenges related to the design, implementation and usability of middleware for IoT

encompass both functional and non-functional requirements to assure a minimum degree of successfulness when deploying and releasing the target set of applications. In terms of non-functional objectives, apart from the already mentioned intrinsic **heterogeneity** nature of the IoT, **scalability and extensibility** are one of the main concerns when tackling the development of such distributed systems. In this direction, middleware for IoT are to be capable of managing the growth and expansion of the network as well as adapting to the inclusion of new devices/services. A proper decoupling of low-level dependencies and underlying platform complexity shall smooth the cross-integration among applications and new participatory elements. Another important aspect to be considered lies on the **reliability** of the system under operation. Since failures and temporal malfunctions of part of the components/objects that compose the distributed system may be present during the operational phase of the overall platform, the middleware needs to be able to confront such situations by contemplating fault tolerance strategies, particularly in terms of communication errors, computational effort handling and energy management/monitoring. **Reusability** is also a key feature that allows the combination of already existing technologies with novel components and services implementation. To this extend, IoT middleware shall not exclusively attempt to create a completely new platform composition but integrate available software entities in such a way that an interoperable environment can be efficient run throughout the overall system. The ease of **development** together with a proper **updating** degree shall also represent the foundations of IoT middlewares, seeking a trade-off between usability and continuous evolution of the platform. To cope with this feature, middleware technologies are commonly provided as a set of frameworks so as to speed up the development and integration functional components, in addition to enhancing the ease-of-implementation of user application services at system-level.

Regarding key functional requirements to be met when approaching middleware capabilities for WSN and IoT networks, automatic resource discovery and management can be distinguished as basic features to be addressed due to the dynamic and heterogeneous nature of the sensor network. This is particularly relevant in distributed systems where resource publication and device announcement are to be performed without central or dedicated points. Apart from node discovery mechanisms, resource management when a smart object is included to participate in the urban network shall be also taken into account, including efficient strategies to handle the associated data acquisition and processing, so that usefulness of the available resources is enhanced accordingly. Service composition is another important feature that IoT middleware technologies need to approach, in which connections of functional components is realized in accordance with the input parameters so as to produce the desired output models, i.e., for a given description of the system and requested services, functional block interfaces are correlated. In this way, the integration of components may follow a wrapper-based model approach upon which properties and configurations of

provided functionalities can be accessed. Event monitoring and processing encompasses efficient awareness and management of application and environment events that may appear during the operation phase of the sensor network, for which the middleware shall be capable of handling real-time events such that data associated to them can be useful to the overall objective of the deployed distributed system. Due to reconfigurable and updating capabilities of the IoT smart objects, services, and applications are to be addressed as a combination of modular codes generated and allocated into the sensor platforms, so that runtime efficient reprogramming can be accomplished at runtime. This approach encompasses three main strategies: component updating/replacement tasks, code dissemination and collaborative reprogramming, and memory and resource management for transparent and efficient reconfiguration and relocation. System updating/reprogramming can be also addressed from the point of view of component correlation modification, where connections among the involved functional blocks can be reconfigured.

Based on these guidelines and overall features that define the basic underlying capabilities of middleware technologies, there are different approaches in the literature that determine the type of model to be adopted. Although different classifications can be found to depict existing middleware solutions, various groups can be distinguished and highlighted as the baseline for IoT implementations: Service-oriented, event-based, message-oriented, virtual-machine-based, and database-oriented approaches. Moreover, combination of these models to create hybrid solutions can be also found, as well as application-specific approaches.

Service-oriented middleware is one of the main focuses when approaching the implementation of IoT functionalities, in which things (including sensing and actuating capabilities, etc.) are abstracted as software services provided by distributed nodes. Seeking the described integration of pervasive sensor networks with internet communication and the Web, the concept of Sensor-Web has been supported by the creation of the Sensor Web Enablement (SWE) framework [17], which is mainly intended to standardize the discovery, access and use of sensors and transducers in the form of web service interfaces including the provision of protocols, communication and modelling capabilities. Other initiatives related to service-oriented approaches can be distinguished, such as SOCRADES [18] which uses Device Profile for Web Services (DPWS) to abstract sensor platforms as application and device services; and MUSIC [19] which is primarily focused on a self-adaptive component-based architecture in order to provide an efficient adaptation to dynamic changes in application/service contexts.

Event-based middleware relies on the definition of states which then allows the subscription to events in order to receive information when changes appears in the parameters associated to the states, i.e., publishers provide events to those subscribers that are interested in component states. When a client desires to be notified about certain events, it registers an event listener and then information will be delivered whenever

an event occurs. It can be seen that the message-oriented approach is an event-driven process, where attributes and topic registration and requests are carried out by using pub/sub message exchanging as the event bus. For instance, Mires [20] implements a basic publish/subscribe service mechanism on TinyOS taking advantage of its event-based nature. Hermes [21] uses a type and attribute based publish/subscribe scheme in order to make simpler and efficient the interest and filtering of diverse events, in addition to providing routing algorithms for scalability purposes.

Virtual-machine approaches focus on the provision of a VM environment for application development to users as well as infrastructure virtualization, so as to meet requirements related to high-level programming abstraction and transparency among platforms. Mate [22] is one example of VM integration into WSN, which has been conceived to be built on TinyOS and contemplates VM application updates, although multi-application loading into single nodes is not supported. On the other hand, in database-oriented middlewares sensor nodes are seen as virtual relational databases, where queries are used to retrieve information from the distributed sensor network. This means that the overall network is viewed as a database system. For instance, Cougar [23] and TinyDB [24] use SQL-like language to perform operations and transactions on the virtual database.

In fig. 2 a basic layer and abstraction scheme for a middleware architecture is shown, which allows decoupling of the low-level device capabilities with the top level user applications access, control and monitoring, as well as cross-correlation among heterogeneous platforms.

IV. A NEW APPROACH TOWARDS IOT URBAN AUTOMATION SYSTEMS

In line with the described innovation trends, current requirements related to urban networking and challenges associated with the integration of WSN techniques into the

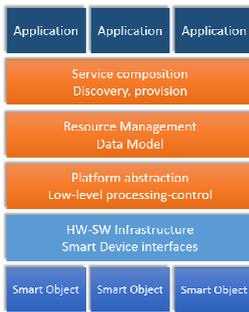


Fig.2. General Perspective of a middleware architecture for WSN-IoT.

IoT-based evolution wave, in this work the experimental experience upon a new approach for urban automation is used to propose the integration of a flexible and modular WSN HW-SW platform with a powerful solution for IoT. The basis of this system implementation lies on the use of an open-source integration platform for gathering the diversity of different vendor solutions and platform heterogeneity into an IoT middleware architecture. OpenHAB [25] has been conceived to meet requirements related to the interoperability and technology synergies into a hardware/protocol transparent support solution upon which different agents can be involved to create a comprehensive automation environment. Though specially focused on home automation systems, OpenHAB features can be extrapolated to a variety of urban networking cases where novel smart objects are to be seamless integrated. The open-source nature of the project, extensibility and its ease of integration broaden the continuous evolution and development maturity of the system to cover a wide range of different technologies and communication strategies into a well-defined IoT framework. As in middleware architectural conceptions, OpenHAB does not focus on replacing manufacturer software capabilities, but instead offers an integration environment for correlated heterogeneous technologies in such a way that an efficient management of the overall system operation is better achieved. This is particularly important considering the fast growing of technologies that come out with the IoT evolution process, event more from the user point of view, where the access and consumption of system services is to be enhanced.

OpenHAB defines as a basic principle the notion of Items, which represents the elemental abstraction for functionalities and capabilities. This produces a totally transparent use of an Item without being aware of implementation dependencies, i.e., no references to the underlying technology (protocol access, hardware interface, etc.) that implements the control of an attribute/parameter (for instance, temperature sensor) will appear at the Item-level management. This indeed fosters a straightforward reusability and replacement of low-level components without changes at the application-user abstraction layer. Likewise, in the other side of the system architecture the concept of *Bindings* is applied to implement platform, services and communication interfaces with the smart objects and *things*. OpenHAB leverages the integration of OSGi as the core runtime architecture of the middleware platform, which produces a pure Java-based modular solution.

The OSGi Alliance [26] has defined a comprehensive Java-based dynamic component model system that allows a modular creation and deployment of functional components, called Bundles, as well as the interaction rules that determines how bundles will be connected. It has been created in such a modular way that bundles can be remotely installed, reprogrammed and deleted at runtime without affecting the overall system, i.e., without been necessary to bring down the whole application. Apart from the Bundles (that are the software components to be developed by users/developers) OSGi specification defines the Service layer which determines how bundles are dynamically connected, and the Modules

layer which establishes the rules for a bundle to export/import code, whereas the Life-Cycle layer provides the API to dynamically install, start, stop, update and uninstall bundles [26].

OpenHAB creates bindings from the basis of OSGi bundles and provides the OpenHAB Event Bus, which allows items to transparently access the services provided by the component implementation, in addition to performing a decoupling communication among bindings. This means that in order to avoid low-level dependencies among components, when events or actions are performed, bundles shall notify other involved bundles via the common communication bus, so that a proper cross-connectivity is achieved. Although this can be successfully applied to stateless services, OpenHAB also implements an Item Repository in order to keep track of the state of the Items. In this way, two main events can be distinguished within the Event Bus operation. On one hand, commands allow triggering particular actions or state changing in a binding and thus in a device, and on the other hand state updates allow properly notifying such a modification to the involved actors. These interactions are supported by a pub/sub mechanism from the basis of the *OSGi EventAdmin* service and associated event delivery strategies.

In fig. 3, a general overview related to how connections among bindings and applications by using Items and Event Bus approaches is represented. Based on the architecture definition of OpenHAB and OSGi, three main levels of user/developer interactions with the middleware can be identified, as depicted in fig. 4. First, user services, which corresponds to the top-level application, and relies on the GUI from which users can interact with the services and the deployed distributed systems, by receiving information about the subscribed items and perform actions on them. The second level refers to the application development, in which users can create and customize their own automation system, by making use of available bindings to create and configure Items as well as OpenHAB runtime environment. For this, openHAB designer allows an ease edition of configuration files such as .items,

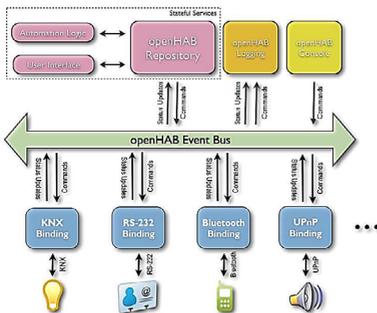


Fig.3. OpenHAB EventBus and Item-Bindings interactions [25].

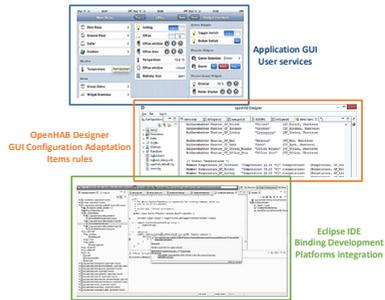


Fig.4. Three different levels of User/developer interaction.

sitemap, cfg and .rules, so that the high-level implementation of the target application can be customized, including the GUI that will be visualized at the user side. The third interaction level lies on the proper development and integration of platforms, technologies and services by using the OpenHAB binding structure, i.e., developers can create their modular components and integrate them in the system architecture. This is carried out by using the OpenHAB development framework under Eclipse IDE, which allows a fast prototyping and debugging by using well-defined interfaces for Event bus communication, command and actions management, and bundle configuration. After validation, bindings are exported and encapsulated as add-ons in the runtime environment, which then are dynamically initialized and started.

In fig. 5, the creation and interaction flow of a developed binding is shown, in which the aforementioned structure is highlighted. Bindings contain the interface and implementation specifics of the technology to be integrated. This can encompass communication protocols, data acquisition and management services, and processing strategies for computational tasks, up to the inclusion of optimization algorithms and system modelling capabilities. The successfulness in their inclusion into the overall runtime system will certainly depend on the development of the component as well as its suitable integration in accordance with the communication and cross-connectivity paradigm previously described.

V. A PRACTICAL INTEGRATION WITH THE WSN MODULAR PLATFORM

Taking advantage of the OSGi and OpenHAB system environment, in this work the integration of this technology with a modular HW-SW platform for WSNs, called Cookies [27] is proposed. The main target of this approach focuses on merging already existing technologies that are present in OpenHAB in the form of available Bindings as well as the runtime environment with the development of new modular components, so as to create an embedded management system



Fig.5. Generation and activation of a modular Binding.

for an urban automation network, from which users can not only receive relevant information of the state of different services but also actively interact with the system when needed. In order to accomplish this, a combination of low-rate and low-power wireless sensor devices and IP-based elements is realized, as shown in fig. 6 where the overall view of the proposed system is highlighted. The kernel of the system is an embedded hardware platform upon which the OpenHAB runtime core is included. For this, a Raspberry Pi B2+ is considered as the central management component while the correlation with the sensor/actuator objects is realized by using three main communication approaches, as follows:

- Wi-Fi based wireless communication: this encompasses binding external interfaces with smart devices in order to control, for instance, the state of home/urban powered-elements.
- Wi-Fi/USB virtual interface to the WSN: the connection of the management elements and bundle components to the wireless distributed sensory systems based on Cookie nodes is performed following two main concepts. On one hand, a WSN Network Interface can be included from the basis of a serial connection between the embedded platform and a Cookie node (see fig. 6). The later provides direct data access to the sensor network by encapsulating

communication control dependencies and hardware details. In this way, the Cookie platform integrates various types of WSN communication protocols such as ZigBee stack and IEEE 802.15.4-based connectivity strategies. On the other hand, a low-power-Wi-Fi-to-Low-rate Cookie Node Adapter allows a seamless integration of the WSN data gathering with the embedded platform, in which a virtual interface is created to establish a direct connection with the sink node.

- AP connection for internal-external data accessibility: while the internal communication among participant agents can be carried out from the basis of Ad-Hoc networking scheme, an AP-based infrastructure suites well for external remote access to the smart management system, so that users have both options to monitor the behaviour of the overall application and related services.

In this direction, the system definition is composed of four main add-ons in line with the aforementioned interface definitions and encompassing the integration of various types of smart objects:

- *_SmartPlug_Binding_*: includes the integration of vendor interfaces to access the services of Smart Plugs such as [28]. Main capabilities to control this smart devices lies on switching Items to manage the state of the plug in relation with events on light and temperature sensor measurements, as well as timing windows to configure the operational duty cycles.
- *_PubSub_Binding_*: encompasses the reusability of binding functionalities already included in OpenHAB such as the MQTT client [29] to configure and establish publish-subscribe mechanisms with a distributed broker.
- *_CookieServiceInface_Binding_*: implements the Cookie service management, considering the configuration of platform and sensing measurement attributes as well as

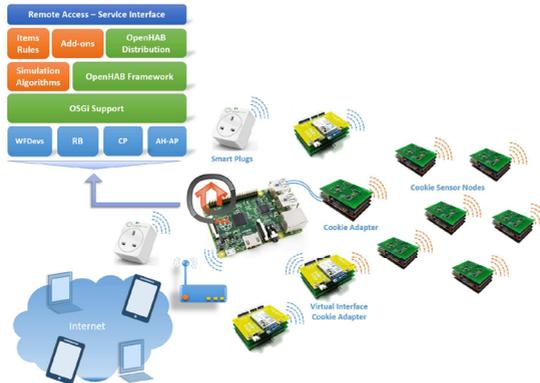


Fig.6. General Overview of the proposed system implementation.

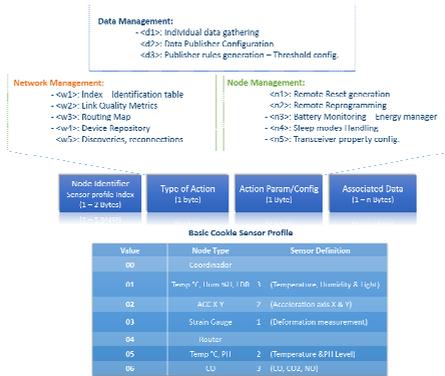


Fig.7. Control frame and management interface for remote accessing and data aggregation of the Cookie-based sensor nodes.

topics subscription, including network event handling and WSN data aggregation.

- *CookieVirtualInface_Binding*: establish the virtual interface with the sensor network through the configuration of the Cookie node adapter connection.

The service provision of the Cookie-based sensor network is realized by considering a light-weight data and control frame format as well as sensor/peripherals repository, which allow the configuration of sensor data publication, attributes monitoring and properties configuration. Fig. 7 highlights common parameters that are considered within the implementation of Cookie WSNs, including some of the available hardware and software controllers to manage the acquisition of modular sensor layers that are already implemented and integrated in the Cookie platform. The repository includes reference identifiers for the type of parameters and the sensor classification profile.

Merging the flexibility and modularity of the Cookie platform with the high degree of heterogeneous and multi-domain integration of the OpenHAB framework can leverage an efficient ease-to-use interoperable support system suitable for addressing IoT urban networking scenarios. However, some challenges are to be faced to guarantee a proper operation and adaptability towards large-scale target application contexts. In particular, distributed binding's management might be considered to assure a better performance in denser environments where the number of participatory objects can grow considerably. A clustering configuration scheme can provide a trade-off between distributed processing in the embedded management platforms and efficient data aggregation/dissemination throughout the network.

VI. CONCLUSIONS AND FUTURE PERSPECTIVES

The evolution of hardware and software platforms towards efficient pervasive sensory environments has produced research initiatives to progressively merge the WSN and IoT paradigms for smart urban application scenarios. In this work, several heterogeneous technologies envisioned to be established as the baseline for the future of the wireless distributed systems have been presented, paying special attention, on one hand, to communication protocols and specifications for IP-based sensor connectivity, and on the other hand middleware capabilities for the integration and interoperability of different WSN smart objects. In this direction, the flexibility and modularity of the Cookie HW-SW platforms has been combined with the system architecture of the OpenHAB framework in order to propose a new approach to smart urban and home automation systems. Important challenges are to be addressed for the success of this technological evolution into an efficient IoT future global scenario, mainly considering attributes related to scalability, reliability, seamless integration, and communication and data dissemination efficiency.

REFERENCES

- [1] Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). IEEE standard, October 2003.
- [2] IEEE 802.5.4-based specification for low-rate low-power wireless personal area networks. ZigBee Alliance.
- [3] J. Ko, A. Terzis, S. Dawson-Haggerty, D. E. Culler, J. W. Hui and P. Levis, "Connecting low-power and lossy networks to the internet," in IEEE Communications Magazine, vol. 49, no. 4, pp. 96-101, April 2011.
- [4] S. Ziegler, C. Cretz, L. S. Krco, et al., "IoT6 - Moving to an IPv6-Based Future IoT", in The Future Internet, vol. 7858, pp.161-172, 2013.
- [5] RFC6282. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks, 2011.
- [6] RFC6775. Neighbour Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LOWPANs), 2012.

- [7] RFC6550, RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, 2012.
- [8] Simple Object Access Protocol, specifications and technical recommendations, <https://www.w3.org/TR/soap/>
- [9] F. Belqasmi, R. Glitho, C. Fu, "RESTful web services for service provisioning in next-generation networks: a survey", in *IEEE Communications Magazine*, vol. 49, no. 12, pp. 66-73, 2011.
- [10] C. Bormann, A. P. Castellani and Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," in *IEEE Internet Computing*, vol. 16, no. 2, pp. 62-67, March-April 2012.
- [11] ZigBee IP Stack specification for IPv6-based full mesh networking, ZigBee Alliance 2009.
- [12] Low-power long range Wi-Fi based on IEEE 802.11ah specification, Wi-Fi Alliance.
- [13] G. R. Hiertz et al., "IEEE 802.11s: The WLAN Mesh Standard," in *IEEE Wireless Communications*, vol. 17, no. 1, pp. 104-111, February 2010.
- [14] Open-source implementation of the IEEE 802.11 wireless mesh standard, available online: <http://www.o1s.org/>
- [15] P. Pandey, S. Satish, J. Kuri and H. Dagale, "Design & implementation of IEEE 802.11s mesh nodes with enhanced features," 2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems, Macau, 2009, pp. 639-644.
- [16] M. M. Wang, J. N. Cao, J. Li, S. K. Dasi, "Middleware for wireless sensor networks: A survey", *Journal of computer science and technology*, vol. 23 no. 3, 305-326, 2008.
- [17] Sensor Web Enablement (SWE) framework and standards, Open Geospatial Consortium (OGC).
- [18] L.M.S. de Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos and D. Savio, "SOCRADES: A Web Service Based Shop Floor Integration Infrastructure," in *Proc. Internet of Things Conf. (IoT'08)*, pp. 50-67, Mar. 2008.
- [19] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, U. Scholz, "MUSIC: Middleware support for self-adaptation in ubiquitous and service-oriented environments", in *Software Engineering for Self-Adaptive Systems*, vol. 5525, pp. 164-182, 2009.
- [20] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, "Mires: a publish/subscribe middleware for sensor networks", in *Personal and Ubiquitous Computing*, vol. 10, no. 1, pp. 37-44, 2005.
- [21] P. R. Pietzuch and J. M. Bacon, "Hermes: a distributed event-based middleware architecture," in *Proceedings. 22nd International Conference on Distributed Computing Systems Workshops*, pp. 611-618, 2002.
- [22] P. Levis, D. Culler, "Maté: a tiny virtual machine for sensor networks", in *international conference on Architectural support for programming languages and operating systems*, pp. 85-95, New York, USA, 2002.
- [23] Bonnet, P., Gehrke, J., & Seshadri, P. "Towards sensor database systems", in *Mobile Data Management, Lecture Notes in Computer Science* pp. 3-14, Springer Berlin Heidelberg, 2001.
- [24] Madden, S. R., Franklin, M. J., Hellerstein, J. M., & Hong, W. "TinyDB: an acquisitional query processing system for sensor networks", in *ACM Transactions on database systems*, vol. 30, no. 1, pp. 122-173, 2005.
- [25] OpenHAB, a framework for home automation systems and IoT technology integration, available online: <http://www.openhab.org/>
- [26] OSGi, Dynamic module framework and specifications for Java programming, OSGi Alliance, <https://www.osgi.org/>
- [27] J. Portilla, A. de Castro, E. de la Torre, T. Riesgo, "A Modular Architecture for Nodes in Wireless Sensor Networks", in *JUCS*, vol. 12, no. 3, pp. 328-339, March 2006.
- [28] Wi-Fi Plug for remote wireless control of powered devices, <http://www.wifiplug.co.uk/>
- [29] U. Hunkeler, H. L. Truong and A. Stanford-Clark, "MQTT-S – A publish/subscribe protocol for Wireless Sensor Networks," in *Conference on Communication Systems Software and Middleware and Workshops*, pp. 791-798, Bangalore, 2008.

Algoritmo de Aprendizaje Automático para la Detección y Captura de Tráfico Usando el Sniffer Multicanal SnifferWSNTOS

Miguel S. Polonio¹, Joaquín Olivares², Jose M. Palomares³, F. León⁴, J.M. Castillo-Secilla⁵ y A. Cubero-Fernández⁶

Resumen— El desarrollo e implantación de redes de sensores inalámbricas o WSN (Wireless Sensor Network) ha sufrido un fuerte empuje en los últimos años, su aplicación en entornos industriales, médicos, incluso en hogares, es cada vez más común. Por tanto, conocer el estado de los canales de transmisión de este tipo de redes es fundamental ante nuevas instalaciones o incluso para redes ya instauradas y que se encuentren trabajando. Junto con la captura de tráfico en los canales de transmisión, son los dos motivos del estudio que se realiza sobre este tipo de redes. Como primera fase del mismo, se desarrolló la herramienta software SnifferWSNTOS[1], la cual permite con una sola mota de la red, detectar y capturar tráfico en cada uno de los 16 canales de la banda. En este artículo se propone una mejora del algoritmo de escaneo y obtención de tráfico. Este algoritmo realiza un análisis de todos los canales centrándose en aquellos en los que se ha detectado envío de tramas y, tras una fase de aprendizaje, obtiene la cadencia de envío de cada uno de ellos. Posteriormente, conmuta el canal de escucha para capturar las tramas de los canales en los que hay tráfico en un momento dado. Este nuevo sistema proporciona unos porcentajes de tramas capturadas mucho mayores que el algoritmo anterior, es decir, hace que la mota la cual lleva instalado este algoritmo, se convierta en un sniffer inteligente de los 16 canales de la banda de canales de transmisión WSN IEEE 802.15.4.

Palabras clave—Sniffer, Wireless Sensor Network, 802.15.4, ZigBee, TinyOS, WPAN.

I. INTRODUCCIÓN

El objeto de este estudio son las redes establecidas en el estándar IEEE 802.15.4, redes WPAN[3] de baja velocidad. Son las adecuadas para WSN debido a su flexibilidad, bajo coste económico, bajo consumo de energía y baja tasa de transmisión de datos. Las redes IEEE 802.15.4 utilizan dos bandas de frecuencia en su capa física[3], PHY 868/915 Mhz (con 11 canales de transmisión de datos y una tasa de transferencia de 40Kb/s) y la PHY 2,4GHz (con 16 canales de transmisión de datos y una tasa de transferencia de 250Kb/s). Para este trabajo se utiliza la banda PHY de 2,4GHz, por tanto este sistema deberá ser capaz de analizar y capturar el tráfico de tramas en cada uno de los 16 canales de esta banda de frecuencias.

El estándar IEEE 802.15.4[3] solamente implementa las dos primeras capas del modelo arquitectónico de red, dejando a cada fabricante el desarrollo de las capas superiores. Dentro de los diferentes modelos arquitectónicos que existen para este tipo de redes, trabajamos con el estándar ZigBee[4][5], el cual aporta una capa de red llamada NWK y una capa de aplicación denominada APS. Existen en el mercado muchos dispositivos hardware que

utilizan el estándar ZigBee, en la primera fase de nuestro trabajo en la cual se desarrolló la herramienta SnifferWSNTOS[1], se realizó una búsqueda y estudio de los mismos, como conclusión se decidió la utilización de dispositivos TelosB[2] de la empresa MEMSIC. Entre algunas de las ventajas de estos dispositivos se encuentra que son los más utilizados para la difusión de estudios en la comunidad científica.

En la primera fase de este trabajo se desarrolló un sistema software capaz de capturar el tráfico de red con un solo dispositivo hardware en todos los canales de la banda de 2,4Ghz. El sistema quedó dividido en dos partes, por un lado la interfaz gráfica de usuario, implementada en lenguaje de programación JAVA, la cual permite seleccionar el tipo de sondeo que queremos realizar, almacenar estadísticas de uso de canal en un determinado ámbito, mostrar históricos de datos y ser el punto de unión para cargar en la mota hardware los algoritmos necesarios para que ésta se convierta en sniffer de un solo canal o de los 16 canales de la banda de frecuencias. La otra parte de nuestro proyecto es el desarrollo de los algoritmos para la mota sniffer. Dichos algoritmos se implementaron utilizando el lenguaje de programación NesC[6] sobre el sistema operativo TinyOS[7]. Se han desarrollado en NesC principalmente dos tipos de algoritmos, uno que permite a la mota convertirse en sniffer de un solo canal durante un tiempo determinado y capturar todas las tramas que circulen por el mismo y otro que permite a la mota alternar el canal de escucha entre los 16 canales de transmisión para capturar el tráfico entre estos. En este artículo se expone la modificación y mejora de este algoritmo, consiguiendo que el sniffer desarrollado sea más eficaz en cuanto a adaptabilidad a los canales con tráfico y mayor número de tramas capturadas.

El nuevo algoritmo desarrollado se adapta al tráfico que existe en cada uno de los 16 canales en un momento determinado, haciendo que la mota hardware conmute y escuche el canal en el cual se prevé que va a haber tráfico, por tanto es un algoritmo adaptativo en contraposición con el anterior que realizaba un escaneo secuencial de cada uno de los canales. El desarrollo de este algoritmo también ha permitido conocer una aproximación de la cadencia de envío de las diferentes redes de sensores que puedan estar transmitiendo por cada uno de los 16 canales, siempre con la premisa de que estas redes suelen tener un comportamiento periódico, donde sus motas están programadas bajo la secuencia "dormir-despertar-enviar datos-dormir". Un gran porcentaje de las redes WSN trabajan de este modo.

¹Dpto. Arquitectura de Computadores, Electrónica y Tecnología Electrónica, Universidad de Córdoba, e-mail: mspolonio@gmail.com

²olivares@uco.es

³jmpalomares@uco.es

⁴fernando.leon@uco.es

⁵jmcastillo@uco.es

⁶acubero@uco.es

II. ANTECEDENTES

En el artículo que precede a éste y donde se explica el desarrollo del sistema SnifferWSNTOS[1], se detallan los diferentes sistemas de análisis de tráfico en redes de sensores que existen en el mercado y/o artículos relacionados con esta funcionalidad. En la actualidad hay diferentes sistemas de captura y análisis de paquetes para WSN. Algunos de ellos son capaces de sondear solo un canal a la vez, (Z-Monitor[8], dispositivo hardware CC2531 USB de TI[9], sistema ZENA[10]) o varios canales simultáneamente como los sistemas propuestos por Choong[11], [12], el sistema Peryton-M[13] y el sistema propuesto por Seong-eun[14]. Otros muestran diferentes tipos de visualización de los datos capturados y/o utilizan herramientas externas software como WireShark[15] ó SmartRF Packet Sniffer[16]. Por último existen sistemas capaces de trabajar en otras bandas diferentes a la usada por las WSN, por ejemplo el dispositivo Arada System[17].

De entre todos, el sistema *Peryton-M-Multi-Channel 802.15.4/ZigBee/LoWPAN Protocol Analyzer*[13] es el que más se asemeja al sistema descrito en este artículo en cuanto a funcionalidad como sistema sniffer multicanal. Peryton-M es analizador software comercial desarrollado por la empresa Perytons Network Visibility. Se basa en utilizar una base conectada a un pc vía USB en la cual se instalan tantos dispositivos sniffers como canales de la banda de 2,4GHz queramos sondear. Está disponible en tres versiones, según el número de módulos sniffers que soporte simultáneamente, para 4, 7 y 16 canales respectivamente. Como hardware sniffers admite diferentes dispositivos, módulo RZU USBSTICK de Atmel[13], módulo deRFusb23E06 de Dresden Elektronik[13], módulo deRFusb12E06 de Dresden Elektronik[13], etc. Proporciona un software que muestra la topología y elementos de la red de sensores y el tráfico de tramas en tiempo real. Permite realizar capturas y almacenarlas para posteriormente analizarlas. Sus principales desventajas son el alto precio (el usuario debe comprar el software analizador Peryton-M y los módulos sniffers hardware), baja flexibilidad y difícil portabilidad e integración del sistema con las redes de sensores que pretenden analizar.

III. ALGORITMO DE ESCANEEO SECUENCIAL

En un primer momento se desarrolló una aplicación que permitía a cualquier mota hardware de una WSN convertirse en un sniffer multicanal de los 16 canales de la banda de frecuencias. Cuando un nodo de la red es cargado con ella, se especifica el tiempo que debe estar escuchando un canal en concreto, o lo que es lo mismo, el tiempo de cambio de canal. El nodo espera la orden reset para iniciar la captura en el canal 11 de la banda de 2,4GHz, si no se ha determinado el comienzo en otro canal. En ese momento envía al PC el número del canal en el que se encuentra y pasa a modo escucha. Durante la escucha, si se detecta tráfico en el canal, el nodo envía al PC la palabra clave TRAMA, a continuación y si se ha capturado la trama completa o parte de la misma, se envía la trama capturada. Si la trama es ilegible debido a una interferencia, no se enviará al PC, sin embargo, al

haber transmitido la palabra clave TRAMA indica que el canal tiene tráfico. Cada vez que el nodo detecte nuevo tráfico repetirá este proceso que consiste en “envío palabra TRAMA - envío (trama capturada completa o trama capturada incompleta o nada)”. La trama capturada incompleta es consecuencia de detectar una nueva trama en el canal de transmisión antes de realizar el envío completo de la anterior trama al PC, ya que tiene prioridad de envío la palabra clave TRAMA que el tráfico capturado. De esta manera se refleja que hay tráfico en el canal y si éste es legible completamente, incompleto o ilegible. Esta información se utiliza posteriormente, al finalizar la escucha, para determinar si un canal tiene tráfico no conflictivo, ha sufrido interferencias, está libre o tiene un elevado volumen de tráfico que puede provocar pérdidas de información, bien por saturación del canal o por saturación de los buffers de los nodos receptores.

Después de un determinado tiempo, la mota sniffer cambia al siguiente canal de escucha de manera secuencial, recorriendo todos los canales desde el canal 11 hasta el canal 26, cada uno de estos canales tienen una diferencia de 0,05GHz comenzando por el canal 11 que se encuentra en la frecuencia de 2,4Ghz.

Los problemas con los que se encuentra este sistema de sniffer multicanal son varios. En primer lugar, hay que determinar el tiempo de salto de canal, si se utiliza un tiempo demasiado reducido, se puede encontrar con que en algunos canales nunca se esté a la escucha en el momento en que se está transmitiendo por ellos, aparte de realizar demasiados cambios de canal en la mota con la consiguiente pérdida de tiempo útil de escaneo. Por otro lado, si el tiempo de escucha en cada canal es demasiado alto, incluso superior al canal con mayor cadencia de envío, se estarán perdiendo mucho tiempo en un canal y posiblemente se estén transmitiendo datos en otros canales. Hay que determinar al principio de la escucha ese tiempo de salto en base al tipo de redes que se vayan a sondear, esto implica un conocimiento previo del usuario de las cadencias aproximadas de envío en el escenario de frecuencias. Por tanto, este sniffer deberá ser modificado para ser utilizado en diferentes ubicaciones no siendo flexible ante la inserción de nuevas redes en el escenario en el cual se está realizando la escucha. Otro de los inconvenientes es la escucha de canales en los cuales no hay tráfico, el sistema no es capaz de determinar que durante un periodo completo de sondeo no se ha detectado tráfico y por tanto eliminar ese canal de futuras escuchas. El algoritmo se ha diseñado como un sistema secuencial, esto tiene el inconveniente de tener que seguir siempre la misma secuencia de escucha cíclica, partiendo del canal 11 al canal 26. Por supuesto, las redes que estén transmitiendo por estos canales no siguen este sistema secuencial y se puede estar sondeando el canal 16 y que en el siguiente instante el canal con tráfico sea el 15, mientras el sniffer pasaría a sondear el canal 17.

Aunque el número de tramas capturadas puede ser muy pobre dependiendo de las cadencias de las redes que están transmitiendo y del tiempo de salto entre canales elegido, nuestro sistema es aconsejable ante la instalación de nuevas redes de sensores en entornos donde se desconocen si existen o no diferentes redes. Por ejemplo, si la nueva

WSN a instaurar tiene una cadencia de envío media de 2 minutos entre trama y trama, se puede configurar nuestro sniffer con un salto entre canal de 4 minutos, el doble de la cadencia de nuestra red. Después de la escucha de todos los canales durante un determinado tiempo, se tendrá una lista de canales en los cuales no se ha detectado tráfico, esto no quiere decir que no haya tráfico en los mismos, sino que su cadencia es mayor al doble de la nueva red que deseamos poner en marcha, por tanto el riesgo de colisión e interferencia en ese canal es muy bajo. Se podría entonces seleccionar uno de los canales sin tráfico e incluso hacer posteriormente una escucha exhaustiva de solo ese canal elegido para comprobar que no hay ningún tipo de red trabajando en esa frecuencia.

IV. NUEVO SISTEMA DE SONDEO DE CANALES

El nuevo algoritmo desarrollado, y objeto de exposición en este artículo, trata de solventar los problemas inherentes del uso del anterior algoritmo de escaneo multicanal. Para ello, se ha modificado la aplicación para que realice las funciones de sniffer multicanal permitiendo la escucha de un canal solo cuando se prevea que en el mismo hay probabilidad de que ocurra una transmisión. Uno de los problemas con los que se encontró inicialmente el primer algoritmo sniffer es el salto secuencial de canal. Para mejorarlo se decide modificar el algoritmo para que realice saltos aleatorios entre los 16 canales. Se implementó con dos opciones, la posibilidad de que un canal pueda ser escuchado varias veces antes de completar la escucha de todos los canales, sistema con repetición, o evitando esto, es decir, salto aleatorio entre los 16 canales sin repetición. Una vez modificado el algoritmo se ha comparado con la detección de tráfico que realiza el algoritmo secuencial, utilizando en los dos sistemas el mismo tiempo de escucha en cada uno de los canales, es decir, el mismo tiempo de salto. Los datos que se obtienen de las diferentes pruebas con el algoritmo aleatorio de salto de canal se encuentran reflejados en la tabla I. La columna TET indica el Tiempo de Escucha Total en minutos, la columna TCC indica el Tiempo de Cambio de Canal en segundos, la columna C1 indica si se ha detectado tráfico en el canal 12, dicho canal tiene una cadencia de envío de 5 segundos, la columna C2 indica si se ha detectado tráfico en el canal 16, dicho canal tiene una cadencia de envío de 10 segundos, la columna C3 indica si se ha detectado tráfico en el canal 19, dicho canal tiene una cadencia de envío de 25 segundos, la columna C4 indica si se ha detectado tráfico en el canal 22, dicho canal tiene una cadencia de envío de 50 segundos y la columna C5 indica si se ha detectado tráfico en el canal 23, dicho canal tiene una cadencia de envío de 100 segundos. Para cada una de las filas de la tabla se han realizado un total de 10 pruebas, para indicar que un canal tiene tráfico, se debe haber detectado como mínimo en el 70 por ciento de las pruebas.

El objetivo de estas pruebas es determinar si con un salto de canal aleatorio se conseguiría detectar tráfico en el menor tiempo de sondeo posible en la mayoría de los canales por los cuales se estaban transmitiendo tramas. Por eso, se utiliza como cambio de canal el tiempo de cadencia mínimo de envío de los canales con tráfico, es decir la

TABLA I
 DETECCIÓN DE TRÁFICO CON SALTO ALEATORIO ENTRE LOS 16 CANALES

TET	TCC	C1	C2	C3	C4	C5
10	5	•	•	×	×	×
20	5	•	•	•	×	×
30	5	•	•	•	×	×
10	2,5	•	•	×	×	×
20	2,5	•	•	•	•	×
30	2,5	•	•	•	•	•

cadencia del canal 12, que es de 5 segundos y un tiempo mucho inferior al mismo, 2,5 seg. La conclusión de las pruebas realizadas es que no se obtienen mejores resultados que con el sistema secuencial, ya que ante el mismo escenario de trabajo la detección de canales es prácticamente la misma, y se sigue cumpliendo como ya se estudió anteriormente, que el tiempo de cambio de canal mejor para el sistema es aproximadamente un valor entre el 0,001 y el 0,002 por ciento del tiempo de cadencia máxima de un canal determinado, siempre y cuando la cadencia no sea menor a 1 segundo, en ese caso, el tiempo de cambio de canal será siempre 1 segundo. Por otro lado, este sistema no arroja tiempos de escucha totales adecuados ya que por ejemplo, para detectar tráfico en el canal C4 (canal 22) con tasa de transferencia de 50 segundos, se necesita como mínimo 20 minutos de escucha, cuando con una escucha secuencial de los 16 canales durante 50 segundos solo haría falta para realizar la detección de tráfico en esos canales en el caso de que todos estuvieran transmitiendo como mucho a esa cadencia, un total de 50 seg × 16 canales = 13.4 minutos. Se desecha por tanto este tipo de cambio de canal.

Posteriormente, se decide dividir la banda en dos grupos, de manera que los primeros ocho canales serán sondeados de manera aleatoria durante la mitad del tiempo de escucha total y posteriormente y durante el mismo tiempo, se sondearan los siguientes ocho. Los porcentajes de detección que se obtienen con este sistema son muy parecidos a los conseguidos con el salto aleatorio entre los 16 canales, no aportando por tanto ninguna mejora.

Siguiendo con la dinámica de agrupar canales se realizan pruebas donde se crean cuatro grupos de cuatro canales cada uno, realizando la escucha durante un tiempo determinado en cada grupo saltando de manera aleatoria entre los canales del mismo. El objetivo de estas pruebas es conseguir reducir al máximo el tiempo de escucha total necesario para que el sniffer detecte tráfico en aquellos canales que lo tengan. El primer grupo abarca los canales del 11 al 14, el segundo los canales del 15 al 18, el tercer grupo los canales del 19 al 22 y el cuarto grupo los canales del 23 al 26.

Para realizar estas pruebas se han utilizado 5 motas transmisoras en los mismos canales y con las mismas cadencias que en las pruebas de la tabla I. El canal con cadencia máxima es el canal 23, con 100 segundos, por tanto realizar un sondeo secuencial por ese tiempo equivaldría a estar a la escucha un total de 1600 segundos (26.7 minutos), es decir, 100 segundos × 16 canales de transmi-

sión. Las primeras pruebas se realizaron reduciendo este tiempo a la mitad, es decir, 800 segundos. Al dividirse en cuatro grupos, el sistema estará saltando aleatoriamente entre los cuatro canales de un grupo un total de 200 segundos. Se debe resaltar que el objetivo de estos ensayos es la detección de tráfico en los diferentes canales, no la obtención del mayor número de tramas, solamente determinar en qué canal hay envío de tramas. En la tabla II se muestran los valores en tanto por ciento de detección de tráfico de canal según los diferentes tiempos de salto entre canales. La columna **TCC** indica el tiempo de cambio de canal en milisegundos. La columna **Prc.T** indica el porcentaje total de detección de todos los canales con tráfico. El tiempo total de escucha es de 800 segundos.

TABLA II
 PORCENTAJES DE DETECCIÓN DE TRÁFICO, GRUPOS DE 4 CANALES, SALTO ALEATORIO Y TIEMPO DE ESCUCHA 800 SEGUNDOS

TCC	C12	C16	C19	C22	C23	Prc.T
500	50	50	80	20	0	40
1000	100	60	100	50	50	72
1500	100	100	60	40	30	66
2000	100	70	60	70	10	62

Como se observa en la tabla II con un tiempo total de 800 segundos, la mitad del tiempo de escucha total máximo permitido según la cadencia del canal con cadencia de envío máxima, un salto de 1000 ms para sondeo de cada canal, es decir, un valor entre el 0.001 y el 0.002 por ciento del tiempo de cadencia del canal con cadencia de envío máximo y grupos de 4 canales cada uno con salto aleatorio entre ellos, se obtiene un 72 por ciento de efectividad a la hora de detectar tráfico en todos aquellos canales por los cuales se está transmitiendo. Valor muy superior al obtenido por el algoritmo secuencial anteriormente descrito en el mismo escenario.

Para aumentar el porcentaje de detección de tráfico que es lo que más relevante en esta fase del nuevo algoritmo, se decide realizar las mismas pruebas pero aumentando el tiempo de escucha total, pasando del 50 por ciento del tiempo del canal con cadencia máxima de envío al 75 por ciento. En el sistema propuesto, equivaldría a realizar una escucha total durante 1200 segundos, es decir, 300 segundos para cada grupo realizando saltos aleatorios con repetición en sus canales. Los resultados obtenidos se muestran en la tabla III. Su formato es el mismo que la tabla II. Para simplificar solamente se muestra los resultados para el mejor porcentaje obtenido en la tabla II, comprobándose que este nuevo porcentaje es sensiblemente superior. Los porcentajes con tiempo de escucha por canal más elevados, proporcionan peores resultados.

TABLA III
 PORCENTAJES DE DETECCIÓN DE TRÁFICO, GRUPOS DE 4 CANALES, SALTO ALEATORIO Y TIEMPO DE ESCUCHA 1200 SEGUNDOS

TCC	C12	C16	C19	C22	C23	Prc.T
1000	100	100	60	100	60	84

Se realizaron la mismas pruebas pero utilizando grupos de dos canales cada uno y conmutando entre ellos. Sin embargo, los resultados obtenidos fueron peores que con grupos de cuatro canales.

Después de realizar estas y otras pruebas con diferentes canales de transmisión y tasas de envío, se determinó que el algoritmo de escaneo o búsqueda de tráfico alcanza su mayor eficacia en cuanto a canales con tráfico detectados y tiempo empleado, cuando el tiempo de escucha total es el 75 por ciento de la cadencia del canal con cadencia máxima determinada, el tiempo de escucha en cada canal es el 1 por ciento del tiempo referente a la cadencia del canal con cadencia máxima, se realiza una agrupación de 4 canales en cada grupo y el salto entre ellos es aleatorio. Con este sistema, los porcentajes de detección de tráfico son muy superiores a los del algoritmo secuencial descrito anteriormente.

V. ALGORITMO DE APRENDIZAJE AUTOMÁTICO

Se ha desarrollado un algoritmo por fases, el cual mejora el algoritmo sniffer multicanal integrado en el sistema original descrito en [1]. Para ello, se ha dividido el mismo en 4 fases.

A. Fase 1: Localización de los canales con tráfico

Utilizando el sistema explicado en la Sec. III de este artículo, la primera función de la mota sniffer es realizar un sondeo de todos los canales para determinar en cual de ellos hay tráfico. Para ello y como único valor introducido por el usuario, se indica la cadencia máxima de envío de los canales que se quieren detectar. Ese valor se multiplicará por 16, a partir de ahora **TSM (Tiempo de Sondeo Máximo)**. Se supone que el funcionamiento de las redes de sensores es periódico. La mota sniffer calcula el 75 por ciento del tiempo de TSM y lo divide entre cuatro, a partir de ahora **TSG (Tiempo de Sondeo de Grupo)**. Durante el TSG saltará aleatoriamente entre los 4 canales del primer grupo, el tiempo de escucha en cada canal hasta el siguiente salto será un 1 por ciento de TSM, cuando se agote el tiempo TSG, el sniffer sondeará los 4 siguientes canales del grupo posterior. Después de esta fase el sistema ya sabe en qué canales de los 16 se ha detectado tráfico.

B. Fase 2: Calculo de cadencias máximas de canal con tráfico

Con los datos obtenidos en la Fase 1, se sondea cada uno de los canales donde se ha detectado tráfico. El tiempo de sondeo es igual al tiempo de cadencia máxima introducido por el usuario. Durante ese sondeo se contabilizan el número de tramas que se detectan en el canal, obteniendo una probabilidad de envío que será la cadencia de ese canal. Al finalizar esta fase se conocen todos los canales donde hay tráfico y la cadencia de envío de los mismos. Si durante la escucha de un canal no se detecta ninguna trama pero se sabe que hay tráfico porque así fue indicado en la Fase 1, se indica como cadencia de ese canal la cadencia máxima introducida por el usuario.

C. Fase 3: Puesta en marcha de los relojes de salto de cada canal

Durante esta fase se escucha cada uno de los canales con tráfico, como máximo en cada canal el tiempo de escucha es la cadencia máxima introducida por el usuario. Durante esa escucha, cuando se detecte la primera trama, se configura un reloj de salto o alarma que se activará de manera periódica según indique la cadencia de envío del canal. La primera vez que se activa el reloj se hace con valor cadencia de envío del canal menos 100 ms, para que el cambio de canal tenga este desfase y la mota tenga tiempo suficiente para permutar de canal, a partir de ese primer salto de reloj, él mismo se configura para saltar con tiempo igual a la cadencia del canal, es decir, cada reloj de cada canal con tráfico saltará 100 ms antes de que en el canal haya tráfico.

D. Fase 4: Captura de tramas

Esta fase es la puramente sniffer, a partir de este momento, la mota esperará a que salte cada uno de los relojes de aquellos canales con tráfico y que han sido configurados en la fase 3. En ese momento, conmutará a ese canal y se pondrá a la escucha, cuando detecte una trama enviará al PC vía USB el número de canal y la trama capturada para su posterior análisis.

VI. EVALUACIÓN Y PRUEBAS

Se han llevado a cabo diferentes pruebas para comprobar el funcionamiento del nuevo algoritmo. Se han modificado las cadencias de envío y adaptado los tiempos de escucha por canal y total según dichas cadencias. Se han realizado un total de 3 pruebas, de cada una de estas pruebas se han ejecutado 10 rondas. Para que se considere que en un canal hay tráfico, éste debe haber sido detectado en al menos 8 de las 10 rondas de cada prueba. Los porcentajes de tramas capturados sobre tramas totales enviadas, se calculan como media aritmética, utilizando solamente aquellas rondas de cada prueba donde se haya detectado tráfico. Para cada prueba, se ha generado una tabla con las siguientes columnas: Canal (indica el canal donde hay motas enviando a una determinada cadencia), Cadencia (cadencia de envío en ese canal), TD (indica si en al menos 8 de las 10 rondas se ha detectado tráfico en ese canal), TE (tramas totales que se han enviado en ese canal en ese tiempo de escucha), PTC (porcentaje de tramas que el sistema ha sido capaz de capturar). Los tiempos de escucha totales de cada ronda vienen calculados según se ha expuesto en la Sec. IV. Con un tiempo mayor de escucha total en cada prueba, los resultados de detección de tráfico en los canales con cadencia máxima hubieran sido mejores. Debido a esto, cuando el sistema sea implantado para funcionar en un entorno real, se debe determinar un tiempo de cadencia máxima límite, sabiendo que, si existen canales con cadencias muy parecidas o superiores a esa cadencia máxima indicada, la probabilidad de que el sistema detecte el tráfico, disminuye.

Después de analizar los resultados de las pruebas, se observa que ante cadencias de envío que superan el minuto, los porcentajes de captura de tramas son muy altos, de hecho, el no obtener porcentajes del 100 por 100 se debe al coincidir el envío de manera simultánea en varios

TABLA IV
PRUEBA 1

Canal	Cadencia	TD	TE	PTC
11	1 seg.	•	132	43
12	2 seg.	•	66	60
18	1 seg.	•	132	30
21	4 seg.		33	0
25	3 seg.	•	44	60

TABLA V
PRUEBA 2

Canal	Cadencia	TD	TE	PTC
11	2 min.	•	72	90
12	4 min.	•	36	95
18	6 min.	•	24	96
21	4 min.		36	0
25	3 min.	•	48	92

TABLA VI
PRUEBA 3

Canal	Cadencia	TD	TE	PTC
11	5 min.	•	72	97
12	10 min.	•	36	98
18	30 min.		12	0
21	20 min.	•	18	98
25	25 min.	•	14	99

canales. Hay que tener en cuenta que el sniffer se pone a la escucha del canal, en el cual prevé tráfico, un porcentaje de tiempo antes de que se produzca el envío. Si en ese momento se activa el salto hacia otro canal, el sistema pasará a la escucha de este nuevo canal dejando de capturar la trama del canal anterior. Esta es una de las mejoras que se contempla. Para solucionarlo, se utilizarán algoritmos de exclusión mutua. Este problema se acentúa ante canales que tienen una intensidad de tráfico muy alta, menores a un minuto. Aunque en estos escenarios los porcentajes de tramas capturados son bajos, el sistema sí ha sido capaz de marcar al canal como canal con tráfico. Se puede indicar, por tanto que, el sistema propuesto actúa como un sniffer multicanal con un elevado porcentaje de captura de tramas para canales con cadencias superiores al minuto y que es una herramienta que puede detectar tráfico en canales con cualquier cadencia de envío.

Como ya se ha citado, el sistema comercial que realiza las funciones de sniffer multicanal es el desarrollado por la empresa Perytons [13]. No se ha adquirido dicho sistema para comprobar su funcionamiento, pero se ha realizado una comparativa en base al hardware que utiliza. El sistema Peryton se basa en el uso de un dispositivo hardware por canal, por tanto, dicho dispositivo estará a la escucha del canal el 100 por 100 del tiempo. Estos dispositivos están conectados a un hub, el cual a su vez se conecta al PC vía USB. Este sistema provoca un cuello de botella ante envíos concurrentes y a altas frecuencias.

Obviando este punto y suponiendo que todas las tramas capturadas por los dispositivos lleguen correctamente al PC, el sistema depende de la capacidad del dispositivo para la captura de tramas, por tanto habría que estudiar las características técnicas y su funcionamiento el campo de trabajo de los diferentes dispositivos que admite Perytons para poder realizar todas las pruebas y comparativas, sobre todo en entornos de trabajo donde las cadencias de envío sean muy altas, que es precisamente donde nuestro sistema no consigue buenos porcentajes de captura de tramas. Por otro lado, el sistema Perytons se comercializa en 3 versiones, con 4, 7 y 16 dispositivos hardware respectivamente. Las pruebas que se exponen en este artículo se han realizado con tráfico en 5 canales, el sistema más básico de Perytons ya ofrecería carencias en ese aspecto al estar limitado a solo 4. Suponiendo que se utiliza el sistema de 7 dispositivos y que se obtienen porcentajes del 100 por 100 de captura de tramas para las pruebas 2 y 3, Tabla V y Tabla VI, el coste hardware es muy superior al tener que utilizar 7 dispositivos. Coste que se multiplicaría si utilizáramos la versión de 16 dispositivos.

La degradación en cuanto a resultados en porcentajes de tramas capturadas del nuevo algoritmo desarrollado para el sistema SnifferWSNTOS, es proporcional al número de canales en los que se detecte tráfico. A mayor número de canales con envío de tramas, mayor probabilidad de que en varios canales se esté transmitiendo al mismo tiempo, por tanto, el sniffer perderá tramas de alguno de ellos. Sin embargo, ante cadencias de envío bajas, esta probabilidad se reduce. Por otro lado, el algoritmo puede ser modificado para que solo busque tráfico en 8 de los 16 canales, utilizando dos dispositivos en vez de uno y así reduciendo la probabilidad de coincidencias de envío. Esta sería una buena solución cuando las cadencias de envío son altas. Para cadencias de envío de menos de un minuto, este sistema proporciona porcentajes muy bajos de captura de tramas. Debido a esto, en el caso de que se desee capturar tramas en este tipo de escenarios, se recomienda usar el algoritmo de escaneo propuesto en este trabajo para determinar en qué canal o canales de los 16 se está transmitiendo y, posteriormente, dedicar un dispositivo hardware específico a cada uno de los canales con tráfico detectado.

VII. CONCLUSIONES

Se ha desarrollado un nuevo algoritmo sniffer capaz de detectar con mayor probabilidad de éxito aquellos canales que cuentan con tráfico y de obtener un mayor porcentaje de número de tramas capturadas. Dicho algoritmo ayudará principalmente en dos cuestiones, conocer el estado de los 16 canales en cuanto a ocupación y convertir cualquier mota en sniffer multicanal adaptativo. Si bien es cierto que, es necesario introducir un valor que debe ser indicado el usuario, la cadencia máxima de envío, este valor puede ser modificado según el tipo de redes de sensores que queramos estudiar. Por otro lado, se ha supuesto la periodicidad de este tipo de redes ya que, en la fase de cálculo de cadencia, lo que se obtiene es la probabilidad de envío del canal y ante redes que no siguen un patrón de este tipo, esta probabilidad puede diferir mucho y no ser válida. Ante redes que cumplan esta periodicidad

y utilizando tiempos de cadencias adecuados, una vez que el sniffer pasa a la fase de obtención de tramas, los resultados de porcentajes de tramas obtenidos son mejores que otros sistemas del mercado que, o bien utilizando un gran número de dispositivos hardware (uno por canal) con el consiguiente coste económico y de energía, o bien, conmutando entre canales, realizan también la labor de sniffer. Cabe destacar que se han cumplido algunas de las mejoras propuestas en el artículo que precede a éste, en el cual se explica el desarrollo del sistema SnifferWSNTOS[1] y que fue punto de partida de esta investigación y que aspira concluir con el desarrollo de un sistema que permita a las redes de sensores inalámbricas adaptarse a las circunstancias del tráfico e interferencias y conmutar su canal de trabajo, si fuera necesario.

Cómo mejoras de este algoritmo se indican las siguientes:

- En la fase de búsqueda de tráfico, no conmutar a canales donde ya se ha detectado tráfico, con esto se sondearían más veces aquellos canales en los cuales aún no se ha detectado actividad.
- Gestionar de mejor manera aquellos canales con cadencias de envío muy pequeñas, o lo que es lo mismo, canales donde exista mucho tráfico.
- Estudiar e implementar un sistema de exclusión mutua para evitar que los relojes de alarma de diferentes canales se disparen a la vez.
- Desarrollar un algoritmo que detecte tráfico e interferencias no solo de redes de sensores, sino también de otras redes inalámbricas que transmiten en la banda de frecuencias de 2,4GHz y gestionar el ruido que pueda haber en dicha banda.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado mediante los proyectos: P11-TIC-7462 y DPI2013-47347-C2-2-R.

REFERENCIAS

- [1] M. Sánchez-Polonio, J. Olivares, J.M. Palomares, F. León, J.M. Castillo-Secilla and A. Cubero SnifferWSNTOS: *Software implementation for multi-channel IEEE 802.15.4 Sniffer* Proc. of the VI Jornadas de Computación Empotrada, 49-56, Septiembre 2015.
- [2] Especificaciones técnicas de TelosB, http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf, 2016.
- [3] IEEE, *IEEE 802.15.4 Standard Part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for a low-rate wireless personal area network (LR-WPANs)*, Junio 2006.
- [4] Sinem Coleri Ergen, *ZigBee/IEEE 802.15.4 Summary*, <http://www.sinemergen.com/zigbee.pdf>, 10 de septiembre de 2004.
- [5] Zigbee Alliance, <http://www.zigbee.org>.
- [6] David Gay, Philip Levis, David Culler, and Eric Brewer *NesC 1.1 Language Reference Manual*, <http://nescc.sourceforge.net/papers/nesc-ref.pdf>, Mayo de 2003.
- [7] Lewis Philips, *Tinyos Programming*, <http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf>, 27 de octubre de 2006.
- [8] ZMonitor, *A Monitoring Tool for IEEE 802.15.4 WPANs*, <http://www.z-monitor.org>, 2011.
- [9] Texas Instruments, *CC2531 USB Hardware User's Guide*, <http://www.ti.com>, Revisión SWRU221, 8 de Mayo de 2009.
- [10] Microchip Technology, *ZENA Wireless Network Analyzer Users Guide*, 2007.
- [11] L. Choong and M. Tadjikov, *Using USRP and gnuradio to decode 802.15.4 packets*, University of California, Los Angeles, USA, 2008.
- [12] L. Choong, *Multi-Channel IEEE 802.15.4 packet capture using software defined radio*, Networked and Embedded Systems Laboratory, UCLA, Technical Report TR-UCLA-NESL-200904-01, Abril 2009.

- [13] Perytons Network Visibility, *Peryton-M Multi-Channel 802.15.4-ZigBee-6LoWPAN Protocol Analyzer. Guía de especificaciones*, <http://www.perytons.com>, Revisión 32210A, Septiembre de 2013.
- [14] Yoo Seong-eun, Chong Poh Kit, Bae Jeonghwan, Kim Tae-Soo, Kim Hiecheo, and Yoo Joonhyuk, *Multi-Channel packet-analysis system based on IEEE 802.15.4 packet-capturing modules*, International Journal of Distributed Sensor Networks, Volume 2014, Article ID 216504, 16 de septiembre de 2014.
- [15] Yoo Joonhyuk, *Wireshark packet analysis*, <http://www.wireshark.org>, 2015.
- [16] Texas Instruments, *SmartRF Packet Sniffer User Manual*, <http://www.ti.com>, 2011.
- [17] Arada System, *Multi-Channel Sniffer Powered by Arada Wireless Solution Software. Guía de especificaciones*, <http://www.aradasystem.com>, Abril de 2009.

Sistema de Arquitectura Abierta: el Gesto Aplicado al Control Robótico

Gonzalo Pomboza-Junez y Juan Antonio Holgado-Terriza

Resumen—Este artículo presenta el uso de una interfaz natural de usuario que permite el uso del gesto de la mano, aplicado al control de un dispositivo robótico. Para lograrlo hemos usado el dispositivo de captura gestual conocido como controlador Leap Motion® y al robot Lego EV3®. Presentamos un sistema de arquitectura abierta que permite la comunicación usando el protocolo Bluetooth o WiFi entre la interfaz de control gestual y el robot. La interfaz permite dos modos de operación que son automático y manual, para el control del robot, el cual puede desplazarse linealmente, realizar giros e inclusive transportar objetos. Se define además una biblioteca gestual que asocia cada gesto a un comando prestablecido. Los que conforman la biblioteca, han sido evaluados en su calidad de reconocimiento lo cual permite ubicarlos en una escala (alto, medio y bajo) de aceptación para tareas de control. Finalmente, un caso de estudio, permite evaluar la efectividad del gesto aplicado a tareas de control, logrando resultados muy alentadores.

Palabras clave—Control Gestual, Interfaces gestuales, Lenguaje gestual, Control robótico, Interfaces naturales de usuario de tipo gestual.

I. INTRODUCCIÓN

LA interacción hombre-máquina continua en constante evolución. Las interfaces, su diseño y aplicación siguen transformado al mundo que nos rodea y a la sociedad en su conjunto [1]. El objetivo de alcanzar una forma de Interacción Humano Computador (IHC), donde una idea del usuario sea reflejada en una acción inmediata y sin restricciones físicas, está muy cercana [2].

El uso del gesto en la comunicación con la máquina no es nueva y, muchos trabajos hacen uso de ellos incorporando gestos corporales [3], faciales [4] y manuales [5] en la IHC. Otros trabajos tratan de orientar el gesto a mundos desafiantes como la realidad virtual [6] y el control de maquinaria industrial robótica [7]. Y otras investigaciones lo orientan al control de dispositivos de casa [8] e inclusive lo adaptan al control de sillas de ruedas [9]. Sin embargo, lograr una interfaz que permita el control de un dispositivo, basado directamente en el comportamiento del usuario, es un concepto que va cobrando más fuerza en los últimos años [10]. Entonces, bajo este enfoque, una interfaz de tipo gestual debería ser considerada como aquella interfaz diseñada para reutilizar las habilidades previas del usuario [11], y permitirle interactuar con su entorno [12] usando movimientos gestuales intuitivos [13] de las manos, o cualquier otra parte del cuerpo.

El interés por aplicar el gesto a tareas especializadas ha permitido el desarrollo de nuevos dispositivos tecnológicos, que fusionan hardware y software en un

Este trabajo esta soportado por la Secretaría de Educación Superior, Ciencia, Tecnología e Innovación del Ecuador (SENESCYT) y por licencia de la Universidad Nacional de Chimborazo a través su programa de formación para Ph.D con la Universidad de Granada, España.

Gonzalo Pomboza-Junez, Universidad Nacional de Chimborazo, Riobamba, Ecuador (e-mail: wpomboza@unach.edu.ec)

Juan Antonio Holgado-Terriza, Universidad de Granada, Granada, España (e-mail:jholgado@ugr.es)

solo aparato [14]. La diferencia, además de la tecnología usada, radica en los límites que esos dispositivos imponen al usuario.

El controlador Leap Motion® (LMC siglas en inglés) combina tecnología óptica e infrarroja, alcanzando una gran exactitud en la detección del gesto en tiempo real [15],[16]. Permite realizar el seguimiento del gesto tanto estático como dinámico [17]. Además, estudios indican que su exactitud está inclusive por debajo del milímetro [18]. Sin embargo, la gran ventaja de LMC es que el reconocimiento del gesto es independiente de la contextura física del usuario (altura, peso, etnia, edad, etc), es decir, el gesto no requiere ser ajustado a un patrón de referencia. Quizá como un punto en contra se podría considerar el hecho de que hasta el momento, no haya liberado los algoritmos de reconocimiento gestual usados, limitando a los investigadores en la búsqueda de nuevas y mejores aplicaciones. Pero, basándonos en sus características tanto técnicas como de reconocimiento, es ideal para lograr una consola de interacción gestual para control supervisado de un dispositivo robótico.

En la actualidad, sistemas robotizados que requieren presencia de un operador para lograr un control reactivo adecuado, están limitados en un gran porcentaje al conocimiento técnico y entrenamiento del operador para lograr una tarea o adaptarse a su entorno [19]. Es notorio entonces que, si deseamos avanzar en este sentido, debe establecerse un método de interacción más versátil que permita su control y logre mayor versatilidad en cualquier entorno.

Este artículo expone el uso del gesto, basado en LMC para el diseño de una interfaz gestual que permite el control reactivo de un dispositivo robótico. Un lenguaje gestual sencillo permite controlar un robot que posee libertad de movimiento (lineal-direccional) e incorpora un brazo electromecánico para transporte de objetos. Responde en forma rápida y efectiva a las órdenes gestuales indicadas por el usuario. Las limitaciones alámbricas para envío y recepción de datos, entre la consola de administración y el robot, han sido superadas usando los protocolos de comunicación Bluetooth-WiFi.

Este artículo se encuentra organizado de la siguiente manera: La Sección II describe los materiales y métodos usados. La Sección III presenta un caso de estudio. La Sección IV presenta los resultados y su discusión. Finalmente, la Sección V expone las conclusiones.

II. MATERIALES Y MÉTODOS

A. Sensor de captura gestual.

Hemos usado a LMC para capturar el gesto que realiza el usuario. Dicho gesto, es realizado con la mano directamente sobre el LMC, en el aire y sin tocar dispositivo alguno. El LMC en su interior contiene dos cámaras estéreo monocromáticas y tres led infrarrojos (longitud de onda de 850 nm) que permiten construir escenas en 3D [20] (Fig. 1). Según el fabricante (Leap Motion Inc., San Francisco, CA, US) LMC es capaz de

reportar posiciones discretas, gestos y movimientos. Su unidad de tiempo es el microsegundo (μ) y la de velocidad es el milímetro por segundo (mm/s). Sus ángulos están dados en radianes y se conecta al ordenador usando un puerto USB. No presentó restricción alguna a los programas que utilizamos y se usó la versión 1.2.0 del SDK de desarrollo. Mayores detalles pueden encontrarse en la página WEB del fabricante (<https://www.leapmotion.com/>).

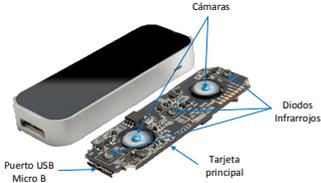


Fig. 1. Vista externa e interna de Leap Motion Controller® y sus componentes principales.

El LMC fue conectado a un ordenador Intel Core i7-X455L de 3.1 GHz con 12 GB en memoria RAM, OS WIN 8.1, HDD 1TB. Tarjeta Gráfica NVIDIA Geforce 820M de 2 GB. El lenguaje de programación usado fue Java. Para obtener los datos de los puntos rastreables de la mano, que LMC capta con su hardware, se utilizó un objeto de tipo *controller* el cual permite el acceso al último cuadro capturado. Este cuadro, es un objeto de tipo *frame*, que contiene la siguiente información: número de manos, tipo de mano (izquierda o derecha), la posición y número de los dedos, circunferencia de la palma y el reconocimiento de herramientas. Una herramienta es considerada recta, más fina y más larga que un dedo.

B. Robot.

Como sistema robótico hemos elegido a EV3 (Fig. 2) con las siguientes características: microprocesador ARM-9 de 300 MHz., 64 MB de memoria RAM, 16 MB de memoria Flash expandible por tarjeta SD, Bluetooth 2.1, USB para conexión WiFi, 4 puertos para sensores y otros 4 para motores, pantalla LCD y altavoz. El sistema es compatible con iOS y Android. Para mayores detalles se puede consultar (<http://www.lego.com/es-es/mindstorms/about-ev3>).

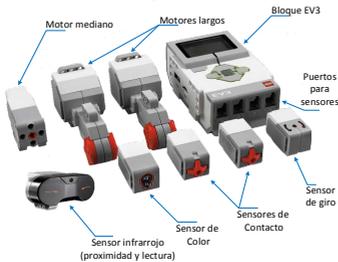


Fig. 2. EV3 y sus componentes.

C. Lenguaje gestual.

Los gestos propuestos se muestran en la Fig. 3. LMC permite identificar al gesto de acuerdo a sus características como número de dedos, posición y movimientos. Por su naturaleza pueden ser dinámicos o estáticos y pueden involucrar movimiento de los dedos, de la mano o ambos a la vez. Las librerías de LMC permiten el reconocimiento de gestos preestablecidos y la implementación de otros, por parte del usuario.

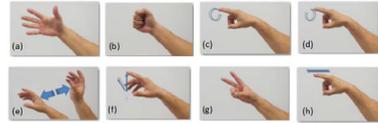


Fig. 3. Gestos propuestos para control.

Los gestos preestablecidos se indican a continuación:

- 1) *Gesto Circulo*, que puede ser a favor y en contra de las manecillas de reloj. En el primer caso, el dedo índice traza un círculo a favor de las manecillas del reloj mediante un movimiento circular (Fig. 3c); en el segundo caso, el dedo índice traza un círculo en contra de las manecillas del reloj mediante un movimiento circular en esa dirección (Fig. 3d). La mano, en los dos casos, permanece estática sobre el LMC.
- 2) *Gesto de barrido o Swipe*, es un rápido movimiento o desplazamiento lineal (de izquierda a derecha o viceversa) de la mano sobre el dispositivo LMC (Fig. 3e).
- 3) *Gesto de tecla o Key Tap*, es el movimiento del dedo índice, manteniendo estática la mano. Dicho movimiento es un desplazamiento de arriba hacia abajo, y trata de simular el haber presionado una tecla (Fig. 3f).
- 4) *Gesto toque de pantalla o Key Screen Tap*, es similar al anterior pero con el dedo índice fijo y apuntando hacia adelante, mientras se desplaza toda la mano longitudinalmente. Describe una posición inicial (sobre el LMC) y una final (con desplazamiento hacia adelante). Los demás dedos levemente recogidos sobre la palma. Este gesto supone tocar un punto en pantalla, simula un gesto toque (touch) de pantalla. (Fig. 3h).

Además de los gestos preestablecidos, fue necesario implementar también los siguientes gestos:

- 5) *Gesto mano abierta*, con los dedos totalmente extendidos y la palma de la mano hacia abajo (Fig. 3a) sobre el dispositivo LMC. La mano permanece estática a una altura aproximada de 15-20 cm.
- 6) *Gesto mano cerrada*, todos los dedos recogidos en la palma formando un puño cerrado (Fig. 3b) a 15-20 cm sobre el LMC.
- 7) *Gesto Two*, se logra extendiendo los dedos índice y medio, mientras los demás están recogidos en la palma y cubiertos con el dedo pulgar. (Fig. 3g). La mano en esta posición permanece estática sobre el dispositivo a 15-20 cm.

Los gestos implementados según Pomboza y Holgado (2015) [8], son confortables (mano abierta 66%, mano cerrada 78% y dos 52%) y además fáciles de recordar en tareas repetitivas para el usuario, según el mismo estudio.

D. Análisis gestual.

Fue necesario establecer la efectividad del gesto en su reconocimiento para ser usado en control y además observar el grado de aceptación para un dispositivo robótico. Se realizó un análisis del reconocimiento del gesto basado en el objeto frame que entrega LMC. Para lograrlo, se desarrolló un programa que permitió identificar al gesto y asociar sus datos al frame. Algunos gestos por su naturaleza dinámica (movimiento de mano, dedos y brazo) a diferencia de otros de naturaleza estática (postura fija) implicaban varios frames para lograr su reconocimiento.

En nuestro análisis participaron ocho voluntarios, 5 hombres y 3 mujeres, cuyas edades estaban comprendidas entre los 19 y 38 años ($M=26.75$, $SD=8.11$). Todos los voluntarios conocían el uso del computador y los hombres indicaron tener experiencia en el uso de video juegos.

Los gestos fueron realizados directamente sobre el dispositivo, manteniendo una altura entre 15 a 20 cm con la suficiente holgura para ejecutarlos. Los gestos estáticos fueron realizados manteniendo la mano en la postura adecuada y firme al ejecutar el gesto; y, los dinámicos, con repeticiones sucesivas al finalizarlo. El tiempo de muestreo de cada gesto fue de 15 segundos. Cada participante completó ocho gestos y se logró un tiempo total de muestreo gestual de 960 s.

Las muestras fueron tomadas en un lugar adecuado, siguiendo las recomendaciones del fabricante, tanto en altura como en amplitud, garantizando en lo posible la menor contaminación por ruido lumínico (no luz con fluorescencia y sin incidencia directa de luz solar). La información, así obtenida de los frames del gesto, fue guardada en un archivo por cada gesto y usuario, para ser procesada posteriormente.

Cada gesto mostró un comportamiento diferente, tanto en los frames necesarios para su identificación como en el tiempo empleado en generar dicho frame. La Figura 4, nos permite observar el comportamiento del gesto (b) en contraste con el gesto (h). El gesto (b) mantiene una frecuencia más o menos estable entre los frames lanzados para reconocer el gesto y el tiempo entre cada intervalo. Mientras que el gesto (h) tiene un comportamiento menos estable, pues los frames son lanzados a intervalos esporádicos tratando de lograr la identificación adecuada del gesto.

Los frames varían su comportamiento en el tiempo, acorde a la complejidad del gesto. Dependiendo entonces del gesto, puede producirse un retardo en su reconocimiento e inclusive una pérdida de información cuando el gesto no es completado, como el caso del gesto (h) (toque de pantalla). Para determinar una relación que permita establecer la precisión del gesto, se realizó un análisis frame-frame. Se pudo determinar un total de frames para cada gesto, considerando que un

gesto se logra definir en al menos dos frames, y que en ocasiones el ruido produce frames vacíos o sin información del gesto.

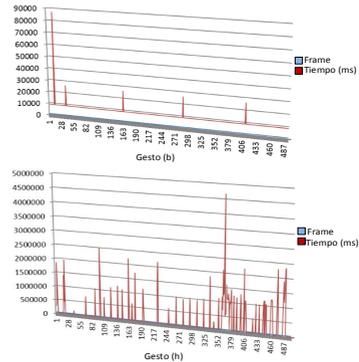


Fig. 4. Muestra del comportamiento entre dos de los gestos estudiados.

Se obtuvo un total de frames válidos (f_n) en la lectura del gesto. De aquellos, se logró determinar un conjunto de frames (positivos) que muy claramente lo identificaban (f_g). El ruido, provocó frames sin información (f_n) o vacíos en algunos casos. Así, hemos logrado la ecuación (1) que trata de medir la precisión (PR) del gesto, basado en los frames positivos que componen el gesto:

$$PR = \frac{f_g}{f_g + f_n} \times 100\% \quad (1)$$

dónde PR lo hemos expresado en porcentaje.

Los resultados de PR obtenidos para cada gesto, permitió ubicarlos en una escala de aceptación para control. Dicha escala de aceptación fue establecida de la siguiente manera: alto (100-76%), medio (66-34%) y bajo (33-0%). Esta escala permitió calificar al gesto, para admitirlo o no en tareas de control, siempre que al menos supere el intervalo más bajo de la misma.

III. CASO DE ESTUDIO

A. Arquitectura del sistema.

La arquitectura tiene características cliente-servidor y fue implementada usando leJOS en el EV3 y combinando librerías de Leap Motion y EV3 en la interfaz gestual. leJOS es un proyecto libre desarrollado para interactuar con NXJ y EV3, diseñado en Linux y que debe ser instalado en una micro SD. leJOS al encendido se apropia de los recursos del EV3, reemplazando al sistema original.

La arquitectura del sistema propuesto se expone en la Fig. 5, y está conformada por tres capas que se describen a continuación:

Capa Uno: Está formada por el LMC como un sensor de captura gestual (GS) y se encarga de la lectura del gesto realizado por el usuario. Envía datos a la capa 2 por conexión USB.

Capa Dos: Está formada por la interface natural de usuario de tipo gestual (INU-G) desarrollada para control y la hemos llamado LEAPEV3 por sus componentes (Leap Motion y EV3). Esta capa se ejecuta en el ordenador y fue programada en Java e integra a LMC y EV3 en sus librerías. Además, agrupa internamente a los componentes para Proceso y Reconocimiento (PR) y de Asignación de Comandos (AC).

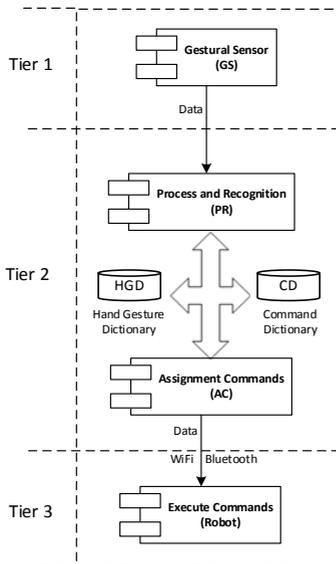


Fig. 5. Arquitectura del Sistema LEAPEV3.

Cada componente de la capa 2 cuenta con un Diccionario Gestual (HGD) y un Diccionario de Comandos (CD). Permite además la comunicación con la capa 3 usando el protocolo Bluetooth o WiFi. En nuestras pruebas hemos usado más frecuentemente el protocolo Bluetooth por su bajo consumo de energía.

- Componente PR: Éste componente aplica los algoritmos que permiten determinar las características del gesto y que logran su reconocimiento. Para identificarlo, compara las características del gesto con aquellas contenidas en el HGD implementado.
- Componente AC: Realiza la asignación de

comandos para cada uno de los gestos identificados por el PR. Cada gesto se asocia a un comando previamente establecido y configurado en el CD (Tabla I) y que será ejecutado en el robot. Es accesible por el usuario para definir las posibles relaciones entre comandos-acciones.

LEAPEV3 posee también la opción de visualización remota de la consola del robot (por defecto). Esta consola refleja en estado del robot y permite, entre otras cosas, tener un informe inmediato de los parámetros de funcionamiento, tales como: su nivel de batería, estado de la conexión, etc. El modo de operación puede alternar entre manual y automático. Por defecto está en automático.

El Algoritmo I muestra el flujo de datos al momento de leer un gesto y sus condiciones.

ALGORITMO I

LECTURA DEL GESTO DE CONTROL

- Mientras (existe operador)?
 - lee el gesto
 - Si (gesto es válido)?
 - Muestra gesto leído en pantalla
 - Ejecutamos el comando
- Señal de alto (stop) a todos los motores

Capa Tres: Está conformada por el bloque EV3 ® en su versión GRIPP3R, un modelo que permite movilidad y transporte. Posee libertad de movimientos (hacia adelante, hacia atrás, giro izquierda y giro derecha) y con una mano electromecánica, con sensor de contacto, que le permite tomar y soltar un objeto.

El Algoritmo II describe la conexión con el robot y la secuencia de inicio de sus componentes. Se debe contar con las librerías de reconocimiento del bloque (*BrickFinder, BrickInfo*), conexión remota (*RemoteBTDevice*), manejo remoto de consola y motores (*RMIMenu, RMIReglatedMotor*) y de identificación remota (*RemoteEV3*). Es necesario establecer conexión con el robot para usar el lenguaje gestual establecido.

ALGORITMO II

CONEXIÓN E INICIALIZACIÓN DEL ROBOT

- Creamos una nueva instancia de conexión Bluetooth o WiFi
 - Si (establecido conexión)?
 - Se asigna conexión por defecto (Bluetooth o WiFi).
 - Se muestra mensaje de conexión en consola
 - Se asigna nombre al EV3 por defecto
 - Se inicializan los motores y sensores
 - Nombre del motor
 - Tipo del motor
 - Sensores disponibles
 - Se inicializa GS
 - No, entonces se generan excepciones

B. Gestos de Control

Los gestos de control usados se muestran en la Tabla I y se ajustan a los porcentajes de aceptabilidad, expresados en la Sección II.d y obtenidos luego del análisis del reconocimiento gestual con el GS. Se han considerado gestos a partir del nivel medio, a excepción del gesto f) que lo usamos para contraste.

TABLA I
 GESTOS DE CONTROL EN EL SISTEMA LEAPEV3

Comando	Acción	Comando	Acción
	Abre la mano del robot. Inicia la interacción.		Cierra la mano del robot. Toma un objeto.
	Ordena giro a la derecha.		Ordena giro a la izquierda.
	Ordena avanzar linealmente.		Ordena retroceder. El inicia reversa.

C. Interface gestual

La interfaz gestual implementada se muestra en la Fig. 6. Permite al usuario el control gestual (automático) y manual del robot, en caso de requerirlo. Ha sido desarrollada en Java, usando objetos Swing por compatibilidad con el SDK de EV3. Cada gesto leído por el GS, es mostrado en pantalla usando una ventana informativa que indica al usuario el gesto detectado y el comando ejecutado. El usuario se mantiene informado de las acciones que el robot realiza en el mismo instante en que se producen.



Fig. 6. Descripción de LEAPEV3.

Como parte de las políticas de seguridad en control reactivo robótico o electrónico, hemos implementado una rutina de parada o stop inmediato en motores, al no detectar operador. El Algoritmo 1 permite observar que al no detectar mano alguna sobre el dispositivo GS, se ejecuta la rutina de parada. Este estado se mantiene hasta detectar el gesto de inicio de interacción (gesto (a)). Las operaciones continúan inmediatamente después de haber iniciado la interacción con el robot.

IV. RESULTADOS Y DISCUSIÓN

A. Del análisis:

Después de realizar el análisis propuesto en la Sección II.d, se obtuvo los totales se muestran en la Tabla II.

Al aplicar la ecuación (1) en los gestos propuestos para análisis en nuestra investigación (Fig. 3), se alcanzaron los siguientes porcentajes de PR: Gesto (a) un 83.21%, gesto (b) un 91.04%, gesto (c) un 76.14%, gesto (d) un 63.20%, gesto (e) un 46.19%, gesto (f) un 7.46%, el gesto (g) un 88.51% y el gesto (h) un 1.27%. Fue

notorio que algunos gestos no llegaron al nivel medio sugerido, por lo tanto no garantizan un adecuado comportamiento en reconocimiento y precisión para ser usados en control de un dispositivo robótico.

TABLA II
 ANÁLISIS DEL FRAME QUE COMPONE UN GESTO

Gestos	Válidos (f_v)	Positivos (f_g)	Negativos (f_n)
(a)	6332	5269	1063
(b)	13318	12125	1193
(c)	6597	5023	1574
(d)	5954	3763	2191
(e)	5365	2478	2887
(f)	4771	356	4415
(g)	6198	5486	712
(h)	7587	96	7491

Se pudo notar además, que un gesto puede alcanzar una buena exactitud y tener una mala precisión o puede tener una excelente precisión pero muy mala exactitud. Este análisis nos permitió determinar los valores referenciales de precisión (PR) para un gesto de control.

B. Del caso de estudio:

La interfaz gestual para control robótico LEAPEV3 basada en el LMC demostró ser válida para esta tarea manteniendo una tasa de muestreo lo suficientemente estable para garantizar el reconocimiento del gesto y la ejecución inmediata del comando. Al alternar entre el modo de trabajo automático y manual, no presentó problema alguno. Los gestos propuestos para control y que cumplen la escala de aceptación, fueron reconocidos y ejecutados adecuada e inmediatamente, permitiendo así un control efectivo del robot. Las tareas realizadas por el robot, tanto de carga y descarga, transporte de objetos, movimientos (circulares y lineales) fue precisa usando los diccionarios gestual y de comandos implementados. Esto permitió validar los resultados obtenidos para un gesto de control en precisión y en escala de aceptación, estableciendo así un método de evaluación para estudios similares.

Se pudo notar además, que el gesto h), tomado como contraste en los gestos de control, presentó en ocasiones un comportamiento errático e inconsistente, provocando inclusive problemas de *overload* en la transmisión de datos hacia el robot (Bluetooth y WiFi). Esto debido a la latencia introducida por su escasa precisión al intentar ser reconocido y ejecutar el comando de control respectivo.

También se pudo determinar el efecto perturbador del ruido lumínico introducido a propósito en las pruebas del sistema, realizadas en ambientes agresivos (con demasiada iluminación fluorescente o solar). Debido a ello, cubrimos al LMC para lograr un ambiente operativo normal con una cubierta lo suficientemente amplia (25 cm de altura x 50 cm de ancho), para garantizar una mejor respuesta y permitir la movilidad de la mano del operador. Se usó una caja, a la cual

llamamos Caja de Control (CC o Control-box) que dio excelentes resultados de aislamiento y cuya posible posición se indica en Fig. 7.

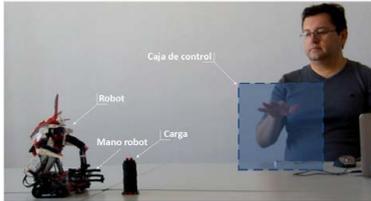


Fig. 7. Caja de Control y detalles del sistema robótico EV3 GRIPP3R.

Hemos llamado Consola de Administración Gestual (CAG) a la unión de la Caja de Control con el dispositivo LMC en su interior.

Por último, es necesario indicar que los gestos a usarse en tareas de control, deben ser fáciles de realizar, cómodos de mantener (inclusive por espacios prolongados de tiempo) y sencillos de recordar. Leap Motion, ha demostrado ser un sensor de captura gestual muy exacto, reconociendo inclusive aquellos gestos manuales que involucran movimientos mínimos de uno o más dedos.

V. CONCLUSIONES

En este artículo se ha implementado un Interfaz Natural de Usuario de tipo gestual llamada LEAPEV3, aplicada a tareas de control reactivo en un dispositivo robótico usando el controlador Leap Motion. Se realizó un análisis de precisión del gesto, para garantizar su uso en tareas de control, concluyendo en un método de análisis para determinar la validez o no de un gesto en tareas de este tipo. Se creó una escala de aceptación del gesto, para tareas de control. Se sugiere que un gesto para ser considerado como gesto de control, en sistemas de control reactivo, debe cumplir al menos con un valor de $PR (>=33\%)$ en la escala de aceptación, creada en esta investigación.

Se pudo establecer además, que el controlador Leap Motion se adapta a tareas de control donde se requiera una rápida identificación del gesto, y en las que, se usen consolas de administración gestual con presencia obligatoria del operador. Se debe evitar en control con LMC, aquellos gestos que involucren movimiento de mano y brazo al mismo tiempo, o de dedos, mano y brazo en su conjunto.

Como trabajo futuro hemos planteado la creación de una nomenclatura gestual completa, para control de dispositivos robóticos o electrónicos. Los gestos propuestos podrán ser usados en la industria, el comercio, la educación o cualquier lugar donde se desee utilizar una Interface Natural de Control de tipo Gestual.

AGRADECIMIENTOS

Este trabajo esta soportado por la Secretaría de Educación Superior, Ciencia, Tecnología e Innovación del Ecuador (SENESCYT) y por licencia de la Universidad Nacional de Chimborazo a través su

programa de formación para Ph.D con la Universidad de Granada, España.

REFERENCIAS

- [1] M. Nielsen, M. Störring, T. B. Moeslund, and E. Granum, "A procedure for developing intuitive and ergonomic gesture interfaces for HCI," in *Gesture-Based Communication in Human-Computer Interaction*, 2004, no. August 2015, pp. 409–420.
- [2] S. Mitra and T. Acharya, "Gesture Recognition: A Survey," *IEEE Trans. Syst. Man, Cybern. - Part C Appl. Rev.*, vol. 37, no. 3, pp. 311–324, 2007.
- [3] A. Sanna, F. Lamberti, G. Paravati, and F. Manuri, "A Kinect-based natural interface for quadrotor control," *Entertain. Comput.*, vol. 4, no. 3, pp. 179–186, Aug. 2013.
- [4] S.-W. Yang, C.-S. Lin, S.-K. Lin, and C.-H. Lee, "Design of virtual keyboard using blink control method for the severely disabled," *Comput. Methods Programs Biomed.*, vol. 111, no. 2, pp. 410–8, Aug. 2013.
- [5] R. Elakkiya, K. Selvamani, S. Kanimozhi, A. Kannan, and V. Rao, "Intelligent system for human computer interface using hand gesture recognition," *International Conference on Modelling Optimization and Computing*, vol. 38, pp. 3180–3191, 2012.
- [6] T. Piumsombon, A. Clark, M. Billinghurst, and A. Canterbury, "User-Defined Gestures for Augmented Reality," *Human-Computer Interact.*, pp. 282–299, 2013.
- [7] S. Lian, W. Hu, and K. Wang, "Automatic user state recognition for hand gesture based low-cost television control system," *IEEE Trans. Consum. Electron.*, vol. 60, no. 1, pp. 107–115, 2014.
- [8] G. Pomboza-Junez and A. Holgado-Terriza Juan, "Control of home devices based on hand gestures," *2015 IEEE 5th Int. Conf. Consum. Electron. - Berlin*, pp. 510–514, 2015.
- [9] D. Anastasiou and C. Stuhl, "LNCS 7383 - Gestures Used by Intelligent Wheelchair Users," no. 1982, pp. 392–398, 2012.
- [10] C. Kühnel, T. Westermann, F. Hemmert, S. Kratz, A. Müller, and S. Möller, "Im home: Defining and evaluating a gesture set for smart-home control," *Int. J. Hum. Comput. Stud.*, vol. 69, no. 11, pp. 693–704, 2011.
- [11] J. Blake, *Natural User Interface in .Net*, MEAP, New York, USA: Manning Publications, 2012.
- [12] O. Erazo and J. Pino, "Estimating the Difficulty of Touchless Hand Gestures," *IEEE Lat. Am. Trans.*, vol. 12, no. 1, pp. 17–22, Jan. 2014.
- [13] T. Vijitha and J. P. Kumari, "Finger Tracking In Real Time Human Computer Interaction," *Int. J. Comput. ...*, vol. 14, no. 1, pp. 83–93, 2014.
- [14] S. V. Rocha Rodriguez, Claudia Patricia Rodriguez; Jhon Alexander, Pineda Arias; Diego, "Traslator prototype of hand signals a text using Kinect," *Avances, Investig. en Ing.*, vol. 10, no. 2, pp. 64–72, 2013.
- [15] P. Breuer, C. Eckes, and S. Muller, "Hand Gesture Recognition with a Novel IR Time-of-Flight Range Camera: A Pilot Study," *Mirage07*, pp. 247–260, 2007.
- [16] F. Erden and A. E. Cetin, "Hand Gesture Based Remote Control System Using Infrared Sensors and a Camera," *IEEE Trans. Consum. Electron.*, vol. 60, no. 4, pp. 675–680, Nov. 2014.
- [17] J. Guna, G. Jakus, M. Poga?nik, S. Toma?i?, and J. Sodnik, "An analysis of the precision and reliability of the leap motion sensor and its suitability for static and dynamic tracking," *Sensors (Switzerland)*, vol. 14, no. 2, pp. 3702–3720, 2014.
- [18] F. Weichert, D. Bachmann, B. Rudak, and D. Fissler, "Analysis of the accuracy and robustness of the Leap Motion Controller," *Sensors (Switzerland)*, vol. 13, no. 5, pp. 6380–6393, 2013.
- [19] A. Ram, R. C. Arkin, K. Moorman, and R. J. Clark, "Case-based reactive navigation: A method for on-line selection and adaptation of reactive robotic control parameters," *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 27, no. 3, pp. 376–394, 1997.
- [20] J. Y. Tung, T. Lulic, D. a Gonzalez, J. Tran, C. R. Dickerson, and E. a Roy, "18. Evaluation of a portable markerless finger position capture device: accuracy of the Leap Motion controller in healthy adults," *Physiol. Meas.*, vol. 36, no. 5, pp. 1025–35, 2015.

eK – Red de control, actuación y optimización de consumo eléctrico para enchufes

José Manuel López Martínez, Jesús González Peñalver y Alberto Prieto Espinosa

Resumen—Una red de control para consumo eléctrico permite disminuir drásticamente el gasto en energía. En este caso, permite la configuración individual de cada dispositivo e integrarlo en una red comunicada inalámbricamente. Mediante la interfaz web disponible, se permite el acceso a los datos de consumo en estado real y cortar la energía eléctrica suministrada por cada uno de los enchufes en la red.

Palabras clave—Consumo eléctrico, enchufes, red, control, actuación, ahorro, energía, sistema empotrado, interfaz web.

I. INTRODUCCIÓN

EN la actualidad, se plantea el gran desafío de la sostenibilidad del planeta como uno de los puntos fundamentales para seguir creciendo. Las ciudades de hoy no son eficientes, consumiendo la mayor parte de los recursos energéticos del mundo, alrededor del 80%, siendo además, donde se generan la mayor parte de los gases que provocan el efecto invernadero.

La eficiencia energética es utilizar la energía de la mejor forma posible, evitando desperdiciarla en su uso y abuso. Disminuyendo el consumo energético en nuestra vivienda, no solo estamos cuidando el planeta, sino que estamos consiguiendo un ahorro energético considerable.

Con pequeños cambios en nuestros hábitos de consumo y desarrollando sistemas eficientes, algo actualmente posible gracias al avance de las TICs, podemos garantizar la protección del medioambiente y conseguir cerca del 100% de energía limpia [1].

El desarrollo de sistemas, similares al que se presenta en este documento, ha demostrado ser rentable en países similares al nuestro, generando beneficios tanto para el usuario como para la economía del país. Estos beneficios intrínsecos serían de entre 2 y 3.5 veces la inversión.

Por tanto, podemos incluir el presente desarrollo dentro del llamado *Smart Grid*, que ya está favoreciendo el crecimiento sostenible de la economía, mejorando el eficiencia energética global y generando empleo, siempre enmarcado dentro de los objetivos de despliegue de energías renovables previstos para el 2020 [2].

En este camino se enfoca el sistema desarrollado, de manera que sirva para obtener datos eléctricos de distintos puntos. Estos datos son analizados para poder actuar adecuadamente, procurando una mejor eficiencia en el uso de los recursos relacionados. Se ha definido un sistema de nodos comunicados con una red de bajo consumo que sigue el estándar IEC61850-15.4 en el que cada nodo tendrá distintos contadores/actuadores sobre el flujo eléctrico que tenga asociado. El usuario es el centro del sistema, pudiendo controlarlo por medio de la interfaz web desarrollada.

II. ESTADO DEL ARTE

En el mercado, encontramos diversas soluciones, empezando por las que permiten un apagado remoto por medio de un mando a distancia, estando en la misma habitación o muy cerca del receptor, como en [3]. Siguiendo por aquellas en las que el apagado remoto por medio de una aplicación web o móvil, véase [4] y [5], que además permite programar este encendido y apagado de forma automática.

Un sistema más avanzado obtiene estadísticas de uso y consumo del enchufe que monitoriza, [6]. Con un panel de control más robusto que los anteriores y siendo mucho más potente que éstos, su conexión es individual con el router *WiFi*, por lo que no forma una red independiente de sensores/actuadores que colaboren entre ellos para transmitir la información.

El producto comercial que más se acerca al aquí presentado es el desarrollado por *wattio*, llamado *POD* (véase [7]). Permite el apagado/encendido del enchufe que controla y obtiene estadísticas de uso de la corriente, poniéndolas a disposición del usuario del usuario a través de una plataforma web y móvil. Además crea una red inalámbrica de bajo consumo, basada en *ZigBee*, a la que se conectan distintos dispositivos de este u otro tipo del mismo fabricante. Todos ellos están controlados por un *GATE* que es el equivalente al coordinador en nuestra red. La principal ventaja de nuestro sistema, en cuanto a usabilidad y flexibilidad, respecto al presentado en [7], es que la red inalámbrica desarrollada es capaz de superar más obstáculos ya que, si el lugar de instalación lo requiere, se pueden utilizar transceptores de 868 MHz que se ajustan al protocolo aquí desarrollado, siendo un punto a favor respecto los de 2.4 GHz que usa *ZigBee* para las comunicaciones del sistema descrito en [7].

La tecnología de comunicación a 868 MHz permite una distancia de transmisión mayor, a costa de reducir la tasa de datos respecto a la que se puede enviar usando la frecuencia de 2.4 GHz. Dado que la tasa de datos a enviar en nuestro sistema es baja, es posible realizar comunicaciones en la frecuencia 868 MHz de manera que tengamos una distancia de transmisión mayor y podamos salvar una cantidad mayor de obstáculos. Esto posibilita que sean necesarios menos cantidad de dispositivos intermedios para comunicar los nodos más alejados entre sí.

Existen dos razones fundamentales que justifican este fenómeno: la potencia de transmisión de radiofrecuencia y las pérdidas por propagación. A medida que la onda de radio se propaga por el aire, su intensidad decrece. Entonces, las señales de radio que se transmiten con una potencia mayor, viajarán más lejos antes de que se hagan demasiado débiles. Además, la señal de las ondas

de radio de mayor frecuencia disminuye mucho más rápidamente.

En Europa, los dispositivos de 2.4 GHz tiene una de RF regulada a un máximo de 100 mW, mientras que los de 868 MHz es de 500 mW, lo que significa que estos últimos tienen una distancia de transmisión fiable teóricamente cinco veces mayor. Típicamente, el rango de distancias para altas frecuencias está entre 30 y 100 metros, y de 1 kilómetro para las bajas.

A mayores frecuencias, la señal tiene mayor atenuación al penetrar obstáculos ya que tienes menos capacidad de doblarse alrededor de un obstáculo y pierde mayor fuerza al reflejarse sobre ellos. En teoría, la atenuación de una señal aumenta cuadráticamente con la distancia de transmisión

Esta última conclusión, se deriva de la investigación realizada por investigadores del CEA (*Commissariat à l'Énergie Atomique*) de Francia, disponible en [8], que llevó a cabo la caracterización de un canal de propagación para una WSN dentro de un vehículo. Para ello, se estudió el efecto del entorno, la orientación de la antena y la ubicación del nodo, para las bandas 2.4 GHz y 868 MHz. Los resultados obtenidos indican que en la banda de 868 MHz, existían menos pérdidas de datos y menos efectos del entorno, en comparación con la banda de 2.4 GHz cuyas mayores pérdidas se deben a la reflexión de la señal dentro del vehículo (obstáculos) y el efecto de la propagación de la señal electromagnética. A esto, hay que añadir la alta congestión de la banda de 2.4 GHz, por ejemplo de routers WiFi en oficinas o casas, ordenadores y teléfonos móviles con Bluetooth activo, hornos microondas, que provocan muchas interferencias en el medio. Así, un espectro sin ruido permite una mayor eficiencia en la comunicación y una reducción del consumo.

Aquí es donde radica la principal ventaja de nuestra comunicación inalámbrica ya que, al ser una implementación propia basada en el protocolo IEEE802.15.4, permite el uso sobre distintas bandas de frecuencia sin tener que hacer uso de implementaciones más a alto nivel y de módulos certificados por ZigBee que pueden o no (como es el caso de *wattio*) comunicar en frecuencias sub-GHz.

III. DISEÑO HARDWARE

A. Contando la Corriente

Para tomar las medidas oportunas de consumo de la red eléctrica se ha utilizado un sensor que envuelve unos centímetros el cable para detectar las variaciones de flujo eléctrico que circula por el mismo. Necesita de un circuito de acondicionamiento formado por una fuente de corriente para la polarización del sensor, un amplificador para darle ganancia a la magnitud de la señal y filtro paso bajo. Con este acondicionamiento se pretende fijar la señal entre unos rangos que le permita ser adquirible para los niveles lógicos de un microcontrolador. Estos sensores se basan en el efecto Hall, descubierto en 1879 por el Dr. Edwin Herbert Hall.

El efecto Hall consiste en la aparición de un campo eléctrico en un conductor cuando es atravesado por un campo magnético. A este campo eléctrico se le conoce



Fig. 1. Sensor de corriente SCT-013 utilizado

como campo Hall. Cuando por un material conductor o semiconductor circula una corriente eléctrica, estando en el seno de un campo magnético, aparece una fuerza magnética que provoca el campo magnético. Ligado al campo Hall aparece la tensión Hall que es medible mediante un voltímetro.

El voltaje obtenido, proporcional al campo magnético y a la corriente eléctrica, es del orden de los microvoltios por lo que debe amplificarse con un circuito de acondicionamiento para los valores de entrada analógicos de nuestro microcontrolador [9].

Se toman una cantidad suficiente de muestras y cada una de ellas, tras ser filtradas, son tomadas en valor absoluto y se acumulan en una variable global. Una vez tomada la cantidad de muestras adecuada, se divide esta variable acumuladora por el número de muestras, obteniendo la media de ellas que es multiplicada por una constante para calibración del sensor, calculada según las características del sensor usado:

- Valor de la resistencia de carga.
- Número de vueltas del sensor.
- Constante de corriente, según el coseno del desfase de la red.

Este valor final, es tomado como Amperios que circulan por la red.

Para obtener el valor de la tensión en ese mismo instante de tiempo, se utiliza un estimador software que sitúa el valor que debería tener según la posición que tendría en su onda sinusoidal asociada. Para ello se considera el tiempo pasado entre una muestra y la anterior.

El sensor mostrado en Fig. 1 (de la serie **SCT-013**, [10]) ha sido el utilizado ya que es capaz de medir corrientes hasta los 100 Amperios y tiene la resistencia de carga ensamblada en su interior. Además ofrece una instalación sencilla, no intrusiva en la red eléctrica.

Se hace necesario el uso de una circuitería intermedia entre éste sensor y el voltaje de entrada de nuestro microcontrolador. En Fig. 2 se muestra el usado a partir del desarrollo explicado en [12].

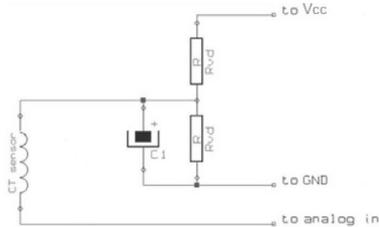


Fig. 2. Circuitría intermedia para sensor de corriente

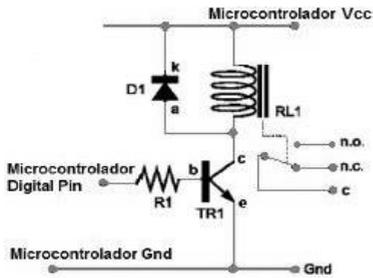


Fig. 3. Circuitría intermedia para relay

B. Actuando sobre la Corriente

Para actuar sobre la red eléctrica se ha utilizado un relé que, como interruptor, podrá cortar/activar el paso de corriente eléctrica por el cable asociado después de que el microcontrolador interprete correctamente las instrucciones recibidas del usuario.

Si se le aplica un voltaje adecuado a la bobina se genera un campo magnético haciendo que un contacto u otro hagan conexión [11].

La gran ventaja de los relays electromagnéticos es la completa separación entre la corriente de accionamiento, la que circula por la bobina del electroimán, y los circuitos controlados por los contactos, lo que hace que se puedan manejar altos voltajes o elevadas potencias con pequeñas tensiones de control.

En nuestro caso, al tener microcontroladores que funciona a 5V durante la operación, utilizaremos relays que necesiten esos 5V para generar el campo magnético necesario para conmutar. La circuitría intermedia para que los valores energéticos se mantengan constantes mientras se desee es la esquematizada en Fig. 3, donde, además del relé (RL1), se ha utilizado un diodo (D1) de tipo 1N4007, un transistor (TR1) de tipo 2N2222 y una resistencia de 1kΩ.

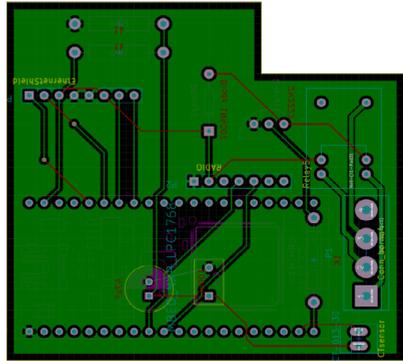


Fig. 4. PCB para dispositivo coordinador

C. Microcontroladores

Nuestro sistema tiene dos tipos de dispositivos ya que uno de ellos debe ser capaz de conectarse vía Ethernet (dispositivo **coordinador**) y será el principal de la red de nodos. El resto de dispositivos en la red será de tipo **esclavo** y, en este caso, serán de un tamaño más reducido.

Siempre intentando disminuir el tamaño final de los dispositivos y, a la vez, facilitar su fabricación y puesta en marcha, se han utilizado placas de desarrollo fabricadas por *mbed* y *Arduino*.

Para el caso del dispositivo **coordinador** se ha usado la placa de desarrollo *mbed NXP LPC1114* debido a que incorpora una interfaz Ethernet que se adapta perfectamente a las comunicaciones que se desarrollan con el usuario. Además del puerto UART que es utilizado para las comunicaciones inalámbricas con el resto de dispositivos de la red.

Para el caso del dispositivo **esclavo** se ha seleccionado un *Arduino Mini Pro* debido principalmente a su reducido tamaño y a que tiene un puerto UART con el comunicarlo con el dispositivo coordinador de su red inalámbricamente.

D. Fabricación

Con los componentes detallados anteriormente, se ha fabricado la primera versión de la PCB para cada tipo de dispositivo y las cajas de plástico (imprimidas con impresora 3D). Véase Fig.4 y Fig. 5.

Estas placas, con sus componentes soldados, han sido las utilizadas para las pruebas experimentales que se detallan más adelante.

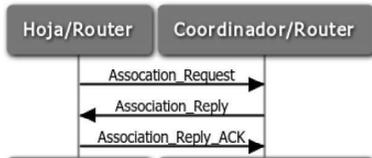


Fig. 7. Flujo de mensajes para operación de asociación

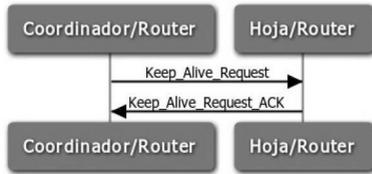


Fig. 8. Flujo de mensajes para control de estados con éxito

- Operación de control de estado de la red:

Dado un determinado tiempo, los nodos padre sondan a sus hijos y esperan respuesta de estos para actualizar sus tablas de enrutamiento. Existen dos posibles situaciones:

- Con éxito. El nodo hijo controlado responde correctamente (véase Fig. 8).
- Con errores: Si el nodo hijo no responde correctamente en el tiempo esperado, se realizan cinco reintentos más. Si todos falla, el enlace se desactive y se elimina al hijo de las tablas de enrutamiento.

- Operación de disociación de nodos:

El nodo padre puede solicitar a un nodo hijo que abandone su red, enviando un mensaje de tipo *Disassociation Request*. Recibirá un mensaje *Disassociation Request ACK* y lo eliminará de sus tablas de nodos hijos.

- Operación de envío de datos:

Cualquier tipo de nodo puede ser tanto emisor como receptor, dependiendo de la función del tipo de mensaje. El proceso de comunicación se compone por el envío de la información:

- COUNTER_DATA*. Contiene datos de consumo de corriente del nodo emisor. Su envío procede desde nodo hijo como emisor y su nodo padre como receptor.
- COUNTER_ACTION*. Contiene información sobre apertura o cierre del relay del receptor. Su envío

procede desde un nodo padre como emisor y un nodo hijo como receptor.

A ambos tipos de mensajes responderá el receptor con la confirmación de recepción (*ACK*), de tipo *COUNTER_DATA_ACK* o *COUNTER_ACTION_ACK* respectivamente.

B. Comunicación Ethernet Sistema-Usuario

Para satisfacer esta importante parte del proyecto, se ha utilizado el protocolo *Message Queue Telemetry Transport (MQTT)*. Un protocolo abierto para envío y recepción de mensajes que permite la transferencia de datos telemáticamente desde distintos tipos de dispositivos a un servidor (*broker*).

Algunas de las características clave para su elección han sido:

- Simplicidad. Construir un elemento sólido que pueda ser integrado para diferentes soluciones.
- Publicación/subscription. Permitir a los dispositivos decir cosas nuevas en los llamados *tópicos*.
- Reducir la administración por parte del usuario. El sistema debe ser capaz de responder con sensatez a nuevos eventos que ocurran y actuar dinámicamente para llevar a cabo su respuesta correspondiente.
- Ser ligero para poder funcionar en un ancho de banda reducido. Se espera que las aplicaciones que usen este protocolo como cliente tengan recursos de procesamiento reducidos.
- No obliga a un formato de contenido concreto. Aumenta la flexibilidad.

Para utilizar correctamente MQTT, necesitamos instalar el *broker*, su IP y los nombres *tópicos* ([13], [14]) a los que llegarán las publicaciones de los dispositivos y en los que reciben las acciones. Éstos son:

- contadores/pub/nombreDisp*
Cada dispositivo publica las medidas de consumo eléctrico tomadas según su nombre o número de dispositivo (*nombreDisp*).
- contadores/sub/nombreDisp*
Cada dispositivo, según *nombreDisp*, recibe en este tópico las órdenes de encendido/apagado de la corriente.
- contadores/red*
El dispositivo coordinador publica en él la red de dispositivos que mantiene en cada momento.
- contadores/names*
En él se escriben los nombres que el usuario desea poner a un dispositivo. Para ello se publicará un mensaje con el número identificativo del dispositivo, seguido por el nuevo nombre que se le asocia.

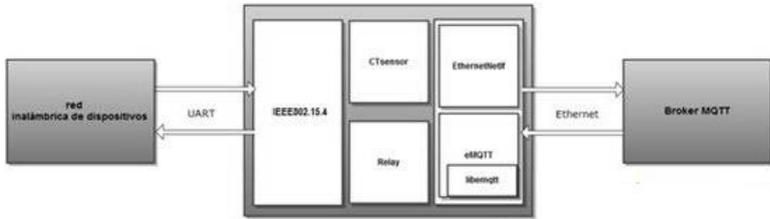


Fig. 9. Módulos software para coordinador.

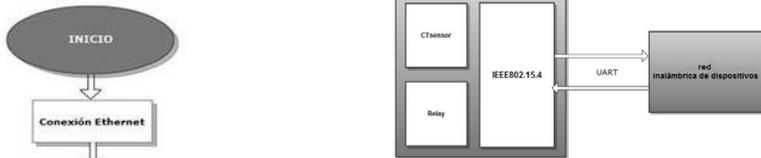


Fig. 11. Módulos software para esclavo.

C. Funcionamiento de dispositivos

Cada tipo de dispositivo (coordinador o esclavo) está compuesto por distintos módulos según las funciones que realiza.

1. Dispositivo coordinador

Se encarga de controlar y actuar sobre el consumo eléctrico en su enchufe. Además se comunica vía inalámbrica con el resto de dispositivos y con el usuario vía Ethernet. En la Fig. 9 podemos ver cuál es el esquema general de los distintos módulos implementados y cómo interactúan entre ellos.

En el diagrama de flujo de Fig. 10, se explica esta interacción durante el ciclo de funcionamiento del dispositivo.

2. Dispositivo esclavo

La diferencia con el anterior, además de su tamaño, es que no se comunica vía Ethernet con el usuario. Los módulos software que utiliza y la interacción entre ellos durante su ciclo de funcionamiento, se resume en los diagrama de las figuras número 11 y 12.

En Fig. 12, vemos como el dispositivo realiza una acción u otra según lo que recibe del dispositivo coordinador. Estas acciones pueden ser cortar o permitir el paso de corriente (valores 0 ó 1) si el usuario así lo ha indicado, u obtener una nueva medida de la corriente en caso de que no se haya indicado otra acción.

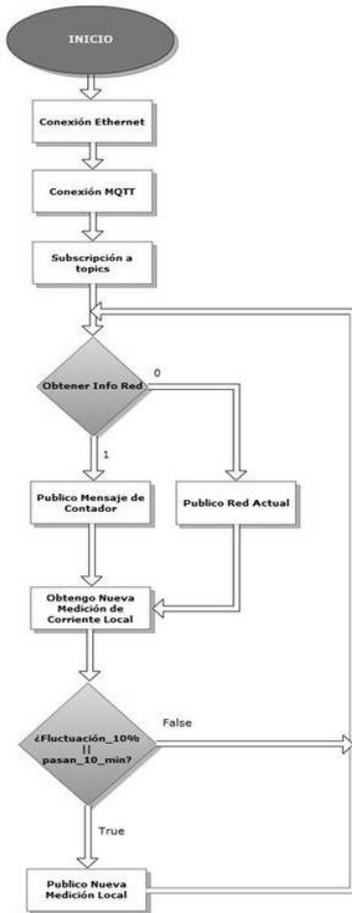


Fig. 10. Diagrama de flujo para coordinador.

V. INTERFAZ DE USUARIO

Para facilitar el manejo del sistema al usuario, se ha realizado una interfaz web que se puede utilizar en cualquier navegador web. Para ello debe estar conectado

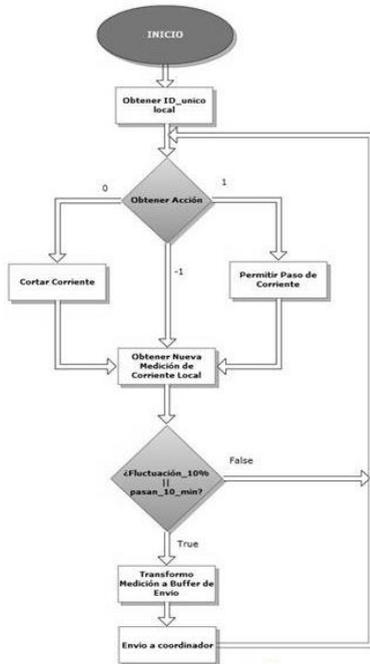


Fig. 12. Diagrama de flujo para esclavo.

a la misma red a la que se conecta el dispositivo coordinador para enviar los datos del sistema via *Ethernet*.

Se ha incluido una base de datos para almacenar los datos de consumo y actuación que lleguen de los dispositivos, así como el historial de nodos que se han conectado.

El panel principal de control que encuentra el usuario facilita el acceso a cada uno de los dispositivos y al sistema completo, tal y cómo se observa en Fig. 14. En él, se pueden modificar el nombre del dispositivo, la foto que lo describe y, pulsando sobre cualquiera de ellos, vemos los datos de consumo registrado. Además de poder encenderlo o apagarlo para que deje de consumir (véase Fig. 14)



Fig. 13. Interfaz web principal de control.



Fig. 14. Ventana de consumo de un dispositivo

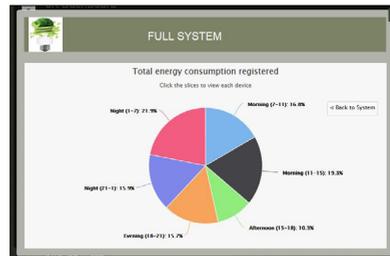


Fig. 15. Ventana de consumo por franjas horarias

Por otro lado, pulsando en el icono identificado como *FULL SYSTEM* en el panel principal, aparece una ventana web con información de consumo de todo el sistema completo. Primero dividido por cada uno de los dispositivos y, pulsando sobre una de las porciones, aparece el consumo repartido por franjas horarias del dispositivo que corresponda, tal y cómo se indica en Fig. 15.

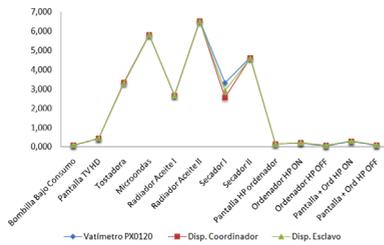


Fig. 16. Gráfica comparativa de medidas.

TABLA I
 COMPARACIÓN DE MEDIDA. TOMADAS CON VATÍMETRO Y CADA TIPO DE DISPOSITIVO DEL SISTEMA. MEDIDAS EN AMPERIOS.

	Vatímetro PX0120	Dispositivo Coordinador	Dispositivo Esclavo
Bombilla bajo consumo	0,073	0,086	0,095
Pantalla TV	0,430	0,430	0,430
Tostadora	3,330	3,330	3,330
Microondas	5,810	5,810	5,800
Radiador Aceite I	2,680	2,660	2,660
Radiador Aceite II	6,530	6,520	6,500
Secador I	3,310	2,550	2,910
Secador II	4,620	4,620	4,580
Pantalla PC	0,136	0,144	0,140
Ordenador ON	0,198	0,201	0,208
Ordenador OFF	0,050	0,060	0,055
Pantalla+Ord ON	0,198	0,201	0,208
Pantalla+Ord OFF	0,070	0,080	0,080

VI. RESULTADOS EXPERIMENTALES

Los resultados obtenidos en el consumo identificado en el sistema han sido verificados con material certificado, como es el vatímetro *PX0120* de *Metrix* [16]. Utilizándose distintos útiles y electrodomésticos eléctricos, se han obtenido las medidas (en Amperios), detalladas en Fig. 16 y en Tabla I.

Calculando el error relativo medio de cada tipo de dispositivo, se obtienen errores muy bajos. Éstos son del 2,85% para el dispositivo coordinador, y del 3,26% para el esclavo.

Podemos decir que, aunque se pueden reducir aún más, dado el bajo coste que se estima del sistema, son muy buenos resultados para facilitar la optimización del consumo eléctrico del usuario.

VII. CONCLUSIONES Y DISCUSIÓN DE RESULTADOS

Para finalizar, se concluye que el sistema puede ayudar significativamente al ahorro en corriente

eléctrica y sobre todo determinar de dónde provienen los distintos gastos derivados en la factura. Esta afirmación se puede justificar con las siguientes razones:

- Se ha comprobado que las medidas de corriente eléctrica obtenidas con este sistema lo suficientemente buenas como para tener una estimación muy aproximada del consumo real que va a tomar la empresa suministradora de electricidad para realizar la factura correspondiente.
- Permite tener un control mucho más restringido de qué es lo que más consumo produce y tras analizar las muestras poder reducir el gasto según las posibilidades de cada usuario.
- Las acciones definidas, activación/corte del flujo eléctrico, permiten al usuario mantener el consumo eléctrico dentro de los márgenes que desee pudiendo corte o activar el flujo de corriente eléctrica según las necesidades que tenga en cada momento y sin necesidad de estar físicamente en el lugar.
- La comunicación inalámbrica entre dispositivos se ha mostrado como muy eficiente en las pruebas realizadas sin tener problema alguno por obstáculos o lejanía, este último dentro de cierto margen.
- La comunicación inalámbrica desarrollada parte de la idea de conectar un gran número de dispositivos a una misma red. Esto puede ser de gran utilidad para comunidades que deseen controlar un gran número de conectores eléctricos.
- La forma en la que funciona el protocolo *MQTT* permite a los distintos dispositivos comunicarse sin esperar algún tipo de confirmación por parte del usuario. Esto le da gran flexibilidad y autonomía al sistema.
- Interfaz web de usuario que facilita enormemente el control y uso del sistema, siendo transparente a las operaciones de comunicación y obtención de consumo que se realizan dentro de él.

Por otro lado, el sistema es susceptible de mejoras, como son la actuación automática sobre la corriente según los datos de consumo, seguridad en comunicaciones inalámbricas, medidas de corriente trifásica o la inclusión de visualizadores en los propios dispositivos.

Respecto a la seguridad en las comunicaciones, que pronostica como primera medida a realizar, se propone proteger la estación base estableciendo caminos redundantes para unir ésta con los distintos sensores, y ocultarla de ataques externos, detectando intrusos.

Para ello, resulta inviable el uso de un IDS (Intrusion Detection System) para todos los nodos a la vez. Por lo que la resolución de este problema se centrará en saber

qué nodos proteger en cada momento, utilizando mecanismos de decisión basados en el aprendizaje o técnicas de teoría de juegos, donde la decisión en cada momento se tomará con la intención de maximizar la recompensa futura, según recompensas anteriores. En esencia, el estudio se basa en la defensa por parte del IDS del nodo mejor en términos de menor coste de protección y de mayor utilidad en la red [17].

REFERENCIAS

- [1] Hernández Muñoz, José Manuel. "¿Qué son las 'Smart Cities' o Ciudades Inteligentes?" [en línea]. Último acceso en Abril de 2016. Disponible en la Web: <https://telos.fundaciontelefonica.com/url-direct/pdf-generator?tipoContenido=articulo&idContenido=2011050916510001>
- [2] Smart Grids European Technology Platform. Varias publicaciones. [en línea]. Último acceso en Abril de 2016. Disponible en la Web: <http://www.smartgrids.eu/>
- [3] Efergy Store. "Efergy Socket Remote Control", [en línea]. Último acceso en Mayo de 2016. Disponible en la Web: <http://www.efergystore.com/es/ahorro-energetico/efergy-socket-remote-control.html>
- [4] Belkin International, Inc. "WeMo Insight Switch", [en línea]. Último acceso en Mayo de 2016. Disponible en la Web: <http://www.belkin.com/us/p/P-F7C029/>
- [5] EDIMAX Technology Co. "SP-1101W", [en línea]. Último acceso en Mayo de 2016. Disponible en la Web: http://www.edimax.es/edimax/merchandise/merchandise_detail/data/edimax/es/home_automation_smart_plug/sp-1101w/
- [6] D-LINK Europe Ltd. "DSP-W215", [en línea]. Último acceso en Mayo de 2016. Disponible en la Web: <http://www.dlink.com/es/home-solutions/mydlink-home/smart-plugs/dsp-w215-smart-plug>
- [7] Wattio, "POD", [en línea]. Último acceso en Mayo de 2016. Disponible en la Web: <https://wattio.com/es/tienda/pod-2>
- [8] R. D'Errico, L. Rudant, J. Keignart, "Channel characterization for intra-vehicle WSNs in the ISM bands", in Antennas and Propagation (EuCAP), 2010 Proceedings of the Fourth European Conference, pp.1-5, 12-15. Abril 2010.
- [9] Tamura Corp. "Current Sensor Information Page", [en línea]. Último acceso en Abril de 2016. Disponible en la Web: https://www.digikey.com/Web%20Export/Supplier%20Content/Tamura_132/PDF/Tamura_Basic_Current_Sensor_Info.pdf
- [10] ITeard Studio. "Split-Core Current Sensor", [en línea]. Último acceso en Abril de 2016. Disponible en la Web: ftp://imall.iteardstudio.com/Sensor/IM120628008/Specs_IM120628008.pdf
- [11] RALUX. "DIL miniature relay. Mod R", [en línea]. Último acceso en Abril de 2016. Disponible en la Web: <http://www.ralux.com/archivos/upload/R.pdf>
- [12] Lea, Trystan. "Mains AC: non-invasive version 3.0 up", [en línea]. Último acceso en Abril de 2016. Disponible en la Web: <http://openenergymonitor.blogspot.com.es/2010/04/mains-ac-non-invasive-version-30-up.html>
- [13] IBM, Information Center. "Publicación/Subscripción. Semántica y uso de temas", [en línea]. Último acceso en Abril de 2016. Disponible en la Web: <http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r1m0/index.jsp?topic=com.ibm.etools.mft.doc/aq13300.htm>
- [14] Mosquitto, An Open Source MQTT v3.1 Broker. "Manual Index", [en línea]. Último acceso en Abril de 2016. Disponible en la Web: <http://mosquitto.org/man/>
- [15] Atmel Corporation. "Single-Chip Solutions. IEEE 802.15.4 MAC", [en línea]. Último acceso en Abril de 2016. Disponible en la Web: http://www.atmel.com/tools/IEEE802_15_4MAC.aspx
- [16] Metrix Corporation. "TRMS three and single phase digital wattmeters", [en línea]. Último acceso en Abril de 2016. Disponible en la Web: <http://datasheet.octopart.com/PX0120-Metrix-Electronics-datasheet-518125.pdf>
- [17] A. Agah, S. K. Das and K. Basu, "A Non-cooperative Game Approach for Intrusion Detection in Sensor Networks", IEEE Vehicular Technology Conference (VTC), Los Angeles, CA. Octubre 2004

Sistema abierto para la adquisición de datos desde GCMs propietarios

J. Berián, A. Gardel, I. Bravo, C. Vázquez, J.L. Lázaro
Dpto. Electrónica – Universidad de Alcalá

Resumen—La diabetes es un problema de talla mundial y existen diferentes sistemas comerciales para la detección del nivel de glucosa en sangre. Al ser sistemas cerrados no se permite modificar su uso, y tampoco mejorarlo por terceras partes. En este artículo se aborda el diseño de un *transceiver* que permite la captura de datos de un sensor continuo de glucemia y que, conectado a un teléfono móvil, sirve de pasarela para almacenar sus medidas en una base de datos fácilmente explotable con servicios en la nube. Esta funcionalidad permite la monitorización de los valores de glucosa en sangre por parte de un médico, el mismo paciente, o persona adulta que esté a cargo de un menor con esta enfermedad y que pueda así tomar las acciones correctivas que más convenga realizar. El sistema es de bajo coste y permite descargar el código de forma libre para que un usuario pueda configurarlo de forma sencilla.

Palabras clave—Monitorización de glucosa, pasarela RF-USB, protocolo, HW/SW abierto.

I. INTRODUCCIÓN

ESTE artículo presenta el diseño de un *transceiver* que permite la captura de datos de un sensor continuo de glucemia y que, conectado a un teléfono móvil, sirve de pasarela para almacenar sus medidas en una base de datos fácilmente explotable.

La problemática que introduce el uso de sensores con protocolos propietarios hace necesario el desarrollo de un sistema abierto que ofrezca los datos recabados por el sensor a terceras partes y aumentar el desarrollo de otros sistemas y dispositivos que puedan mejorar el estado actual de los sistemas de control de la glucosa en sangre.

Tras el estudio del protocolo de comunicaciones del sensor utilizado, se hizo el desarrollo de la plataforma utilizando un *System-On-Chip* de Texas Instruments que permitiera trabajar en la misma banda de frecuencia (TI CC1111). A su vez, se diseñó el *transceiver* con el objetivo de presentar un interfaz amigable al sistema en el que se desee utilizar. En este caso la opción fue un puerto serie estándar USB y el kit de desarrollo del fabricante encajó perfectamente en las necesidades, evitando así el desarrollo de plataformas HW propietarias.

Para demostrar su viabilidad, se ha hecho uso de un software ad-hoc para teléfonos Android que permite el tratamiento de estos mensajes y, posteriormente, se almacenan en una base de datos en la nube. El sistema completo proporciona un mecanismo, no invasivo en cuanto al sistema del fabricante, para que un usuario pueda almacenar todos los registros relacionados con la

diabetes en un servidor online, con acceso remoto en la nube [1]. Este desarrollo no forma parte del trabajo presentado pero sirve de ejemplo en cuanto a los múltiples usos que se puede hacer del mismo.

II. SISTEMAS DE MONITORIZACIÓN DE GLUCOSA

Actualmente existen diversas tecnologías para monitorizar el nivel de glucosa. Idealmente, los sensores utilizados para tal fin deberían tener una alta sensibilidad respecto a variaciones de la glucosa en sangre, repetibles, con una respuesta rápida y sobre todo de bajo coste dado el elevado número de pacientes con diabetes en el mundo.

Dentro de los sensores que permiten monitorización continua [2], encontramos aquellos que se basan en la reacción de oxidación de la glucosa y los que utilizan tecnologías ópticas o de ultrasonidos. En [3] y [4] se realiza una revisión de las tecnologías utilizadas en algunos dispositivos comerciales y de las tendencias en las tecnologías de monitorización de glucosa.

Los medidores de glucosa continuos GCM (Glucose Continuous Monitoring) necesitan de una calibración inicial, siempre que se reponga el elemento sensor [5]. Así pues, cuando se sustituye el sensor hay que dejar un tiempo muerto de 120 minutos hasta que puede realizarse la primera calibración, que requiere dos muestras. Tras la primera calibración, el fabricante recomienda calibrar al menos una vez cada 12 horas mediante una medida de glucosa en sangre.

Los dispositivos comerciales disponen de un sistema de calibración que solicita al usuario medidas tomadas con un dispositivo *fingerstick*, cuya función principal es ajustar la ganancia entre la glucosa medida en el fluido intersticial y la glucosa en sangre real, ignorando el retraso entre ambos valores [6].

Por otro lado, los datos que se muestran por los sistemas que capturan la información del sensor GCM y dan la información al usuario son datos filtrados, siendo el tipo de filtro utilizado desconocido en soluciones propietarias [7]. En particular para el sistema GCM, éste realiza una media ponderada de k muestras. Así pues, el hecho de poder recuperar la muestra real obtenida desde el GCM sin pasar por otros sistemas de procesamiento puede proporcionar una mayor información al usuario o ser útil para aplicar otros modelos de procesamiento, a partir de un sistema de captura de datos abierto para usar con cualquier dispositivo GCM.

En la actualidad existen diversas soluciones comerciales para la medición intersticial de glucosa y algunas de ellas pueden ser utilizadas para el fin de este trabajo. Entre los fabricantes de este sector podemos destacar como más influyentes a Medtronic, Dexcom y

Abbott. Sólo los dos primeros disponen en la actualidad de sensores que envíen sus datos por radiofrecuencia sin interacción del usuario, mientras que el último depende de un lector NFC y la interacción del paciente para su lectura. Medtronic tiene en el mercado dos tecnologías distintas para transmisión de datos: una basada en modulación OOK en la banda ISM de 868/916MHz y otra basada en IEEE 802.15.4 en la banda ISM 2.4GHz. En el caso de Dexcom sucede algo similar: dispone de una solución basada en SimpliciTI (protocolo abierto de Texas Instruments) en la banda de 2.4GHz y otra basada en Bluetooth Low Energy en la misma banda.

Para el sistema aquí propuesto se tomó como objetivo el análisis de los datos procedentes de un sensor Medtronic con modulación OOK.

III. DIAGRAMA DE BLOQUES DEL SISTEMA PROPUESTO

El bloque de funcionamiento propuesto para el sistema de adquisición GCM de código abierto y libre uso se muestra en la Fig. 1. En nuestra propuesta consideramos que el usuario del sistema tiene un medidor de glucosa GCM de Medtronic que conecta de forma inalámbrica con un protocolo propietario en la banda de ISM de 868MHz.

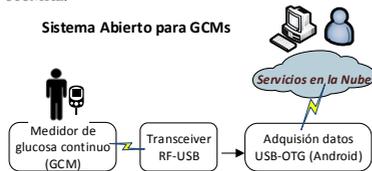


Fig. 1. Diagrama de bloques del sistema abierto para adquisición de datos GCM y posterior tratamiento con servicios en la nube.

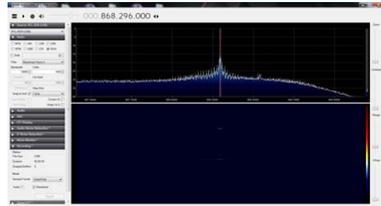
Se ha incluido un receptor de la señal inalámbrica procedente del sensor GCM con el código necesario de interpretación de los datos en el microprocesador embebido del sistema-en-chip el cual es objeto de desarrollo en el presente trabajo. En este caso se describe el proceso sobre la tecnología de Medtronic pero si se tratara de otro fabricante o modelo se debería modificar el transceptor. El resto del sistema quedaría inalterado, pudiendo reutilizarse de esta manera cualquier añadido al propio sistema para cualquier otro sensor que pudiera integrarse en el futuro.

Este *transceiver* se conecta de forma sencilla a través de un puerto USB-OTG a un dispositivo capaz de realizar un mínimo tratamiento de los datos y que en general podrá comunicarse con servicios adicionales ofrecidos por terceras partes en la nube.

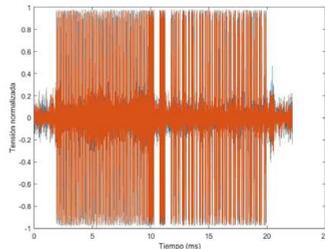
El sistema propuesto tiene una concepción de sistema abierto para que posibles desarrolladores adicionales amplíen las capacidades de la aplicación. De esta manera se ha utilizado el sistema operativo abierto Android para que a través de dispositivos móviles se transmitan los datos capturados a servicios en la nube desde cualquier ubicación en la que se encuentre el paciente.

IV. DEMODULACIÓN Y DECODIFICACIÓN DE LA SEÑAL

Como el protocolo y el esquema de modulación no eran conocidos a priori, se hizo un estudio de la señal utilizando técnicas de SDR (Software Defined Radio), software libre de captura (SDRSharp) y un receptor DVB-T USB comercial. La banda de trabajo era conocida pues es un dato que el fabricante hace público para poder comercializar su dispositivo, lo que hizo realmente sencillo encontrar las transmisiones.



Una vez encontradas las señales del dispositivo, se graban en ficheros de forma de onda estándar (WAV) y se importan en MatLAB. Mediante análisis temporal y frecuencial, se logra demodular la señal y extraer los bits del cuerpo del mensaje.



Es tras este proceso de análisis cuando se ve la posibilidad de portar este proceso a un System-On-Chip de Texas Instruments: permite la sincronización a nivel de bit con el preámbulo de la señal y se puede deshabilitar el resto de opciones que nos proporciona el circuito integrado. Una vez recibida la información en crudo, se puede terminar la decodificación del mensaje por software.

V. TRANSCEIVER RF-USB

A continuación se describe el *transceiver* inalámbrico que interpreta el código propietario del medidor GCM utilizado y reenvía los datos de glucosa monitorizados a un host con conexión usb 2.0.

El fabricante proporciona un entorno integrado de desarrollo y placa de evaluación que facilita las tareas de ingeniería inversa necesarias para obtener los códigos de transmisión de los datos procedentes del dispositivo GCM.

A. Texas Instruments - CC1111 USB Dongle

En la Fig. 2 se muestra una imagen del sistema on chip montado en una placa de evaluación [8] y que ha sido utilizado en el sistema desarrollado para la captura de datos inalámbricos de frecuencias por debajo de 1 GHz.



Fig. 2. Imagen del transceiver RF-USB de Texas Instruments CC1111

Este transceiver de Texas Instruments permite ser utilizado para capturar los datos emitidos de forma inalámbrica por el sistema de monitorización de glucosa (CGM).

En la Fig. 3 se muestra el diagrama de bloques del microcontrolador que incorpora el sistema-en-chip del TI-CC1111.

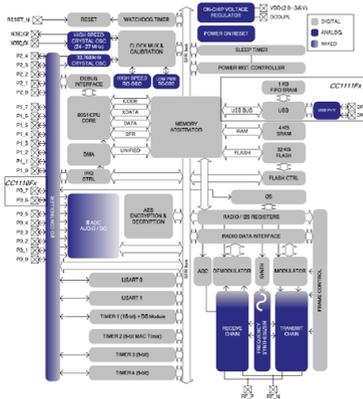


Fig. 3. Diagrama de bloques del transceiver RF-USB TI-CC1111.

El *dongle* opera en las bandas de frecuencia estándar de menos de 1 GHz, en particular la de 868 MHz y 915 MHz. La CPU está basada en un procesador 8051 por lo que permite el uso de programas ya realizados en anteriores desarrollos.

Los diferentes periféricos hardware que incluye el sistema hacen uso de los canales DMA para tener una operación eficiente (bloque AES, controlador Flash, canales de comunicación USART, *timers*, ADCs, etc.). Dispone de un *timer* específicamente diseñado para permanecer en un modo de ultra-bajo consumo que sólo consume 0.3 uA, ello permite hacer funcionar el sistema de forma autónoma, con batería.

B. Entorno de programación y desarrollo del transceiver CC1111 RF-USB

El sistema de depuración y carga del código de funcionamiento del transceiver utilizado para la conversión entre la información RF y un host USB es el que se muestra en la Fig. 4 (*CC Debugger*).



Fig. 4. Sistema de depuración y carga del transceiver RF-USB.

El sistema codificado se basa en una aplicación de puerto serie USB realizando funciones de una pasarela inalámbrica. De esta manera se hace uso del *dongle* USB para capturar los paquetes emitidos desde el medidor de glucosa.

VI. PROGRAMACIÓN DEL TRANSCEIVER

En esta sección se describe brevemente las acciones y configuraciones programadas en el código necesarias para tener un sistema de adquisición de datos del GCM

A. Inicialización del transceiver

El programa del CC1111 configura diferentes elementos: los necesarios en cualquier sistema microcontrolador como son las tablas de vectores de interrupción, variables globales, entradas y salidas, oscilador, etc. y en particular la configuración del modo RF del sistema-en-chip para comunicar correctamente con el módulo GCM y la configuración del canal de comunicación UART vía USB-OTG.

A continuación se muestra parte del código con las llamadas necesarias para configurar inicialmente el sistema:

```

/* Configure system */
initGlobals();
configureIO();
configureOsc();
crc64Init();
configureMedtronicRFMode();
enablePushButtonInt();
halUartInit(HAL_UART_BAUDRATE_57600, 0);
    
```

B. Bucle de operación del transceiver

El programa que se ejecuta en el *transceiver* consta de un bucle indefinido en el cual se van repitiendo una serie de acciones.

En primer término se comprueba si se ha recibido o no algún dato/mensaje procedente del GCM, en el caso del ejemplo se trabaja con un GCM comercializado por Medtronic [9].



Fig. 5. Sistema GCM para la monitorización de glucosa de Medtronic.

El sistema repite el envío de ciertos mensajes así que se ha programado en el dispositivo de enlace una función de comprobación de si hay nuevos datos en el canal de comunicación para no reenviar de nuevo datos pasados a *host* conectado por USB. A continuación se intercambian datos de forma inalámbrica y si existe transacción nueva de datos recibidos, éstos se transmiten al *host*.

A continuación se muestra parte del código con las funciones incluidas en el bucle de operación del transceiver:

```
receiveMedtronicMessage(dataPacket, &dataLength);  
repeatedMessage();  
uartRxTxBuffer();  
usbRxTxData();
```

En la función `receiveMedtronicMessage` se han codificado e interpretado todos aquellos mensajes del protocolo propietario que son necesarios para un correcto funcionamiento y configuración del sistema GCM.

En el resto de funciones se ha hecho uso de librerías y código de comunicaciones UART/USB estándar, y configurando el interfaz inalámbrica según indica el fabricante del sistema-en-chip.

Para un mejor uso y configuración del programa por parte de otros usuarios con distintas funcionalidades y dispositivos GCM se ha diseñado todo el programa para que se haga uso de una serie de parámetros de configuración definidos en fichero externo.

VII. RESULTADOS Y CONCLUSIONES

El desarrollo realizado se puede descargar del repositorio <https://github.com/jberian/mmccommander> para usarlo según se proporciona o bien para abrir una nueva rama de desarrollo y generar cambios con nuevas funcionalidades/protocolos incluidos en la aplicación.

En el sistema que se ha desarrollado, un teléfono móvil realiza las tareas de captura de mensajes, análisis y almacenamiento en una base de datos. El MMCCommander se presentará al sistema operativo del teléfono como un puerto serie USB, lo que hace realmente sencillo trabajar con él en cualquier plataforma. La aplicación del teléfono discrimina los mensajes provenientes del CGM del usuario y, mediante un proceso de calibración y conversión, se obtiene el valor de glucemia medido por el CGM sin filtrar.

En la Fig. 6 se muestran los componentes unidos para la monitorización de la glucosa en sangre. Atendiendo al

coste de los elementos necesarios se puede decir que el sistema utilizado es de bajo coste, teniendo a día de hoy un precio por debajo de los 100 € para el sistema de monitorización aparte del coste del GCM utilizado.



Fig. 6. Componentes del sistema de monitorización de glucosa.

Este mismo esquema se puede utilizar en otros sistemas de medición de glucosa, siguiendo todo lo desarrollado para el equipo Medtronic MiniMed Veo. También es válido para el Medtronic 530 pump con el transmisor MiniLink CGM incluido.

Puesto que los datos son transmitidos por RF y el transmisor del CGM debe llevarlo la misma persona que lleva el receptor, se utiliza muy poca potencia para maximizar la duración de las baterías. Esto, agravado por el uso de una banda ISM, hace que las transmisiones no sean del todo fiables y que sea necesario implementar mecanismos de detección de errores y recuperación de datos. En este caso, la detección de errores se realiza mediante un CRC-8 estándar y se logra recuperar errores gracias a que se mandan dos mensajes con la misma información por cada medida del sensor. A su vez cada mensaje contiene las últimas 9 medidas del sensor, añadiendo la posibilidad de recuperar una medida incluso 45 minutos más tarde de haberse tomado. El teléfono, de esta manera, podrá detectar si ha perdido alguna medida y recuperarla en mensajes posteriores.

Una vez el teléfono recibe las medidas del sensor, actualizará una base de datos MongoDB y subirá las medidas a una colección. Cada entrada contendrá, entre otros parámetros, el valor de la glucemia medida, el valor de la medida del sensor, una estimación de la relación señal a ruido existente en las medidas y una marca de tiempo para saber a qué instante corresponde cada medida.

A continuación se muestra una entrada de la base de datos, la cual tendría la siguiente estructura:

```
{  
  "_id": {  
    "$oid": "5741e7958d78da15211af3aa"  
  },  
  "sysTime": "2016-05-22T19:08:36.153+0200",  
  "unfiltered": 138320,  
  "filtered": 138320,  
  "direction": "FortyFiveDown",  
  "device": "MMCCommander",  
  "rssi": 100,  
}
```

```
"sgv": 134,  
"dateString": "2016-05-22T19:08:36.153+0200",  
"type": "sgv",  
"date": 1463936916153,  
"noise": 1  
}
```

Poder tener los datos almacenados en una base de datos de estas características, facilita enormemente la explotación de los mismos para múltiples usos: desde monitorización y alarmas hasta el análisis de datos con fines científicos.

Como primer paso se puede ver el uso de este dispositivo de pasarela de datos en un servidor online (<http://www.nightscout.info>) donde además de almacenarse todos los registros de glucemia se pueden incorporar otros datos del usuario relacionados con la diabetes como son las inyecciones de insulina, tipo de insulina, comidas realizadas, y ejercicios/actividades físicas, con lo que se pueden extraer datos muy relevantes por parte de los facultativos en relación a la diabetes que padece dicha persona.

VIII. AGRADECIMIENTOS

El presente trabajo ha sido financiado parcialmente mediante el proyecto UAH REF-01/2015 y UAH CCG2015/EXP-041.

IX. REFERENCIAS

- [1] Nightscout.info. Aplicación de monitorización DIY de Sistema GCM en la nube.
- [2] Klonoff, D. C. (2005). Continuous glucose monitoring roadmap for 21st century diabetes therapy. *Diabetes care*, 28(5), 1231-1239.
- [3] Oliver N. S., C. Toumazou, A. E. G. Cass, and D. G. Johnston, "Glucose sensors: a review of current and emerging technology," *Diabetic Medicine*, vol. 26, no. 3, pp. 197 – 210, 2009.
- [4] Matzeu G., L. Florea, and D. Diamond, "Advances in wearable chemical sensor design for monitoring biological fluids," *Sensors and Actuators B: Chemical*, vol. 211, pp. 403 – 418, 2015.
- [5] Yue, X. Y., Zheng, Y., Cai, Y. H., Yin, N. N., & Zhou, J. X. (2013). Real-time continuous glucose monitoring shows high accuracy within 6 hours after sensor calibration: a prospective study. *PLoS one*, 8(3), e60070.
- [6] Boland, E., Monsod, T., Delucia, M., Brandt, C. A., Fernando, S., & Tamborlane, W. V. (2001). Limitations of Conventional Methods of Self-Monitoring of Blood Glucose Lessons learned from 3 days of continuous glucose sensing in pediatric patients with type 1 diabetes. *Diabetes care*, 24(11), 1858-1862.
- [7] Facchinetti, A. G. Sparacino, and C. Cobelli, "Online denoising method to handle intraindividual variability of signal-to-noise ratio in continuous glucose monitoring," *Biomedical Engineering, IEEE Transactions on*, vol. 58, pp. 2664-2671, Sept 2011.
- [8] CC1111 USB Evaluation Module Kit 868/915 <http://www.ti.com/tool/CC1111EMK868-915> (enlace comprobado 05/2016).
- [9] Continuous Glucose Monitoring - Medtronic Diabetes www.medtronicdiabetes.com/.../continuous-glucose-monitoring

A Novel Indoor Localization Scheme for Autonomous Nodes in IEEE 802.15.4a Networks

G. Rébel^{*†}, J. González[†], P. Glösekötter,^{*} Francisco Estevez^{*} and Adrian Romero^{*}

^{*} Lab. for Semiconductor Devices & Bussystems

University of Applied Sciences of Münster, Münster, Germany

Email: gregor@fh-muenster.de, gloesek@ieee.org, fjestevz@ieee.org, ar461827@fh-muenster.de

[†] Dept. of Computer Architecture and Technology

University of Granada, Granada, Spain

Email: gregorrebel@correo.ugr.es, jesusgonzalez@ugr.es

Abstract—Ultra Wide Band (UWB) transceivers that are compliant to IEEE 802.15.4a standard do not provide distance measurement on their own. Instead they incorporate a high frequency counter (e.g. 64 Ghz) and a precise timestamp mechanism for outgoing and incoming data frames. The distance measurement has to be implemented by an external software.

Different localization schemes exist but these are optimized to provide the position information in the fixed anchor nodes first. The anchor nodes then have to use additional transmissions to send the information to the mobile node.

The term autonomous (mobile) node should mean that the position data is required by each mobile node itself. The new proposed localization algorithm accumulates the required localization information in the mobile node first by using less transmissions (and therefore less time and energy) than existing algorithms.

Index Terms—IEEE 802.15.4a, Ultra Wide Band, Localization, Indoor, Algorithm

I. INTRODUCTION

A. Indoor Localization using Ultra Wide Band Radios

The localization of a mobile node in 3D space requires to take ranging measures to four other nodes which positions are already known. The amount of required ranging measures can be reduced if some coordinates are fixed (e.g. the node is always located on a flat surface) or can be calculated (e.g. the node is located on a surface which height map is known). The algorithm used to locate a node is not part of this paper. Instead the focus is on the key technique of range measurement to one or more fixed anchor nodes.

B. Metric of Energy Consumption

IEEE 802.15.4a UWB transceivers require more energy to transmit or receive data frames than previous technologies. The power draw of an UWB transceiver is typically the same, if not higher, while its receiver is switched on as during transmitting. In order to minimize the energy consumption per localization, the sum of all time periods when the transmitter or receiver is enabled has to be minimized.

II. STATE OF ART

A. Time Difference Of Arrival (TDOA)

This ranging algorithm uses one mobile node M and two or more fixed anchor Nodes A_1, \dots, A_n as shown in fig. 1. The

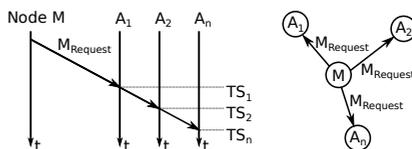


Fig. 1: Localization via Time Differential of Arrival (TDOA)

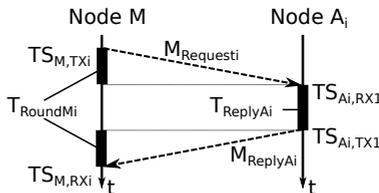


Fig. 2: Ranging via Time Of Flight Single Sided (TOF-SS)

mobile node M sends out a message that is received and timestamped by A_1, \dots, A_n . The anchor nodes then communicate with each other to locate M based on the different arrival timestamps $TS - i, i = [1, \dots, n]$. This algorithm only requires one message to be transmitted by the mobile node. The big disadvantage of this method is that the anchor nodes must precisely synchronize their clocks. This synchronization can be difficult (e.g. if the anchor nodes cannot communicate with each other directly). TDOA is not part of this paper and is only presented for completeness.

B. Time Of Flight Ranging Single Sided (TOF-SS)

TOF ranging measures the distance between two nodes. In the context of localization these typically are a mobile node M and a fixed anchor node A_i . The single sided version is the

(1)	$T_{ReplyA_i} = TS_{A_i,TX_1} - TS_{A_i,RX_1}$
(2)	$T_{RoundM_i} = TS_{M,RX_1} - TS_{M,TX_1} - T_{ReplyA_i}$
(3)	$T_{ReplyM} = TS_{M,TX_1} - TS_{M,RX_1}$
(4)	$T_{RoundA_i} = TS_{A_i,RX_2} - TS_{A_i,TX_1} - T_{ReplyM}$

TABLE I: Formulas - Time Of Flight Ranging

most simple TOF implementation. For this, Node M has to transmit one request- and to receive one reply-message named $M_{Request_i}$ and M_{ReplyA_i} in fig. 2. Node M takes timestamps TS_{M,TX_1} and TS_{M,RX_1} when transmitting $M_{Request_i}$ and receiving M_{ReplyA_i} . Node B will transmit a ranging reply message M_{ReplyA_i} after a fixed delay time T_{ReplyA_i} . The physical distance between M and A_i is then proportional to the round trip time T_{RoundM_i} , as calculated in table I. T_{ReplyA_i} should be as short as possible but may vary among individual nodes. T_{ReplyA_i} must be long enough to allow Node A_i to read and process the received ranging request and to prepare the answer message. The required minimum delay time depends on the actual processing speed of the used microcontroller and its firmware. T_{ReplyA_i} may vary between different radio nodes with same hardware. As node M requires the exact delay time to calculate the distance, T_{ReplyA_i} is sent with M_{ReplyA_i} . Different transceivers always differ in their clock speed. T_{ReplyA_i} is measured by using the clock of node A_i while T_{RoundM_i} is measured by using the clock of node M. The effect of clock induced error in single sided TOF ranging also depends on T_{ReplyA_i} and the frame length. E.g. for the DecaWave DW1000 transceiver, varying $T_{ReplyA_i} = 0.1 \dots 5$ ms and $ClockError = 2 \dots 40$ parts per million (ppm) will affect the measurement of T_{ReplyA_i} from 0.1 ns to 100ns. Where 1 ns corresponds to a 30 cm error in measured distance [1].

C. Time Of Flight Ranging Double Sided (TOF-DS)

Clock induced errors can be reduced by using double sided TOF ranging as shown in fig. 3. This basically requires one additional transmission from node M to A_i using an exact reply time T_{ReplyM} . T_{ReplyM} is required to calculate a second round trip time T_{RoundA_i} as shown in table I. As T_{ReplyM} is measured using the clock of node M, the different clock speeds of M and A_i will compensate each other. Even a relative clock error of 40 ppm will induce an error in distance measure of less than 2.2 mm for a practical setup [1]. Double Sided TOF ranging is implemented as software libraries for various commercially available IEEE 801.15.4a transceivers like DecaWave DW1000 [1] and Nanotron nanoLOC [2].

D. TOF Ranging Symmetric Double Sided (TOF-SDS)

Double Sided TOF ranging with equal reply times in both participating nodes is called Symmetric Double Sided TOF (SDS TOF). SDS TOF minimizes the clock induced error as crystal frequency variations are halved [3]. For asymmetric double sided TOF ranging [4] has shown a method to minimize the clock induced error to a value similar to SDS TOF ranging.

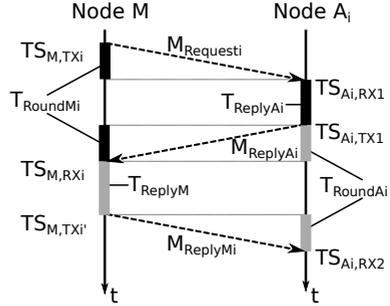


Fig. 3: Ranging via Time Of Flight Double Sided (TOF-DS and TOF-SDS)

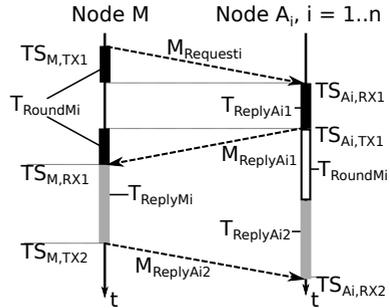


Fig. 4: Ranging TOF-TRADS with Forward Reporting

E. TOF Ranging Traffic Reduced Asymmetric Double Sided (TOF-TRADS)

Mi-Kyung Oh et al [5] presented a ranging scheme that reduces clock induced error without increasing the amount of transmitted messages. The basic idea is to measure the same response delay on two nodes. The two time measures are then used to calculate the relative frequency offset. Two basic variants exist which each require three messages to be exchanged. Similar ranging schemes, that also measure the same time period on different nodes, have later been proposed by Hakyong Kim [6] and by Myungkyun Kwak and Jongwha Chong [7].

The basic TOF-TRADS scheme works as follows: Node M sends a Request to Node A_i . Node A_i sends a reply message

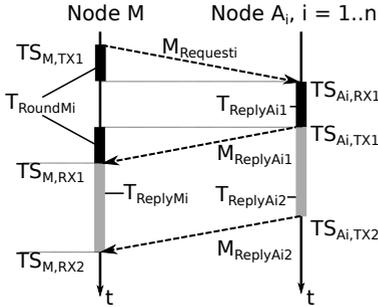


Fig. 5: Ranging TOF-TRADS with Backward Reporting

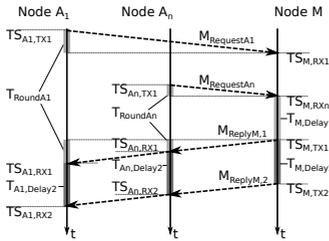


Fig. 6: Localization with Traffic Reduced Asymmetric Double Sided Time Of Flight Ranging (TOF-TRADS)

back to Node M after a fixed delay T_{ReplyA_i1} . With forward reporting (fig. 4), node M sends a reply message after a fixed delay time after receiving the reply message from A_i . Node A_i measures time period $T_{ReplyA_i2} = TS_{A_i,RX2} - TS_{A_i,TX1} - T_{RoundM_i}$. The CFO_{Ratio} can then be calculated the same way as in backward reporting scheme. With backward reporting (fig. 5), node A_i then sends a second reply message after a fixed delay. Node M knows the timestamps $TS_{M,RX1}$ and $TS_{M,RX2}$ of both reply messages. T_{ReplyM_i} is simply the difference between both timestamps. The clock frequency offset ratio $CFO_{Ratio} = T_{ReplyM_i}/T_{ReplyA_i2}$ is the relative clock frequency offset of both radios. This ratio allows to compensate the clock induced error.

F. Localization using TOF-TRADS

Mi-Kyung Oh et al [5] describe how to use their traffic reduced asymmetric double sided time of flight ranging for indoor localization. The basic localization scheme is shown in fig. 6. Multiple anchor nodes $A_i, i = [1, \dots, n]$ each send

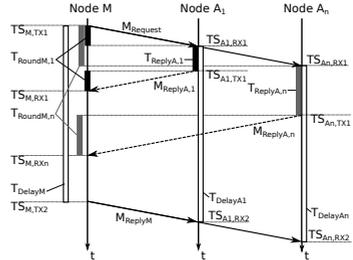


Fig. 7: Localization via Double Sided TOF Ranging with Cascaded Replies from n Anchors (TOF-CR)

one localization request $M_{RequestA_1}$ to the same mobile node M. The requests arrive at M at time stamps $TS_{M,RX1}$. Node M sends a first broadcast reply message $M_{ReplyM,1}$ after delay time $T_{M,Delay1}$ after receiving the last ranging request $M_{RequestAn}$. Node M broadcasts a second reply message $M_{ReplyM,2}$ after another delay time $T_{M,Delay2}$. The anchor nodes receive both reply messages and measure their receiving timestamps $TS_{A_i,RX1}$ and $TS_{A_i,RX2}$. The difference between both timestamps represents the same delay time but it is measured with the local clock and used to calculate CFO_{Ratio} as described above. The delay times are embodied in $M_{ReplyM,1}$ and $M_{ReplyM,2}$ respectively. Each anchor node is then able to calculate the round trip time T_{RoundA_i} and compensate the CFO. The transmit times $TS_{A_i,TX1}$ must be timed exactly among all anchor and mobile nodes to minimize the total receiver on time of node M while avoiding collisions. At the end of this localization scheme, each anchor node knows its distance to the mobile node.

III. PROPOSED LOCALIZATION SCHEME

The TOF-TRADS scheme requires $n + 2$ messages to calculate the distance from n anchor nodes to a mobile Node M. Using TOF-SDS ranging n times would require 3 n messages. TOF-TRADS therefore requires less messages than n times TOF-SDS for $n \geq 1$. But TOF-TRADS it has two disadvantages:

- The localization process is initiated by the anchor nodes. This requires that the mobile node switches on its receiver at a predefined time which increases its energy usage.
- At the end of the localization scheme, the gathered range information is distributed over n anchor nodes. This information has to be sent to the mobile node by n additional transmissions.

A. Localization via TOF with Cascaded Replies (TOF-CR)

A novel localization scheme called TOF Ranging with Cascaded Replies (TOF-CR) is intended to overcome these

disadvantages as shown in fig. 7. In TOF-CR, a mobile node broadcasts a ranging request message MRrequest to n anchor nodes $A_i, i = [1, \dots, n]$. Each anchor node answers after a unique reply time T_{ReplyA_i} . The reply messages are received by node M at timestamps T_{SM,RX_i} . From these, round trip times T_{RoundM_i} are calculated analogue to table I. Node M then sends all calculated round trip times plus its own reply time T_{ReplyM} in one reply message M_{ReplyM} . This final reply message is sent after a known delay T_{DelayM} after T_{SM,TX_i} via broadcast to all anchor nodes. The delay time is also measured in each anchor node as T_{DelayA_i} . As for TOF-TRADS, these multiple measures of the same time period allow to compensate CFO.

B. Predicting Relative CFO in Periodic Localization (TOF-CRP)

When multiple localizations take place in a short time, then the relative CFO between a mobile node and the available anchor nodes is expected to be stable. This stationary state allows to cache relative CFOs for each anchor in the mobile node. The n anchor nodes will not send their calculated relative CFO in the current localization process. Instead, each anchor node A_i will send the relative CFO in its MReplyA.i message at the beginning of the next localization process as shown in fig. 7. This makes it possible to provide a compensated position in the mobile node with only n + 2 messages beginning from the second localization.

C. Localization via TOF with Dynamic Cascaded Replies (TOF-DCRP)

TOF-DCRP shall be an extension to TOF-CRP where a large amount of anchor nodes is deployed in a region or a building. When the amount of anchor nodes is high, then TOF-CR will result in a very long T_{ReplyM} . This means that node M has to switch on its receiver for a long time to receive even the anchor node with longest T_{ReplyA_i} . Node M may switch off its receiver after receiving enough round trip times for a localization. Unfortunately, in a large scale setup, most anchor nodes will be out of reach. If only An-3, ..., An are in reach, then node M has to wait the complete T_{DelayM} to obtain enough ranging data to localize itself. As an optimization, anchor nodes which are far away from each other may reuse the same reply time T_{ReplyA_i} in a dynamic way. It is assumed, that the anchor nodes are located in a sensefull setup. Which means that, for every mobile location, not much more than four anchor nodes are in reach. As UWB-Transceivers are reflected by typical walls, this assumption seems to be realistic inside buildings. Each room may be equipped with four anchor nodes. Adjacent rooms or a floor may add to the group of anchor nodes in reach. But anchor nodes in not adjacent rooms should ideally be out of reach. Different dynamic scheduling schemes for UWB ranging are described in [8] and a joint scheduling algorithm based on integer linear program (ILP) is described in [9] and may be investigated during this research line.

Scheme	1st Data	Initiator	Example Application Scenario
TDOA	Anchors	Mobile	Indoor Asset Tracking
TOF-SDS	Initiator	Anyone	General Purpose
TOF-TRADS	Anchors	Anchor	Indoor Asset Tracking
TOF-CRP	Mobile	Mobile	Autonomous Indoor Navigation
TOF-DCRP	Mobile	Mobile	Autonomous Indoor Navigation

TABLE II: Localization Scheme Comparison

D. Best Localization Scheme for Autonomous Nodes

Individual localization schemes differ in the place where the 3D location of a node is available first during the process of localization. For autonomous mobile nodes, we claim that the 3D location is required in the mobile node first. Localization schemes that gather ranging information in n anchor nodes will require n additional messages to concentrate all required data in the mobile node. The TOF-CRP and TOF-DCRP are initiated by the mobile node and provide the position in the mobile node first. This makes them the best localization schemes for autonomous mobile nodes. Table II categorizes different localization schemes by their typical application scenario. The table shows where all required localization data is available first and which node type initiates the localization.

IV. METHODOLOGY

The communication protocols for TOF-CRP and TOF-DCRP localization schemes will be implemented using the C programming language. The software development bases on an open source operating system and development toolchain for 32 bit microcontrollers called The ToolChain [10]. A new software driver for an IEEE 802.15.4a compliant Ultra Wide Band transceiver from DecaWave Corp called DW1000 has been developed for The ToolChain. The new proposed localization scheme will be implemented on top of this software driver. The localization scheme will be tested on prototype boards equipped with a STM32L100 microcontroller and this DW1000 transceiver. The network simulations for a large amount of distributed anchor nodes will be performed using a software simulator.

V. CONCLUSION

We have presented a novel localization scheme TOF-CR for IEEE 802.15.4a UWB sensor networks that requires less transmissions than state of the art schemes in a special scenario where localization is initiated by each mobile node and the localization data is required in the mobile node first. An extension TOF-CRP has been described that allows CFO compensation without extra transmissions during periodic localizations. Another extension TOF-DCRP has been described for setups with large amount of anchor nodes.

VI. ACKNOWLEDGEMENT

We would like to thank Robin Weiß for fruitful discussions.

REFERENCES

- [1] DecaWave Ltd. DW1000 User Manual rev 2.05. Source: <http://decawave.com>, 2015.
- [2] Christof Röhrig and Marcel Müller. Indoor Location Tracking in Non-line-of-Sight Environments Using a IEEE 802.15.4a Wireless Network. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.
- [3] J. Lampe, R. Hach, L. Menzer, K.-K. Lee, and J.-W. Chong. DBO-CSS PHY Presentation for 802.15.4a. *IEEE 15-05-0126-01-004a*, 2005.
- [4] Y. Jiang and V. Leung. An Asymmetric Double Sided Two-Way Ranging for Crystal Offset. *IEEE 1-4244-1449-0/07*, pp. 525- 528, 2007.
- [5] Mi-Kyung Oh, Jae-Young Kim, and HyungSoo Lee. Traffic-Reduced Precise Ranging Protocol for Asynchronous UWB Positioning Networks. *IEEE Communication Letters*, 14, 2010.
- [6] Hakyong Kim. Double-Sided Two-Way Ranging Algorithm to Reduce Ranging Time. *IEEE Communications Letters*, 13, 2009.
- [7] Myungkyun Kwak and Jongwha Chong. A new double two-way ranging algorithm for ranging system. In *Proceedings of IC-NIDC2010*, 2010.
- [8] B. Denis. On the Scheduling of Ranging and Distributed Positioning Updates in Cooperative IR-UWB Networks. *ICUWB*, 2009.
- [9] Gabriel E. Garcia, Wuhua Hu, Wee Peng Tay, and Henk Wymeersch. Joint Scheduling and Localization in UWB Networks. *IEEE ICC 2015 - Workshop on Advances in Network Localization and Navigation*, 2015.
- [10] Gregor Rebel. The ToolChain - An Open Source Development Environment and Operating System for 32 Bit Microcontrollers. Source: <http://thetoolchain.com>, June 2016.

V Congreso Español de Informática
Salamanca, 13 al 16 de septiembre, 2016

