Selección de una arquitectura *many-core* comercial como plataforma de tiempo real

David García Villaescusa, Michael González Harbour y Mario Aldea Rivas

Dpto. Ingeniería Informática y Electrónica
Universidad de Cantabria, Santander (España)
{garciavd, mgh, aldeam}@unican.es

Resumen Los procesadores *many-core* representan la evolución natural de las arquitecturas de computadores de propósito general. Su aumento de prestaciones y la contribución a la disminución de tamaño, peso y consumo del sistema completo con respecto a los procesadores actuales hace prever que serán usados también como plataformas para sistemas de tiempo real en el futuro. En este artículo se identifican los requisitos que debería cumplir una arquitectura *many-core* para su utilización en sistemas de tiempo real y se analizan varios procesadores *many-core* existentes en el mercado para acabar seleccionando el mejor candidato. Finalmente el artículo adelanta algunos de los retos esperables en el desarrollo del soporte para aplicaciones de tiempo real sobre estos sistemas.[1]

Palabras clave: Tiempo real · Many-core · Mapeado · Planificación · Sistemas operativos · Intel Xeon Phi · Tilera · Kalray

1. Introducción

Históricamente los procesadores mejoraron su potencia de cómputo a base de subir la velocidad de su reloj interno. Esto elevaba el consumo implicando un límite al crecimiento impuesto por la disipación del calor generado por los procesadores. Como solución, desde hace ya varios años se está orientado la evolución de los procesadores al aumento del número de núcleos de cómputo. En la actualidad, los procesadores *multi-core* son un referente en la electrónica de consumo. Esto es debido a las múltiples ventajas que proporcionan respecto a un *mono-core*:

- Mejor rendimiento de manera más eficiente.
- Permiten la ejecución simultánea de múltiples aplicaciones en un solo procesador, reduciendo el consumo y las altas exigencias de refrigeración con respecto a los procesadores con muy altas frecuencia de reloj. En sistemas embebidos de altas prestaciones esto permite una reducción del número de sistemas de computación, que equivale a una reducción de peso y cableado.

[1] Este trabajo ha sido financiado en parte por el Gobierno de España en el proyecto TIN2014-56158-C4-2-P (M2C2).

- Permiten que se efectúen diferentes tareas de una misma aplicación de manera simultánea utilizando diversos hilos de ejecución con concurrencia física. Esto supone un aumento en la capacidad de cómputo, pero requiere de un esfuerzo a la hora de paralelizar el código.

Sin embargo, el uso de un bus para el acceso a la memoria compartida y el mantenimiento de la coherencia de cachés introduce incertidumbre en los tiempo de acceso a la memoria y con ello en el tiempo de ejecución de peor caso, lo que lleva a desaconsejar en general la implantación de *multi-cores* en sistemas de seguridad crítica [1][4][10]. Estos sistemas requieren de un proceso de certificación para asegurar que son capaces de cumplir con ciertos criterios de seguridad y fiabilidad. Existen sectores, como la aviación, que poseen sus procesos de certificación específicos.

La utilización de un único bus, compartido por todos los núcleos, para el acceso a la memoria constituye el principal cuello de botella en las arquitecturas *multi-core*. Dicho bus compartido es la principal causa de la baja escalabilidad de este tipo de arquitecturas que ha conducido a la búsqueda de nuevas estrategias de acceso como las que se emplean en los sistemas *many-core* en los que el número de núcleos puede aumentar gracias a un cambio en la arquitectura de comunicación con la memoria. Estas arquitecturas son novedosas y apenas hay experiencia de su uso en sistemas de tiempo real. En este artículo pretendemos analizar las arquitecturas de algunos sistemas *many-core* comerciales para estudiar cuál de ellos se puede adaptar mejor a los requisitos de un sistema de tiempo real en el que es preciso tener garantías de predictibilidad de los tiempos de respuesta.

La estructura del artículo consta de una exposición de los requisitos generales que ha de cumplir una plataforma de tiempo real seguida de un análisis de tres procesadores *many-core* comerciales: Intel, Tilera y Kalray. Sobre ellos se ha realizado un estudio del cumplimiento de los requisitos planteados, ubicado en la sección 2. A continuación, en la sección 3 se describen aspectos generales sobre el mapeado de hilos de ejecución a núcleos y en la sección 4 se hace un breve resumen de algunos sistemas operativos disponibles para arquitecturas *many-core*. Finalmente, en la sección 5 se expone la conclusión de qué procesador es el candidato con mayor potencial para llevar a cabo un estudio más completo de la implementación de sistemas de tiempo real en procesadores *many-core*.

2. Procesadores *many-core* como plataformas de tiempo real

Las principales diferencias entre procesadores *multi-core* y *many-core* son el número de núcleos y la forma en la que estos se conectan. Las arquitecturas *many-core* utilizan una arquitectura más escalable que el bus, tal como una malla o un toro. De esta manera se mantienen, incluso se mejoran, las ventajas que supone el uso de múltiples núcleos a la vez que se aumenta la potencia de cómputo.

Como hemos comentado, la actual generación de procesadores *multi-core* no es adecuada para ejecutar software de tiempo real a su máxima potencia, debido a la impredecibilidad en el tiempo de acceso a memoria. Se recomienda su uso solo si se cumplen determinadas restricciones, por ejemplo una de las más drásticas consiste en que la aplicación se ejecute solo en un núcleo [1]. Por ello es de gran interés poder incluir la próxima generación de procesadores, los *many-core*, como plataformas para sistemas de tiempo real y explorar la posibilidad de hacerlo incluso en sistemas de seguridad crítica. Es preciso investigar si los procesadores *many-core* poseen una arquitectura más predecible en cuanto a los tiempos de respuesta.

2.1. Requisitos generales

Para poder utilizar procesadores *many-core* en sistemas de tiempo real hay que cumplir ciertos requisitos en las especificaciones del mismo:

1. **Núcleos predecibles.** Los núcleos deben proporcionar de manera individual un comportamiento acotable temporalmente. Un aspecto que puede incidir negativamente en la predictibilidad es la ejecución fuera de orden *out of order*. Esto afecta a los tiempos de ejecución e incide negativamente en los cambios de contexto en los que de manera asíncrona el procesador abandona un punto de la ejecución de un hilo para pasar a ejecutar otro hilo, debiendo por tanto descartar cálculos hechos fuera de orden. Este retraso se puede añadir con facilidad al tiempo de cambio de contexto. Pero por otro lado la ejecución fuera de orden presenta el inconveniente de dificultar el análisis estático de tiempos de ejecución.
2. **Memoria local.** La existencia de memoria compartida en un procesador genera una gran incertidumbre en el cálculo de los tiempos de respuesta. En muchos sistemas *multi-core* tanto el último nivel de memoria caché (L2 o L3) como el bus de memoria son compartidos. Además, la memoria caché privada (L1) necesita mantener la coherencia de sus datos, lo cual provoca que determinados accesos a datos contenidos en esta caché no sean independientes si se hacen accesos a ellos desde diferentes núcleos. La coherencia de las memorias caché genera una gran incertidumbre en el tiempo de respuesta máximo salvo que se tengan mecanismos de coherencia de caché con tiempo de ejecución acotada o se pueda deshabilitar dicha coherencia de caché. Para sistemas de tiempo real, se necesitan cachés privadas o memorias locales a los núcleos de tamaño suficiente para albergar la mayor parte de los accesos de un thread o hilo de ejecución alojado en cada núcleo.
3. **Mapeado de memoria.** La existencia de memorias locales o cachés privadas hace necesario disponer de mecanismos para mapear la memoria usada por un hilo de ejecución y bloquearla en la caché privada a voluntad.
4. **Red de interconexión.** La Network-on-Chip (NoC) también tiene que proporcionar un comportamiento temporal predecible. No basta con que tenga un gran *throughput* (anchura de banda) y baja latencia.
5. **Periféricos.** Diferentes aplicaciones pueden intentar acceder a un mismo periférico. El acceso debe de ser controlado y sincronizado.

6. **Acceso a memoria.** Se requieren tiempos de acceso a memoria acotados.
7. **Envío de mensajes.** Es preciso poder de acotar el tiempo de envío de mensajes entre los diferentes núcleos del procesador. Puesto que en muchas redes el envío de mensajes a procesadores vecinos es más rápido que a otros más lejanos la posibilidad de controlar el mapeado de hilos de ejecución a procesadores puede facilitar el agrupamiento de aquellos hilos que tienen mayores relaciones entre sí.

Vamos a analizar a continuación tres arquitecturas de procesadores comerciales disponibles actualmente y estudiaremos el cumplimiento de los requisitos expuestos arriba (con las limitaciones de la documentación pública existente).

2.2. Intel Xeon Phi

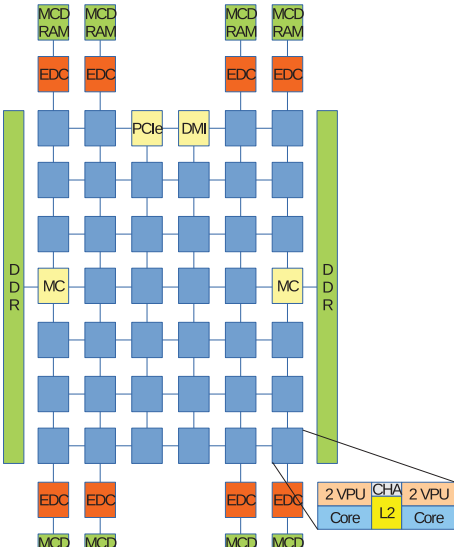

Figura 1. Arquitectura del procesador Intel Xeon Phi Knight's Landing (izq.) y arquitectura de un núcleo del procesador (der.)

La Intel Xeon Phi es una familia de procesadores que lleva evolucionando desde 2010 y cuya generación actual se llama Knights Landing (KNL) [15]. Esta arquitectura está orientada a la computación de altas prestaciones. Como se puede observar en la figura 1 su arquitectura consiste en una malla 2D que conecta celdas, cada una con 2 núcleos y cada uno con 2 Vector Processing Units (VPUs) que comparten 1MB de caché L2. La coherencia de la caché L2 y la conexión con la NoC se realizan a través de la Caching/Home Agent (CHA). El procesador Intel® Xeon Phi™ Processor 7290 ofrece esta arquitectura Knights Landing y tiene 72 núcleos.

Analicemos cómo se adecúa este procesador a los requisitos de tiempo real:

1. **Núcleos predecibles.** Cada núcleo dispone de ejecución fuera de orden (*out-of-order*) y cuatro hilos simultáneos (*hyperthreading*). Son una modificación de los Intel Atom con mayor búfer *out-of-order*.
2. **Memoria local.** Los núcleos solo disponen de memoria privada en el nivel L1, separada en datos e instrucciones ambas de 32kB. La L2 se comparte en las celdas y se mantiene coherente a través de la malla.
3. **Mapeado de memoria.** La documentación pública no especifica si es posible realizar el mapeado de memoria en cachés por medio de la precarga y bloqueo de determinadas líneas.
4. **Red de interconexión.** Conecta las diferentes celdas de núcleos, controladores de memorias, controladores de entrada/salida y demás elementos del procesador. La malla soporta el protocolo de coherencia de caché MESIF (Modified, Exclusive, Shared, Invalid, Forward).
5. **Periféricos.** Dispone de un coprocesador para PCIe (PCI express).
6. **Acceso a memoria.** Los controladores de memoria están situados en las celdas EDC y MC de la figura 1.
7. **Envío de mensajes.** La NoC utiliza enrutado XY estático (cada salto en dirección Y cuesta 1 ciclo y 2 ciclos en dirección X).

Particionado Se puede particionar el conjunto de celdas en 2 o 4 grupos para crear zonas aisladas entre sí para mejorar las prestaciones. Estas particiones se pueden seleccionar desde la BIOS.

Conclusiones La familia Intel Xeon Phi está muy dirigida a un mercado que nada tiene que ver con el tiempo real y eso se nota en que, aparentemente, no se permite la desactivación de la coherencia de caché L2 y que en cada celda se encuentre un sistema de doble núcleo. El pequeño tamaño de la caché privada L1 hace difícil evitar los conflictos debidos a la coherencia de la caché L2.

2.3. Tilera

Como se puede observar en la figura 2, los procesadores de la arquitectura Tilera-GX [16] se dividen en celdas, cada una con un procesador de 64-bits, memoria caché (niveles 1 y 2) y un *switch* que actúa como interfaz con la malla

que comunica todas las celdas y que proporciona coherencia de la caché L2 a todos los núcleos. El procesador Tile GX-100, de esta familia, es un modelo que presenta 100 núcleos.

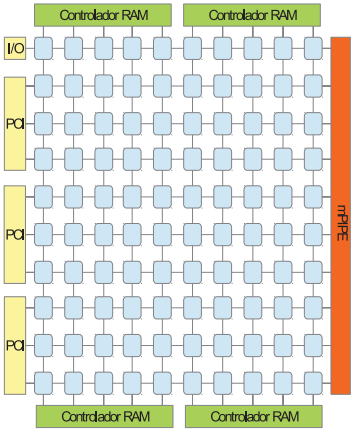

Figura 2. Arquitectura del procesador TILE-Gx100

Vamos a analizar el cumplimiento de los requisitos de tiempo real en este procesador:

1. **Núcleos predecibles.** Los núcleos son homogéneos con una arquitectura *Very Long Instruction Word* (VLIW) de 3 vías con instrucciones de 64-bits y ejecución en orden. Disponen de 32kB de caché L1 de instrucciones y otros tantos de datos. Cada núcleo también tiene 256kB de caché L2.
2. **Memoria local.** El espacio de direcciones físicas compartidas tiene coherencia de caché L2 con todo el *hardware*. Utiliza un modelo de memoria compartida y las lecturas/escrituras de entrada/salida se realizan directamente por la caché.
3. **Mapeado de memoria.** Se mapea cada dirección física a un núcleo propietario, con lo que si esa dirección se cachea es en la caché L2 de ese núcleo donde se guarda el dato. El resto de celdas acceden al dato mediante la malla de comunicaciones. De esta manera se minimizan los accesos a la memoria

RAM y se tiene una visión global de una caché compartida, que hace las veces de una caché L3 virtual.

4. **Red de interconexión.** El número de núcleos varía según el modelo pero siempre se distribuyen en una malla 2D. En la NoC se computa y envía en la misma instrucción. Esta posee 5 redes independientes *full-duplex* (IDN: sistema y entrada salida; MSN: fallos de caché, DMA (Direct Memory Access) y demás memorias; TDN: acceso a memoria entre celdas; UDN: streaming a nivel de usuario y STN: transferencias escalares). Las cachés llevan integrados mecanismos DMA para acceder a/desde la malla. Cada celda tiene un *switch*.
5. **Periféricos.** Como se puede observar en la figura 2 el acceso a los periféricos es a través de los núcleos situados en las filas de la izquierda y la derecha del procesador.
6. **Acceso a memoria.** Las instrucciones y datos son manejados por controladores de caché que proporcionan una interfaz al sistema de memorias. Traducen las direcciones de memoria virtuales a físicas además de proporcionar una vista coherente de la memoria. Cuando se requieren datos que no se encuentran en la caché, el controlador utiliza la NoC para comprobar otras caches y la memoria principal.
   La memoria está compartida de manera global y las transacciones se realizan a través de la malla. El *hardware* proporciona una vista de la memoria de las aplicaciones con coherencia de caché, aunque el valor esté en la caché de otra celda.
   Los datos tienen una celda asignada, donde las otras celdas buscarían el valor del dato en caso de fallo de la caché L2. En caso de no encontrarse ahí, se requiere un acceso a la memoria principal DDR.
7. **Envío de mensajes.** Los datos pueden transferirse punto a punto: celda a memoria, entre celdas y de celdas a entrada/salida. Hay conmutación de paquetes, se utiliza enrutado *wormhole* [13] con control de flujo en las celdas cercanas y enrutado X-Y.
   Se tarda un ciclo en pasar de un *switch* a la entrada de un *switch* vecino. Los *switches* soportan dos arbitrajes: *round robin* y *network priority*.
   - Con *round robin* varias entradas están dirigidas al mismo puerto de salida siguiendo estas normas: cada puerto de entrada tiene la misma prioridad y se procesan todas las peticiones al mismo puerto de salida utilizando política cíclica (*round robin*).
   - Con *network priority* la NoC proporciona prioridades dinámicas a los paquetes. Para las entradas de cada *switch* con la misma prioridad se utiliza *round robin*. Se proporciona un contador de antigüedad al núcleo con el objetivo de evitar la inanición y se impide que los núcleos de los bordes del procesador perjudiquen el acceso a los dispositivos de entrada/salida.

Particionado Se puede particionar el conjunto de celdas en grupos de 1 a 36 celdas para crear grupos zonas aisladas entre sí. Este particionado se establece

por software a nivel de hipervisor. En el particionado también se parte y asigna memoria a cada grupo.

Conclusiones El procesador con ejecución en orden es predecible. Pese a ser un sistema que no está orientado a tiempo real se permite cierta independencia entre zonas del procesador gracias al sistema de particionado. También se tiene una NoC de comportamiento predecible. Hay una cierta noción de memoria local si mediante el particionado se consigue que cada celda acceda solo a los datos de su propia caché L2 y no a los de otras deldas. Esta caché L2 tiene un tamaño de 256kB, que puede ser suficiente para un gran número de aplicaciones empotradas.

2.4. Kalray

Kalray es un fabricante que con su chip *many-core* MPA-256 ha conseguido un procesador expresamente dirigido a sistemas embebidos y de tiempo real. Cuenta con 288 núcleos repartidos en 16 agrupaciones (*clústers*) de cómputo y 4 subsistemas de entrada/salida cada uno con 4 núcleos capaces de acceder a la memoria externa. Como se observa en la parte derecha de la figura 3, cada agrupación contiene 16 núcleos de ejecución más un núcleo que actúa como gestor de recursos. Pueden llegarse a conectar 4 procesadores Kalray MPPA-256 a través de una placa TurboCard3 en caso de requerirse mayor potencia de cálculo.

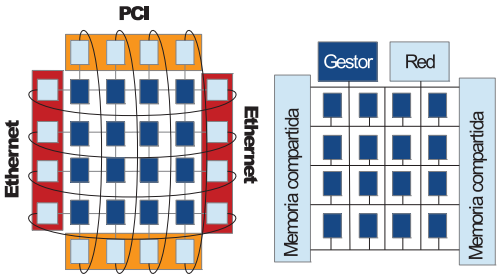

Figura 3. Arquitectura del procesador Kalray MPA-256. A la izquierda el procesador y a la derecha cada agrupación de cómputo

Vamos a analizar el cumplimiento en este procesador de los requisitos de tiempo real:

1. **Núcleos predecibles.** Los 17 núcleos que forman una agrupación de cómputo son idénticos y cada uno posee cachés privadas de 8kB de instrucciones y datos. Cada uno de ellos implementa una arquitectura VLIW de 5 vías con ejecución en orden.
2. **Memoria local.** Cada agrupación tiene una memorial local de 2MB dividida en 16 bancos de 128kB de acceso independiente y sin mecanismo de coherencia. No hay direccionamiento directo de otros espacios de memoria de otros agrupamientos o de la memoria externa. El acceso a cada banco se hace por política cíclica pero es posible configurar la memoria de modo que cada núcleo de ejecución tenga acceso a un banco de memoria diferente y de este modo se evitan todas las interferencias entre ellos.
3. **Mapeado de memoria.** No hay mapeado de memoria directo desde los procesadores de ejecución a la memoria principal o a las memorias locales de los demás agrupamientos.
4. **Red de interconexión.** Tanto las 16 agrupaciones de cómputo como los 4 subsistemas de entrada/salida están conectados por dos NoC de enlaces bidireccionales, una de datos y la otra de control. Ambas presentan una topología toroidal 2D aumentada con conexiones directas entre los subsistemas de I/O. El tráfico de la NoC que pasa por un *router* no interfiere con los buses de memoria, el subsistema de entrada/salida o la agrupación de cómputo salvo que ese nodo sea el destinatario.

   Los canales de datos y control aseguran el envío y los mensajes que siguen la misma ruta llegan en orden. No se envía un mensaje de reconocimiento al nodo fuente. El procesador gestor de recursos tiene conexiones a la interfaz de la NoC a través de líneas de eventos e interrupciones.

   Con objeto de garantizar el cumplimiento de requisitos temporales cada conexión lleva asociada una cuota de anchura de banda, controlada por el nudo emisor.

   Dentro de cada agrupación de cómputo (imagen de la derecha de la figura 3) se utilizan buses que van directamente a los bloques de memoria (divididos en 2 partes, una a la izquierda y otra a la derecha) pudiéndose configurar el acceso a dichos bloques de manera independiente y por tanto con tiempos de acceso acotados.
5. **Periféricos.** El acceso a los periféricos (PCIe y Ethernet) y memoria DDR se realiza a través de los núcleos situados a los bordes del procesador.
6. **Acceso a memoria.** Cada núcleo ve la memoria local a través de cachés L1 de datos e instrucciones con política de reemplazo LRU (Least Recently Used) y sin coherencia hardware, ya que es posible particionar la memoria local entre los núcleos. Es posible implementar mecanismos de coherencia software si fuese necesario compartir algún área de memoria.

   Cada bloque de la memoria compartida del agrupamiento de cómputo es accedido mediante enlaces directos desde los diferentes núcleos mediante arbitrio *round robin*.
7. **Envío de mensajes.** Se puede configurar tanto la ruta y como el flujo para garantizar límites y latencias en el paso de datos desde la fuente. También

se puede configurar el algoritmo de inyección. Van Amstel [3] detalla cómo se garantizan los servicios.
En la temporización hay que considerar las interrupciones entre parejas de núcleos.
La NoC de datos puede operar con garantías debido a los *routers* no bloqueantes y el control de flujo realizado en el origen.

Aplicación a tiempo real Toda agrupación de computo y subsistema de entrada/salida contiene una unidad de soporte de depuración (*Debug Support Unit*) con un contador de 64-bits. Cada contador es direccionable en memoria local y puede leerse por cualquier núcleo. El mensaje de inicialización de este contador, enviado en modo *broadcast*, produce un pequeño desfase entre agrupaciones. Cada núcleo implementa su propio reloj de tiempo real, que da soporte a una implementación ligera de los *timers* POSIX.

Conclusiones Con Kalray nos encontramos con procesadores *many-core* dirigidos a sistemas de tiempo real. La arquitectura anidada nos proporciona un sistema de particionado directo. La NoC tiene tiempos de paso de mensajes acotables y cada núcleo presenta ejecución en orden. Se cuenta con una noción de memoria local, aunque su tamaño de 128kB puede ser algo escaso.

3. Mapeado y análisis

3.1. Mapeado

Para conseguir que un sistema de tiempo real sea ejecutado en procesadores *many-core*, se pueden asignar ciertas tareas de manera permanente a núcleos del procesador, pudiendo llegar a reducir así el tiempo de respuesta de peor caso al minimizar los cambios de contexto. El mapeado de tareas en procesadores *many-core* es un problema NP-Completo por lo que las soluciones utilizadas actualmente son heurísticas. En estos algoritmos se tienen en cuenta la distancia entre los núcleos que se van a comunicar y la congestión que pueda haber en los enlaces que se van a utilizar. Debido a que la complejidad de la colocación en un sistema aumenta de manera exponencial con el tamaño del software (número de tareas) y del *hardware* (cantidad de núcleos), la gran mayoría de algoritmos son heurísticos genéticos [12], simulación [6] o ramificación y poda [9].

También se puede buscar reducir en lo máximo posible el consumo energético utilizando mapeados eficientes de tareas y reduciendo la frecuencia de funcionamiento del procesador mientras se mantiene la planificabilidad [14].

Que hay que tener en cuenta a la hora de mapear tareas en un procesador:

- Las tareas que accedan a dispositivos deben de estar lo más cerca posible a los mismos.
- Las tareas que utilicen asiduamente la memoria RAM deben de colocarse lo más cercanas a su bloque.

- Las tareas que se comuniquen entre ellas deben colocarse cerca.
- Hay que evitar congestiones en la NoC.
- Las tareas más críticas o con menor plazo tienen que estar "mejor" situadas que las demás.

En aplicaciones complejas que demandan flexibilidad puede ser de gran interés poder modificar el mapeado en tiempo de ejecución, en momentos de baja carga del sistema, según las necesidades que tenga el propio sistema.

3.2. Técnicas de análisis

Existen ciertos elementos a modelar, muy dependientes de la arquitectura y de la distancia entre los elementos:

- **Accesos a memoria**: Cuando se accede a memoria local no se compite con otros núcleos. Al acceder a memoria compartida no solo se compite por el propio acceso a la memoria si no por el uso de los enlaces de la NoC. Según la arquitectura, la zona de memoria compartida puede tener bloques reservados a ciertos núcleos, pero sigue siendo necesario analizar la contención en la NoC.
- **Paso de mensajes**: Se compite con el tráfico que utilicen los mismos enlaces que el mensaje. Hay arquitecturas que dividen el tráfico en diferentes *streams* para segregar lo máximo posible el tipo de tráfico.
- **Accesos a dispositivos**: En la mayoría de las arquitecturas los dispositivos están distribuidos a lo largo de la NoC, y por tanto se compite por los enlaces para acceder a ellos.

Los elementos de modelo y las técnicas de análisis que se deriven se podrán añadir en el futuro a una herramienta de análisis de planificabilidad tal como MAST [8], que ya proporciona modelos de sistemas distribuidos.

3.3. Protocolos de sincronización

Los núcleos tienen que ser capaces de sincronizarse a la hora de interactuar entre ellos y cuando utilicen variables compartidas. Es por esto que el sistema operativo debe proporcionar un protocolo de sincronización que suponga poca sobrecarga en la NoC.

3.4. Técnicas de particionado espacial y temporal

Se requieren protocolos con los que mantener las diferentes particiones aisladas de manera espacial y temporal. Las particiones no podrán estar completamente aisladas pues es preciso tener en cuenta que hay cierta contención para el intercambio de mensajes y por el tráfico que atraviesa otras particiones para acceder a recursos compartidos.

En la actualidad el particionado se realiza a la vez que el mapeado de manera estática a través de algoritmos heurísticos. Los algoritmos genéticos se usan

mucho en este contexto y pueden evaluar diferentes características como son: los ciclos que van a tardar en comunicarse diferentes tareas, la probabilidad de que haya comunicación entre las tareas, la contención en los enlaces debida al tráfico y la latencia y probabilidad de acceso a recursos compartidos.

4. Sistemas operativos

Realizar software para sistemas de tiempo real en arquitecturas *many-core* requiere conocimientos de la arquitectura a la hora de diseñar servicios del sistema operativo tales como sincronización de tareas, localización de tareas, localización del código y datos, actualización de datos, acceso a periféricos, etc.

Unos de los elementos más críticos de cualquier sistema operativo es el planificador. Gu realizó un trabajo sobre un planificador de un sistema *many-core* sin memoria compartida [7] proporcionando un protocolo para simplificar la programación paralela y utilizando paso de mensajes entre objetos.

Un sistema operativo de gran interés es Manycore Operating System for Safety-Critical (MOSSCA) [11] que se basa en cumplir los siguientes requisitos y propiedades:

- El sistema debe tener un comportamiento temporal predecible y conducir a la implementación de sistemas analizables.
- El particionado en tiempo y espacio evita interferencias y facilita el análisis. Hay que tener un cuidado especial con los recursos compartidos.
- Los procesadores de la misma aplicación pueden comunicarse entre ellos. Para comunicarse a través de los límites de particionado hay que utilizar el sistema operativo.

MOSSCA es un sistema operativo basado en una arquitectura *microkernel* distribuida en cada núcleo, donde se ofrecen servicios en relación al *hardware* del que dispone cada núcleo.

Otro ejemplo de este tipo de sistemas es eSOL eMCOS [5], que es un sistema operativo de tiempo-real diseñado para embebidos con *many-cores*. Su arquitectura de *microkernel* distribuida incluye servicios básicos tales como paso de mensajes, planificación local y gestión de hilos. Actualmente es compatible con los *many-core* Kalray MPPA2-256 y Tilera Gx8036. La planificación se realiza mediante dos planificadores, uno que asigna las tareas de mayor prioridad a cada núcleo para que puedan ejecutarse siempre que estén listos. El resto de tareas están bajo otro planificador que las distribuye sobre los núcleos balanceando su carga y buscando alto *throughtput*.

5. Conclusión

Se ha seleccionado como mejor contendiente para el reto de llevar el tiempo-real a los procesadores *many-core* la arquitectura de Kalray ya que proporciona características que resultan de gran interés:

- Ejecución en orden dentro de cada núcleo.
- Posibilidad de tener tareas con memoria privada cuando se mapea directamente a un núcleo, disponiendo cada uno de ellos de 128kB de la memoria de la agrupación. El acceso a esta memoria se puede configurar para que no haya interferencias con el resto de núcleos de la agrupación, aunque sí podría haber interferencias con la NoC o el núcleo de gestión.
- No hay coherencia de caché hardware.
- Paso de mensajes para la comunicación entre núcleos con tiempos acotables.
- Sistema anidado escalable.
- Posibilidad de determinar el mapeado de las tareas en el procesador, pudiendo implementar un algoritmo de mapeo.
- Aislamiento entre tareas de diferentes agrupaciones, al menos mientras no necesiten acceder a la NoC para intercambiar mensajes entre sí.

Como se dice en [2] Kalray dispone además de un kit de desarrollo de software junto con un sistema *software* ligero que proporciona un *run-time* capaz de correr software que utiliza servicios POSIX y permite crear sistemas operativos personalizados. También existe un sistema operativo para esta plataforma llamado eSOL eMCOS [5].

Queda como trabajo futuro la adquisición de dicho dispositivo para realizar un estudio más a fondo. Será objeto de este estudio el desarrollo de modelos y técnicas de análisis, heurísticas de mapeado y la implementación de un sistema operativo de tiempo real en el mismo.

Referencias

1. Certification Authorities Software Team: Position Paper-32, Multi-core Processors (2014), http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-32.pdf
2. de Dinechin, B.D., Ayrignac, R., Beaucamps, P.E., Couvert, P., Ganne, B., de Massas, P.G., Jacquet, F., Jones, S., Chaisemartin, N.M., Riss, F., Strudel, T.: A clustered manycore processor architecture for embedded and accelerated applications. 2013 IEEE High Performance Extreme Computing Conference (HPEC) pp. 1–6 (2013)
3. de Dinechin, B.D., Durand, Y., van Amstel, D., Ghiti, A.: Guaranteed services of the noc of a manycore processor. In: Proceedings of the 2014 International Workshop on Network on Chip Architectures. pp. 11–16. NoCArc '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2685342.2685344
4. EASA: CERTIFICATION MEMORANDUM Subject Development Assurance of Airborne Electronic Hardware (2011), http://www.easa.europa.eu/system/files/dfu/certification-docs-certification-memorandum-EASA-CM-SWCEH-001-Issue-01-Rev-01-Development-Assurance-of-Airborne-Electronic-Hardware.pdf
5. eSOL: emcos, http://www.esol.com/embedded/emcos.html
6. Giannopoulou, G., Stoimenov, N., Huang, P., Thiele, L., de Dinechin, B.D.: Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. Real-Time Systems 52(4), 399–449 (2016), http://dx.doi.org/10.1007/s11241-015-9227-y

7. Gu, X., Liu, P., Yang, M., Yang, J., Li, C., Yao, Q.: An efficient scheduler of {RTOS} for multi/many-core system. Computers & Electrical Engineering 38(3), 785 – 800 (2012), http://www.sciencedirect.com/science/article/pii/S0045790611001340, the Design and Analysis of Wireless Systems and Emerging Computing Architectures and Systems
8. Harbour, M.G., Gutiérrez, J.J., Medina, J.L., Palencia, J.C., Drake, J.M., Rivas, J.M., Martínez, P.L., Cuevas, C.: Mast: Bringing response-time analysis into real-time systems engineering, http://mast.unican.es/mast_analysis_techniques.pdf
9. Indrusiak, L.S., Harbin, J., Burns, A.: Average and Worst-Case Latency Improvements in Mixed-Criticality Wormhole Networks-on-Chip. 2015 27th Euromicro Conference on Real-Time Systems pp. 47–56 (2015)
10. Jean, X., Berthon, M.G.G., Fumey, M.: MULCORS - Use of Multicore Processors in airborne systems. Journal of Chemical Information and Modeling 53, 160 (1989), https://www.easa.europa.eu/system/files/dfu/CCC_12_006898-REV07-MULCORSFinalReport.pdf
11. Kluge, F., Triquet, B., Rochange, C., Ungerer, T.: Operating systems for manycore processors from the perspective of safety-critical systems. 8th annual workshop on Operating Systems for Embedded Real-Time applications p. 16 (2012)
12. Mesidis, P., Indrusiak, L.S.: Genetic mapping of hard real-time applications onto noc-based mpsocs; a first approach. In: Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on. pp. 1–6 (June 2011)
13. Ni, L.M., McKinley, P.K.: A survey of wormhole routing techniques in direct networks. Computer 26(2), 62–76 (Feb 1993), http://dx.doi.org/10.1109/2.191995
14. Sayuti, M.N.S.M., Indrusiak, L.S.: A function for hard real-time system search-based task mapping optimisation. In: 2015 IEEE 18th International Symposium on Real-Time Distributed Computing. pp. 66–73 (April 2015)
15. Sodani, A., Gramunt, R., Corbal, J., Kim, H.S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., Liu, Y.C.: Knights landing: Second-generation intel xeon phi product. IEEE Micro 36(2), 34–46 (Mar 2016)
16. Tilera Corporation: Tile Processor Architecture Overview for the Tile-Gx Series (2012), http://www.mellanox.com/repository/solutions/tile-scm/docs/UG130-ArchOverview-TILE-Gx.pdf

