Búsqueda Paralela Exhaustiva Aplicada a Cadenas de ARNi

Jesús García-Ramírez, Ivo H. Pineda T. ,María J. Somodevilla, Mario Rossainz, Concepción Pérez de Celis

Benemérita Universidad Autónoma de Puebla,
Facultad de Ciencias de la Computación,
México
gr_jesus@outlook.com,
{ipineda,mariasg,rossainz,perezdecelis}@cs.buap.mx

Resumen Dentro de la bioinformática la búsqueda de cadenas de ADN es un problema que requiere procesamiento masivo, por esta razón el enfoque paralelo es una buena opción para reducir los tiempos de ejecución. En este artículo se propone un algoritmo paralelo y se muestran los resultados de su implementación mediante la interfaz de paso de mensajes, para encontrar si es que una sub-cadena esta contenida dentro una cadena de ADN o ARNi, así mismo se propone una forma de guardar los datos para que estos sean enviados dentro de la red de computadoras.

Palabras clave: Bioinformática, Búsqueda en archivos, Cadenas de ADN,ARNi.

1. Introducción

Los métodos de búsqueda en anchura y profundidad son técnicas para encontrar solución a problemas dentro de un gran campo de soluciones. Estos son los principales enfoques dentros de la inteligencia artificial, las principales desventajas que éstos presentan son que sus complejidades en cuanto a tiempo son exponenciales. Muchos enfoques se han desarrollado en los últimos años como las heurísticas no garantizan que se obtenga una solución óptima. Otro enfoque es la Búsqueda Exhaustiva Paralela (BEP), la cual se realiza desde diferentes puntos o en orden distinto dentro del espacio de soluciones, ya que ésta se realiza como procesos independientes, se puede implementar mediante un enfoque paralelo.

Implementar un enfoque paralelo en las búsquedas puede ser de gran ayuda, ya que se reducen tiempos de ejecución, siempre y cuando se cuenten con los recursos necesarios, para que éstas se pueda entregar los resultados correctos y completos. Existen diferentes herramientas para realizar enfoques paralelos, algunos de estos son: Message Passing Interface (MPI) que utiliza los recursos de diferentes computadoras conectadas en red, asignando tareas a cada una, de esta forma se pueden asignar distintas búsquedas a cada computadora (nodo), los programas son escritos de manera secuencial y las librerias son llamadas para

enviar y recibir mensajes, de esta manera se debe pueden realizar implementaciones paralelas [1]; programación multicore, en este enfoque se utilizan todos los núcleos que tienen los procesadores de las computadoras, asignando a cada uno búsquedas diferentes; programación con CUDA, las tarjetas gráficas que se utilizaron inicialmente para procesamiento de gráficos, ahora se utilizan para realizar programación paralela, aprovechando la totalidad de los núcleos con los que cuentan las tarjetas.

Para este trabajo se pretende implementar una búsqueda exhaustiva paralela con datos de cadenas de ARNi mediante un enfoque paralelo (MPI), de esta manera se pueden reducir tiempos de ejecución, optimizando la manera en que se realizan las búsquedas cadenas de ARNi de gran longitud para encontrar algún patrón de coincidencia en dichas cadenas.

2. Estado del Arte

Dentro de la bioinformática un tema que se ha estudiado desde los años ochenta a la fecha es la búsqueda en cadenas de ADN desde su representación computacional (archivos de texto con una representación como cadenas texto). Dada la amplia variedad de algoritmos de búsqueda que se han propuesto, mención especial merece el algoritmo BLAST inspirado en el algoritmo conocido como "Máxima subsecuencia común"(Longest Common Subsequence LCS) el cual se concibió inspirado en la la técnica conocida como Programación Dinámica [2], dicha técnica resuelve subproblemas y se guarda su resultado para encontrar la solución global cuando todos son resueltos [3].

Un nuevo algoritmo para la búsqueda en cadenas de ADN es reportado por William Pearson y David Lipman se discute en [4]. En él se presenta una modificación al algoritmo de búsqueda de cadenas de ADN FASTP, a dicha modificación se le da el nombre de FASTA, el cuál es más rápido que el algoritmo FASTP. FASTP es un algoritmo que sirve para comparar cadenas de ADN, buscar proteinas o cadenas de ADN y comparar dichas cadenas con las que contenga alguna base de datos.

Gulio Pavesi et al. reportan un algoritmo que realiza una búsqueda exhaustiva en patrones de cadenas de ADN. En él se buscan subcadenas de forma que se encuentre un número determinado de veces en la cadena [5].

Por lo regular, las cadenas de ADN se guardan en archivos de texto, los cuales contienen miles de datos. Por esta razón el tiempo de búsqueda en este tipo de archivos es considerable, Ning Zemin et al. proponen el algoritmo SSAHA (Sequence Search and Alignment by Hashing Algorithm) que implementa búsquedas de subcadenas de ADN y crea una tabla Hash para realizarlas con una mayor rapidez [6], otro enfoque que también utiliza tablas Hash es reportado en [15].

La forma de representar las cadenas de ADN puede ayudar a mejorar la búsqueda, así como muestran Sonnhammer y Durbin, quienes proponen que la forma de representación sea una forma matricial, lo cual permite ver las cadenas de ADN de manera gráfica [7], al igual que en [14], donde se reporta un sistema en

el cual se pueden visualizar de manera gráfica las cadenas de ADN, ya que muchas veces la observación puede ser de gran ayuda para el desarrollo de proyectos de éste tipo.

En los últimos años se han desarrollado diferentes enfoques para éste tipo de búsquedas, como se mencionó anteriormente el enfoque paralelo puede ser una buena alternativa para implementarlas, por ejemplo, en [10] se muestra un enfoque paralelo que utiliza tarjetas gráficas CUDA, el cuál puede ser más rápido a medida que el número de cores de las tarjetas va aumentando, hay que tomar en cuenta que se debe tener el hardware adecuado para que éste enfoque resulte realmente eficiente, ya que, el tiempo de carga de datos en la memoria de la tarjeta es costoso por el bus que implementa. El trabajo presentado en [13] se utiliza un enfoque multi-hilo el cual tiene la ventaja de que los datos son manipulados directamente en la computadora y no se pierde tiempo en enviar los datos como en el enfoque anterior.

Otras formas de realizar la búsqueda dentro de las cadenasson mencionadas a continuación, [11] propone que se aplique el metodo de búsqueda k-mer, donde la principal desventaja que se ve es la memoria RAM, ya que este método requiere la utilización de espacio en memoria. Otros enfoques implementan algoritmos iterativos como es reportado en [12].

En éste trabajo se propone una la BEP, la cual se realiza de una manera paralela implementando MPI, siendo enviado a cada nodo un diccionario con los datos extraídos de las bases de datos encontradas, de esta manera se realiza la búsqueda de manera exhaustiva, balancenado la carga de trabajo para así reducir los tiempos de ejecución tomando en cuenta el identificador de cada nodo. Para las pruebas se aumenta la longitud de la cadena a ser búscada para determinar si impacta en el tiempo de ejecución.

2.1. Levaduras

Las levaduras son los agentes de la fermentación y se encuentran naturalmente en la superficie de las plantas, el suelo es su principal hábitat encontrándose en invierno en la capa superficial de la tierra. En verano, por medio de los insectos, polvo y animales, son transportados hasta el fruto, por lo que su distribución se produce al azar. Existe un gran número de especies que se diferencían por su aspecto, sus propiedades, sus formas de reproducción y por la forma en la que transforman el azúcar. Las levaduras del vino pertenecen a varios géneros, cada uno dividido en especies. Las especies más extendidas son Saccharomyces ellipsoideus, Kloeckera apiculata y Hanseniaspora uvarum, las cuales representan por sí solas el 90 % de las levaduras utilizadas para la fermentación del vino. Como todos los seres vivos, tienen necesidades precisas en lo que se refiere a nutrición y al medio en que viven. Son muy sensibles a la temperatura, necesitan una alimentación apropiada rica en azúcares, elementos minerales y sustancias nitrogenadas, tienen ciclos reproductivos cortos, lo que hace que el inicio de la fermentación sea tan rápido, pero así como se multiplican, pueden morir por la falta o el exceso de las variables mencionadas.

Tomando como referencia estos elementos es que se decide utilizar el conjunto

de datos asociados a la especie de levadura conocida como *Schizosaccharomyces pombe* o *S. pombe*, también llamada ''fission yeast" en inglés. S. pombe, la cual se divide por fisión binaria y produce dos células de igual tamaño. Es muy usada como organismo modelo en el estudio del ciclo celular, ARNi, entre otros. Estas características sobre el comportamiento del hongo unicelular eucariota os motivó al diseño de la herramienta para poder encontrar de manera rápida y eficiente información asociada a las cadenas de la levadura.

3. Metodología Propuesta

En esta sección se propone una forma de realizar una BEP dentro de una red de computadoras. Las búsquedas se realizaron en archivos de texto que contienen un representación de cadenas de ARNi,con su respectivo nombre. La arquitectura que se adopta en ésta propuesta es tener un nodo principal, el cual realiza el pre-procesamiento del archivo, dejando los datos de los archivos en un diccionario, los cuales contienen las características y las cadenas, el cual se envía a todos los nodos para realizar la búsqueda de manera exhaustiva. La arquitectura, así como el funcionamiento general del sistema se puede observar en la Fig. 1, en la que existe un nodo principal y n nodos para el procesamiento paralelo.

El pre-procesamiento de los datos se realiza mediante un archivo de texto de entrada que contiene el nombre de una cadena de texto, así como sus características. El balance de datos a procesar se realiza en cada nodo con base en su número de identificador. Posteriormente se realiza la búsqueda en cada nodo de procesamiento y si se encuentra una subcadena prestablecida dentro de la cadena de ARNi se avisa al nodo principal y se imprime en que cadena de ARNi se encontró la subcadena.

Para realizar la búsqueda en cada nodo se realiza un pre-procesamiento, el cual es implementado mediante el análisi del archivo de texto en el que se guardan las características y las siguientes líneas se realiza una unión de todas estas para dejarlas como una sola cadena, posteriormente se distribuye la carga a cada nodo de procesamiento que se tenga dentro de la red de computadoras estableciendo los límites de donde se empieza y donde se termina a realizar la búsqueda en cada nodo. Finalmente se realiza la búsqueda en cada nodo dependiendo de la carga de trabajo que se le asigno. En el Algoritmo 1 se muestra el algoritmo para el nodo principal donde se realiza el pre-procesamiento de los archivos, donde, si la línea contiene el caracter punto y coma (;), significa que la línea contiene las características, de lo contrario se concatena la cadena para formar la cadena general, finalmente los datos son enviados a todos los nodos donde se realizará la búsqueda. En el Algoritmo 2 se muestra el algoritmo para los nodos de procesamiento donde se establece los rangos de búqueda en el nodo, si se encuentra la cadena retorna un mensaje al nodo principal.

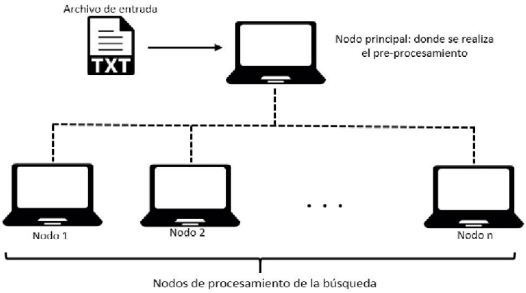

Figura 1. Arquitectura general del sistema.

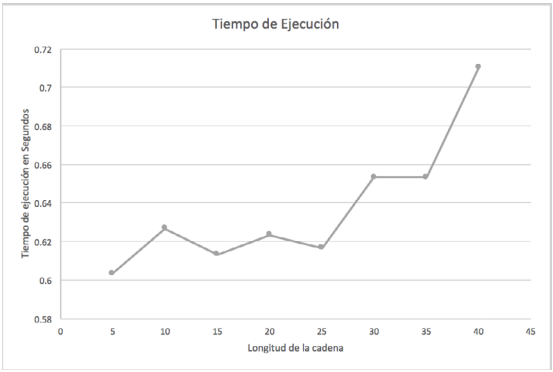

Figura 2. Gráfica de tiempos de ejecución dependiendo del tamaño de la cadena de búsqueda.

Algoritmo 1 Pseudocódigo para el nodo principal

Entrada: Archivo de entrada con cadenas de ARNi.
Salida: Diccionario con el nombre y las características de la cadena (*diccionarioCadenas*).

```
for linea in ArchivoCadenas do
    if linea contiene ";" then
        diccionarioCadenas[linea]=Cadena
        Nombre=linea
    else
        Cadena=linea
    end if
end for
```

Algoritmo 2 Pseudocódigo para los nodos de procesamiento

Entrada: Entrada: Diccionario con las cadenas de texto (*diccionarioCadenas*), Cadena a buscar (*cadBusqueda*).
Salida: Mensaje de en qué cadena encontró *cadBusqueda*.

```
NoProcesadores=número de nodos de procesamiento
Id=identificador del nodo
Longitud=diccionarioCadenas.length()
LongitudProcesamiento=int(Longitud/(NoProcesadores)+1)
Inicio=id*LongitudProcesamiento+1
Fin=Inicio+LongitudProcesamiento-1
if Fin es mayor que Longitud then
    Fin=Longitud
end if
for Inicio to Fin do
    if diccionarioCadenas in diccionarioCadenas[Inicio] then
        Print Se encontró la cadena en: Inicio
    end if
end for
```

4. Experimentos y Resultados

Para los experimentos se utilizó el lenguaje de programación Python con las librerías para implementar MPI, en una máquina virtual, con un sistema opertivo Linux Ubuntu con un solo núcleo y una frecuencia de reloj de 2.6 GHz, 2GB de Memoria RAM y un disco duro virtual de 10 GB.

Los archivos que se utilizaron para las pruebas fue la base de datos de cadenas de ARNi que se encuentra en del sitio oficial de Pombase [9]. Dichos archivos están organizados de la siguiente manera. Primero se tiene una descripción de la cadena de ARNi que se va a procesar, seguida de la representación con letras, el cual contiene 5280 cadenas de ARNi.

Se realizaron experimentos con diferente número de nodos para determinar si la carga de trabajo se realiza correctamente. En la Tabla 1 se reporta cuantas coincidencias encuentra por nodo, por lo que se puede observar se distribuye la carga de trabajo a medida que se aumenta el número de nodos, ya que se

van encontrado menor número de coincidencias en cada nodo, además la suma de coincidencias debe ser la misma para todas las pruebas ya que lo único que cambia es el número de nodos en los que se ejecuta el programa.

Cuadro 1. Número de coincidencias encontradas en cada nodo del sistema.

Número de nodos	Número de coincidencias en cada nodo
3	562,508,575
4	417,405,396,427
5	329,341,294,329,352
6	270,292,260,249,272,302
7	233,249,230,206,236,241,250
8	205,212,214,191,176,220,207,220

Para determinar si la longitud de las cadenas que se tiene que buscar tiene un impacto directo sobre el tiempo de ejecución, se realizaron experimentos con diferentes longitudes de cadena para su búsqueda, como se puede observar en la Figura 2 los tiempos de ejecución van aumentando conforme la longitud de la cadena de búsqueda crece, esto es lógico, ya que, el número de comparaciones va aumentando a medida que la longitud de la cadena va creciendo.

5. Conclusiones

Haber adoptado un algoritmo con enfoque paralelo en problemas que requieran búsquedas exhaustivas o comparaciones dentro de un espacio de soluciones muy grande, puede ayudar a encontrar la solución del problema en menos tiempo, ya que el trabajo que correspondería a un solo procesador en una computadora convencional, se distribuye entre una red de computadoras o en diferentes procesadores dentro una sola computadora.

Como se pudo observar en la Figura 2 y en la Tabla 1 la distribución del trabajo en diferentes computadoras ayudó a que se redujeran los tiempos de ejecución, también dependiendo de la longitud de la cadena a buscar se aumenta el tiempo de ejecución.

Después de varios experimentos hemos llegado a la conclusión que un posible enfoque puede consistir en tener a disposición mayores cantidades de datos y poder contar con la opinión de expertos del área para brindar opciones de búsuqeda pensando en subcadenas de una determinada longitud y con significado. Como elemento importante es el hecho que con las configuraciones actuales de cómputo se pueden obtener resultados favorables sin necesidad de tener una computadora de alto costo y tener resultados en tiempos de ejecución aceptables..

Referencias

1. Foster I.: *Designing and Building Parallel Programs*, Addison Wesley, primera edición (1995).
2. Altschul S, Gish W.,Miller W., Myers E., Lipman D.: *Basic Local Alignment Search Tool.* Journal of molecular biology, 215(3), pp. 403-410 (1990).
3. Levitin, A.: *Introduction to Design and Analysis of Algorithm*, Pearson Education, tercera edición (2012).
4. Pearson W., Lipman D.: *Improved tools for biological sequence comparison.* In: Proceedings of National Academy of Science (Biochemistry), Vol. 85, pp. 2444-2448 (1988).
5. Pavesi G., Mauri G., Pesole, G.: *An algorithm for finding signals of unknown length in DNA sequences*, Bioinformatics, 17(1), pp. S207-S214 (2001).
6. Nimg Z., Cox A., Mullikin J.: *SSAHA: A Fast Search Method for Large DNA database*, Genome Research, pp. 1725-1729 (2001).
7. Sonnhamer E., Durbin R.: *A dot-matrix program with dynamic threshold control suited for genomic DNA and protein sequence analysis*, Gene, Vol. 104 (1996).
8. Russel S., Norving P.: *Artificial Intelligence, A Modern Approach*, Pearson Education, tercera tercera edición (2010).
9. Wood V, Gwilliam R, Rajandream MA, Lyne M, Lyne R, Stewart A, Sgouros J, Peat N, Hayles J, Baker S, et al.: *The genome sequence of Schizosaccharomyces pombe.*Nature (2003).
10. Encarnaco G., Sebastiao N.: *Advantages and GPU implementation of high-performance indexed DNA search based on suffix arrays.* En: Proceedings de Conferencia Internacional en Computo de Alto rendimiento y simulación, pp. 49-55 (2011).
11. AIslam T., Pramanik S.; Ji X., James R. Cole J., Zhu Q.: *Back translated peptide K-mer search and local alignment in large DNA sequence databases using BoND-SD-tree indexing*, En proceedings de IEEE 15th International Conference of Bioinformatics and Bioengeenering (BIBE), pp.1-6 (2015).
12. Cheng P., Chen H., Kao J.: *Protein surface search in DNA-binding protein prediction by Delaunay triangulation modeling*, en Proceedings de: International Computer Symposium, pp. 783-788 (2010).
13. Erodula K., Bach C., Bajwa H.: *Use of Multi Threaded Asynchronous DNA Sequence Pattern Searching Tool to Identifying Zinc-Finger-Nuclease Binding Sites on the Human Genome*, en Proceedings de: Eighth International Conference on Information Technology, pp. 985-991 (2011).
14. Glisovic N.: *System for DNA visualization and clustering in searching through information.* en Proceedings de: International Symposium on Computational Intelligence and Informatics, pp.169-170 (2010).
15. *Fast searching in biological sequences using multiple hash functions.* In Proceedings de: International Conference on Bioinformatics & Bioengineering, pp. 175-180 (2012).