

# **Whole genome alignment in high performance computing environments**

## *Informe del progreso de trabajo de investigación*

**Julio César García Vizcaíno**

Departamento de Arquitectura de Computadores y Sistemas Operativos

Universidad Autónoma de Barcelona

jcgarcia@aomail.uab.es

8 de mayo de 2013

## **Datos del doctorando**

Nombre Completo: Julio César García Vizcaíno  
NIE: Y1497678R

## **Datos del proyecto de tesis**

Estudio de doctorado: Computación de Altas Prestaciones

Título del proyecto de tesis doctoral: *Whole genome alignment in high performance computing environments.*

Directores de la tesis: Antonio Espinosa, Juan Carlos Moure.

Línea de investigación: Aplicaciones bioinformáticas.

Régimen de dedicación: completa.

Año de inscripción: 2011.

Año previsto para la presentación de Tesis Doctoral: 2014

Firmado

Director	Director	Estudiante
Antonio Espinosa	Juan Carlos Moure	Julio César García Vizcaíno

# 1. Resumen

Actualmente la generación de nueva información genómica ha creado la necesidad de almacenar y procesar estos nuevos datos computacionalmente hablando. Una de las tareas en la que nos centramos es el alineamiento de genomas ya que es una tarea computacionalmente intensiva. La información básica de un genoma es el ADN. El ADN es una cadena compuesta por el alfabeto  $\sigma = a, c, g, t$ , cada uno de estos elementos es llamado nucleótido. El alineamiento de genomas consiste en encontrar un mapeo de cada posición en un genoma de consulta a su correspondiente posición en el genoma de referencia. Es decir, encontrar las similitudes y diferencias de las cadenas de ADN entre genomas.

El alineamiento de genomas es un modelo matemático utilizado por los biólogos para determinar la similitud entre genomas. Existe diferentes algoritmos propuestos para el alineamiento de dos secuencias como [8] y [11]. Estos algoritmos funcionan bien con tamaños de secuencias pequeños como un gen, 20-30kbp<sup>1</sup>. Sin embargo, son ineficaces al alinear genomas enteros. La complejidad computacional y espacial es  $O(nm)$ , donde  $n$  y  $m$  son las longitudes de los genomas a alinear. Estos algoritmos tienen problemas de mayores requerimientos de memoria o de tiempos de ejecución inaceptables.

Para grandes secuencias, como el genoma humano (3Gbp), el uso de recursos computacionales pueden ser extremadamente grande y con un largo tiempo de ejecución. Es por ello que han surgido alternativas para reducir el tiempo de ejecución del alineamiento de secuencias, como [7, 5, 2, 12, 10, 4, 9]. Una de esas alternativas es la utilización de la heurística basada en las coincidencias únicas máximas en las secuencias a alinear. Al encontrar las coincidencias únicas se puede realizar el alineamiento de genomas enteros reduciendo el espacio del alineamiento a las regiones entre las coincidencias únicas máximas. Esto permite reducir la complejidad de un alineamiento óptimo  $O(nm)$  a la búsqueda de coincidencias únicas que puede resolver de manera lineal  $O(m)$  mediante estructuras de datos de indexación como el árbol de sufijos.

Formalmente el problema de la búsqueda de coincidencias únicas se define como:

**Definición 1** *Dadas dos secuencias (genomas)  $R = r_1r_2 \dots r_n$  y  $Q = q_1q_2 \dots q_m$ , y una longitud mínima  $l$ , se requiere encontrar todas las ocurrencias de las coincidencias únicas de longitud mínima  $l$ , entre  $R$  y  $Q$ .*

Existen diferentes aproximaciones para resolver el problema ver Figura 1, todas ellas involucran un uso intensivo de recursos de cómputo (memoria, procesador).

En consecuencia se busca **acelerar la búsqueda eficiente de coincidencias**

---

<sup>1</sup>bp es la unidad básica de medida en las secuencias de ADN y representa la longitud de un nucleótido

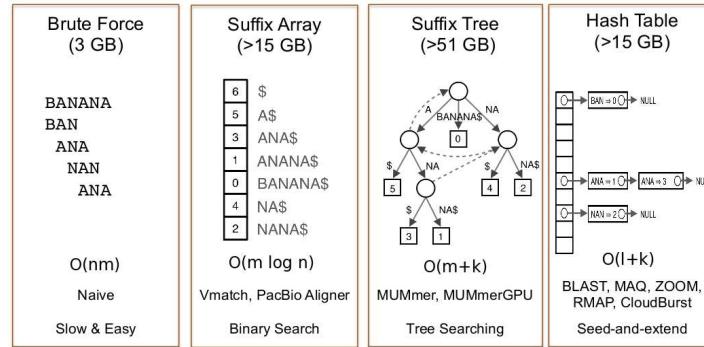


Figura 1: Diferentes estructuras de indexación para el problema de la búsqueda de coincidencias exactas máximas para el genoma humano como referencia.

**máximas únicas y que tenga en cuenta el uso de recursos de cómputo y memoria en un clúster multicore.** Las consideraciones que se toman en cuenta son:

- Reducir el número de operaciones en la búsqueda de MUMs: al agilizar la búsqueda se reduce el tiempo utilizado por cada MUM encontrado.
- Almacenar genomas de mayor tamaño con la memoria disponible: se hace necesario diseñar estructuras de datos que permitan la indexación de genomas mayores con menor uso de memoria, manteniendo la funcionalidad de la búsqueda de MUMs.
- Mejorar la localidad de los datos para reducir los fallos en cache y garantizar un menor tiempo de ejecución.

## 2. Introducción

### 2.1. MUM y MEM

La heurística basada en la búsqueda de coincidencias exactas máximas para el alineamiento de genomas requiere de la definición formal de una coincidencia exacta máxima (MEM).

**Definición 2** *Un MEM (Maximal Exact Match) es una subcadena común a dos genomas que es mayor que una longitud mínima específica  $d$  de tal manera que es máxima, esto es, que no puede ser extendida en ambos extremos sin incurrir en una discrepancia.*

Adicionalmente, a partir de la definición de MEM surge el concepto de la coincidencia única máxima (MUM).

**Definición 3** *Un MUM (Maximal Unique Match) es una subcadena única común a dos genomas que es mayor que una longitud mínima específica  $d$  de tal manera que es máxima, esto es, que no puede ser extendida en ambos extremos sin incurrir en una discrepancia.*

El objetivo es buscar todas aquellas subcadenas que son comunes a las secuencias de referencia  $R$  y consulta  $Q$  que son máximas de longitud.

## 2.2. Definición del problema a resolver

El problema de la búsqueda de las coincidencias máximas únicas de una longitud mínima ha sido identificado en varias aplicaciones, entre ellas MUMmer [6]. Aunque el algoritmo de MUMmer realiza la búsqueda de coincidencias exactas máximas, los requerimientos de recursos de cómputo aún son altos debido al tamaño de los datos de entrada.

Si la longitud de los genomas son muy grandes, ver Cuadro 1, la cantidad de búsquedas aumentan, el tiempo de búsqueda crece linealmente y el uso de memoria se convierte en un problema a resolver si se considera la disponibilidad finita de memoria en la mayoría de sistemas de cómputo personales. Por otro lado si la memoria no es una restricción, existe entonces problemas de acceso aleatorio a la estructura de datos de indexación, que ocasiona una degradación en el tiempo de ejecución de búsquedas de coincidencias únicas máximas.

De acuerdo a los datos del Cuadrs 1 se hace necesario la búsqueda de alterna-

Estructura	L [bp <sup>2</sup> ]	Cantidad de búsquedas	Búsqueda [s]	Uso de memoria [MB]
Árbol de sufijos	20	$9,87 \times 10^{18}$	169189,4	48665,12

Cuadro 1: Búsqueda coincidencias exactas para una cadena de referencia de 2960,21Mbp y una cadena de consulta de 2716,96Mbp

tivas que permitan llevar a cabo la búsqueda de manera eficiente. El uso de HPC nos permitiría resolver los problemas de escalabilidad (tiempo de búsqueda, uso de memoria) cuando se tienen cadenas muy grandes, como el genoma humano. En consecuencia, para resolver este problema se utiliza alguna estructura de datos que permita realizar una búsqueda distribuida de coincidencias máximas únicas de forma ágil en el genoma entero.

En resumen la búsqueda de coincidencias únicas máximas en una cadena, como el genoma humano, es un problema computacionalmente intensivo que es posible resolverse con estructuras de datos de indexación de cadenas.

Independientemente de la estructura de datos utilizada, lo que se desea es poder realizar búsquedas de coincidencias exactas a esa estructura de manera eficiente. Una de las consultas en que se requiere realizar rápidamente, para el alineamiento de genomas, es la búsqueda de coincidencias únicas máximas.

Una propuesta para reducir el tiempo de búsqueda de coincidencias únicas máximas es el uso de la paralelización de datos. Es decir dividir el genoma de consulta y realizar la búsqueda de coincidencias únicas máximas con threads a la estructura de indexación compartida por todos los threads.

### 2.3. Árbol de sufijos

Cualquier cadena de referencia de longitud  $n$  puede ser descompuesta en  $s$  sufijos, ver Figura 2, y estos sufijos pueden almacenarse en un árbol de sufijos. Para crear esta estructura de datos se requiere de un tiempo  $O(n)$  y para buscar una cadena en él requiere de un tiempo  $O(l)$  donde  $l$  es la longitud de la cadena [3]. Estas dos propiedades hacen al árbol de sufijos una estructura útil para un rango diverso de aplicaciones bioinformáticas, incluyendo: alineamiento de genomas [6].

La búsqueda de coincidencias únicas máximas en un árbol de sufijos se realiza

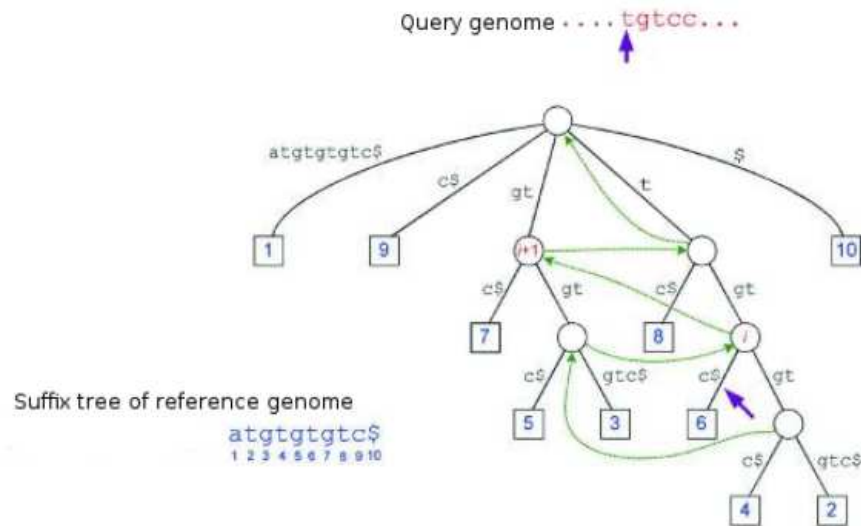


Figura 2: Árbol de sufijos para la palabra atgtgtgtc\$.

recorriendo el árbol con la cadena que se requiere encontrar la coincidencia única

máxima. Si la coincidencia termina en un nodo hoja, la coincidencia máxima es única en la secuencia de referencia. Verificando el carácter inmediato anterior al inicio de la posición de la coincidencia es posible determinar si es máxima.

Por lo tanto es posible identificar todas las coincidencias exactas máximas en tiempo proporcional a la longitud del genoma de consulta. Es importante resaltar que las coincidencias encontradas no son necesariamente únicas en el genoma de consulta. Esto se debe a que al recorrer el genoma de consulta, las coincidencias encontradas no nos permiten identificar si son únicas ya que no es posible determinar que coincidencias se encontrarán posteriormente en el genoma de consulta.

En la Figura 2 se muestra como una secuencia de consulta es buscada en el árbol de sufijos. El árbol representa la secuencia de referencia `atgtgtgtc$`. Las hojas representadas en la figura 2 por cuadrados indican la posición en la que inicia el sufijo. Por ejemplo, la hoja 7 representa el sufijo `gtc$` que inicia en la posición 7 de la secuencia de referencia y que está formado por la secuencia de aristas etiquetadas desde la raíz hasta el nodo hoja 7. En el punto mostrado en la figura, se ha encontrado la coincidencia en la posición  $i$ , indicada por la flecha. La coincidencia se extiende a la correspondiente posición en el árbol. En este caso sabemos que la coincidencia es única en la referencia por que estamos ubicados en un nodo hoja. El número del nodo hoja nos da la posición de inicio de la coincidencia en la referencia.

Para encontrar la siguiente coincidencia en la cadena de consulta, usamos los enlaces de sufijos, señalados en la Figura 2 por flechas punteadas. Estos enlaces son contruidos para cada nodo interno en el árbol. Un enlace apunta de un nodo  $x$  a un nodo  $y$  si el sufijo de la raíz a  $y$  es igual al sufijo desde la raíz a  $x$  con el primer carácter eliminado. Por ejemplo, la cadena del nodo  $i$  en la Figura 2 es `tgt` y del nodo  $i + 1$  es `gt`. Esa es la posición en el árbol correspondiente a la siguiente posición en la secuencia de consulta. Desde el nodo  $i$  podemos continuar la coincidencia bajando en el árbol para determinar que tan grande es la coincidencia puede extenderse. Las coincidencias son máximas hacia la derecha cuando se buscan en el árbol de sufijos. Para verificar si son máximas a la izquierda comparamos los caracteres precedentes en cada cadena. En el ejemplo de la Figura 2, la coincidencia inicia en  $i + 1$  en la cadena de consulta y en el nodo 7 de la cadena del árbol no es máxima a la izquierda porque el carácter precedente en ambas cadenas es `t`.

La clave para la búsqueda veloz en un árbol de sufijos es que hay un camino desde la raíz para cada sufijo del texto. Esto significa que se necesitan  $l$  comparaciones para encontrar una cadena de longitud  $l$ .

Otra mejora de implementación necesario para conseguir un tiempo y espacio lineal es el uso de enlaces de sufijos. Un enlace de sufijo es un puntero de un nodo interno etiquetado  $xS$  a otro nodo interno etiquetado  $S$ , donde  $x$  es un carácter arbitrario y  $S$  es posiblemente una subcadena vacía. Los enlaces de sufijos permi-

ten agilizar el recorrido del árbol sin visitar la raíz del árbol debido a que apuntan a la siguiente extensión del sufijo. Los enlaces de sufijos y la compresión de las etiquetas de las aristas son los requerimientos claves para la implementación del árbol de sufijos en  $O(m)$ .

## 2.4. Arreglo de sufijos mejorado

El arreglo de sufijos mejorado es una estructura de datos que consiste de un arreglo de sufijos, un arreglo de sufijos invertido, una tabla de valores LCP y una tabla hijo que almacena los nodos virtuales del árbol de sufijos. Cada algoritmo basado en un árbol de sufijos puede ser implementado para usar un arreglo de sufijos mejorado. Esto permite resolver el problema de la búsqueda de coincidencias únicas máximas con la misma complejidad que el árbol de sufijos.

Un arreglo de sufijos es un arreglo de enteros en el rango de 0 a  $n$ , especificando el orden lexicográfico de los  $n+1$  sufijos de la cadena  $R\$$ . Requiere  $4n$  bytes.

Un arreglo de sufijos inverso es un arreglo de enteros en el rango de 0 a  $n$  tal que  $ISA[SA[i]]=i$ . Puede calcularse en tiempo lineal del arreglo de sufijos. Requiere  $4n$  bytes.

La tabla de LCP almacena los valores del prefijo común más largo entre dos sufijos de  $SA$ . Se define como  $LCP[0]=0$ , y  $LCP[i]=lcp(R[SA[i]], R[SA[i-1]])$  para  $i \leq i \leq n$ . Requiere  $4n$  bytes en el peor caso.

La tabla hijo contiene tres valores por índice: arriba, abajo, sig/índice. Esta tabla mantiene de manera virtual los nodos internos del árbol de sufijos. La tabla hijo puede comprimirse para usar un solo campo en lugar de los tres valores mencionados anteriormente, ya que contiene información redundante. El requisito de espacio es  $4n$  bytes.

## 3. Estado de la investigación

La búsqueda de MUMS entre un genoma de Referencia y uno de Consulta tiene una característica importante que puede ayudarnos a paralelizar su búsqueda: podemos buscar más de un sufijo a la vez del genoma de Consulta. Utilizando una estructura de datos de indexación para buscar MUMs, encontramos MUM candidatos al ir recorriendo el genoma de Consulta. Un MUM candidato es un MUM que se almacena como un posible MUM pero que se tiene que verificar al final de la fase de búsqueda para garantizar que es un MUM único en el genoma de Consulta. El estado actual de la investigación consiste en la evaluación de la estructura



de datos (árbol de sufijos) utilizada en el algoritmo actual de la aplicación MUMmer, determinar los problemas de rendimiento entre el algoritmo y la estructura de datos y finalmente estudiar, evaluar y modificar la estructura de datos (Arreglo de sufijos mejorado, [1]) que permite reducir el tiempo de ejecución, mejor localidad espacial y temporal y la reducción en el uso de memoria. Finalmente se ha realizado una paralelización de la búsqueda de MUMs mediante OpenMP para su utilización en arquitecturas multi-core con el árbol de sufijos y con el arreglo de sufijos mejorado. Todos los experimentos se han desarrollado en el siguiente entorno:

1. GCC 4.7 con OpenMP + Linux
2. MUMmer 3.23
3. 2 procesadores Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz, con 6 cores cada procesador: 32KB cache L1 por core, 256KB cache L2 por core y 12MB cache L3 compartida.
4. RAM 96GB

### **3.1. Problemas de rendimiento en el árbol de sufijos**

La búsqueda de MUMs mediante un árbol de sufijos tiene una complejidad lineal, con respecto a la longitud del genoma de consulta, debido al uso de los enlaces de sufijos. El enlace de sufijo permite saltar a otra posición del árbol de sufijos con el siguiente sufijo evitando la necesidad de recorrer el árbol de sufijos desde el inicio del sufijo. Sin embargo el árbol de sufijos sufre de un problema de localidad espacial y temporal. Lo que convierte a la estructura en el cuello de botella del rendimiento del algoritmo de búsqueda de MUMs. Para demostrar que el árbol de sufijos sufre de problemas de localidad espacial y temporal se ha realizado una evaluación experimental para determinar el efecto de mantener un árbol de sufijos en el último nivel de cache y mantener el árbol de sufijos en memoria principal.

#### **3.1.1. Evaluación experimental**

Al mantener la estructura de datos en cache la búsqueda de MUMs será más rápida. Al crecer la estructura de datos se debe mantener en memoria principal, lo que ocasiona una disminución en las prestaciones del algoritmo de búsqueda de MUMs. Para demostrar la falta de localidad espacial del árbol de sufijos se estudiaron dos escenarios:

1. Árbol de sufijos cabe en el último nivel de cache.

## 2. Árbol de sufijos cabe en memoria principal.

Los genomas de Referencia utilizados fueron de 312Kbp y 170Mbp, el genoma de Consulta fue de 169Mbp en todos los experimentos. Al mantener la longitud del genoma de consulta constante garantizamos mantener la complejidad de la búsqueda de MUMs. El primer parámetro medido fue el CPI, al tener el árbol de sufijos en cache el CPI es de 1.61 y al tener el árbol de sufijos en memoria principal el CPI aumenta a 4.98, 3 veces mas que mantener el árbol de sufijos en cache. La razón de este aumento es el crecimiento de los ciclos de reloj en el segundo escenario, ya que las instrucciones ejecutadas prácticamente es el mismo en ambos escenarios. Respecto a los fallos de cache por instrucción, en L1 aumentan 12 % con respecto al escenario 1, en L2 23.79 % y en L3 los fallos pasan de 0.01 % a 2.83 %. Eso nos demuestra que la falta de localidad espacial del árbol de sufijos es relevante cuando el genoma de referencia indexado no se puede almacenar en algún nivel de cache. Los fallos de cache por instrucción aumentan el tiempo de ejecución de la búsqueda de MUMs. Las instrucciones de Load y Store se mantienen prácticamente iguali, en ambos escenarios los valores son 119.13 y 120.29 instrucciones de memoria Load y Store, respectivamente. El total de accesos a memoria por sufijo en ambos escenarios es de 0.013 y 4.107. En el segundo escenario: existen 14038588 accesos a memoria (49.14 %) para almacenar los MUM candidatos, 12775089 accesos a memoria (44.7 %) al utilizar los enlaces de sufijos y 1754614 accesos a memoria (6.14 %) al navegar el árbol de sufijos desde la raíz. Adicionalmente el ancho de banda (máximo práctico de 10927.706 MB/s) en la jerarquía de memoria es mayor en cache 2 y 3 en el escenario 1; en el escenario 2 el ancho de banda es mayor en memoria principal, pero en cache 2 cae un 64 % y en cache 3 cae un 65 % con respecto al escenario 1. Lo que indica que el algoritmo de búsqueda de coincidencias únicas máximas sufre de una falta de localidad espacial y temporal y no debido a una saturación de ancho de banda de memoria.

## 3.2. Paralelización de búsqueda de MUMs en Árbol de sufijos

Debido a los problemas de localidad espacial del árbol de sufijos se elaboró una propuesta de paralelización, ver Figura 3, para aprovechar el tiempo de espera del core. Los resultados de los genomas en la Tabla 2 se muestran en la Figura 5.

Los resultados muestran que aunque hay una reducción en el tiempo de ejecución, existe una pobre escalabilidad al utilizar más de un thread por core.

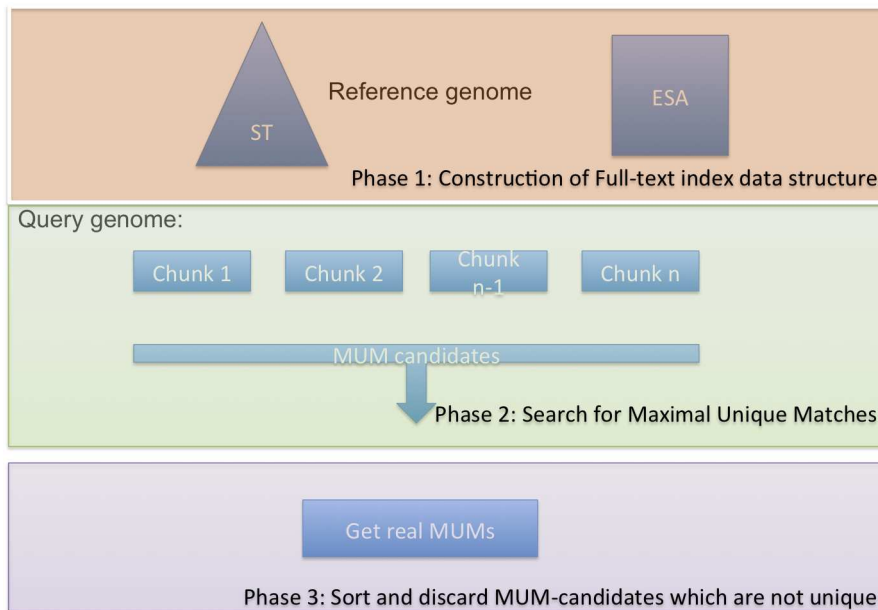


Figura 3: Búsqueda de MUMs en arquitecturas Multi-core.

Cuadro 2: Genomas usados en experimentos.

	1	2	3
Referencia	4.64 [Mbp]	169 [Mbp]	1031 [Mbp]
Consulta	5.5 [Mbp]	167 [Mbp]	1357 [Mbp]

### 3.3. Paralelización de búsqueda de MUMs en Arreglo de sufijos mejorado

Ya que se ha propuesto cambiar la estructura de datos para la búsqueda de MUMs por el arreglo de sufijos mejorado, hemos realizado también su paralelización con OpenMP, siguiendo el esquema de la Figura 3. Los resultados nos demuestran que existe una escalabilidad casi lineal hasta 12 cores y cuando aumentamos más de un thread por core, el tiempo de ejecución ya no escala linealmente. El efecto del uso de multithreading se sigue investigando para mejorar la escalabilidad de la paralelización.

## 4. Publicaciones

Se envió un artículo al Workshop on "Biomedical and Bioinformatics Challenges for Computer Science" que forma parte de International Conference on

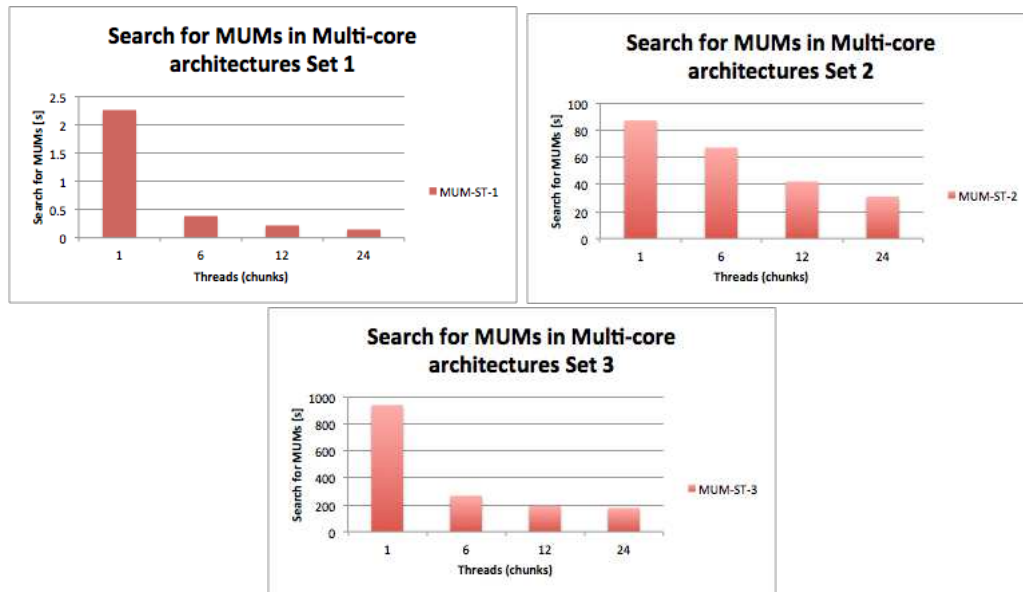


Figura 4: Búsqueda de MUMs en Árbol de sufijos para arquitecturas Multi-core.

Computational Science. El artículo fue rechazado.

## 5. Seguimiento

En el primer año se ha estudiado el árbol de sufijos y el algoritmo para la búsqueda de MUMs. En el segundo año se ha determinado una estrategia de paralelización de búsqueda de MUMs en arquitecturas Multi-core. Se ha evaluado el cambio del árbol de sufijos por el arreglo de sufijos mejorado y se ha implementado la estrategia de paralelización utilizada con el árbol de sufijos. Se está definiendo la estrategia de distribución del arreglo de sufijos mejorado en un cluster bajo el paradigma master/worker para la búsqueda de MUMs.

## 6. Planificación

En el tercer año se implementará y evaluará el algoritmo paralelo y distribuido de búsquedas de coincidencias exactas/únicas máximas. Se adaptará este algoritmo a la aplicación MUMmer para su ejecución en entornos de HPC. Se refinará la ejecución de MUMmer en entornos de HPC. Se tomará como base el programa original, MUMmer, y se realizarán las modificaciones pertinentes para mejorar la fase de agrupamientos de MUMs y MEMs. Estancia en otra unidad de investigación, EMBL Heidelberg Alemania. Adaptar la propuesta de búsqueda distribuida

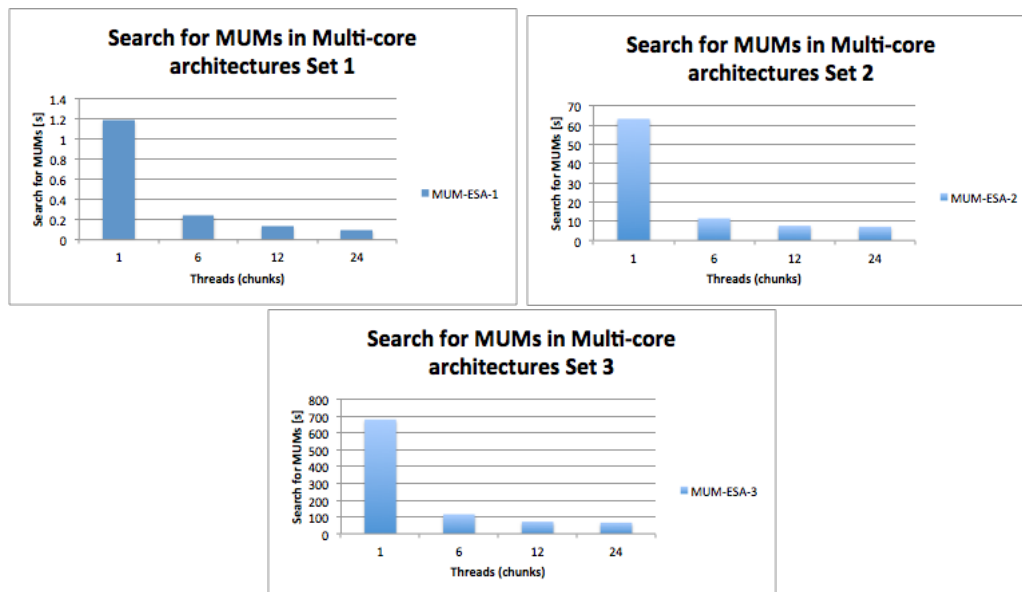


Figura 5: Búsqueda de MUMs en Arreglo de sufijos mejorado para arquitecturas Multi-core.

de coincidencias exactas desarrollado para su ejecución en las instalaciones de EMBL. Redactar la tesis doctoral y preparar su defensa.

## 7. Análisis de las publicaciones pendientes de publicar

Search for Maximal Unique Matches in Multi-core architectures en Workshop on Parallel Computational Biology, que forma parte de International Conference on Parallel Processing and Applied Mathematics. La fecha de aceptación es el 22 de junio de 2013.

## 8. Publicaciones previstas

IEEE Computational Systems Bioinformatics Conference, CORE A, 30 de junio de 2013.

Symposium on Principles and Practice of Parallel Programming, CORE A, 2014.

IEEE Bioinformatics and Bioengineering, CORE C, 1 de julio de 2013.

BioMed 2014, Microsoft academic search #1, 2014.

International Conference on Intelligent Systems for Molecular Biology, Microsoft academic search #2, Septiembre 2013.

Revista: Bioinformatics, Impact factor: 5.468

Revista: BMC Bioinformatics, Impact factor: 2.75

## 9. Conclusiones

Durante este segundo año de investigación doctoral se ha identificado el cuello de botella del árbol de sufijos: la falta de localidad espacial.

Se ha estudiado y evaluado otra estructura de datos para la búsqueda de MUMs.

Se ha paralizado la búsqueda de MUMs en arquitectural Multi-core. Aún falta resolver los problemas de escalabilidad de la paralelización.

En el siguiente año se busca determinar si la propuesta de búsqueda de coincidencias exactas/únicas máximas en el arreglo de sufijos mejorado distribuido permite reducir el tiempo de ejecución bajo entornos de HPC multi-core. Adicionalmente se tendrán que resolver los siguientes problemas

- Política de carga de trabajo para cada thread/core: garantizar que todas las búsquedas se resuelvan de manera local.
- Optimizar las fases de agrupamiento y alineamiento local de la metodología de alineamiento de genomas.

## Referencias

- [1] ABOUELHODA, M. I., KURTZ, S., AND OHLEBUSCH, E. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms* 2, 1 (Mar. 2004), 53–86.
- [2] ENCARNAC, G., AND ROMA, N. Advantages and GPU Implementation of High-Performance Indexed DNA Search based on Suffix Arrays. *Architecture* (2011), 49–55.
- [3] GUSFIELD, D. *Algorithms on strings, trees, and sequences : computer science and computational biology*. Cambridge Univ. Press, Jan. 2007.
- [4] KHAN, Z., BLOOM, J. S., KRUGLYAK, L., AND SINGH, M. A practical algorithm for finding maximal exact matches in large sequence datasets using sparse suffix arrays. *Bioinformatics* 25, 13 (July 2009), 1609–1616.

- [5] KOUZINOPOULOS, C. S., MICHAILIDIS, P. D., AND MARGARITIS, K. G. Parallel processing of multiple pattern matching algorithms for biological sequences : Methods and performance results. *Systems and Computational Biology - Bioinformatics and Computational Modeling* (2005).
- [6] KURTZ, S., PHILLIPPY, A., DELCHER, A. L., SMOOT, M., SHUMWAY, M., ANTONESCU, C., AND SALZBERG, S. L. Versatile and open software for comparing large genomes. *Genome Biology* 5, R12 (January 2004), R12.1–R12.9.
- [7] MONGELLI, H. Efficient Two-Dimensional Parallel Pattern Matching with Scaling.
- [8] NEEDLEMAN, S. B., AND WUNSCH, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology* 48, 3 (Mar. 1970), 443–453.
- [9] OHLEBUSCH, E., GOG, S., AND KÜGEL, A. Computing matching statistics and maximal exact matches on compressed full-text indexes. In *SPIRE* (2010), pp. 347–358.
- [10] OĞUZHAN KÜLEKCI, M., HON, W.-K., SHAH, R., SCOTT VITTER, J., AND XU, B.  $\Psi$ -RA: a parallel sparse index for genomic read alignment. *BMC genomics* 12 Suppl 2 (Jan. 2011), S7.
- [11] SMITH, T. F., AND WATERMAN, M. S. Identification of common molecular subsequences. *Journal of Molecular Biology* 147 (1981), 195–197.
- [12] VYVERMAN, M., DE BAETS, B., FACK, V., AND DAWYNDT, P. *essa-MEM: finding maximal exact matches using enhanced sparse suffix arrays. Bioinformatics (Oxford, England)* (Feb. 2013), 2–4.