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

17 de septiembre de 2012

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.

Rgimen de dedicación: completa.

Año de inscripción: 2011.

Año previsto para la presentación de Tesis Doctoral: 2014

Firmado

Director	Director	Estudiante
Antonio Espinos	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 positición 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. Un alineamiento involucra tres operaciones básicas:

- Coincidencia: los nucleótidos en ambos genomas son iguales.
- Discrepancia: los nucleótidos no coinciden en ambos genomas.
- Mutaciones: uno o varios cambios de nucleótidos en el genoma:
  - Sustitución: un nucleótido es reemplazado por otro.
  - Inserción: uno o varios nucleótidos se insertan en una posición en el genoma.
  - Eliminación: uno o varios nucleótidos se borran del genoma.

Existe diferentes algoritmos propuestos para el alineamiento de dos secuencias como [Needleman and Wunsch, 1970] y [Smith and Waterman, 1981]. Estos algoritmos funcionan bien con tamaños de secuencias pequeños como un gen, 20-30kbp[1]. 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. Una de esas alternativas es la utilización de la heurística basada en las coincidencias exactas máximas en las secuencias a alinear. Al encontrar las coincidencias exactas máximas se puede realizar el alineamiento de genomas enteros reduciendo el espacio del alineamiento a las regiones entre

[1]bp es la únidad básica de medida en las secuencias de ADN y representa la longitud de un nucleótido

las coincidencias exactas máximas.

Formalmente el problema de la búsqueda de coincidencias exactas máximas se define como:

Definición 1 *Dadas dos secuencias (genomas)* $R = r_1 r_2 \dots r_n$ *y* $Q = q_1 q_2 \dots q_m$*, y una longitud mínima* l*, se requiere encontrar todas las ocurrencias de las coincidencias exactas máximas de longitud mínima* l*, entre* R *y* Q*.*

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

El árbol de sufijos es una estructura de datos que permite la búsqueda de co-

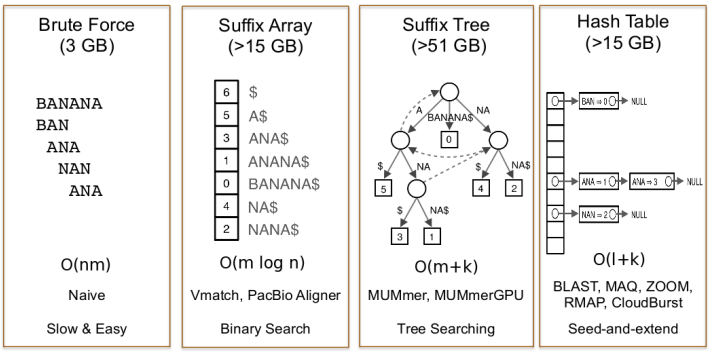

Figura 1: Diferentes estructuras de indexación para el problema de la búsqueda de coincidencias exactas máximas para el genoma humano como referencia.

incidencias exactas de longitud variable en tiempo lineal respecto a la longitud de la secuencia de consulta. Una coincidencia exacta de longitud w ocurre en $r_i r_{i+1} \dots r_{i+l-1}$ y $q_j q_{j+1} \dots q_{j+l-1}$ si y solo si los sufijos $r_i r_{i+1} \dots r_n$ y $q_j q_{j+1} \dots q_m$ comparten un prefijo común de longitud mínima l. La búsqueda de coincidencias máximas exactas en un árbol de sufijos se realiza recorriendo el trayecto que es común a la secuencia de consulta hasta encontrar una discrepancia. Si la búsqueda termina en un nodo hoja, la coincidencia es única en la secuencia de referencia. Verificando el carácter inmediato anterior al inicio de esta coincidencia se puede determinar si es una coincidencia máxima.

Este problema se presenta en aplicaciones de alineamiento de genomas donde el tamaño de los genomas son de millones de caracteres y en consecuencia la búsqueda de MEMs es una tarea computacionalmente intensiva. Además, la razón de utilizar un árbol de sufijos radica en que la búsqueda de coincidencias exactas de longitud variable en un árbol de sufijos se realiza en tiempo lineal, gracias al uso de los enlaces de sufijos.

En consecuencia se busca **acelerar la búsqueda eficiente de coincidencias exactas 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 MEMs: al agilizar la búsqueda se reduce el tiempo utilizado por cada MEM encontrado debido a que los MEMs son dependientes de los datos de entrada.
- Almacenar genomas de mayor tamaño con la memoria disponible: al utilizar un árbol de sufijos como estructura de indexación el tamaño del árbol se convierte en un problema cuando este no cabe en la memoria disponible. Se hace necesario diseñar estructuras de datos que permitan la indexación de genomas mayores.
- Reducir los problemas de localidad en el árbol de sufijos: al navegar en un árbol de sufijos puede ocurrir que ocurran fallos de cache.

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 maxima (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.*

Identificar las cadenas, k, mas grandes en el genoma Q que tienen una coincidencia de longitud l idéntica en el genoma R, tiene la siguiente complejidad computacional:

- Método simple: $O(nl)$
- Usando árbol de sufijos: $O(l + k)$

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. 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 [Gusfield, 2007]. Estas dos propiedades hacen al árbol de sufijos una estructura útil para un rango diverso de aplicaciones bioinformáticas, incluyendo: alineamiento de genomas [Kurtz et al., 2004].

La búsqueda de coincidencias exactas máximas en un árbol de sufijos se re-

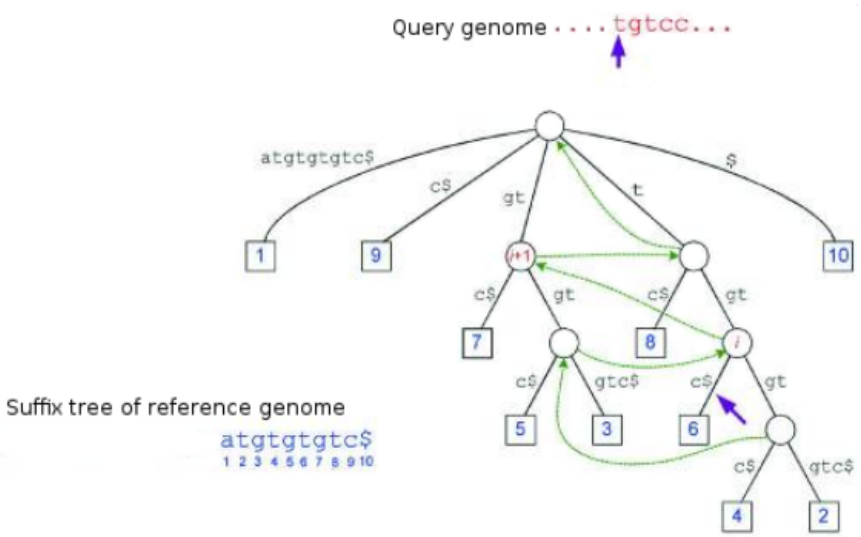

Figura 2: Árbol de sufijos para la palabra atgtgtgtc$.

aliza recorriendo el árbol con la cadena que se requiere encontrar la coincidencia exacta máxima. Si la coincidencia termina en un nodo hoja, la coincidencia exacta 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 construidos 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.

El primer algoritmo para la construcción de un árbol de sufijos en tiempo lineal y eficiente en espacio fue propuesto en [McCreight, 1976] , y Ukkonen produjo una variante “on-line” de ese algoritmo en [Ukkonen, 1992]. 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 permiten 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 $O(n)$.

2.3. Definición del problema a resolver

El problema de la búsqueda de las coincidencias exactas máximas de una longitud mínima ha sido identificado en varias aplicaciones, entre ellas MUMmer [Kurtz et al., 2004]. 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.

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

Estructura	L [bp^2]	Cantidad de búsquedas	Búsqueda [s]	Uso de memoria [MB]
Árbol de sufijos	20	$9{,}87\times10^{18}$	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 exactas máximas de forma ágil en la cadena entera. Los árboles de sufijos permiten hacer búsquedas con complejidad $O(m + k)$, donde m es el tamaño de la cadena a buscar y k es el número de ocurrencias.
En resumen la búsqueda de coincidencias exactas máximas en una cadena, como el genoma humano, es un problema computacionalmente intensivo que es posible resolverse con estructuras de datos como el árbol de sufijos. 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 exacta de coincidencias máximas.

3. Estado de la investigación

3.1. Paralelización a nivel de datos: alineamiento de genomas

La versión secuencial del algoritmo de MUMmer[3], requiere mayores recursos de cómputo conforme los genomas a alinear son de longitudes mayores. Un primer problema es cuando la referencia a indexar mediante el árbol de sufijos resulta en un árbol de mayor tamaño a la memoria disponible. El segundo problema es el tiempo empleado para procesar un genoma de consulta.

Es por ello que se realizó una evaluación de la viabilidad de utilizar la técnica de paralelización de datos en el alineamiento de genomas. Específicamente se evalúa que técnica de paralelización es mejor: dividir el genoma de referencia o dividir el genoma de consulta. En la figura 3 se muestra la propuesta evaluada de paralelización a nivel de datos. La paralelización involucra que se realice una división del genoma con un tamaño fijo y una área de solapamiento. El tamaño del bloque es determinado por el número de instancias de ejecución que se tendrán y el solapamiento es utilizado para evitar perder alguna coincidencia debido a la división del genoma. Una vez hecha la división se procede a ejecutar la búsqueda de coincidencias exactas máximas. Al realizar la búsqueda de MEMs se tiene un conjunto de listas parciales de MEMs que se mezclan para obtener la lista global de MEMs que debe ser la misma a una ejecución serie.

En las figuras 4 y 5 se muestran los resultados de la paralelización del alineamiento de genomas reseñado en la figure 3. El problema que surge al realizar este tipo de paralelización es en la fase final de obtener la lista global de MEms. Esto es, se requiere contar con un algoritmo que permita descartar aquellos MEMs que están contenidos por otro MEM de mayor tamaño y que no son posible eliminar en la fase previa debido a que se requiere evaluar el resto de los MEMs.

Una primera solución es evaluar para cada MEM si está cubierto por un MEM mayor en toda la lista. Esta solución simple ayuda debido a que tiene una complejidad $O(g^2)$ donde g es la cantidad total de MEMs encontrados, es por ello que se necesita un algoritmo que permita efizcamente descartar aquellos MEMs que están repetidos y/o contenidos por otro MEM.

La propuesta para un nuevo algoritmo de mezcla final de MEMs toma en cuenta el principio de que basta con evaluar solo aquella región que contiene al MEM que estamos evaluando. De esa manera es posible disminuir la complejidad del algoritmo y el tiempo de cómputo total de la paralelización no sería penalizado por la última fase. El algoritmos es descrito a continuación:

Conjunto de MEMs parciales: A.

[3]Aplicación para el alineamiento de genomas con más de 700 referencias en artículos de investigación.

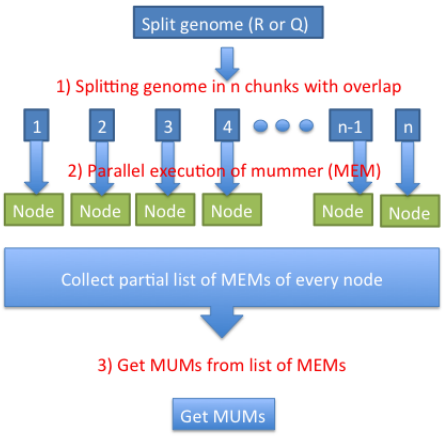

Figura 3: Paralelización del alineamiento de genomas.

Quicksort de MEMs parciales por la posición en la consulta.
for $i = 0$ a N **do**
  $A[i].Good = TRUE$
end for
for $i = 0$ a $i < N - 1$ **do**
  if $!A[i].Good$ **then**
    Continuar.
  end if
  $i\_diag = A[i].Q - A[i].R$
  $i\_fin = A[i].Q + A[i].Len$
  for $j = i + 1; j < N \&\& A[j].Q \leq i\_fin; j++$ **do**
    if $!A[j].Good$ **then**
      Continuar.
    end if
    $j\_diag = A[j].Q - A[j].R$
    if $i\_diag == j\_diag$ **then**
      $j\_extent = A[j].Len + A[j].Q - A[i].Q$
      if $j\_extent > A[i].Len$ **then**
        $A[i].Len = j\_extent$
        $i\_end = A[i].Q + j\_extent$
      end if
      $A[j].Good = FALSE$

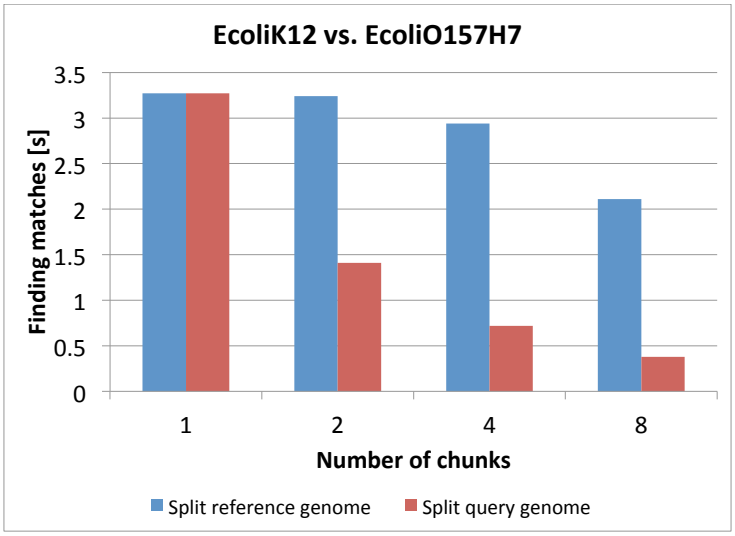

Figura 4: Paralelización del alineamiento del genoma Ecoli con longitud de 5,4Mbp.

else if $A[i].R == A[j].R$ **then**
  $olap = A[i].Q + A[i].Len - A[j].Q$
  if $A[i].Len < A[j].Len$ **then**
    if $olap \geq A[i].Len/2$ **then**
      $A[i].Good = FALSE$
      Romper.
    end if
  else if $A[j].Len < A[i].Len$ **then**
    if $olap \geq A[j].Len/2$ **then**
      $A[j].Good = FALSE$
    end if
  else
    if $olap \geq A[i].Len/2$ **then**
      $A[j].Tentative = TRUE$
      if $A[i].Tentative$ **then**
        $A[i].Good = FALSE$
        Romper.
      end if
    end if

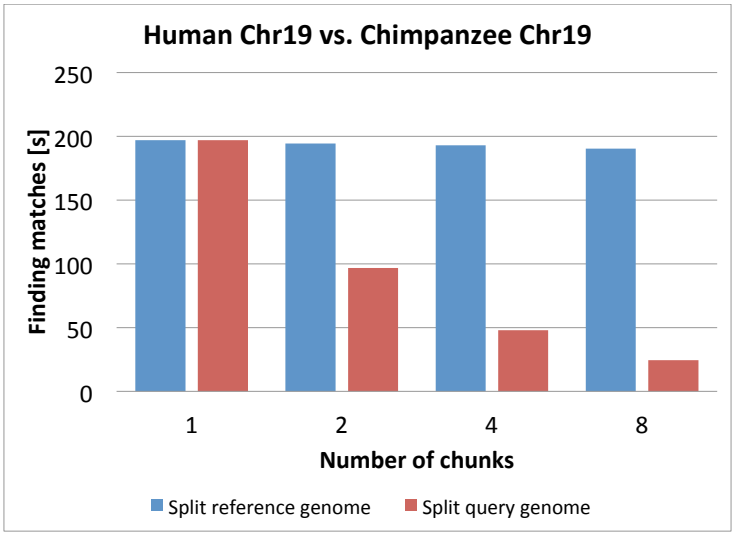

Figura 5: Paralelización del alineamiento del cromosoma 19 del homo sapiens y el chimpancé con longitud de 59Mbp.

end if
else if $A[i].Q == A[j].Q$ **then**
$olap = A[i].R + A[i].Len - A[j].R$
if $A[i].Len < A[j].Len$ **then**
if $olap \geq A[i].Len/2$ **then**
$A[i].Good = FALSE$
Romper.
end if
else if $A[j].Len < A[i].Len$ **then**
if $olap \geq A[j].Len/2$ **then**
$A[j].Good = FALSE$
end if
else
if $olap \geq A[i].Len/2$ **then**
$A[j].Tentative = TRUE$
if $A[i].Tentative$ **then**
$A[i].Good = FALSE$
Romper.
end if

```
            end if
          end if
        end if
      end for
    end for
    for i = j = 0; i < N; i++ do
      if A[i].Good then
        if i ≠ j then
          A[j] = A[i]
        end if
        j++
      end if
    end for
    N = j
    for i = 0; I < N; i++ do
      A[i].Good = FALSE
    end for
```

La complejidad del algoritmo descrito en el peor de los casos es $O(g)$ donde g es el tamaño de la lista total de MEMs parciales, en el mejor de los casos es $O(g)$. Para el genoma Ecoli de longitud 5,4Mbp en donde se buscaron MEMs de longitud mínima de 20, la figura 6 muestra el tiempo de cómputo utilizado en la fase de búsqueda de MEMs, en las figuras 7 y 8 se muestra el tiempo utilizado en la fase de obtención de la lista de MEMs global.

Un segundo experimento se llevo a cabo evaluando la viabilidad de la propuesta descrita utilizando el cromosoma 19 del homo sapiens y del chimpancé. La búsqueda de MEMs de longitud mínima de 2000 se muestra en la figura 9 y en la figura 10 se muestra el tiempo de cómputo para la fase final.

Un detalle importante de resaltar de la mejora realizada al algoritmo simple es la reducción de tiempo obtenido ya que el algoritmo simple tuvo un tiempo de ejecución de más de 7 horas. La mejora en el algoritmo de mezcla de las listas parciales de MEMs en una lista global de MEMs es debido a que el algoritmo simple era muy lento debido a su complejidad computacional $O(g^2)$, donde g es la lista total de MEMs.

El primer algoritmo simple realizaba la comparación de cada MEM contra toda la lista de MEMs para determinar si estaba contenido o no por otro MEM. La segunda versión propuesta permite realizar la mezcla de MEMs parciales considerando solo la región en la que se podrían encontrar MEMs cubiertos por un MEM mayor. Está mejora ha permitido reducir el tiempo de cómputo y garantizar que la última fase de esta primera aproximación de paralelización del alineamiento de genomas no se vea afectado por la última fase incluida en la figura 3.

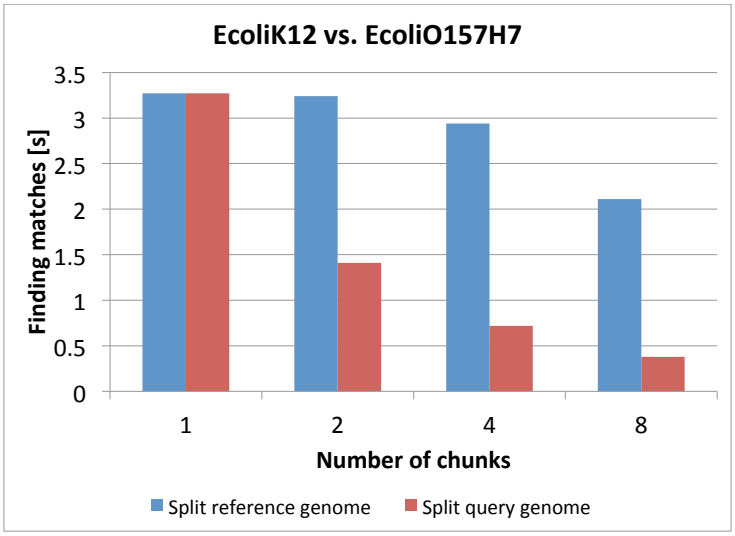

Figura 6: Búsqueda de MEMs para el genoma Ecoli de longitud mínima 20.

3.2. Búsqueda distribuida de coincidencias exactas: árbol de sufijos distribuido

El problema de mejorar el tiempo de cómputo de las búsquedas de coincidencias exactas nos da una oportunidad de utilizar el HPC para reducir el tiempo de procesamiento. Se propone modificar la estructura de datos utilizada en MUMmer (árbol de sufijos) para mejorar las búsquedas de coincidencias exactas entre un genoma de referencia y uno de consulta dada una longitud mínima de coincidencia, considerando un uso eficiente de los recursos. Para ello se tomarán en cuenta los siguientes parámetros:

- Optimizar el número de operaciones en la búsqueda de MEMs.
- Tamaño del genoma de referencia y consulta.
- Memoria principal disponible en cada nodo de cómputo.

Basado en dichos parámetros se distribuirá el árbol de sufijos en subárboles, ver Figura 11. Dicha aproximación ha sido llevada a cabo de distintas maneras por [Mansour, 2012], [Japp, 2004], [Ghoting and Makarychev, 2010] y [Sadakane,]. Los subárboles aprovecharán una característica de la búsqueda de coincidencias: la frecuencia de prefijos en los sufijos del genoma de referencia.

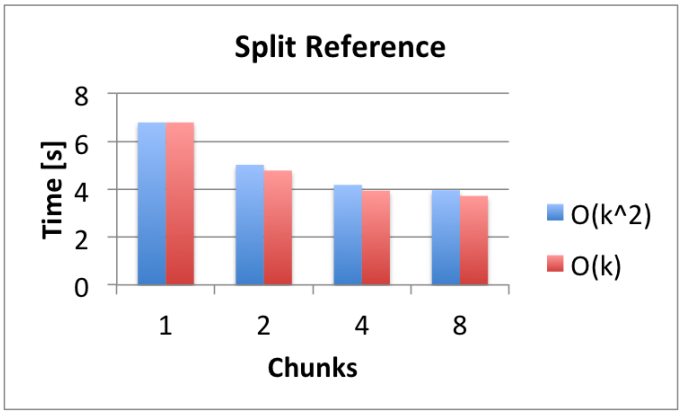

Figura 7: Evaluación de algoritmo de obtención de lista de MEMs global para el genoma Ecoli.

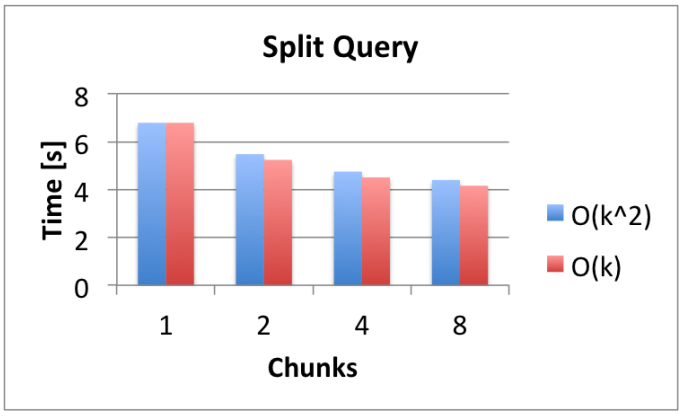

Figura 8: Evaluación de algoritmo de obtención de lista de MEMs global para el genoma Ecoli.

Para definir formalmente el árbol de sufijos distribuido es necesario mostrar algunos conceptos, extraídos de [Clifford, 2005] y [Ghoting and Makarychev, 2010].

Definición 4 *Si $t = uvw$ entonces u es un prefijo y w es un sufijo de t.*

Un sufijo de t se repite si ocurre al menos una vez como una subcadena no sufijo de t. Además requerimos especificar que sufijos de t se incluyen en un subárbol de sufijos. Esto se hace considerando un prefijo de longitud variable z, y el conjunto de prefijos de longitud variable representan los subárboles que se almacenan en cada uno de los nodos de cómputo del clúster multicore. El conjunto de prefijos de longitud variable es una idea ya propuesta en [Ghoting and Makarychev, 2010] que consiste en encontrar la frecuencia de prefijos que cumplen la ecuación 3.2,

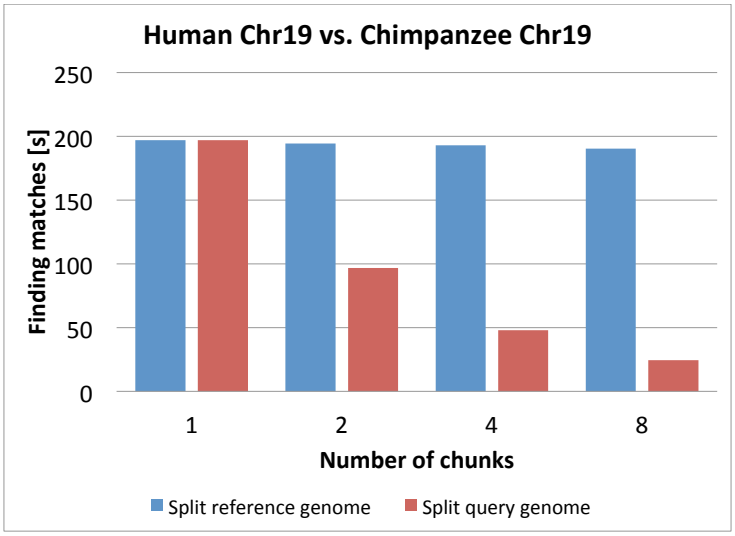

Figura 9: Búsqueda de MEMs para el cromosoma 19 del homo sapiens y el chimpancé de longitud mínima 2000.

en la que $f(p)$ es la frecuencia del prefijo p, MTS es el tamaño máximo del árbol de sufijos y NS es el tamaño del nodo de un árbol de sufijos en bytes. El factor de 2 es debido a que podemos tener a lo sumo $f(p)$ nodos internos y $f(p)$ nodos hojas.

$$f(p) \leq \frac{MTS}{2 * NS} \tag{1}$$

Ecuación para el cálculo de prefijos de longitud variable.

Para generar el árbol de sufijos distribuido DST:

1. Construir el árbol de sufijos entero en tiempo lineal.
2. Generar el conjunto de prefijos de longitud variable que serán la raíz de cada SST.
3. Para cada prefijo de longitud variable

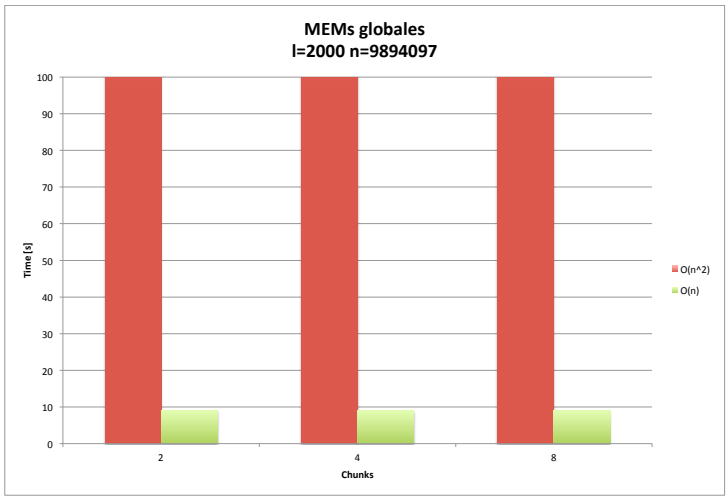

Figura 10: Evaluación de algoritmo de obtención de lista de MEMs global para el cromosoma 19 del homo sapiens y el chimpancé.

a) Extraer el subárbol debajo de ese prefijo, ese subárbol se convierte en el SST.

b) Generar la tabla de direccionamiento de los enlaces de sufijos que apuntan hacia nodos que están fuera del SST.

4. Refinar la tabla de direccionamiento de los enlaces de sufijos distribuidos con el identificador del SST al que corresponde el nodo hacia el cual apunta.

El algoritmo resultante usa el espacio en cada nodo que es proporcional al tamaño del SST construido. La generación de los SSTs obliga a contar con un esquema de enlaces de sufijos que apuntan hacia nodos que no están dentro del SST recien creado. Este esquema de enlaces de sufijos distribuido será almacenado en cada uno de los SSTs generados.

3.3. Importancia del enlace de sufijo

Los enlaces de sufijos juegan un rol crítico en la construcción lineal de árboles de sufijos y en su recorrido.

Definición 5 *Si hay un nodo v en el árbol de sufijos con la etiqueta* $c\alpha$*, donde c es un carácter y* $alpha$ *es una cadena (no vacía), entonces el enlace de sufijo de*

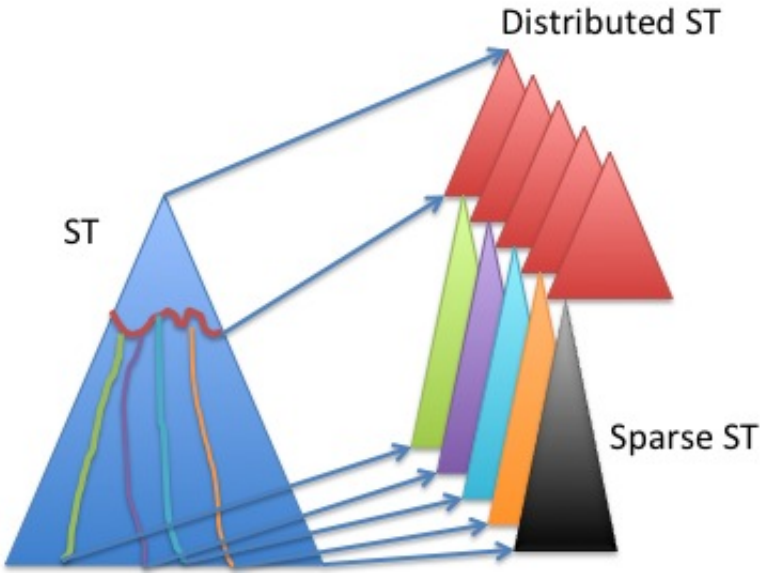

Figura 11: Árbol de sufijos distribuido.

v apunta a s(v), que es un nodo con la etiqueta α. Si α está vacío, entonces el enlace de sufijo de v, por ejemplo s(v) es la raíz.

Los enlaces de sufijo existen para cada nodo interno (no hoja) de un árbol de sufijo. Siguiendo un enlace de sufijo, se puede saltar de un sufijo a otro, cada sufijo iniciando exactamente un carácter siguiente del primer carácter de su sufijo precedente. De tal manera que usando **enlaces de sufijos** es posible obtener un **algoritmo lineal para la búsqueda de coincidencias exactas máximas**. Esto es porque los enlaces de sufijos hacen un seguimiento de que o cuanto de la cadena ha coincidido hasta ahora. Cuando encontramos una discrepancia, podemos saltar a lo largo del enlace de sufijo para ver si hay cualquier coincidencia después en la referencia y en la consulta.

La creación de un DST genera un nuevo problema a resolver, los enlaces de sufijos. Esto es, al construir árboles dispersos SST puede ocurrir que existan enlaces de sufijos que apunten a nodos de otro árbol SST. Para ello es necesario contar con algun esquema que permite seguir el nuevo enlace de sufijo distribuido. Si el número de enlaces de sufijos distribuido de un SST es muy grande pueden ocurrir problemas de sobrecarga de procesamiento en algunos SSTs. Adicionalmente puede trasladar nuestro problema de procesamiento a uno de comunicaciones en el que se tendría que lidiar con el uso de los enlaces de sufijos distribuido.

La solución es contar con un esquema que permita la utilización de enlaces de sufijos distribuido:

1. Tabla de búsqueda directa del nodo al que apunta el enlace de sufijo distribuido, compartido por todos los SSTs.

2. Sistema de cola para el procesamiento de las peticiones de búsqueda en el SST hacia el cual apunta el enlace de sufijo distribuido.

Mediante la organización del nuevo árbol de sufijos distribuido de esta manera, podemos realizar búsquedas de coincidencias exactas máximas de forma distribuida. Al utilizar una búsqueda distribuida es posible mejorar el tiempo de cómputo de las búsquedas exactas. Se requiere que el tiempo de cómputo sea menor que el mostrado por propuestas existentes en el estado del arte.

3.4. Evaluación experimental

Los experimentos realizados se ejecutaron en el siguiente ambiente de pruebas:

1. GCC 4.6.3

2. MUMmer 3.23

3. SparseHash 2.0.2-1

4. Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz

5. RAM 6GB

Los genomas utilizados en las pruebas fueron:

- Genoma humano (2,96Gbp).
- Cromosoma 2 humano (238,6Mbp).
- Cromosoma 1 chimpancé (232,7Mbp).
- EcoliK12 (4,6Mbp).
- EcoliO157H7 (5,3Mbp).

El primer elemento a evaluar fue la utilización de los enlaces de sufijos al buscar MEMs de diferentes longitudes. Este experimento se realiza para el cromosoma 2 del ser humano como referencia y el cromosoma 1 del chimpancé como consulta. La búsqueda a realizar es de MEM. Se evalúan dos mecanismos de búsqueda de MEMs:

- Cada sufijo inicia la búsqueda desde la raíz.
- Cada sufijo utiliza el enlace de sufijo.

En la figura 12 se muestra que para diferentes longitudes de MEMs y se comprueba que la ganancia en tiempo de cómputo es siempre mayor al utilizar los enlaces de sufijos que si no se utilizarán. Esta mejora se explica porque al utilizar un enlace de sufijo se evita una comparación cada que se utiliza el enlace de sufijo.
El tiempo de procesamiento también se ve afectado, evidentemente por la canti-

Figura 12: Comparación de búsqueda de MEMs con y sin utilización de enlaces de sufijos.

dad de MEMs encontrados, como se muestra en la figura 13.

Una vez que se ha comprobado la efectividad de los enlaces de sufijos se

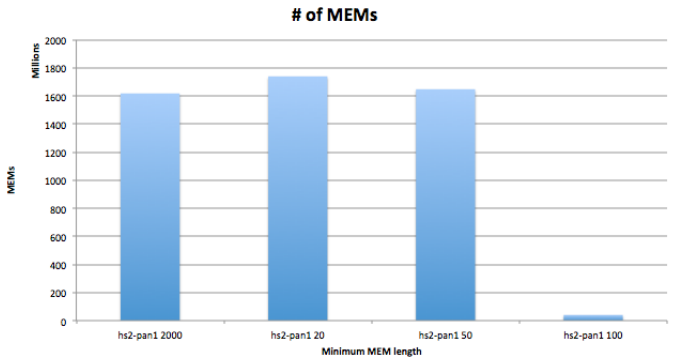

Figura 13: Conjunto de MEMs encontrados para distintos tamaños mínimos de longitud.

procede a evaluar como son accesados estos enlaces de sufijos y en que zonas del árbol se encuentran. Esto se hace con el fin de evaluar como se verá afectado la búsqueda de MEMs al hacer la división vertical del árbol de sufijos en nuestra propuesta.

Para ello se realizó la búsqueda de MEMs con el genoma de referencia EcoliK12 y con el genoma de consulta EcoliO157H7 con una longitud mínima de 20 de MEM. En la figura 14 se muestra que la longitud mínima de MEM restringe la profundidad del árbol en el cual los enlaces de sufijos son utilizados.

Una posible mejora de optimización consistiría en mejorar la localidad de los

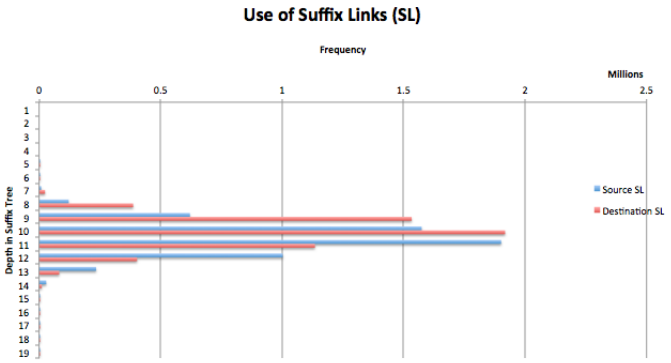

Figura 14: Frecuencia de uso de los enlaces de sufijos al buscar MEMs de longitud mínima de 20bp.

nodos que se encuentran en los niveles de profundidad de 8 a 12 que son en donde se utilizan los enlaces de sufijos con mayor frecuencia. Esta posible heurística estaría basada en la longitud mínima de MEM a buscar y que depende del usuario.

Una vez que se ha evaluado la importancia del enlace de sufijo en un árbol de sufijos para la búsqueda de coincidencias exactas máximas se procede a evaluar experimentalmente la propuesta de distribución del árbol de sufijos. Se debe de mantener el enlace de sufijo al hacer la distribución del árbol debido a que esto nos garantiza la complejidad lineal de la búsqueda de MEMs.

El primer paso es determinar el conjunto de prefijos de longitud variable necesario para distribuir el árbol de sufijos. Es por ello que se realizó un experimento para ubicar la longitud necesaria para la elección de los prefijos.

Al utilizar prefijos de longitud variable es posible tener árboles de sufijos dispersos balanceados debido a que se podrían agrupar aquellos prefijos con mayor frecuencia de ocurrencia. Es por ello que debemos encontrar un conjunto valido de prefijos para dividir el árbol de sufijos en árboles de sufijos dispersos con el requerimiento de que puedan construirse en memoria principal.

Definición 6 *Sea $f(p)$ el número de veces que un prefijo ocurre en el genoma de referencia R. Sea MTS (tamaño máximo del árbol de sufijos) la cantidad máxima de espacio de memoria en bytes que puede ser reservado al árbol de sufijos durante la construcción del árbol. Sea NS el tamaño de un nodo del árbol de sufijos en bytes. El objetivo es encontrar un conjunto de prefijos de longitud variable en R, tal que $\forall p \epsilon R, f(p) < \frac{MSSTS}{2*NS}$.*

Una vez definido el conjunto de prefijos se procede a la construcción del árbol de sufijos disperso. Para ello se ubican en cada árbol de sufijos disperso el prefijo de longitud variable que será la raíz de cada árbol de sufijos disperso. Una vez definido el conjunto de prefijos se procede a la división del árbol de sufijos para obtener el árbol de sufijos disperso. Finalmente se genera la tabla de enlaces de sufijos distribuido correspondiente a cada árbol de sufijos disperso.

Para demostrar como la creación de los prefijos de longitud variable permiten generar árboles de sufijos dispersos, se muestra la frecuencia de prefijos para el genoma humano en las figuras 15 y 16.

Si definimos $F_m = \frac{MTS}{2*NS}$ entonces de las figuras 15 y 16 se obtendrían las

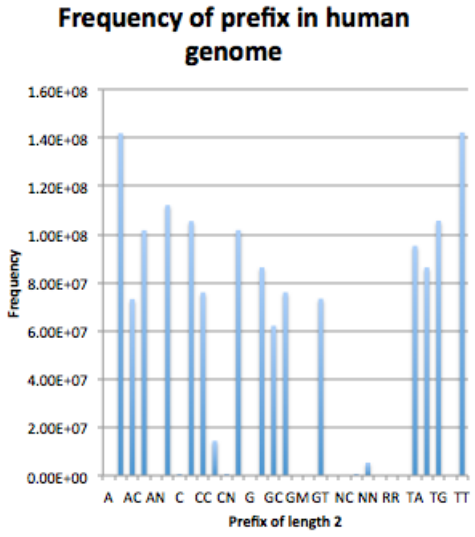

Figura 15: Frecuencia prefijos de longitud variable en genoma humano.

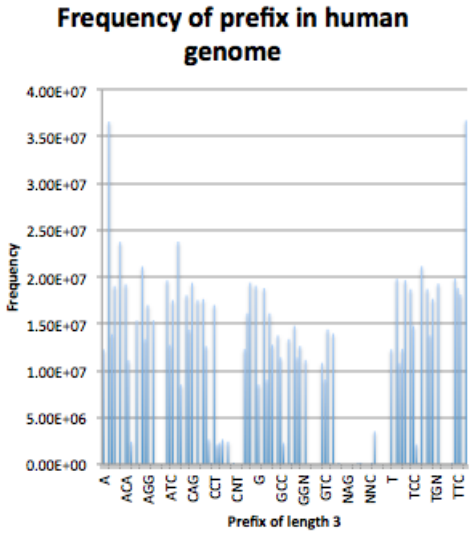

Figura 16: Frecuencia prefijos de longitud variable en genoma humano.

raíces de los árboles de sufijos dispersos que son menores que el valor F_m. Si F_m es igual a 1.2×10^8 entonces de la figura 15 se muestra que se debe de extender los prefijos AA y TT con una longitud mayor hasta que se cumpla que sea menor que F_m.

Para un genoma real se evalúa la creación del árbol de sufijos distribuido. Con

el genoma EcoliK12 se procede a generar el árbol de sufijos distribuido. Se toma en consideración que $F_m = 150K$, se recorre el genoma de referencia para encontrar el conjunto de prefijos de longitud variable que cumplen la ecuación 3.2. El resultado de este proceso se muestra en las figuras 17 y 18.

El siguiente paso es determinar los enlaces de sufijos distribuidos para cada uno

Length=2	Freq.
AC	128412
AG	118868
CC	135729
CT	117983
GA	133800
GG	134744
GT	127767
TA	105926
TC	133759

Figura 17: Frecuencia prefijos de longitud variable en genoma EcoliK12.

Length=3	Freq.
AA[ACGT]	90384
AT[ACGT]	80758
CA[ACGT]	106860
CG[ACGT]	114085
GC[ACGT]	125862
TG[ACGT]	105767
TT[ACGT]	111341

Figura 18: Frecuencia prefijos de longitud variable en genoma EcoliK12.

de los árboles de sufijos dispersos creados a partir del conjunto de prefijos de longitud variable. En las figuras 19 y 20 se muestra que el porcentaje de enlaces de sufijos distribuido es inferior 1 %.

El resultado del árbol de sufijos distribuido es garantizar la búsqueda de coinci-

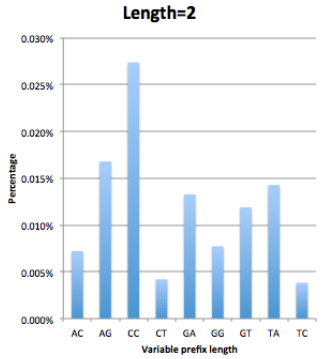

Figura 19: Enlaces de sufijo distribuidos para genoma EcoliK12.

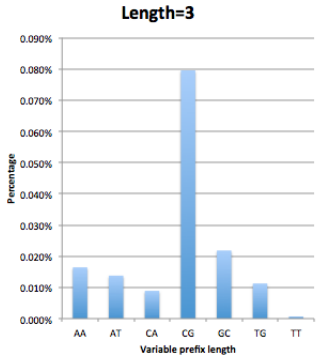

Figura 20: Enlaces de sufijo distribuidos para genoma EcoliK12.

dencias exactas máximas en el árbol de sufijos disperso y minizar el traslado de peticiones de búsqueda en árboles externos, las figuras 19 y 20 demuestran que el impacto de distribuir el árbol de sufijos será mínimo.

En la figura 21 se muestra el árbol de sufijos distribuido con cada uno de los

árboles de sufijos disperso. Para una cadena aacacccacacaccacaaa$ con sus respectivos nodos raíz etiquetados como $r_{aa}, r_{ac}, r_{ca}, r_{cc}, r_{a\$}$ and $r_{\$}$. Una característica

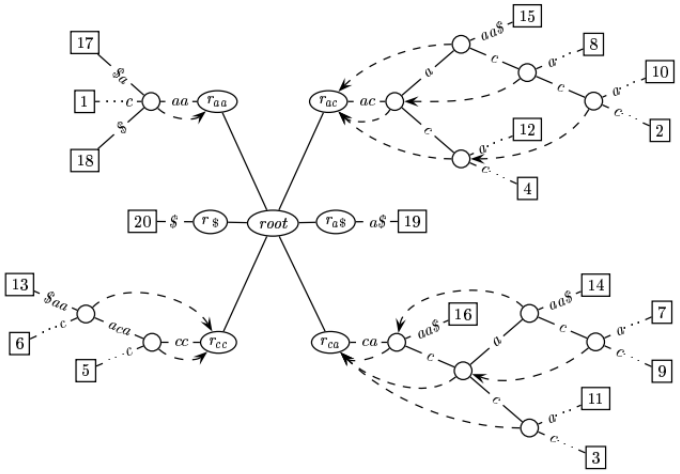

Figura 21: Árbol de sufijos distribuido para la cadena: aacacccacacaccacaaa$.

que el árbol de sufijos distribuido tiene es que únicamente mantiene los enlaces de sufijos que pertenecen al árbol de sufijos disperso. Aquellos enlaces de sufijos que apunten a otros nodos que no están en el árbol de sufijos disperso son agregados a la tabla de enlaces de sufijos distribuidos.
Como se mencionó anteriormente el enlace de sufijo no solo permite la construcción en tiempo lineal del árbol de sufijos sino que hace que las búsquedas de coincidencias sean también lineales. En consecuencia surge ahora un nuevo problema al proponer esta estructura de datos modificada:

¿Es posible mantener la complejidad de búsqueda de coincidencias en un árbol de sufijos $O(m + k)$, en donde m es la longitud de la secuencia de consulta y k es el número de ocurrencias al utilizar un árbol de sufijos distribuido?

3.5. Búsqueda de coincidencias exactas en un DST

El propósito de adaptar una estructura de datos como el árbol de sufijos es lograr reducir el tiempo de búsqueda de coincidencias exactas. Para ello se propone una primera aproximación a la búsqueda de coincidencias exactas máximas en un árbol de sufijos distribuido.
La idea general se basa en dividir la secuencia de consulta en segmentos y asignar

la búsqueda de cada sufijo dentro de cada segmento en el árbol de sufijos disperso correspondiente. La búsqueda de coincidencias exactas se realiza de manera paralela en el árbol de sufijos disperso y posteriormente determina las posiciones en las cuales se encuentran las coincidencias. Al evaluar el siguiente sufijo pueden ocurrir dos casos:

1. El siguiente sufijo se encuentra en el mismo árbol de sufijos disperso. La búsqueda continúa de manera local.

2. La búsqueda del siguiente sufijo se realiza en otro árbol de sufijos disperso.

El segundo caso es de especial relevancia ya que implica el proponer una estrategia de búsqueda del árbol de sufijos disperso en donde se debe realizar la búsqueda y adicionalmente continuar la búsqueda en ese árbol.
Para solucionar este caso, se propone el envío de la petición de búsqueda del sufijo en el árbol correspondiente utilizando la tabla de enlaces de sufijos distribuido. El árbol que recibe la petición la encola en la lista de sufijos que tiene que procesar y resuelve la búsqueda de coincidencias exactas máximas del sufijo que se le ha asignado.
En la figura 22 se muestra de manera gráfica la búsqueda de coincidencias exactas para una secuencia de consulta, la división en segmentos de la misma y la búsqueda del primer sufijo de cada segmento en el árbol de sufijos distribuido.

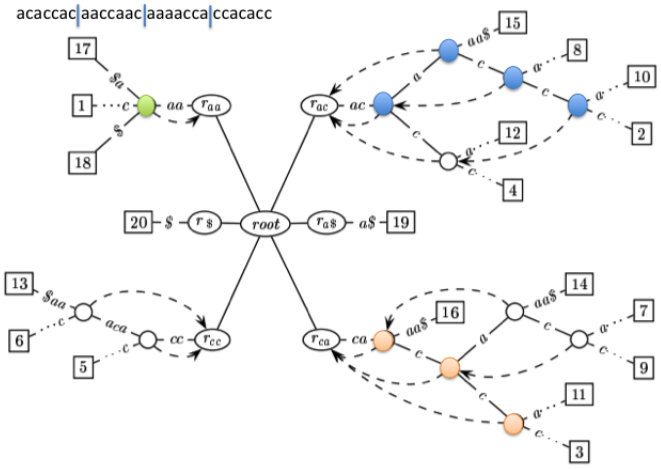

Figura 22: Búsqueda de coincidencias exactas máximas en un árbol de sufijos distribuido.

4. Publicaciones

Ninguna.

5. Seguimiento

En el primer año se ha cumplido con la adaptación de una estructura de datos que permita su utilización en el alineamiento de datos biológicos a gran escala. Dicha estructura será la base del desarrollo de MUMmer en HPC. La estructura es capaz de manejar grandes volúmenes de datos y se pueden realizar operaciones de consulta sobre ella.

6. Planificación

En el segundo año se realizará el diseño y evaluación experimental de un algoritmo paralelo y distribuido de búsquedas de coincidencias exactas máximas. Para ello se aprovechará la estructura de datos diseñada en el primer año. Se adaptará este algoritmo a la aplicación MUMmer para su ejecución en entornos de HPC. En el tercer año 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 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

Ninguna.

8. Publicaciones previstas

IEEE International Symposium on Parallel and Distributed Processing with Applications - ISPA - Enero 2013 : La estructura del árbol de sufijos distribuidos forma parte de los temas de investigación en este congreso.
International Conference on Parallel Processing - Euro-Par - Febrero 2013 : El algoritmo de búsqueda de coincidencias exactas máximas distribuida forma parte de los temas aboradados en este congreso.

European Symposium on Algorithms - ESA - Abril 2013 : El algoritmo de búsqueda distribuida de coincidencias exactas máximas distribuida forma parte de los temas abordados en este congreso.
International Symposium on Algorithms and Computation - ISAAC - Junio 2013 : El algoritmo de búsqueda distribuida de coincidencias exactas máximas distribuida forma parte de los temas abordados en este congreso.

9. Conclusiones

Durante este primer año de investigación doctoral se ha estudiado y evaluado la estructura de datos utilizada en la aplicación MUMMer. Se ha mejorado la fase de obtención de la lista final de MEMs reduciendo la complejidad del algoritmo utilizado previamente. Se han diseñado diferentes experimentos para determinar la mejor manera de dividir dicha estructura y aprovechar la nueva estructura en entornos de HPC. Se evalúo la utilidad de los enlaces de sufijos en un árbol de sufijos y como se pueden aprovechar para la búsqueda de coincidencias exactas máximas.

Se ha adaptado el árbol de sufijos para su ejecución en entornos de HPC y se ha propuesto una solución al problema de la restricción de memoria cuando la cadena de referencia crece. La estructura de datos adaptada se ha utilizado para proponer una primera aproximación a la solución de mejorar el tiempo de búsqueda de coincidencias exactas máximas.

En el siguiente año se busca determinar si la propuesta de búsqueda de coincidencias exactas máximas en el árbol de sufijos distribuido permite mejorar el tiempo de ejecución bajo entornos de HPC. Adicionalmente se tendrán que resolver los siguientes problemas

- Política de carga de trabajo para cada árbol de sufijos disperso: garantizar que todas las búsquedas se resuelvan de manera local.
- Búsqueda de coincidencias exactas máximas en el árbol de sufijos disperso de forma paralela, es decir, realizar múltiples búsquedas para distintos sufijos bajo el paradigma de las arquitecturas multicore.

Referencias

[Clifford, 2005] Clifford, R. (2005). Distributed suffix trees. *Journal of Discrete Algorithms*, (October 2006).

[Ghoting and Makarychev, 2010] Ghoting, A. and Makarychev, K. (2010). I/O efficient algorithms for serial and parallel suffix tree construction. *ACM Transactions on Database Systems*, 35(4):1–37.

[Gusfield, 2007] Gusfield, D. (2007). *Algorithms on strings, trees, and sequences : computer science and computational biology*. Cambridge Univ. Press.

[Japp, 2004] Japp, R. (2004). The Top-Compressed Suffix Tree A Disk-Resident Index for Large Sequences. *Bioinformatics Workshop 21st Annual British national Conference on Databases*, (January).

[Kurtz et al., 2004] Kurtz, S., Phillippy, A., Delcher, A. L., Smoot, M., Shumway, M., Antonescu, C., and Salzberg, S. L. (2004). Versatile and open software for comparing large genomes. *Genome Biology*, 5(R12):R12.1–R12.9.

[Mansour, 2012] Mansour, E. (2012). ERA : Efficient Serial and Parallel Suffix Tree Construction for Very Long Strings. 5(1):49–60.

[McCreight, 1976] McCreight, E. M. (1976). A space-economical suffix tree construction algorithm. *J. ACM*, 23:262–272.

[Needleman and Wunsch, 1970] Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453.

[Sadakane,] Sadakane, K. Compressed Suffix Trees with Full Functionality. *Communication*, 4:1–18.

[Smith and Waterman, 1981] Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197.

[Ukkonen, 1992] Ukkonen, E. (1992). Constructing Suffix Trees On-Line in Linear Time. In Van Leeuwen, J., editor, *Proceedings of the IFIP 12th World Computer Congress on Algorithms Software Architecture Information Processing 92 Volume 1*, volume 1, pages 484–492. North-Holland Publishing Co.