Análisis paralelo de secuencias de ADN sobre computadores con múltiples cores

Antoni Sanjuan[1], Vicente Arnau[2] y José M. Claver[3]

Resumen—**En este trabajo presentamos la paralelización de dos aplicaciones bioinformáticas utilizadas el análisis de largas secuencias de ADN. Está basada en un trabajo previo que transforma una secuencia de ADN en una secuencia binaria y se proponen diversas optimizaciones para su paralelización sobre computadores actuales de bajo coste con múltiples núcleos de procesamiento (*cores*). Las aplicaciones paralelas se han realizado utilizando el API de paralelización de aplicaciones sobre multiprocesadores con memoria compartida OpenMP. Los resultados experimentales obtenidos para diversos tamaños de secuencias de ADN muestran que las aplicaciones paralelas consiguen buenos resultados para computadores personales con múltiples procesadores, pero existen limitaciones cuando éstos se ejecutan en computadores que poseen un procesador con múltiples cores.**

Palabras clave— **Análisis de ADN, Paralelismo, Múltiprocesadores, Multicore, OpenMP.**

I. INTRODUCCIÓN

LA cantidad de datos disponibles en bioinformática está creciendo de forma exponencial en los últimos años y cada vez está siendo más necesario crear aplicaciones para el análisis de esta ingente cantidad de información. En particular, la cantidad de genomas secuenciados estos últimos años requiere herramientas para su análisis y clasificación. Es más, podemos afirmar que estas herramientas serán masivamente utilizadas en un futuro próximo si tenemos en cuenta que, en la actualidad, secuenciar el genoma completo de una persona tiene un coste de unos 40.000€. En el momento en que se reduzca un poco más este coste, obtener el genoma de una persona será una práctica habitual y la mejor forma de prevenir las futuras enfermedades que puede llegar a padecer.

Hoy en día hay muchas especies de las cuales ya se ha obtenido su genoma completo y estos son relativamente fáciles de conseguir a través de la página WEB del NCBI (ftp://ftp.ncbi.nih.gov/genomes/) de donde se pueden descargar los ficheros en formato FASTA. Se trata de ficheros de texto que contienen una largas secuencias formadas por combinaciones de 4 letras, correspondientes a las iniciales de los 4 nucleótidos presentes en el ADN (A=*Adenina*, C=*Citosina*, G=*Guanina* y T=*Timina*). En una primera observación estas secuencias parecen generadas aleatoriamente, pero no es así, podemos encontrar patrones de repetición o de ausencia que nos aportarán una valiosísima información sobre su contenido, como por ejemplo la localización de los genes, que son secuencias únicas en un cromosoma.

[1] Dpto. de Informática, Univ. Valencia, e-mail: *ansanva@alumni.uv.es*
[2] Dpto. de Informática, Univ. Valencia, e-mail: *Vicente.Arnau@uv.es*
[3] Dpto. de Informática, Univ. Valencia, e-mail: *Jose.Claver@uv.es*

La comparación de secuencias cortas de ADN es, posiblemente, el análisis más repetido en bioinformática. Así, con mucha frecuencia, una secuencia de ADN obtenida experimentalmente en un laboratorio es comparada con las secuencias contenidas en grandes bases de datos. A partir de esta comparación, se obtienen similitudes entre la secuencia de estudio y las ya existentes, de las cuales se conoce sus propiedades funcionales, que podrán ser similares a nuestra secuencia desconocida si comparten parte o la totalidad de los datos secuenciados [1]. Aunque no están diseñadas para ello, nuestras aplicaciones también pueden ser utilizadas con estos fines.

Pero cuando se habla de estudiar o comparar genomas completos de especies, los métodos de comparación de secuencias cortas no son aplicables. Se necesitan nuevas metodologías y, a la vez, grandes computadores [2]. En el trabajo presentado en 2007 por Arnau *et al.* [2] se ofrece una nueva perspectiva de análisis que llaman "*oligonucleotide profiling*", basado en el análisis exhaustivo de palabras de nucleótidos (hasta k=14 nucleótidos) que nos ofrece la posibilidad de realizar diversos tipos de análisis para el estudio de secuencias de ADN sobre un ordenador personal. Algunos trabajos previos habían utilizado estudios parecidos pero con tamaños de palabra más pequeños [4][5], lo cual no permitía el estudio de grandes cadenas de ADN, como por ejemplo los cromosomas humanos, y mucho menos una descripción estadísticamente significativa de los mismos.

Hay dos factores que han motivado la realización de nuestro trabajo: el primero es que los ordenadores personales de hoy en día poseen ya múltiples cores que no son aprovechados si no diseñamos las herramientas bioinformáticas teniendo en cuenta el paralelismo, y como segunda cuestión, decir que, aunque los análisis son muy rápidos (3 Mega nucleótidos por segundo para palabras de 13 nucleótidos), cuando hablamos de genomas completos como el humano, que posee 3 Giga nucleótidos, y lo queremos comparar por ejemplo con el del Chimpancé, a pesar de la rapidez de los algoritmos los tiempos de análisis son considerablemente altos. De aquí surge la motivación de nuestro trabajo, mejorar y paralelizar los algoritmos secuenciales presentados en [3] para optimizar su ejecución en computadores con múltiples cores y memoria compartida.

El número posible de combinaciones que puede haber de palabras distintas de k nucleótidos en una secuencia de ADN es igual a 4^k, teniendo en cuenta los 4 posibles valores de los nucleótidos {A, C, G, T}. Por lo tanto tenemos que guardar en memoria una estructura de 4^k elementos de tipo entero para obtener los datos de las frecuencias de todas las palabras de k nucleótidos presentes en una secuencia de ADN. Para un

computador con 2 GB de RAM el valor más alto de *k* es igual a 14, pues 4^{14} x 4 Bytes (tamaño del tipo de dato entero), que es igual a 1 GB.

En este trabajo se han desarrollado dos metodologías de análisis de secuencias de ADN:

1. La primera es la obtención de las frecuencias de aparición de todas las palabras de *k* nucleótidos presentes en una secuencia de ADN y el volcado de esta Tabla de Frecuencias en un fichero.
2. La segunda aplicación consiste en obtener una Tabla de Perfiles formada por los valores de las frecuencia de aparición de cada palabra de una secuencia *diana* en una segunda secuencia *fuente* (definido como "*oligonucleotide profiling*" en [3]).

Este segundo tipo de análisis ofrece muchas posibilidades de análisis biológicos, pues no solo es útil comparando grandes secuencias de ADN, también ha sido utilizado utilizando secuencias *diana* o *fuente* de tamaño pequeño (genes por ejemplo) y ofreciéndonos la posibilidad de realizar diferentes tipos de análisis sobre las secuencias de ADN [6]. Por poner un ejemplo práctico, si disponemos de una secuencia corta de ADN y queremos saber en qué lugares de un cromosoma aparece esta secuencia o secuencias similares, utilizando la secuencia corta como fuente y el cromosoma completo como diana nos ofrecerá una grafica con valores cero en casi la totalidad de la misma, excepto en aquellos lugares en donde está presente la secuencia corta a estudio.

A continuación se describe la organización de este artículo. En la siguiente sección describimos con detalle las aplicaciones sobre las que se centra este trabajo. En la sección III se realiza un estudio de los algoritmos realizados y de las técnicas de programación utilizadas para el desarrollo de estas las dos aplicaciones descritas en la sección anterior. En la sección IV se describirán las pruebas experimentales realizadas y los resultados obtenidos. Por último, en la sección V se presentan las conclusiones de este trabajo y sus posibles líneas de continuación.

II. ANÁLISIS DE SECUENCIAS DE ADN

Como ya se ha comentado en la introducción, las aplicaciones que se han desarrollado se basan en el análisis de los datos provenientes de unos ficheros de texto donde se encuentran las secuencias de nucleótidos que forman un cromosoma en formato FASTA. El formato de estos ficheros consiste en sucesivas líneas de preferentemente 80 nucleótidos precedidas de una línea de comentario que describe la secuencia y que comienza con el carácter ">". Un ejemplo de este tipo de ficheros es el que se muestra a continuación:

```
>gi|89027|ref=1| Homo sapiens chr_22
GTGTCTCATGCCTGTAATCCCAACACTTTGGGAGCCTGAGGTAG
AAGTTGAGGTCAGGAGGTCCAAGACACACTTTGGGAGCCTGAGG
TAGAAGAGCCTGGGCAACATAGTGAGTCTACAAAAAAAATATTG
AGAATCTACATAAATATAGTGGGGGTGGGGGTATATAGTGGGGG
TGGGGGTATATAGCAAAACCCAGCCGGGCATGGTG ...
>gi|89028|ref=2| Homo sapiens chr_22
TGAGGTAGAAGTTGAGGTCAGGAGGTCCAAGACACACTTTGGGA
ATAGTGAGTCTACAAAAAGGGTATATAGTGGGGGTGGGGGTATA
TAGCAAAAGGCATG ...
```

Cuando analizamos una secuencia de ADN, leemos *k* nucleótidos y formamos la primera palabra. A continuación, cada nuevo nucleótido leído genera una nueva palabra con los *k*-1 nucleótidos anteriores. Es como una ventana deslizante de *k* nucleótidos que se mueve a lo largo de la secuencia. En este análisis se deben discriminar las líneas que empiezan por el carácter ">" y las letra "N", que reinician la cuenta de *k* nucleótidos de una nueva palabra.

La primera aplicación, como se ha comentado, consiste en leer este tipo de ficheros y obtener las frecuencias de aparición de palabras de un tamaño determinado. El acceso a la tabla de frecuencias de las palabras encontradas en el cromosoma lo realizamos utilizando la codificación binaria de los nucleótidos propuesta en [3]. Para ello, el carácter A lo codificamos como "00", la C como "01", la G como "10" y la T como "11". De esta forma, y concatenando estas codificaciones según vamos conformando las palabras leídas en la secuencia de ADN, obtenemos un código binario de *k*2* bits que se utiliza para acceder directamente a la posición de la tabla que tiene las frecuencias de aparición de las 4^k palabras posibles de *k* nucleótidos. Podemos ver un ejemplo de este funcionamiento en la Fig. 1.

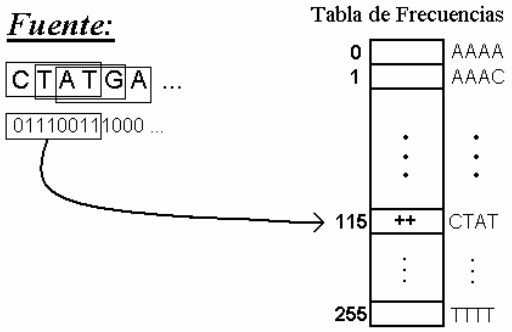

Fig. 1. Codificación de nucleótidos y Tabla de Frecuencias para palabras de tamaño *k*=4.

La segunda aplicación utiliza la tabla de frecuencias generada al analizar un fichero de ADN, al que llamamos secuencia *fuente*, pero con otra finalidad, generar un perfil de una segunda secuencia que llamaremos secuencia *diana*. La idea es generar una tabla de perfiles de forma que leeremos el fichero *diana* y para cada palabra de este anotaremos en la tabla de perfiles la frecuencia con que aparece esta palabra en la secuencia *fuente*. En algunos estudios biológicos, la secuencia *diana* y *fuente* pueden ser la misma secuencia.

El resultado de este análisis sería una tabla con tantos valores como palabras consecutivas de k nucleótidos posea la secuencia diana. La Fig. 2 muestra un esquema del análisis comparativo de 2 cromosomas para un tamaño de palabra *k*=4.

Dado que la cantidad de palabras que puede haber en la secuencia *diana* es casi igual al tamaño del fichero, para poder visualizar el perfil utilizando programas de representación gráfica de datos, procedemos a agrupar los valores de la tabla en grupos de R elementos y calculamos la media de ellos, anotando solo este valor. De esta forma la tabla de perfiles es R veces más pequeña. Estamos trabajando en la posibilidad de aplicar

sucesivamente la transformada Wavelet, seleccionando la componente de baja frecuencia, para la reducción de los datos en lugar de utilizar los promedios.

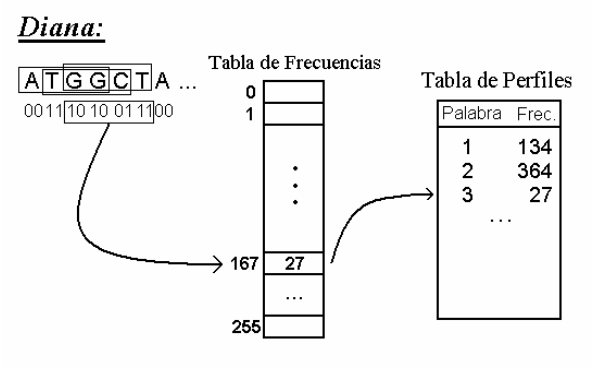

Fig. 2. Generación de la Tabla de Perfiles para el fichero *diana*, utilizando la Tabla de Frecuencias generada para el fichero *fuente* y con tamaño de palabras k=4.

III. OPTIMIZACIÓN Y PARALELIZACIÓN

En esta sección se presentan las estrategias y metodologías utilizadas para optimizar el tiempo de acceso a los ficheros de datos y la paralelización de las aplicaciones de análisis de las secuencias de ADN presentadas en la sección II.

En la aplicación secuencial descrita en [3] la lectura del fichero que contiene la secuencia de ADN se realizaba leyendo del propio fichero carácter a carácter. Así, el algoritmo consistía en una lectura iterativa de la secuencia de nucleótidos hasta encontrar el final de ésta. Al mismo tiempo que se leía un carácter, se reconocía una nueva palabras de k nucleótidos y utilizando la misma codificación que presentamos en este trabajo, se accedía a la tabla de frecuencias para incrementar el contador de la palabra leída (similar a lo presentado en la ver Fig. 1).

En nuestra versión se ha conseguido optimizar el tiempo de ejecución modificando el modo en que el algoritmo realiza la lectura de los caracteres del fichero. Para ello hemos utilizado la función *Posix* de *C* denominada *mmap*, que realiza un mapeado del fichero en memoria, optimizando los tiempos de lectura de los datos del fichero de entrada. El objetivo de *mmap* es permitir el acceso a los datos mediante direcciones del sistema de memoria virtual, por lo que los datos del fichero pueden ser tratados como estructuras de memoria estándar. Utilizando esta llamada se puede acceder al fichero como si de un vector se tratara, consiguiendo así una mejora de rendimiento considerable como se presentará en la siguiente sección.

Para acelerar el procesamiento de los datos en esta aplicación se han considerado 2 propuestas de paralelización de este algoritmo. En la primera se ha buscado el máximo rendimiento, mientras que en la segunda se ha pretendido buscar la eficiencia en el uso de memoria del computador, con el fin de poder tratar problemas de mayor tamaño (para palabras con k>14).

En la primera propuesta, una vez se ha cargado el fichero en memoria, se reparte equitativamente el procesado de la secuencia de ADN entre el número de procesadores o cores sobre los que se vaya a trabajar. Cada uno de los hilos de estos elementos de proceso trabajará en paralelo, realizando los mismos pasos que realizaba el algoritmo secuencial sobre el segmento de ADN asignado y actualizando los resultados en una estructura de datos privada. En la última etapa de esta fase, cada uno de los hilos, menos el último, debe leer los k-1 siguientes nucleótidos que se solapan con los tratados por el hilo contiguo, y así completar el procesamiento de todas sus secuencias. Como puede verse en la Fig. 3 para un ejemplo con 4 hilos.

Al finalizar el procesamiento se realiza una agrupación de las tablas privadas de cada hilo, haciendo una suma en paralelo sobre una tabla global, en la que en cada paso los hilos escriben en posiciones distintas para evitar datos erróneos y conflictos de acceso a memoria al escribir en las mismas posiciones de memoria al mismo tiempo.

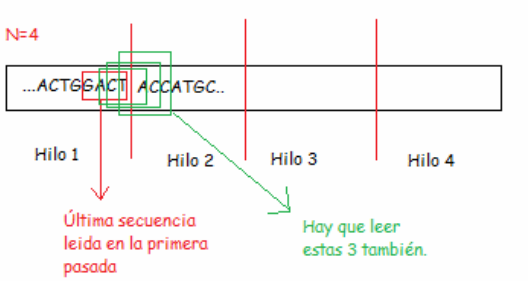

Fig. 3. Explicación gráfica de la división de los datos entre hilos de ejecución y resolución del procesado de las palabras frontera.

La segunda versión paralela propuesta, como se dijo anteriormente, se ha realizado para optimizar la cantidad de memoria que consumirá cada proceso. Esta versión es una adaptación del trabajo presentado en [7] donde se utilizaba un sistema multicomputador, en lugar de un procesador con múltiples cores como utilizamos nosotros en este trabajo. En este caso el algoritmo se basa en que todos los hilos recorran toda la secuencia de ADN (no se distribuye en segmentos iguales la secuencia a analizar). Pero cada hilo solo se encargará de procesar las palabras de k nucleótidos que empiecen por el prefijo destinado a ese hilo. Por ejemplo, si tenemos 4 hilos, el primer hilo se encargará de incrementar los contadores de la Tabla de Frecuencias asociados a las palabras que empiecen con A, el segundo con las que empiecen con C, etc. El número de hilos a ejecutar debe ser potencia de 4, al ser éste el número de nucleótidos. Con ello se consigue reducir la cantidad de memoria que utiliza cada hilo para almacenar la Tabla de Frecuencias de 4^{k+1} bytes a $4^{k+1}/t$ bytes, donde t es el número de hilos, que son los necesarios para almacenar los resultados de su prefijo. Al final los resultados de todos los hilos se unen para obtener la tabla final, aunque, debido a que los hilos trabajan sobre partes disjuntas de la tabla resultado, no serían necesarias las copias privadas de esta tabla. Esta propuesta es interesante para aplicarla en sistemas multiprocesador con memoria distribuida para poder obtener análisis con palabras de k mayores, y es una propuesta interesante para computadores con múltiples cores donde hay compartición del segundo nivel de cache (L2).

En la segunda aplicación descrita en la sección II distribuimos equitativamente el procesado de la secuencia *diana* entre los procesadores o cores. Así, cada hilo de ejecución obtendrá una parte de la tabla final de frecuencias de aparición del segundo cromosoma (como mostramos en la Fig. 2). La compresión de la Tabla de Perfiles se pospone al final del cálculo de ésta, pues necesitamos agrupar los valores obtenidos en grupos de R elementos para calcular su media y escribir el resultado final en un fichero de salida. Para cada valor promedio escrito en el fichero se calculará también la varianza de los R elementos.

IV. RESULTADOS EXPERIMENTALES

La versión paralela se ha diseñado utilizando OpenMP, ya que está pensado para ser ejecutado en procesadores con varios núcleos o multiprocesadores que tienen memoria compartida. El OpenMP es actualmente un estándar de facto y consiste en un API que permite la inclusión de una serie de directivas dentro de un programa escrito en C o Fortran. Las directivas de OpenMP son reconocidas por el compilador, creando un código que puede ser ejecutado sobre varios hilos de ejecución que se distribuyen entre los procesadores o cores de los procesadores disponibles.

Para la obtención de resultados se han llevado a cabo una serie de pruebas ejecutando las diferentes versiones de los programas y variando el tamaño de las cadenas a analizar a si como, para las versiones paralelas, su ejecución con 2, 3 o 4 hilos. Las pruebas se han llevado a cabo en dos tipos de computadores. El primero de ellos (AMD4P) es un computador que tiene 4 procesadores AMD Opteron 848 a 2,2 GHz, 1 MB de cache L2 y 8 GB de memoria principal. El segundo (INTEL4C) es un computador que posee dos procesador Intel Xeon X5355 a 2,66 GHz con 4 cores cada uno, 8 MB de cache L2 (4 MB por cada 2 cores), y 16 MB de memoria principal. Para compilar los programas se ha utilizado el compilador ICC de Intel, tanto para los programas secuenciales como para los desarrollados con OpenMP, en ambos computadores.

Las pruebas se han realizado comprobando que la máquina no estaba cargada, y se han tomado 5 ejecuciones para cada caso, obteniendo como valor observable la media de esas ejecuciones. Se han utilizado para las pruebas los cromosomas humanos CH1, CHX, CH14 y CH21 que tienen unos 219, 147, 82 y 33 Millones de nucleótidos, respectivamente.

El primer estudio realizado ha sido la comparación de las versiones secuenciales del programa que analiza un determinado cromosoma y obtiene las frecuencias de aparición de sus palabras de tamaño *k*. Se comparara la versión secuencial que lee y procesa carácter a carácter el fichero con la versión que utiliza la llamada al sistema "mmap" para volcar la información del fichero.

En la figura 4 se muestran los resultados en MB/s (Millones de bytes por segundo) para el algoritmo secuencial sobre el cromosoma humano CH1, donde podemos ver que hasta valores de k=7 en AMD4P y k=9 en INTEL4C el rendimiento es lineal. Sin embargo, cuando k aumenta por encima de estos valores, el rendimiento decae debido a que la Tabla de Frecuencias ya no cabe en la memoria caché y se producen reemplazos en la misma para su utilización.

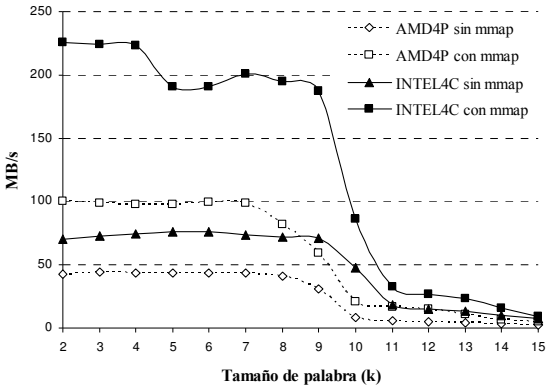

Fig. 4. Velocidad de procesamiento del análisis de frecuencias del cromosoma humano CH1 para todos los posibles tamaños de palabra (k) sobre los dos computadores utilizados en este estudio.

El resultado de las pruebas realizadas sobre el cromosoma CH1 refleja cómo la optimización de la lectura de datos con "mmap" proporciona un rendimiento mucho mayor que sin su utilización. Para tamaños pequeños de secuencia la mejora es evidente, consiguiéndose una reducción del 50% en la ejecución de la aplicación, pero al aumentar el tamaño de k, los resultados obtenidos con y sin "mmap" se igualan.

En la Fig. 4 podemos observar que, aunque las características arquitectónicas de los procesadores de INTEL y de AMD influyen en el rendimiento del algoritmo, dado que el procesador de AMD posee 1 MB de L2 mientras que un core del procesador de INTEL puede disponer de 4 MB, las mejoras en tiempos de ejecución son considerables y la utilización de la función *mmap* es imprescindible si buscamos una rápida ejecución de nuestros algoritmos.

Para la primera aplicación se ha implementado la versión que optimiza el tiempo de ejecución y se ha realizado un estudio sobre la aceleración obtenida, comparando el tiempo de ejecución de la versión secuencial mejorada con "mmap" con la ejecución paralela con 2, 3 y 4 procesos. En este caso se ha utilizado un tamaño de palabra con k= 14, que nos permite obtener unos resultados indicativos sobre el aumento de rendimiento con la utilización de varios procesadores.

Dado que el tiempo de ejecución de las dos aplicaciones implementadas depende del tamaño de las secuencias de ADN analizadas, resulta más interesante mostrar la velocidad de procesado de las secuencias en MB/s. Así, en las Figuras 5 y 6 podemos observar la velocidad de procesado de los cromosomas estudiados al aumentar el número de hilos en los computadores AMD4P e INTEL4C, respectivamente. Como vimos anteriormente, para estos valores de k las versiones secuenciales, aun utilizando la función "mmap", obtienen rendimientos muy bajos, entre 4 y 7 MB/s para el AMD4P y entre 7 y 14 MB/s para el INTEL4C. Para un valor de k= 14, la Tabla de Frecuencias resultante es de 4^{14+1} = 1 GB, lo que conlleva un alto trasiego de datos entre la memoria principal y la cache del procesador. Este comportamiento viene agravado por la falta de

localidad en los accesos a la Tabla de Frecuencias. Debido a ello, las velocidades de procesado para estos valores de *k* son tan pobres. Los resultados para el caso de un hilo son el doble de rápidos para el computador de Intel por el simple hecho de que el procesador funciona a una frecuencia el doble de rápida y la cache de segundo nivel (L2) también es mucho mayor.

Cuando se incrementa el número de hilos se observa un incremento en la velocidad de procesado, que es mayor cuanto mayor es el tamaño del cromosoma a analizar, alcanzándose en el mejor de los casos (CH1) una velocidad con 4 hilos de 24 MB/s para el AMD4P y de 26,5 MB/s para el INTEL4C. Este comportamiento es más lineal en el caso del computador AMD4P con 4 procesadores, pues cada uno de ellos posee una cache L2 propia de 1MB, mientras que en el computador INTEL4C, la cache L2 de 4 MB es compartida por dos cores.

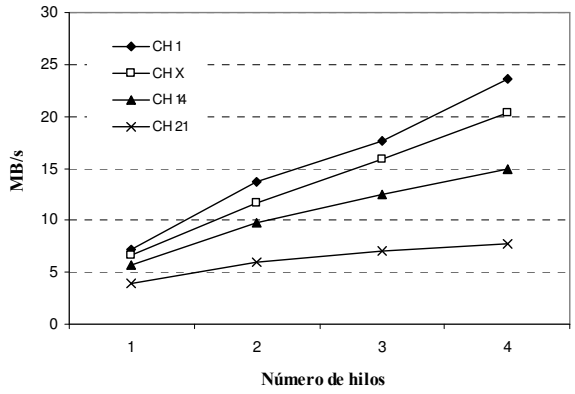

Fig. 5. Velocidad de procesado en MB/s de la primera aplicación para *k*= 14 con diferente número de hilos en el computador AMD4P.

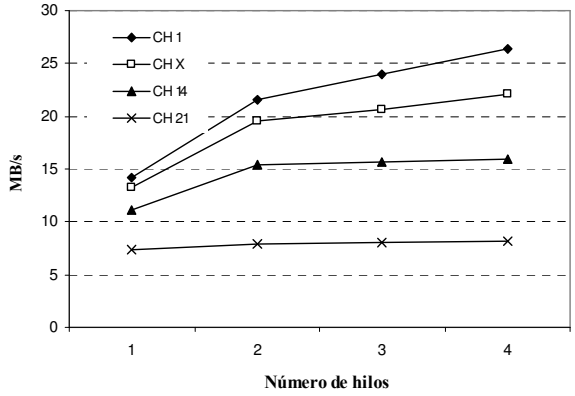

Fig. 6. Velocidad de procesado en MB/s de la primera aplicación para *k*= 14 con diferente número de hilos en el computador INTEL4C.

Este hecho hace que las aceleraciones obtenidas en esta primera aplicación para el computador AMD4P sean mejores que las obtenidas en el computador INTEL4C, como muestran las Figuras 7 y 8, respectivamente. Las aceleraciones para cromosomas de gran tamaño, como el CH1, se acercan a la aceleración teórica en el computador AMD4P. Siendo modestas para cromosomas de menor tamaño como el CH21, en el que no se supera una aceleración de 2 para 4 hilos, lo que supone una eficiencia del 50%.

Las aceleraciones de esta aplicación en el computador INTEL4C son muy bajas, no superándose en ningún caso una aceleración de 2, y siendo prácticamente de 1 para cromosomas de tamaño pequeño como el CH21. Esto nos indica que tal vez el algoritmo paralelo diseñado para esta aplicación puede tener un buen comportamiento para arquitecturas con múltiples procesadores, pero no es la adecuada para la arquitectura de los procesadores multicore actuales como el de Intel.

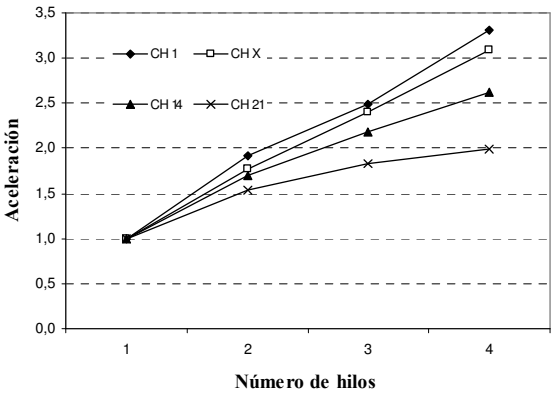

Fig. 7. Aceleración de la primera aplicación para *k*= 14 con diferente número de hilos en el computador AMD4P.

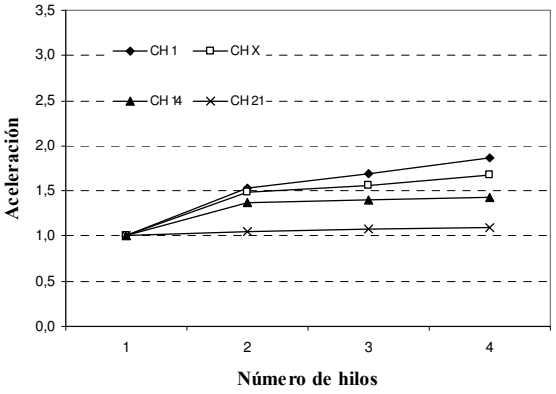

Fig. 8. Aceleración de la primera aplicación para *k*= 14 con diferente número de hilos en el computador INTEL4C.

Para la segunda aplicación, al igual que para la primera, se ha realizado un estudio de la velocidad de procesado en MB/s, tomando como referencia la ejecución secuencial, y comparándola con la versión paralela para 2, 3 y 4 hilos de ejecución. Se han elegido para las pruebas realizadas los pares de cromosomas humanos siguientes: (CH1-CH21), (CHX-CH21), (CH14-CH21) y (CH21-CH21), donde el cromosoma de la izquierda es el *fuente* y el de la derecha el *diana*.

En la segunda aplicación la carga computacional es mayor que en la primera, por lo que se obtiene un incremento más uniforme en las velocidades de procesado al utilizar un mayor número de hilos. Sin embargo, sigue observándose una mejora mayor en los análisis en los que las secuencias son mayores (CH1-CH21) frente a aquellas en las que las secuencias son de menor tamaño (CH21-CH21), como puede verse en las Figuras 9 y 10.

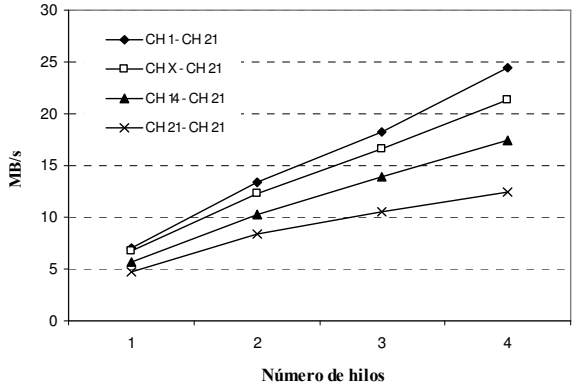

Fig. 9. Velocidad de procesado en MB/s de la segunda aplicación para *k*= 14 con diferente número de hilos en el computador AMD4P.

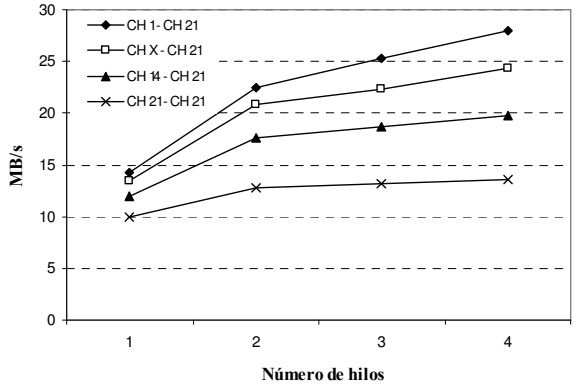

Fig. 10. Velocidad de procesado en MB/s de la segunda aplicación para *k*= 14 con diferente número de hilos en el computador INTEL4C.

Se vuelve a poner de manifiesto el mejor comportamiento del computador AMD4P frente al INTEL4C al aumentar el número de hilos. Así, para el computador AMD4P se obtienen aceleraciones entre 2,7 y 3,5 para 4 hilos (ver Fig. 11), mientras que en el computador INTEL4C las aceleraciones están entre 1,4 y 2 (ver Fig. 12), aunque para valores de *k* menores los resultados han sido mejores. Se hacen evidentes de nuevo los problemas de aceleración del algoritmo paralelo para el procesador de Intel con 4 cores y que requerirán de un estudio detallado para adaptarlos a estas arquitecturas.

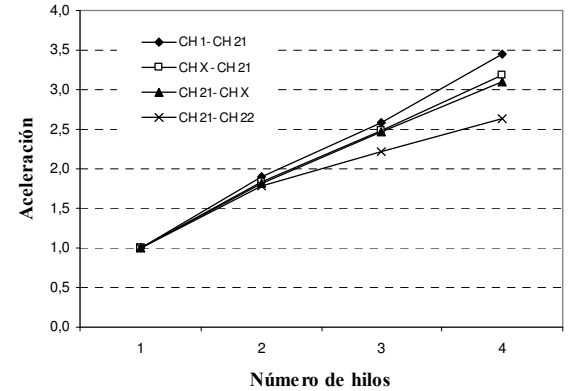

Fig. 11. Aceleración de la segunda aplicación para *k*= 14 con diferente número de hilos en el computador AMD4P.

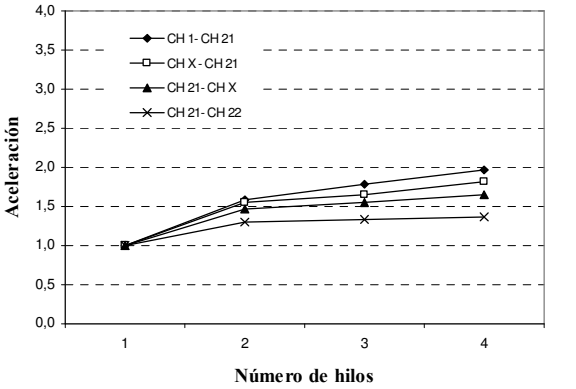

Fig. 12. Aceleración de la segunda aplicación para *k*= 14 con diferente número de hilos en el computador INTEL4C.

V. CONCLUSIONES

Se ha demostrado que se puede reducir considerablemente el tiempo de ejecución de aplicaciones relacionadas con el análisis de secuencias de ADN sobre procesadores con múltiples cores. En los casos de prueba basados en secuencias de cromosomas reales se demuestra que esta mejora es significativa, y estamos seguros de que este estudio preliminar puede ser la base para el análisis más profundo de cómo adaptar los algoritmos propuestos para estas nuevas arquitecturas emergentes y ya disponibles para todo el mundo. Destacar también la importancia de una lectura eficiente de los datos, que puede conseguir una optimización importante incluso para las aplicaciones secuenciales. Para finalizar, comentar la relevancia de los análisis propuestos por la importancia que el estudio de los Genomas está teniendo en la sociedad por las innumerables aplicaciones presentes y futuras.

AGRADECIMIENTOS

El presente trabajo ha sido financiado mediante el programa europeo FEDER y el proyecto del MEC ``Consolider Ingenio-2010 CSD 2006-00046" y ``TIN 2006-15516-C04-02".

REFERENCIAS

[1] S. Vinga, J. Almeida. “Alignment-free sequence comparison – a review”. *Bioinformatics* 2003, 19:513-523.

[2] J. Healy, E.E.Thomas, J.T. Schwartz , M. Wigler, “Annotating large genomes with exact word matches”. *Genome Res* 2003, 13:2306-2315.

[3] V. Arnau, M. Gallach , I. Marin. "Fast comparison of DNA sequences by oligonucleotide profiling". *BMC Research Notes* 2008, 1:5. 28 February 2008.

[4] P.J. Deschavanne, A. Giron, J. Vilain , G. Fagot, B. Fertil. “Genomic signature: characterization and classification of species assessed by chaos game representation of sequences”. *Mol Biol Evol* 1999, 16:1391-1399.

[5] L. Mariño-Ramírez,J.L. Spouge, G.C. Kanga, D. Landsman. “Statistical analysis of over-represented words in human promoter sequences”. *Nucl Acids Res* 2004, 32:949-958.

[6] Miguel Gallach, Vicente Arnau, Ignacio Marin. "Global patterns of sequence evolution in Drosophila". *BMC Genomics* 2007, 8:408.

[7] X.Y. Yang, A. Ripoll, V. Arnau, I. Marín, E. Luque.“Genomic-Scale Analysis of DNA Words of Arbitrary Length by Parallel Computaction”. NIC Series Vol. 33: *Parallel Computing Current & Future Issues of High-End Computing*, 2006, pp.: 623-630.