Análisis paralelo de secuencias de ADN mediante el uso de GPU y CUDA

A. J. Peña[1], J. M. Claver[2], A. Sanjuan[2], V. Arnau[2],

[1]Dpto.de Ing. y Ciencia de los Computadores, Univ. Jaume I, Castellón, España, tpenya@gmail.com

[2]Dpto. de Informática, Univ. de Valencia, Burjassot, España, {jose.claver, ansanva, vicente.arnau}@uv.es

Resumen

En este trabajo en curso presentamos las propuestas de aceleración de dos aplicaciones bioinformáticas utilizadas para el análisis de largas secuencias de ADN. Se proponen diversas aproximaciones para su paralelización sobre computadores actuales de bajo coste con múltiples núcleos de procesamiento (cores) que poseen una tarjeta GPU como elemento de co-procesado. Las aplicaciones paralelas se han desarrollado combinando el uso de OpenMP sobre procesadores multicore de la placa base y la librería CUDA sobre las tarjetas de NVIDIA GeForce 8800 GTX y Tesla C870. Los resultados preliminares para diversos tamaños de secuencias de ADN muestran algunos problemas para la obtención de buenos rendimientos sobre estas arquitecturas y se trabaja en el desarrollo de nuevas técnicas de tratamiento de los datos que mejoren las limitaciones iniciales.

1. Motivación

El número de genomas secuenciados es cada vez mayor y es necesario que las herramientas informáticas para su análisis y clasificación aprovechen las nuevas tecnologías de diseño de los nuevos sistemas de computación, como por ejemplo las tarjetas GPUs. Podemos acceder a ellos conectándonos a 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 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 y secuencias únicas que nos aportarán una valiosísima información biológica.

La comparación de secuencias cortas de ADN es, posiblemente, el análisis más repetido en bioinformática [1]. Pero cuando se habla de estudiar o comparar genomas completos, 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.* [3] se ofrece una nueva perspectiva de análisis que llaman "*oligonucleotide profiling*", basado en el análisis exhaustivo de palabras de nucleótidos (hasta k=14 nucleótidos). Algunos trabajos previos habían utilizado estudios parecidos pero con tamaños de palabra más pequeños [4]. Otros trabajos han realizado representaciones gráficas a partir de ADN [5], pero con poca aplicación práctica. Pretendemos en este trabajo mostrar algunas posibles realizaciones de los algoritmos 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. 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 valor de k igual a 14, necesitamos 4^{14} x 4 Bytes (tamaño del tipo de dato entero), es decir, 1 GB de memoria.

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

- La primera es la obtención de las frecuencias de aparición de todas las palabras de k nucleótidos de una secuencia de ADN.
- La segunda aplicación consiste en obtener una Tabla de Perfiles de una secuencia de ADN (definido como "*oligonucleotide profiling*" en [3]).

Esta segunda aplicación ofrece muchas posibilidades de análisis biológicos, permitiendo estudiar y caracterizar genomas completos de una especie, como se muestra en [6] donde se estudian y comparan entre sí los 7 genomas completos de la mosca.

A continuación se describe la organización de este artículo. En la siguiente sección se describen con detalle las aplicaciones sobre las que se centra este trabajo. En la sección 3 se describen los aspectos más importantes de la arquitectura de las GPU y la metodología de programación con CUDA. En la sección 4 se describen algunas de las optimizaciones iniciales llevadas a cabo y las estrategias que se proponen para realizar la paralelización del problema tratado. En la sección 5 se realiza un estudio de los algoritmos realizados y de las técnicas de programación utilizadas para el desarrollo de la primera de las aplicaciones descritas en la sección anterior y se describirán las pruebas experimentales realizadas y algunos de los resultados obtenidos hasta el momento. Por último, en la sección 6 se presentan las conclusiones de este trabajo y sus posibles líneas de continuación.

2. Análisis de secuencias de ADN

Como ya se ha comentado en la introducción, analizaremos ficheros de texto 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 “>”.

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 letras “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 todas las palabras de *k* nucleótidos para su posterior volcado en un fichero de texto. 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 Figura 1.

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.

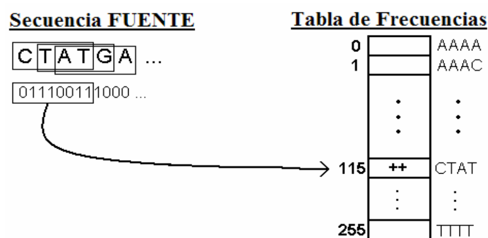

Figura 1. *Codificación de nucleótidos y Tabla de Frecuencias para palabras de tamaño k=4*

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

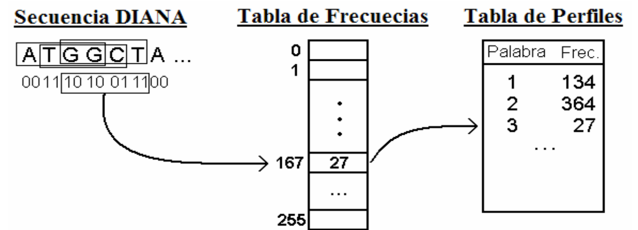

Figura 2. *Generación de la Tabla de Perfiles para el fichero diana, utilizando la Tabla de Frecuencias generada para el fichero fuente y con tamaño de palabras k=4.*

Dado que la cantidad de palabras que puede haber en la secuencia *diana* es prácticamente 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.

3. GPU y su programación con CUDA

La creciente necesidad de cómputo en los actuales sistemas gráficos por ordenador ha propiciado la aparición de dispositivos que pueden ser utilizados para acelerar otras aplicaciones más generales y que tienen como característica común su gran paralelismo e intensa carga computacional.

Las características principales de este tipo de dispositivos son una gran cantidad de elementos de proceso, o cores, integrados en un único circuito integrado a costa de una importante reducción de su memoria cache. Por lo tanto su arquitectura está altamente segmentada y se requiere un modelo de programación SIMD para obtener el máximo partido de sus características.

Las características arquitecturales de las GPU vienen determinadas por las aplicaciones gráficas, que son diferentes a las necesarias en las CPU del computador. En particular, en los procesos gráficos existe un alto paralelismo en las operaciones y en el acceso a los datos, tanto local como temporal, por lo que no necesita ni la complejidad de las actuales CPU, aunque sí su potencia aritmética, ni sus complejos sistemas de memoria cache.

Las empresas que lideran actualmente el mercado de GPUs son NVIDIA y ATI/AMD. De ellas, NVIDIA es la líder actual en el campo de la GPGPU debido a su bajo precio y el uso de CUDA (Compute Unified Device Architecture), que permite la programación de su gama más potente de tarjetas (a partir de la serie 8, incluyendo GeForce, Quadro y la línea Tesla) de forma similar a la programación en C, manteniendo la compatibilidad binaria entre ellas y acercando su modelo de programación a la de los programadores no expertos en lenguajes específicos para GPU como OpenGL y Cg.

3.1. CUDA

Es habitual emplear CUDA [9] para utilizar las GPU de NVIDIA como coprocesador para acelerar ciertas partes de un programa, generalmente aquellas con una elevada carga computacional por dato, ya que es en este tipo de cálculos donde más rendimiento se suele obtener.

En CUDA, los cálculos se distribuyen en una malla o grid de bloques de *threads*, donde todos los bloques tienen el mismo tamaño (número de threads). Estos threads son los que ejecutan el código de la GPU, conocido como *kernel*. Las dimensiones de la malla y de los bloques que contiene deben ser cuidadosamente escogidas para obtener el máximo rendimiento posible, basándose habitualmente en el problema específico a tratar.

En general, se puede ver un grid como una representación lógica de la propia GPU, un bloque como un procesador multinúcleo y un thread como uno de estos núcleos de proceso. Cada multiprocesador ejecuta una serie de bloques en segmentos de tiempo y un bloque siempre se ejecuta en un mismo multiprocesador. Los threads de un bloque se agrupan en *warps*, de modo que todos los threads de un *warp* en todo momento ejecutan la misma instrucción, habitualmente sobre distintos datos. En dispositivos de *'compute capability'* 1.0, como lo es la GeForce 8800 GTX, un *warp* está computesto por 32 threads.

Los threads de un mismo bloque pueden cooperar a través de 16KB de memoria compartida de rápido acceso y se dispone de mecanismos de sincronización a modo de "barrera" que permiten la cooperación de forma segura. No ocurre lo mismo entre distintos bloques, donde no se dispone de mecanismo de sincronización a través de la memoria global (la DRAM del dispositivo).

4. Optimización y Paralelización

En nuestra versión secuencial del algoritmo se optimiza el tiempo de ejecución modificando el modo en que el algoritmo realiza la lectura de los caracteres del fichero. Para ello se ha 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.

Se han llevado a cabo pruebas en dos tipos de computadores. El primero de ellos (AMD4P) es un computador que tiene 4 procesadores AMD Opteron 848 a 1 GHz, 1 MB de cache L2 y 8 GB de memoria principal. El segundo (INTEL4C) es un computador que posee dos procesadores 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 GB de memoria principal.

La Figura 3 muestra el resultado de las pruebas realizadas sobre el cromosoma humano CH1. Reflejan 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 van igualando, debido a que la tabla de frecuencias ya no cabe en la cache y se producen reemplazos.

Se han considerado 2 propuestas de paralelización, 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 analizar largas cadenas de ADN con valores de k altos (*k*>14).

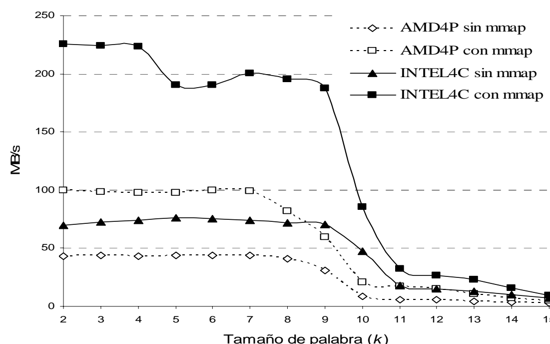

Figura 3. *Velocidad de procesamiento de la tabla de frecuencias del cromosoma CH1 paras los posibles tamaños de palabra (k) sobre los dos computadores utilizados en este estudio.*

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 threads 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. Cada uno de los threads, menos el último, debe leer los *k*-1 siguientes nucleótidos que se solapan con los tratados por el thread contiguo, y así completar el procesamiento de todas sus secuencias, como puede verse en la Figura 4 para un ejemplo con 4 threads.

Al finalizar el procesamiento se realiza una agrupación de las tablas privadas de cada thread, haciendo una suma en paralelo sobre una tabla global, en la que en cada paso los threads 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. Una primera propuesta de implementación de estas optimizaciones sobre computadores personales con múltiples cores se puede ver en [8].

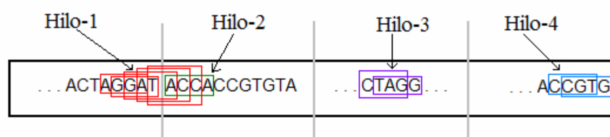

Figura 4. *Explicación gráfica de la división de los datos entre threads de ejecución y resolución del procesado de las palabras frontera para k=4.*

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, de memoria distribuida, en lugar de un procesador con múltiples cores como el utilizado en este trabajo. En este caso el algoritmo se basa en que todos los threads recorran toda la secuencia de ADN (no se distribuye en segmentos iguales la secuencia a analizar). Pero cada thread solo se encargará de procesar las palabras de k nucleótidos que empiecen por el prefijo destinado a ese thread. Por ejemplo, si tenemos 4 threads, el primero 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 threads 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 thread para almacenar la Tabla de Frecuencias de 4^{k+1} bytes a $4^{k+1}/t$ bytes, donde t es el número de threads. Al final los resultados de todos los threads se concatenan para obtener la tabla final 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, que genera la Tabla de Perfiles, distribuimos equitativamente el procesado de la secuencia *diana* entre los procesadores o cores. Así, cada thread de ejecución obtendrá una parte de la tabla de perfiles del segundo cromosoma (como mostramos en la Figura 2). La compresión de esta tabla se pospondrá 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.

5. Aceleración del análisis con GPUs.

En esta sección se detallan las diversas aproximaciones que se han probado para abordar la aceleración del análisis de frecuencias de una secuencia de ADN que hemos descrito en la sección 2 sobre las GPU de gama alta de NVIDIA. En particular hemos utilizado un computador con un Intel Core 2 Duo E6550 a 2.33GHZ y 2GB de RAM con una tarjeta GeForce G8800 GTX y un computador con un Intel(R) Core(TM)2 Duo CPU E8400 a 3GHz con una tarjeta Tesla C870. Se han utilizado para las pruebas los cromosomas humanos CH1, CHX, CH14 y CH21. Por espacio solo se presentan resultados para CH1 que tiene unas 219 Millones de bases.

En la primera aproximación del algoritmo sobre la GPU, la secuencia de ADN se almacenará en la memoria global de ésta y los threads se reparten por igual las 4^k posiciones de la tabla de frecuencias. La secuencia de ADN será procesada por partes, de modo que su tamaño no exceda el disponible en la memoria compartida de los bloques. En la Figura 5 se muestra esquemáticamente el algoritmo implementado como primera aproximación a la resolución del problema en la GPU.

Cada parte de la secuencia de ADN cargada en memoria compartida es recorrida por los *threads* tantas veces como palabras de la tabla de frecuencias tienen asignadas, acumulando en un registro las veces que aparece la palabra que se esté buscando en ese instante. Al final de cada iteración se acumulan en la tabla de frecuencias de memoria global los valores de aparición de estas palabras. En esta primera aproximación se pretende explotar la característica de difusión de memoria compartida al acceder todos los threads de un bloque a la misma posición de memoria al explorar la secuencia de ADN.

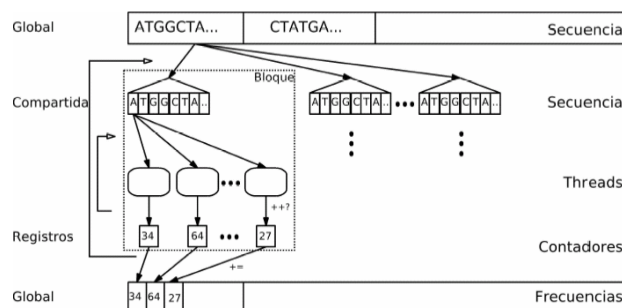

Figura 5. *Cálculo de la Tabla de frecuencia de palabras en la GPU según la primera aproximación.*

Los resultados obtenidos en esta primera aproximación son muy malos. Así, a partir de k=5 (tabla con 1024 frecuencias) el rendimiento cae significativamente, debido a que se configuró un GRID con 16 bloques de 64 threads cada uno (1024 threads en total). De todos modos, el rendimiento máximo obtenido ronda los 2 MB/s, muy por debajo del rendimiento que ofrece la CPU. Este algoritmo está sin duda limitado por 2 factores: (1) la necesidad de recorrer la secuencia tantas veces como frecuencias sean asignadas a cada thread y (2) la repetida acumulación sobre contadores de frecuencia residentes en la tabla de memoria global.

En la segunda versión se mapea la secuencia de ADN sobre la memoria de texturas, para así provechar su menor tiempo de acceso y el hecho de que está cacheada, con una estructura unidimensional. Los resultados son similares a la primera aproximación y por tanto sin mejoras significativas, como cabía esperar del uso de las propiedades del espacio de memoria de texturas, tales como difusión y cache.

En la tercera versión nos hemos inspirado en el ejemplo "Histogram256" del SDK de CUDA que se detalla en [10]. La idea principal de este algoritmo es posibilitar la actualización de la misma posición de memoria compartida por distintos threads, en lo que en [10] se denomina *simulating atomic updates in software*.

De este modo, para resolver las posibles escrituras concurrentes en la misma posición de memoria compartida, se aprovecha la arquitectura SIMD por *warp*. Se utiliza una etiqueta identificativa del thread "escritor", constituida por los 5 bits de mayor peso de los datos a escribir, lo que posibilita la comprobación de la efectividad de la escritura, asignando una tabla de frecuencias por *warp* en la memoria compartida de cada bloque. Así, los accesos a la memoria compartida se dividen entre los distintos *warps* de cada bloque.

Cada thread se encarga de analizar una parte de la secuencia de ADN: cada thread recorre una parte de la secuencia tantas veces como sea necesario dividir la tabla de frecuencias para que quepa en la memoria del warp. Al finalizar cada iteración, se hace una reducción por cada bloque, sumando los valores de las distintas tablas (una por warp), y el resultado se almacena en la memoria global de la GPU. En la memoria global se mantiene una tabla de frecuencias por cada bloque, para evitar accesos concurrentes. Todos estos pasos, que se muestran de forma esquemática en la Figura 6, constituyen el primer kernel de esta aproximación. Al finalizar éste se hace una reducción en memoria global de las tablas de cada bloque para así recopilar las frecuencias de una misma palabra obtenidas por threads de diferentes bloques. Ello requiere la ejecución de un nuevo kernel.

Los resultados temporales obtenidos para esta aproximación son mucho mejores en comparación con las aproximaciones anteriores, aunque no se mejora el rendimiento obtenido únicamente con la CPU. En este caso, el rendimiento comienza a caer a partir de k=4. Esto es debido a que hasta ese valor de k la tabla de frecuencias cabe en la memoria compartida asignada a cada warp, minimizándose así el flujo de datos con memoria global.

La filosofía de la cuarta aproximación es similar a la anterior. En este caso, la secuencia de ADN a analizar se divide en 32 partes, una por cada uno de los thread que forman un warp y la tabla de frecuencias se divide entre los distintos warps de todos los bloques.

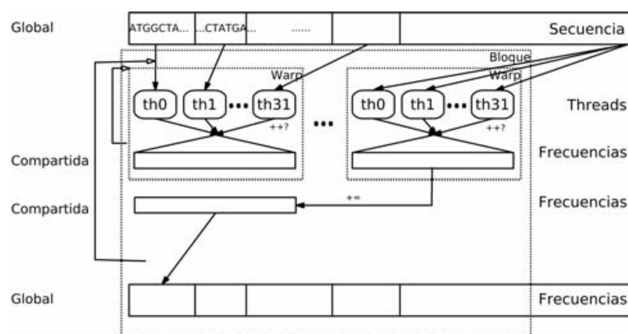

Figura 6. *Cálculo de la Tabla de frecuencia de palabras en la GPU según la tercera aproximación. Primer kernel de la tercera version.*

Al igual que en la versión anterior, se utilizan *tags* para comprobar las escrituras en memoria compartida. De este modo se aborda el problema de las escrituras concurrentes en memoria compartida mediante otra aproximación. En este caso no se necesita el kernel de reducción global ni replicación de la tabla de frecuencias en memoria global, ya que no existe posibilidad de escrituras concurrentes en este espacio de memoria, debido a que la tabla de frecuencias se encuentra particionada entre los distintos warps en ejecución. Su esquema se muestra en la Figura 7.

Cada thread recorre su parte de la secuencia comprobando si las palabras encontradas están dentro del rango de frecuencias actual, actualizando el contador asociado en caso afirmativo, tras lo cual se copia la parte de la tabla de memoria compartida a la tabla principal residente en memoria global. Al finalizar el rango de frecuencias asignado, se pasa al siguiente, repitiéndose este proceso hasta haber tratado toda la tabla de frecuencias. Como se puede apreciar en las gráficas de las Figuras 8 y 9, los resultados obtenidos con esta aproximación son peores que en el caso anterior.

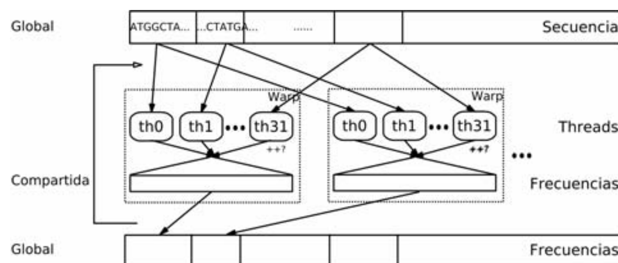

Figura 7*. Cálculo de las frecuencias de aparición en la GPU según la cuarta versión.*

Para la quinta aproximación realizamos un preprocesado del fichero de entrada en la CPU. Este preprocesado consiste en la obtención de las palabras de tamaño k de la secuencia de ADN, para ser posteriormente enviadas a la GPU en lugar de la secuencia de caracteres inicial. Para acelerar este proceso, y teniendo en cuenta que disponemos de un procesador con 2 cores, utilizamos OpenMP para obtener paralelismo en el procesador. Así, una vez las palabras de la secuencia de ADN preprocesada se encuentran en la GPU podemos evitar el acceso secuencial a memoria global que obligaba la secuencia original, posibilitando accesos "coalesced" o alineados a la memoria global.

Se sigue el mismo patrón que en la tercera aproximación, pero en este caso en una posición de 32 bits de memoria compartida se almacenan 2 contadores de frecuencias del modo que se indica en la Tabla 1. Al igual que en la versión de partida, tras cada iteración del algoritmo, en la que se recorre la parte de la secuencia asignada a cada thread y se hace una reducción por bloque para sumar los resultados de los distintos warps. Para evitar desbordamientos indeseados cada posición se acumula en un registro de 32 bits que se lleva a memoria global. En memoria global se almacenan las frecuencias en tablas con posiciones de 32 bits (al igual que en el caso de referencia, una tabla por bloque). En este caso se vuelve a usar de nuevo el kernel de reducción en memoria global de las tablas de cada bloque.

Campo	Bits
Etiqueta del thread:	5
No utilizado:	1
Contador de frecuencia:	13
Contador de frecuencia:	13

Tabla 1*. Contador doble de utilizado en la 5ª versión.*

Si se compara el rendimiento de esta versión con el de la versión tercera (Figuras 8 y 9), se observa que en esta nueva versión el rendimiento disminuye más lentamente con el aumento de k que en la versión anterior. Se trata de la mejor opción obtenida hasta ahora para tamaños de k suficientemente elevados. No obstante, el rendimiento

alcanzado no es significativamente superior al obtenido con la ejecución del código exclusivamente en la CPU, como se muestra en las Figuras 8 y 9.

Las principales dificultades a las que nos enfrentamos al abordar el análisis de frecuencia de aparición de oligonucleótidos son: en primer lugar el elevado movimiento de datos entre la memoria del Host y la GPU, así como entre las memorias global y compartida de la propia GPU y por otra parte, la poca carga computacional asociada al mismo. Las posibles vías de mejora que estamos abordando en estos momentos son las siguientes:

- Una versión más eficiente basada en las 2 primeras, en la que la tabla se divide entre los diferentes núcleos de proceso, tal y como se sugiere en [7].
- Almacenamiento disperso de las tablas de frecuencia en memoria compartida, tal y como se hace en [11] para matrices de co-ocurrencia de imágenes biomédicas de gran tamaño.

Aunque no se obtuvieran ventajas en la utilización de la GPU para acelerar la aplicación 1, pensamos que sería interesante su utilización en la segunda parte del análisis comparativo de 2 secuencias de ADN, en lo que hemos denominado aplicación 2, sobre la GPU, puesto que en este caso existe un mayor número de cálculos, aunque sigue siendo un problema de tipo "streaming" y por lo tanto con una baja reutilización de los mismos.

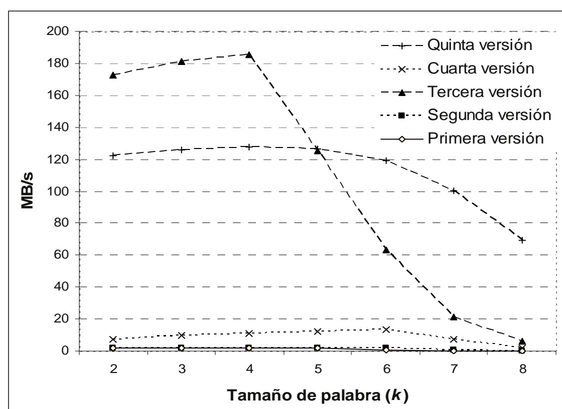

Figura 8. *Velocidad de procesamiento del análisis de frecuencias del cromosoma CH1 sobre la GPU GTX.*

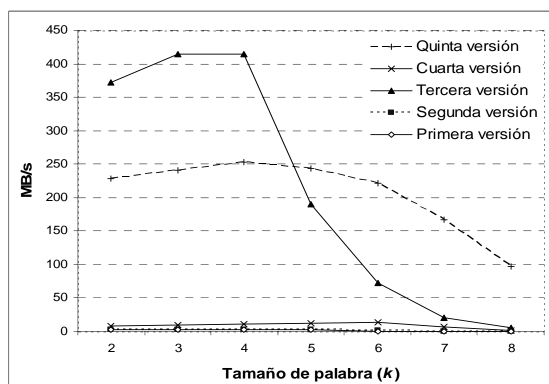

Figura 9. *Velocidad de procesamiento del análisis de frecuencias del cromosoma CH1 sobre la GPU Tesla.*

6. Conclusiones

En este te trabajo se ha presentado el estado actual del trabajo desarrollado para la aceleración del análisis de secuencias de ADN mediante el uso de GPUs. Así, se han planteado las diversas opciones de optimización y paralelización de este problema y se han descrito las diversas aproximaciones implementadas hasta el momento sobre este tipo de arquitecturas.

Los resultados obtenidos muestran una aceleración de los tiempos de análisis respecto de los resultados obtenidos con una sola CPU, pero hay una caída de las prestaciones cuando el tamaño de palabra utilizado para el análisis de frecuencias supera el valor k= 6. Se han propuesto varias líneas de continuación con el fin de hacer más eficiente el análisis de secuencias para palabras de mayor tamaño.

Agradecimientos

Este trabajo ha sido financiado por el programa europeo FEDER y el proyecto MEC ``Consolider Ingenio-2010 CSD 2006-00046", ``TIN 2006-15516-C04-02" y el proyecto nacional de biotecnología BIO2008-05067.

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] M. Gallach, V. Arnau, I. 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 Computation, NIC Series Vol. 33: Parallel Computing Current & Future Issues of High-End Computing, 2006, pp.: 623-630.

[8] A. Sanjuan, V. Arnau, J. M. Claver, Análisis paralelo de secuencias de ADN sobre computadores con múltiples cores, XIX Jornadas de Paralelismo, 2008.

[9] NVIDIA CUDA Compute Unified Device Architecture Programming Guide, http://www.NVIDIA.com/object/ cuda_develop.html. Última visita, 27 de septiembre de 2008.

[10] R. Shams and R. A. Kennedy, Efficient Histogram Algorithms for NVIDIA CUDA Compatible Devices, International Conference on Signal Processing and Communication Systems. ICSPCS' 2007. Australia , Gold Coast, 17-19 December 2007.

[11] A. Ruiz, M. Ujaldón, F. D. Igual, R. Mayo, BIPGPU: Una biblioteca para el procesamiento de imágenes biomédicas optimizada sobre procesadores gráficos, XIX Jornadas de paralelismo, 2008.