Aceleración del Algoritmo Wavefront Aligner para Emparejamiento de Secuencias Genéticas

Ernesto Hernández Chinappi

Resumen– El alineamiento de secuencias se define como la búsqueda de las diferencias que existen entre dos cadenas de caracteres: patrón y texto. Este trabajo utiliza técnicas de Ingeniería de Rendimiento para caracterizar, optimizar y paralelizar el algoritmo Wavefront Aligner de alineamiento de secuencias. Dicho algoritmo mejora el caso medio comparándolo con otros como el Smith-Waterman. Después de hacer una comparativa de la ejecución del algoritmo en procesadores Intel y ARM, se han desarrollado versiones con paralelización en CPU a distintos niveles de granularidad, obteniendo hasta un 7.5x de speedup. Además, se ha desarrollado una versión en GPU que es capaz de calcular la distancia de edición 13 veces más rápido utilizando secuencias de 1000 bases con una tasa de error de hasta el 10 %. De la misma forma, aplicando el algoritmo a secuencias con una tasa de error del 60 %, se ha logrado obtener un speedup de 69x.
Palabras clave– Alineamiento de secuencias, Distancia de edición, Wavefront Aligner, GPU, Ingeniería de Rendimiento, Computación de Altas Prestaciones, Bioinformática.

Abstract– Sequence alignment consists in searching for differences between two strings: pattern and text. The present work employs Performance Engineering techniques to characterize, optimize, and parallelize the Wavefront Alignment algorithm. This algorithm improves on the average case compared with others such as the Smith-Waterman. First, we performed a comparison of the execution of the algorithm Intel and ARM processors. Then, we developed a CPU parallel version, exploiting different levels of granularity, obtaining speedups of 7.5x. Finally, we developed a GPU version that is able to calculate the edit distance 13x faster on sequences of 1000 bases allowing up to 10 % of error. In the same way, using the algorithm on sequences up to 60 % of error we achieve speedups of 69x.
Keywords– Sequence alignment, Edit distance, Wavefront Alignment, GPU, Performance Engineering, High Performance Computing, Bioinformatics.

1 Introducción

La secuenciación del Ácido Desoxirribonucleico (ADN), consiste en determinar el orden de los cuatro componentes básicos químicos, llamados ”bases”, que forman la molécula de ADN de un individuo. Con la comparación de secuencias de ADN se puede generar una enorme y valiosa cantidad de información sobre la herencia en la propensión de enfermedades y las respuestas medioambientales a las mismas, contribuyendo al diagnóstico y al tratamiento [8].

La identificación de subsecuencias máximamente homólogas entre conjuntos de secuencias largas es un problema importante en el análisis de secuencias moleculares [14]. Existe una gran variedad de algoritmos que resuelven este problema con diversos costes computacionales. Dos ejemplos de estos algoritmos son Needleman-Wunsch [7] y Smith-Waterman [13]

Pese a que ambos están basados en programación dinámica, el algoritmo Needleman-Wunsch se utiliza para realizar alineamiento global entre dos secuencias, sobre todo, cuando ambas son de longitud similar y presentan una alta similaridad [7].

En el alineamiento local sólo se alinean las partes más parecidas de las secuencias, de manera que favorece encontrar patrones similares dentro de la secuencia. Un alineamiento local es la combinación de muchos globales de distintos patrones [9]. El alineamiento local resulta adecuado cuando se comparan secuencias sustancialmente diferentes, que difieren significativamente en longitud y contienen pequeñas subsecuencias similares [7].

En términos de alineamiento de secuencias, las que se quieren analizar toman nombres específicos; *query* repre-

• E-mail de contacto: ernestohernandez11@gmail.com
• Mención realizada: Ingeniería de Computadores
• Trabajo tutorizado por: Santiago Marco Sola, Juan Carlos Moure López (Departamento de Arquitectura de Computadores y Sistemas Operativos)
• Curso 2019/20

senta la secuencia seleccionada que debe encontrarse en la base de datos, mientras que *database* determina la base de datos que se comparará utilizando el algoritmo de emparejamiento con la secuencia de query.

Históricamente, el algoritmo más utilizado para el alineamiento local de secuencias ha sido el Smith-Waterman [4], sin embargo, la implementación del mismo tiene un tiempo de ejecución proporcional a $O(mn)$ donde m y n son las longitudes de las secuencias a alinear [11]. Aunque esta complejidad puede parecer asequible, el crecimiento en el número de bases de datos de secuencias genéticas ha sido exponencial la última década. Esto significa que cada vez se necesitan mejores algoritmos que resuelvan el mismo problema en un tiempo menor y de manera más eficiente.

Un algoritmo más eficiente que el Smith-Waterman es el Wavefront-Aligner [1] objeto de estudio de este trabajo de investigación. El citado algoritmo está ideado para mejorar el caso medio, realizando una alineación por banda diagonal y analizando únicamente espacios de la matriz donde es posible que se encuentre la solución óptima. [1]

A lo largo de este proyecto se ha estudiado el rendimiento del algoritmo Wavefront-Aligner y la posibilidad de implementar una versión que mejore los tiempos obtenidos utilizando unidades de procesamiento gráfico (GPU).

2 Objetivos

El objetivo principal de este trabajo de investigación es desarrollar una versión del algoritmo Wavefront-Aligner que se ejecute en una unidad de procesamiento gráfico (GPU), mejorando así el tiempo de ejecución con respecto a la versión CPU base del algoritmo. Este propósito engloba una serie de objetivos específicos para los que se establece la siguiente planificación progresiva:

1. Recopilar fuentes de información y artículos referentes al tema a tratar.

2. Seleccionar dos bases de datos con las que trabajarán los distintos algoritmos.

3. Realizar un estudio sobre los algoritmos Smith-Waterman y Wavefront Aligner. Analizar el rendimiento de ambos, utilizando las bases de datos previamente definidas con la ayuda de herramientas y métricas de ingeniería de rendimiento.

4. Realizar una caracterización completa de los algoritmos: comparar los resultados obtenidos en arquitecturas x86-64 Skylake con los que se obtengan de una ejecución en una máquina con arquitectura ARM, así como argumentar y justificar las diferencias obtenidas.

5. Implementar y validar una versión del algoritmo Wavefront Aligner ejecutado en GPU.

3 Estado del arte

Actualmente, los algoritmos para comparar secuencias genéticas más utilizados por los investigadores son aquellos que realizan una alineación de una molécula de ADN contra todo un gran conjunto de moléculas en una base de datos. Utilizan un algoritmo heurístico aproximado que es aproximadamente dos órdenes de magnitud más rápido que el algoritmo exacto de Smith-Waterman [6]. No obstante, algoritmos exactos como el Smith-Waterman se utilizan ampliamente en muchas aplicaciones bioinformáticas, ya sea como la última etapa de búsqueda de similitud de secuencia realizada con algoritmos aproximados, o dentro de algoritmos más complejos.

En los últimos años, los métodos para mejorar el rendimiento de las CPU han agotado su potencial esperado. Esto ha obligado a muchos investigadores a implementar versiones de sus algoritmos para otros dispositivos de aceleración como GPUs. Hace unos años, era común implementar algoritmos paralelos utilizando la biblioteca gráfica proporcionada por las GPU de NVIDIA. Sin embargo, tras la introducción del entorno CUDA para la programación de GPUs, el desarrollo de este tipo de algoritmos paralelos en GPU se ha simplificado y extendido a muchos ámbitos científicos e industriales [6].

El algoritmo Wavefront Aligner se basa en explotar las similaridades entre texto y patrón logrando una mejora de tiempo en el caso medio. Este algoritmo no ha sido tan investigado como el Smith-Waterman y eso plantea la oportunidad de encontrar nuevas aproximaciones GPU que mejoren su rendimiento. No obstante, en el trabajo de investigación 'Aceleración de Algoritmos de Emparejamiento de Secuencias Genéticas [5]' realizado en el año 2018, el autor logra una aceleración de 8.5 veces, con una paralelización en CPU de grano grueso. La aceleración en este caso se encontró limitada por el número de cores físicos con los que contaba el procesador.

El presente trabajo constituye una contribución a la optimización de algoritmos que comparan secuencias genéticas.

4 Metodología

La metodología de este proyecto, en primera instancia, consistirá en resumir la información necesaria sobre del algoritmo para así entender el trabajo del procesador y cuáles son las tareas más importantes a analizar y optimizar. Posteriormente, se empleará una metodología incremental. Se llevarán a cabo reuniones semanales; en las mismas se evaluarán y discutirán los resultados obtenidos de las tareas semanales y se establecerán nuevos objetivos y tareas. Fundamentalmente, se utilizarán técnicas, métricas y herramientas de análisis de rendimiento computacional, así como de análisis de algoritmos.

4.1. Bases de datos

Para que el algoritmo Wavefront Aligner pueda ser explotado en escenarios reales, se han utilizado bases de datos representativas de casos de usos en producción. Para su generación, se ha empleando un programa generador de secuencias resultando en las siguientes bases de datos:

1. 1000000 cadenas de 100 caracteres de longitud y una tasa de error del 10 %

2. 100000 cadenas de 1000 caracteres de longitud y una tasa de error del 10 %

El programa introduce errores aleatorios y uniformemente distribuidos en una secuencia genética de longitud previamente fijada. Crea aleatoriamente n secuencias genéticas de longitud l y para cada una genera una secuencia a comparar con un error predefinido e.

Es importante tener dos referencias a la hora de analizar el comportamiento de este algoritmo para poder evaluar el desempeño del mismo frente a un crecimiento en el tamaño del problema.

4.2. Smith-Waterman

El algoritmo Smith-Waterman es un método para caracterizar con precisión la alineación óptima de dos secuencias. Cuenta con opciones adicionales para comenzar y terminar en la posición de la secuencia más larga. [10].

En la revisión bibliográfica sobre el algoritmo de alineación de secuencias Smith-Waterman, resulta interesante tomar en cuenta el artículo [13] de sus creadores. En él, los autores realizan una explicación detallada del algoritmo en cuestión:

Dadas dos secuencias moleculares A y B con longitudes n y m respectivamente:

1. Se construye una matriz F de $(n+1)*(m+1)$ rellenando la primera fila y la primera columna con ceros, tal que se cumpla que: $F_{0,j} = F_{i,0} = 0$ para todo i comprendido entre 0 y n y para todo j entre 0 y m.

2. Se calculan todas las celdas de la matriz, a través de la siguiente fórmula:

$$F(i,j) = \text{máx} \left\{ \begin{array}{l} 0 \\ F_{i-1,j} - d, \\ F_{i,j-1} - d, \\ F_{i-1,j-1} + s(x_i, y_j) \end{array} \right\}.$$

3. Traceback: El traceback comienza en la puntuación más alta en la matriz de puntuación F y termina en una celda de matriz que tiene una puntuación de 0. Para obtener la alineación local óptima, se comienza con el valor más alto en la matriz (i, j), luego, retrocedemos a una de las posiciones (i - 1, j), (i, j - 1) y (i - 1, j - 1) dependiendo de la dirección del movimiento utilizado para construir la matriz. Mantenemos el proceso hasta que alcanzamos una celda de matriz con valor cero o el valor en posición (0, 0).

Las dependencias para el procesamiento de una celda en la matriz de puntuación son las reflejadas en la figura 1. Si se analiza las dependencias del algoritmo, se puede encontrar una solución que ejecute instrucciones vectoriales empleando un cambio en la forma de procesar los datos. No existe dependencia entre datos de una misma antidiagonal, de manera que estas celdas se pueden procesar en paralelo. Se pueden utilizar instrucciones SIMD (Single Instruction Multiple Data) para procesar varias posiciones de la antidiagonal simultáneamente. Esta estructura de datos se muestra en la figura 2.

4.3. Wavefront Aligner

El Wavefront Aligner es un algoritmo que mejora el caso medio del alineamiento local de secuencias valiéndose

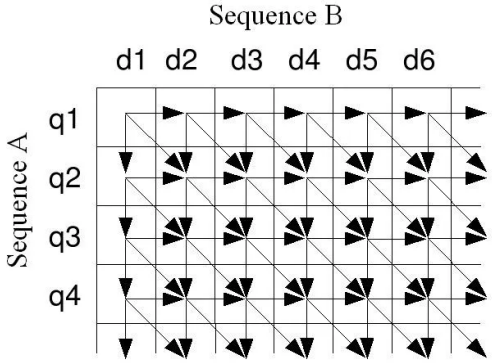

Fig. 1: Dependencias Smith-Waterman

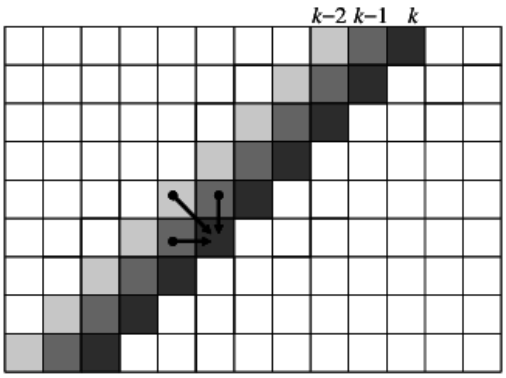

Fig. 2: Smith-Waterman por antidiagonales

de las similitudes entre las secuencias a alinear. Comienza computando celdas de la diagonal principal (K_0) de la matriz de puntuación mientras no se encuentre dos caracteres diferentes [1]. Este proceso de calcular caracteres coincidentes se conoce como extensión.

Al encontrarse una casilla para la cual los caracteres correspondientes son distintos entre sí, el algoritmo incrementa una unidad la distancia de edición y amplía la cantidad de diagonales efectivas. [1]. El proceso de ampliación significa incluir dos diagonales adicionales para ser computadas en la siguiente iteración del algoritmo ($k-1$ y $k+1$). Este proceso se repite hasta que una diagonal alcanza el final de la tabla (esquina inferior derecha). De este modo, la distancia de edición mínima corresponde al número de veces que la banda diagonal ha sido ampliada, es decir, al número total de iteraciones del algoritmo.

En el algoritmo Wavefront Aligner se pueden diferenciar dos partes:

- Extender: Este proceso corresponde al cálculo de los offsets de cada diagonal, es decir, de los elementos coincidentes entre el patrón y el texto desde la posición de cada diagonal. En el ejemplo presentado en la figura 3 se puede observar que como las dos letras de ambas secuencias son coincidentes, el offset almacenado para la primera diagonal será dos, debido a que el primer error se encuentra en la posición número 3.

- Ampliación (compute): Usando las diagonales actuales, se calcula el máximo offset entre una eliminación, una sustitución y una inserción. Además, se incrementa en dos el número de diagonales a ser computadas

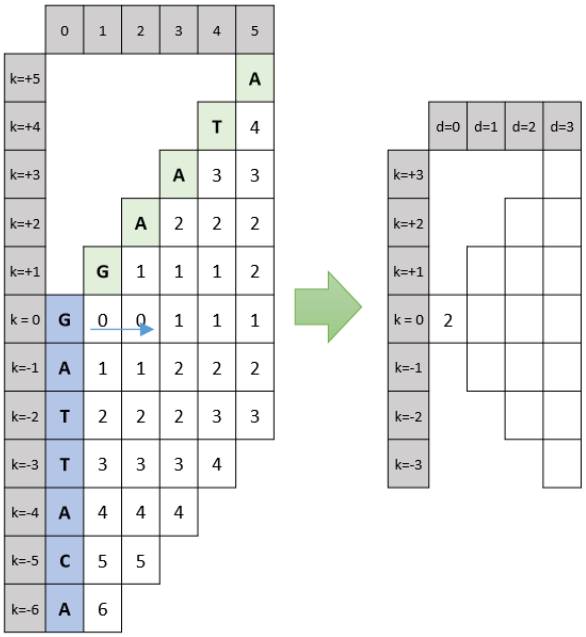

Fig. 3: Proceso de extensión de diagonal

en la siguiente iteración. En el ejemplo de la figura 4 se puede observar que se añaden dos diagonales cuyos offsets representan el cálculo máximo entre una eliminación, una sustitución y una inserción.

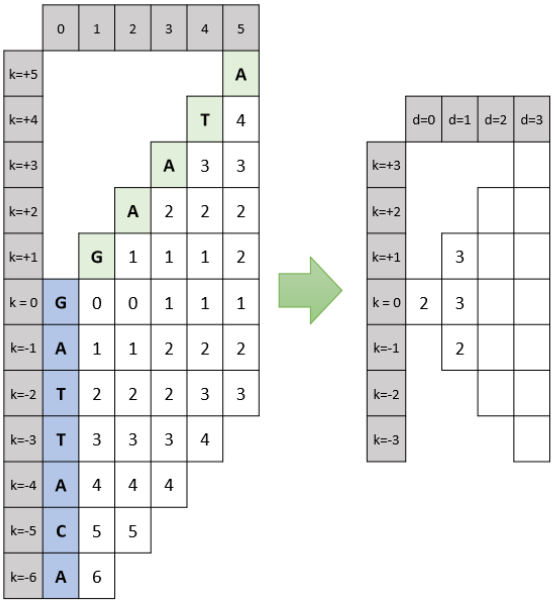

Fig. 4: Proceso de ampliación de diagonales

4.4. X86-64 vs ARM

Para caracterizar un algoritmo, es importante comparar su rendimiento en distintas arquitecturas. Para ello se seleccionaron dos procesadores: un ARM y un Intel Core i9-9900K con arquitectura Skylake.

La diferencia entre estas arquitecturas es principalmente su ISA y filosofía de diseño. Mientras que el x86 de Intel está desarrollado a partir de una ISA más amplia, un procesador ARM está basado en RISC (Reduced Instruction Set Computer) cuya filosofía de diseño de CPU se basa en un repertorio pequeño de instrucciones. [3]. Otra diferencia significativa e importante radica en que los procesadores ARM son diseñados para un consumo menor de energía sin mucha pérdida en capacidad de procesamiento. El procesador de Intel posee una potencia de diseño térmico de 95 W y alcanza una frecuencia de 5.00 GHz. Mientras que el procesador ARM sólo llega a 1.2 GHz de frecuencia disipando como máximo 3W.

En definitiva, Intel apuesta por una arquitetura más compleja en el procesador para un mejor rendimiento y ARM por una menos compleja para una alta eficiencia energética.

4.5. Paralelización en CPU

En informática, el tamaño (o grano) de una tarea es una medida de la cantidad de trabajo (o computación) que se realiza en ella [2]. Este concepto tiene en cuenta la sobrecarga de comunicación entre múltiples procesadores y define el tamaño de grano como la relación entre el tiempo de cálculo y el tiempo de comunicación [5].

Según la frecuencia con la que las subtareas se sincronizan o comunican entre sí y la cantidad de trabajo realizado se definen dos tipos de paralelismo: de grano fino y de grano grueso.

4.5.1. Grano fino

Existen dos funciones principales que consumen la mayoría de los ciclos de ejecución del algoritmo, estas son las encargadas de ampliar los offsets del wavefront y la que se encarga de extender el mismo. Dichas funciones ocupan el 55 % de la ejecución total del algoritmo en ambas arquitecturas. Sin embargo, antes de paralelizar, es necesario conocer cuantas instrucciones ejecutan los bucles más internos de estas funciones. Para una base de datos con secuencias de 100 caracteres y una probabilidad de error del 10 % se obtiene que cada vez que se llama a la función de computar se ejecutan 34,6 iteraciones en el bucle correspondiente al núcleo de la función. Este dato representa un promedio total, no obstante, esta cifra de iteraciones va aumentando con cada llamada a la función. En promedio, por cada alineamiento, dicha función se ejecuta 235,5 veces.

Paralelizar con OpenMP significaría, en una sola alineación, crear 235 threads que se repartirán 35 iteraciones (en promedio). Eso supondría introducir una sobrecarga en la ejecución del algoritmo. De igual forma, en la función que se encarga de extender las diagonales, el paralelismo se basa en repartir a cada hilo de ejecución una diagonal y .[ex]tender"hasta que se produzca un error. La cantidad de iteraciones totales del bucle dependerá de los errores entre secuencias. Nuevamente se generaría una gran cantidad de mensajes de sincronización para repartirse; en el mejor de los casos, el máximo número de diagonales.

4.5.2. Grano grueso

Un paralelismo de grano grueso, en el cual el programa se divide en tareas grandes, podría ofrecer una mejora en el tiempo de ejecución. Esto se debe a la reducción en el tiempo de comunicación de datos entre procesos (comparado con la versión de grano fino). Aprovechando la independencia de datos entre alineamientos, se puede definir un

paralelismo en el que los threads sólo se tengan que comunicar (sincronizar) para repartirse la cantidad de alineamientos que realizará cada uno. Pese a que es una buena opción para obtener mejor rendimiento con respecto a la versión base, este tipo de paralelismo se añade únicamente para completar el estudio y la caracterización del algoritmo Wavefront-Aligner. Este modelo no logra explotar el paralelismo en el programa, ya que la mayor parte del cálculo se realiza secuencialmente en un procesador, por lo que la versión optimizada estará limitada por el número de núcleos físicos de cómputo que tenga el mismo. La ventaja principal de este tipo de paralelismo es la baja sobrecarga de comunicación y sincronización que perjudica la versión en grano fino.

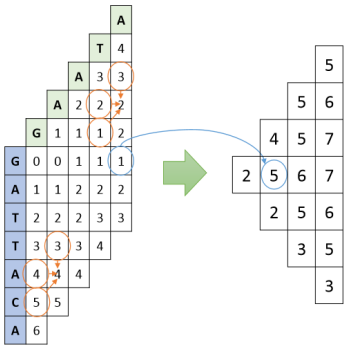

Fig. 5: Estructura de offsets utilizada en el algoritmo Wavefront Aligner

4.6. Paralelización en GPU

Una unidad de procesamiento de gráficos (GPU por sus siglas en inglés) es un circuito electrónico diseñado para manipular rápidamente gráficos y para acelerar la creación de imágenes que posteriormente se proyectarán en un dispositivo de visualización. Su estructura altamente paralela los hace más eficientes que las unidades de procesamiento central de propósito general (CPU). Sin embargo, para poder ejecutar una aplicación en la GPU, hace falta hacer cambios en el código.

CUDA (Compute Unified Device Architecture) es un framework de computación paralela y un modelo de interfaz de programación de aplicaciones (API) creado por Nvidia. Permite a los desarrolladores de software utilizar una unidad de procesamiento de gráficos (GPU), compatible con CUDA, para el procesamiento de propósito general; un enfoque denominado GPGPU (computación de propósito general en unidades de procesamiento de gráficos).

No obstante, es deseable que se minimice la cantidad de veces que hay intercambio de información entre la memoria de la GPU y la CPU, ya que es una operación costosa en tiempo. Esto se puede lograr garantizando que todo el código referente a la alineación de secuencias se ejecute en la GPU; inclusive, aquel código totalmente secuencial y así la GPU cuente con todos los datos necesarios en su memoria antes del proceso de alineación. Para ello, se plantea un algoritmo en el que la GPU recibe como datos la cantidad total de secuencias a alinear y devuelve la distancia de edición como resultado.

Para realizar el cálculo de la distancia entre ambas secuencias en GPU, son necesarios dos vectores de offsets, ambos de longitudes igual a la suma del número de caracteres del patrón y el texto a alinear. Estos vectores estarán ubicados en la memoria compartida de la GPU, de manera que serán compartidos por todos los threads dentro de un mismo bloque. Este buffer de offsets representa de manera lineal y con tamaño estático la estructura de offsets representada en la figura 5.

Continuando con el ejemplo en el que se alinean las secuencias ”GATTACA”, y ”GAATA”, el estado del doble buffer de offsets será el reflejado en la figura 6.

Después de realizar la ampliación de las diagonales, el estado de los buffers es el reflejado por la figura 7. En este momento de la ejecución, la cantidad total de threads en la GPU que interfieren en el alineamiento es de 3.

En este momento, se realiza una operación de intercambio entre los buffers, en el que offsets pasa a contener los valores de next_offsets y next_offsets los valores de offsets. Finalmente se comienza de nuevo el proceso hasta que la condición de salida se cumple, momento en el que todos los threads del bloque acaban la ejecución y la distancia de edición es transladada de la memoria de la GPU a la memoria de la CPU.

Offsets						
-3	-2	-1	0	1	2	3
0	0	0	2	0	0	0
next_offsets						
-3	-2	-1	0	1	2	3
0	0	0	0	0	0	0

Fig. 6: Estado de los buffers de offsets luego de la primera extensión.

Cada diagonal puede ser extendida y ampliada de manera independiente y no existen dependencias entre diagonales, por lo que un thread puede encargarse de ampliar y extender una diagonal. De esta manera, mientras más diagonales efectivas se estén utilizando, mayor cantidad de threads estarán siendo utilizados. Los threads dentro de un bloque que no se correspondan a ninguna diagonal, estarán esperando a que se amplíe la cantidad de diagonales efectivas y sean necesitados.

Cada bloque de la GPU se encarga de realizar un alineamiento entre dos secuencias. Por lo que contará con tantos threads como diagonales máximas tenga el wavefront. Las secuencias a ser alineadas por los distintos bloques se encuentran de manera contigua en la memoria global de la GPU, por lo que es necesario que cada bloque acceda a la secuencia que le corresponde. Es por esto que se utiliza un vector de índices donde cada celda indica la posición de inicio de una secuencia dentro del total de secuencias. Por ejemplo, el bloque 0 accede a la posición índices[0] y obtiene como resultado la posición de la secuencia que debe alinear.

La figura 8 representa un ejemplo con tres patrones y tres bloques donde se ha rellenado el vector de índices para dichas cadenas.

Offsets						
-3	-2	-1	0	1	2	3
0	0	0	2	0	0	0
next_offsets						
-3	-2	-1	0	1	2	3
0	0	2	3	3	0	0

Fig. 7: Estado de los buffers de offsets luego de la primera ampliación.

Posición	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Patterns	A	G	A	T	A	T	A	G	A	T	A	T	A	A	G	G	A	T	G	A
Índices	0	7	14																	

Fig. 8: Vector de índices de secuencias en la memoria de la GPU.

5 Resultados

5.1. X86-64 vs ARM

Tras realizar una comparación entre ejecuciones del algoritmo Wavefront Aligner en ambas arquitecturas, se puede observar que el rendimiento en el procesador Intel con arquitectura Skylake es mejor. Esto se puede observar en la tabla 1 en la cual, pese a que el IPC es similar, el tiempo de ejecución en el procesador Intel es 4 veces menor. Estas pruebas están realizadas con alineamientos para cadenas con 100 caracteres de longitud.

El compilador utilizado para realizar estas pruebas ha sido GCC, utilizando la opción -O3, que habilita muchas de las funciones avanzadas del compilador tal como la autovectorización.

Si bien es cierto que el procesador Intel realiza el cómputo en menos tiempo, se puede evidenciar que el IPC es muy parecido en ambas arquitecturas y que la diferencia en tiempo de ejecución es proporcional a la diferencia de frecuencias entre ambos procesadores.

Procesador	**Intel Core i9-9900K**	**ARM**
Tiempo (segundos)	8,40	35,78
Instrucciones (Gi)	127,00	134,00
IPC	3,20	3,02
Frecuencia (GHz)	4,99	1,19

TAULA 1: Comparación entre arquitecturas de procesadores (Base de datos 1)

Para cadenas con 10 veces más caracteres, se puede observar en la tabla 2 que pese a que la máquina Intel ejecuta más instrucciones con un peor IPC, obtiene mejor tiempo de ejecución como resultado de tener una frecuencia máxima 4,19 veces superior al ARM.

Según estos resultados, en el que el algoritmo en arquitecturas distintas presenta un IPC tan similar, cabe la posibilidad de que sea más económicamente rentable el hecho de tener varios procesadores ARM independientes realizando diferentes alineamientos de secuencias y de esta manera aumentar la cantidad de secuencias alineadas por unidad de tiempo, pese a tener un sistema cuya latencia sea menor que utilizando un procesador Intel, ahorrando una gran cantidad de energía.

Procesador	**Intel Core i9-9900K**	**ARM**
Tiempo (segundos)	37,89	137,06
Instrucciones (Gi)	689,00	615,00
IPC	3,63	3,80
Frecuencia (GHz)	4,99	1,19

TAULA 2: Comparación entre arquitecturas de procesadores (Base de datos 2)

5.2. Paralelización en CPU

En la paralelización en CPU, se utiliza la interfaz de aplicaciones OpenMP que representa un estándar para la programación paralela en multiprocesadores de memoria compartida [12]. OpenMP resulta interesante debido a su modelo de programación paralela portable y fácil de usar. Lejos de ser un lenguaje de programación nuevo, extiende otros como C/C++ y Fortran.

Utilizar OpenMP para analizar una posible paralelización a nivel de threads facilita el desarrollo de la misma, ya que sin modificar el código, se utiliza un conjunto de directivas de compilador y rutinas de biblioteca para modificar la ejecución del algoritmo.

Con el objetivo de obtener los tiempos de ejecución y distintos datos de interés en ambas optimizaciones (grano fino y grueso) se utiliza el procesador Intel Core i-9900K Skylake.

5.2.1. Grano fino

En esta versión paralela del algoritmo, al principio de cada bucle principal los threads son sincronizados para repartirse la carga de trabajo que ejecutará cada uno. No obstante, el tiempo para ejecutar las instrucciones de sincronización es mayor a medida que se van agregando más threads. Como se puede observar en la figura **??**, el tiempo de ejecución aumenta al aumentar el número de threads añadidos.

Las funciones de sincronización y creación de threads, que representan el 50 % de la ejecución con 2 threads, añaden instrucciones responsables del aumento de tiempo proporcional a el número de threads añadidos (véase la relación que existe entre el tiempo de ejecución y las instrucciones en la figura 9).

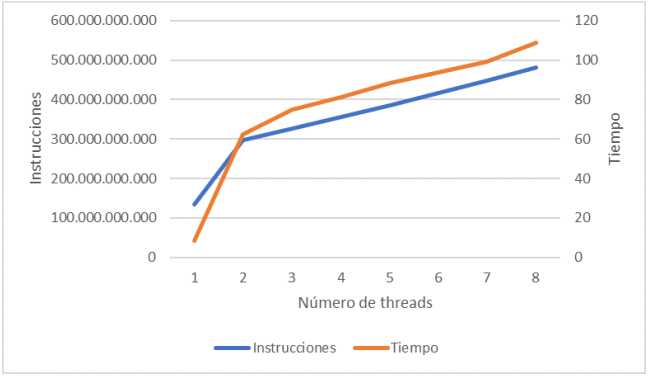

Fig. 9: Incremento de instrucciones y tiempo de ejecución en función del número de threads

5.2.2. Grano grueso

En esta estrategia de paralelización se reparten los alineamientos a realizar entre los distintos threads y cada uno realiza una carga de trabajo de manera individual. De esta forma se maximiza la cantidad de operaciones de cómputo frente a las operaciones de sincronización del thread. Si el 100 % del código fuera paralelizable estaríamos ante una situación ideal, donde se obtendría un speedup equivalente al número de núcleos de cómputo utilizados. Por el contrario se obtiene una mejora proporcional al número de threads (hasta 8 cores físicos que dispone el procesador) como se puede ver en la gráfica 10.

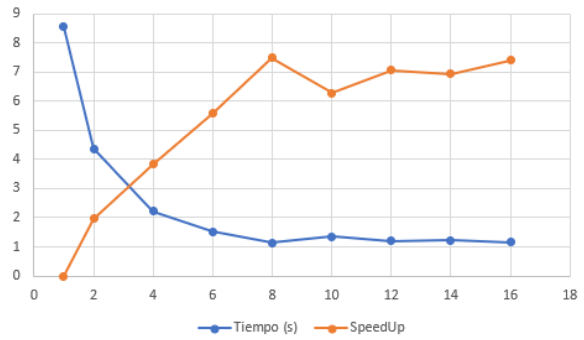

Fig. 10: Tiempo (azul) y speedup (naranja) en función del número de threads.

En oposición a la paralelización de grano fino no existe una gran cantidad de instrucciones para sincronización. De hecho, aumentando el número de threads se obtiene para todos los casos un total de instrucciones similar al del caso base (134 Gi).

La eficiencia del sistema para un sistema con n procesadores se define como:

$$E(n) = \frac{S(n))}{n} \tag{1}$$

La eficiencia es una comparación del grado de speedup conseguido frente al valor máximo. Dado que $1 \leq S(n) \leq n$, tenemos $1/n \leq E(n) \leq 1$. En este caso, la eficiencia máxima se obtiene utilizando dos procesadores. Manteniéndose en valores próximos a uno hasta que el número de threads iguala el número de cores físicos, donde la eficiencia es del 94 %, véase la figura 11.

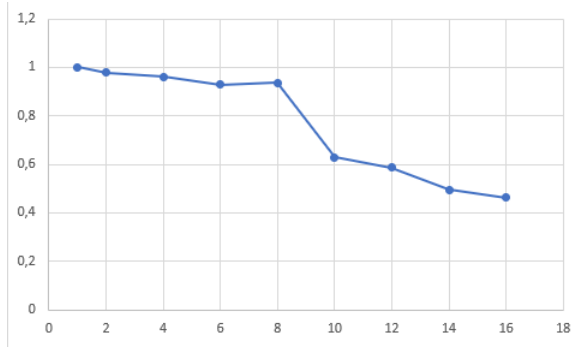

Fig. 11: Eficiencia en función del número de threads.

5.3. Paralelización en GPU

En la versión en GPU desarrollada, el alineamiento de una secuencia es una operación más lenta que en la CPU. Esto se debe a que se tiene que reservar memoria en la GPU, transferir los datos desde la memoria del host, lanzar el kernel y transferir el resultado desde la memoria de la GPU. Sin embargo, la capacidad de cómputo masiva que otorga la GPU facilita la posibilidad de realizar más de un alineamiento a la vez, gracias a la ejecución de varios bloques de threads alineando distintas secuencias de manera independiente entre sí.

Utilizando la primera base de datos de 100 caracteres, se obtiene como máximo un speedup 1.10x, con un tamaño de bloque de 32 threads, alineando 2000 pares de secuencias simultáneamente.

Bloq/Grid	Thr./Bloq	Tiempo GPU	Speed-Up
500	32	0,42	0,97
1000	32	0,40	1,02
2000	32	0,37	1,10
5000	32	0,37	1,10
500	64	0,42	0,96
1000	64	0,38	1,07
2000	64	0,35	1,15
5000	64	0,39	1,04

TAULA 3: EJECUCIÓN DE LA VERSIÓN GPU DEL ALGORITMO: CADENAS DE 100 CARACTERES.

Para la segunda bases de datos, con cadenas de 1000 caracteres y un 10 % de tasa de error, se han obteniendo un speedup de 6.17x utilizando 64 threads/bloque y 2000 bloques/grid. Es decir, 2000 alineaciones simultaneas. Esta base de datos en la versión secuencial en CPU tiene un tiempo de ejecución de 1.11 segundos.

Bloq/Grid	Th/Bloq	Tiempo GPU	Speed-Up
500	32	0,23	4,71
1000	32	0,18	5,82
2000	32	0,19	5,56
5000	32	1,20	0,91
500	64	0,20	5,36
1000	64	0,17	6,17
2000	64	0,19	5,61
5000	64	1,21	0,91

TAULA 4: EJECUCIÓN DE LA VERSIÓN GPU DEL ALGORITMO: CADENAS DE 1000 CARACTERES.

Utilizando una nueva base de datos consistente en 1000000 pares de secuencias de 1000 caracteres se realizan las pruebas correspondientes para obtener la variación del tiempo de ejecución con respecto a la tasa de error de las cadenas a comparar. Se utiliza los tamaños de grid y bloque con los que se obtuvo el mayor speedup en la tabla 5.

6 CONCLUSIONES

Después de realizar una revisión bibliográfica, que permitió obtener una perspectiva global de los algoritmos de

% error	Tiempo CPU	Tiempo GPU	speedup
10 %	11,02	0,86	12,87
20 %	31,68	0,88	36,19
30 %	58,53	1,36	42,92
40 %	87,84	1,50	58,45
50 %	116,00	1,88	61,72
60 %	145,47	2,11	68,84

TAULA 5: SPEEDUP EN FUNCIÓN DE LA TASA DE ERROR ENTRE SECUENCIAS.

alineamiento de secuencias, se logró establecer una planificación que, además de definir los casos que interesaban estudiar, permitió crear bases de datos realistas utilizadas a lo largo del trabajo de investigación.

Se utilizó un procesador Intel Core i9-9900K y ARMv9 para obtener datos de rendimiento. El procesador ARM, que no cuenta con el módulo NEON capaz de ejecutar instrucciones vectoriales, presenta mejor IPC y menos cantidad de instrucciones que en el procesador Intel. Sin embargo, el procesador ARM tiene una frecuencia de reloj 4 veces menor lo que hace que el alineamiento de las secuencias sea más lento que en el procesador Intel. Estos resultados, teniendo en cuenta que el procesador ARM consume menos energía, abre una posible opción en la que se podrían utilizar varios procesadores ARM procesando cadenas en paralelo y así aumentar el throughput de cadenas alineadas por segundo, realizando el mismo trabajo que en el procesador Intel pero ahorrando energía.

En los casos estudiados, el algoritmo Wavefront Aligner es más eficiente que el Smith-Waterman. De esta manera, todos los experimentos realizados fueron enfocados a acelerar el algoritmo Wavefront Aligner. La paralelización de grano fino resultó ser perjudicial en la ejecución del algoritmo y el tiempo de ejecución resulta proporcional al número de threads que se ejecutan en paralelo. Siguiendo con la paralelización en CPU, el paralelismo de grano grueso alcanza un speedup de 7.5x; encontrándose limitado por el número de cores físicos con que cuenta el procesador utilizado.

La versión en GPU implementa un algoritmo de doble buffer en el que se computa iterativamente la mínima distancia de edición entre dos secuencias. Debido a la gran capacidad de cómputo paralelo que es capaz de realizar, muchas cadenas pueden ser procesadas a la vez. La GPU debe comunicarse con la CPU para transferir los datos correspondientes a las cadenas que va alinear y la GPU devuelve a la CPU una lista con todas las distancias de edición computadas. Para cadenas de 1000 caracteres se ha logrado obtener obtiene un speedup de 12x.

Además del paralelismo entre secuencias independientes, cada diagonal es computada de manera paralela: tanto en la extensión como en la ampliación. De esta manera, mientras se computen más diagonales efectivas, habrá mayor paralelismo. De esta forma, para cadenas con una tasa de fallos del 60 %, el speedup obtenido es de 69x.

Como trabajo futuro, se desea implementar una versión en GPU en la que toda la matriz de programación dinámica esté alojada en la GPU para poder realizar el backtrace y obtener mayor información sobre el alineamiento entre secuencias.

REFERENCIAS

[1] Francisco Gutiérrez Martín, dir. (Universitat Autònoma de Barcelona. Departament d'Arquitectura de Computadors i Sistemes Operatius) Moure López, Juan Carlos, and Santiago Marco Sola. Aceleración de algoritmos de emparejamiento de secuencias genéticas. Master's thesis, 2018-07-11.

[2] Kai Hwang. *Advanced Computer Architecture: Parallelism,Scalability,Programmability*. McGraw-Hill Higher Education, 1st edition, 1992.

[3] Tariq Jamil. Risc versus cisc. *IEEE Potentials*, 14(3):13–16, 1995.

[4] Neil C Jones, Pavel A Pevzner, and Pavel Pevzner. *An introduction to bioinformatics algorithms*. MIT press, 2004.

[5] Jan Kwiatkowski. Evaluation of parallel programs by measurement of its granularity. In Roman Wyrzykowski, Jack Dongarra, Marcin Paprzycki, and Jerzy Waśniewski, editors, *Parallel Processing and Applied Mathematics*, pages 145–153, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[6] L. Ligowski and W. Rudnicki. An efficient implementation of smith waterman algorithm on gpu using cuda, for massively parallel scanning of sequence databases. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–8, May 2009.

[7] Vladimir Likic. The needleman-wunsch algorithm for sequence alignment. *Lecture given at the 7th Melbourne Bioinformatics Course, Bi021 Molecular Science and Biotechnology Institute, University of Melbourne*, pages 1–46, 2008.

[8] D. Pacheco Bautista, M. Gonzalez, and I. Algredo Badillo. De la Secuenciacion a la Aceleracion Hardware de los Programas de Alineacion de ADN, una Revision Integral. *Revista mexicana de ingenieria bio.*, 36:259 – 277, 12 2015.

[9] Jonathan Pevsner. Pairwise sequence alignment. *Bioinformatics and functional genomics, 2nd edition. Hoboken: John Wiley & Sons*, pages 47–97, 2009.

[10] Valery O Polyanovsky, Mikhail A Roytberg, and Vladimir G Tumanyan. Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences. *Algorithms for molecular biology*, 6(1):25, 2011.

[11] Torbjørn Rognes and Erling Seeberg. Six-fold speedup of smith–waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, 16(8):699–706, 2000.

[12] Mitsuhisa Sato. Openmp: parallel programming api for shared memory multiprocessors and on-chip multiprocessors. In *15th International Symposium on System Synthesis, 2002.*, pages 109–111. IEEE, 2002.

[13] Temple F Smith, Michael S Waterman, et al. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[14] Gianvito Urgese, Giulia Paciello, Andrea Acquaviva, Elisa Ficarra, Mariagrazia Graziano, and Maurizio Zamboni. Dynamic gap selector: A smith waterman sequence alignment algorithm with affine gap model optimization. In *IWBBIO*, pages 1347–1358, 2014.