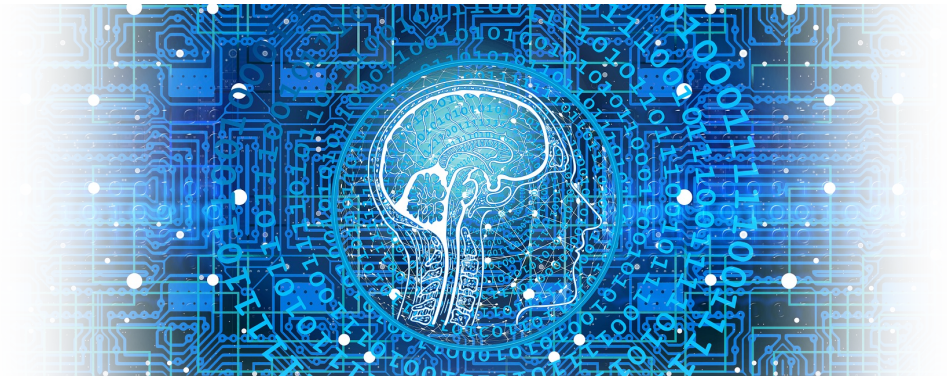

Trabajos de la asignatura

Programación Avanzada en Bionformática

Grado en Ingeniería de la Salud

Curso 2023-24

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

Prólogo

Este libro, editado por el Prof. Alberto G. Salguero Hidalgo, contiene los trabajos realizados por los alumnos de la asignatura Programación Avanzada en Bioinformática, del Grado en Ingeniería de la Salud de la Universidad de Málaga.

Los alumnos son plenos responsables de su contribución al libro y conservan todos los derechos de autoría del contenido de sus respectivos capítulos.

Índice general

Capítulo 1

Aproximaciones concurrentes al algoritmo de Smith-Waterman para el alineamiento de secuencias

Hugo Salas Calderón

1.1. Introducción

El problema original del alineamiento pareado consiste en encontrar la mejor forma de comparar y alinear dos secuencias biológicas (como ADN, ARN o proteínas), insertando eventualmente huecos —*gaps*, de modo que se maximice una función de puntuación, que refleja la similitud entre ellas.

Este problema fue formalizado por Needleman y Wunsch en 1970, quienes aplicaron por primera vez programación dinámica para garantizar la obtención de la solución óptima en el alineamiento global de dos secuencias completas Koerkamp (2025).

Su algoritmo construye una matriz de puntajes que evalúa inserciones, deleciones y sustituciones, y realiza un trazado de retroceso—*traceback* desde la celda final hasta el origen para reconstruir el alineamiento que minimiza el coste total de las ediciones Likic (n.d.).

Este procedimiento resuelve el desafío de explorar de forma exhaustiva un espacio exponencial de posibles alineamientos, reduciendo la complejidad a $\theta(mn)$ en tiempo y espacio, donde m y n son las longitudes de las dos secuencias Rosenberg (n.d.).

A principios de la década de 1980, Smith y Waterman refinaron este planteamiento para atender casos en los que solo interesan regiones de alta similitud dentro de secuencias más largas. En 1981 propusieron el algoritmo de alineamiento local, que introduce la regla de reiniciar a cero las celdas con puntación negativa, de modo que el *traceback* comienza en la celda de mayor valor y finaliza al llegar a un valor igual a 0, identificando así el fragmento óptimo de alineamiento.

Esta aproximación mantiene la complejidad espacial y temporal de $\theta(mn)$, lo que lo vuelve costoso para bases de datos de gran escala, aunque garantiza encontrar el alineamiento local óptimo de acuerdo con la matriz de sustitución y el esquema de penalización de gaps seleccionado.Wikipedia (n.d.)

En este proyecto se desarrollan tres aproximaciones al algoritmo Smith-Waterman:

una implementación secuencial, una versión concurrente a nivel de CPU, y una implementación paralela que aprovecha el potencial de la computación en GPU mediante CUDA.

El objetivo es evaluar la eficiencia y escalabilidad de cada aproximación, proporcionando una base experimental para determinar qué técnica resulta más adecuada en distintos contextos bioinformáticos.

1.2. Marco teórico

1.2.1. Alineamiento de secuencias

Una secuencia se puede representar como una sucesión finita ordenada –o cadena– de elementos sobre un alfabeto finito, tal que $A = A[1 \dots n]$ y $B = B[1 \dots m]$. de J. Pérez Jiménez (2014)

Un alineamiento óptimo consiste en dos secuencias extendidas A' y B' (ambas de longitud L) obtenidas al insertar *gaps* "–" para maximizar la puntuación total:

$$S(A', B') = \sum_{k=1}^{L} s\big(A'[k], B'[k]\big) + \sum_{\text{gaps}} g(\ell),$$

donde:

- $s(x, y)$ evalúa cada par de símbolos valores tomados de una matriz de similitud.
- $g(\ell)$ es la penalización que se aplica al introducir un *gap*.

Para resolver este problema se emplea programación dinámica, construyendo una matriz M de dimensión $(n+1) \times (m+1)$ según la recurrencia:

$$M[i,j] = \text{máx} \begin{cases} 0 & \\ M[i-1, j-1] + s\big(A[i], B[j]\big), & (match) \\ M[i-1, j] + g, & (gap\ en\ B) \\ M[i, j-1] + g, & (gap\ en\ A) \end{cases} \quad \text{para } 1 \le i \le m,\ 1 \le j \le n \tag{1.1}$$

Se plantea un primer caso máx$(0, \dots)$ para permitir reinicios parciales en la puntuación.

Existe una optimización: el algoritmo de Myers-Miller, basado en el algoritmo de Hirschberg, emplea una estrategia recursiva de didive y vencerás para computar alineamientos globales con una complejidad de $\theta(m+n)$ en espacio. Okada et al. (2015)

Matriz de similitud

El cálculo de la similitud entre secuencias biológicas se basa en el uso de matrices de sustitución, las cuales asignan una puntuación cuantitativa a cada posible emparejamiento entre caracteres (nucleótidos o aminoácidos). Estas matrices reflejan la probabilidad

biológica de que un carácter haya sido sustituido por otro a lo largo de la evolución, y son fundamentales para guiar el alineamiento hacia regiones funcional o estructuralmente conservadas.

En alineamientos de ácidos nucleicos (ADN o ARN), se suelen emplear matrices simples, dado que el alfabeto es reducido (A, C, G, T/U). La matriz de similitud más básica otorga una puntuación positiva a las coincidencias (por ejemplo, $+1$) y una negativa a los desajustes (por ejemplo, -1).

También es posible utilizar matrices específicas que consideren transiciones (cambios entre purinas o entre pirimidinas) y transversiones (cambios entre una purina y una pirimidina), ya que estas ocurren con diferentes probabilidades.

Además de las sustituciones, los alineamientos deben considerar *gaps*, que representan inserciones o eliminaciones en la evolución. Para reflejar su impacto biológico, se introducen penalizaciones asociadas:

- Penalización lineal: cada gap tiene un costo fijo, por ejemplo, -2 por cada carácter insertado.
- Penalización afín (gap opening + extension): modelo más realista en el que abrir un gap tiene un costo alto (por ejemplo, -5) y extenderlo tiene un costo menor (por ejemplo, -1 por cada carácter adicional). Esto refleja la observación de que es más probable que ocurran inserciones/deleciones largas que múltiples cortas.

Para una implementación sencilla del algoritmo de Smith-Waterman en el lenguaje de bajo nivel C, vamos a emplear una matriz de similitud $s(A[i], B[j])$, tal que:

	A	**C**	**G**	**T**
A	$+3$	-1	$+1$	-1
C	-1	$+3$	-1	$+1$
G	$+1$	-1	$+3$	-1
T	-1	$+1$	-1	$+3$

Figura 1.1: Matriz de similitud para el cálculo de la puntuación.

y una constante de penalización $g = -2$.

Reconstrucción del alineamiento: Traceback

Una vez completada la matriz de puntuaciones mediante programación dinámica, el alineamiento óptimo no se encuentra explícitamente codificado en la matriz. En su lugar, se utiliza un procedimiento denominado *traceback* para reconstruir la trayectoria que llevó a la máxima puntuación local, recuperando así las subsecuencias alineadas.

A diferencia del algoritmo de alineamiento global (Needleman-Wunsch), donde el traceback comienza desde la celda inferior derecha de la matriz, en Smith-Waterman el

procedimiento de reconstrucción comienza desde la celda con la puntuación máxima de toda la matriz. Esta celda corresponde al final del alineamiento local óptimo.

Desde la celda de puntuación máxima, se sigue el camino inverso de las decisiones tomadas durante el llenado de la matriz. A cada celda $H(i, j)$ se puede haber llegado por una de las siguientes tres opciones:

- Diagonal $(i - 1, j - 1)$: si el carácter a_i de la secuencia A se alinea con b_j de la secuencia B (coincidencia o desajuste).
- Izquierda $(i, j - 1)$: si se insertó un gap en la secuencia A.
- Arriba $(i - 1, j)$: si se insertó un gap en la secuencia B.

Estas opciones están determinadas por la función de recurrencia (1.1) utilizada para construir la matriz de puntuaciones

1.2.2. Programación concurrente

La programación concurrente busca mejorar el rendimiento computacional mediante la ejecución simultánea de múltiples tareas. En el caso del algoritmo de Smith-Waterman, dado que empleamos programación dinámica, esta estrategia resulta útil debido a su elevado costo computacional, especialmente cuando se alinean secuencias largas o grandes conjuntos de datos biológicos.

Dado que el algoritmo requiere calcular una matriz de dimensiones $m \times n$, su complejidad es $\theta(mn)$, lo cual puede ser prohibitivo sin paralelización. La programación concurrente permite distribuir este trabajo entre múltiples hilos de ejecución o unidades de procesamiento, ya sea en CPU o GPU.

Dependencias de datos

Una de las principales dificultades para paralelizar Smith-Waterman es la existencia de dependencias entre celdas. Para calcular el valor de la celda $M[i, j]$, se necesita conocer previamente $M[i - 1, j - 1]$, $M[i - 1, j]$ y $M[i, j - 1]$. Esto implica que la matriz no se puede calcular en cualquier orden arbitrario.

Sin embargo, existe una solución natural: las celdas de la matriz pueden calcularse en ondas diagonales (antidiagonales), ya que todos los elementos en una misma diagonal $i + j = k$ dependen solo de valores calculados en diagonales anteriores.

Paralelización en CPU con OpenMP

OpenMP (Open Multi-Processing) es una API de paralelización basada en directivas para sistemas de memoria compartida, ampliamente utilizada en C/C++ y Fortran. Permite distribuir el trabajo entre múltiples hilos que se ejecutan en paralelo en los núcleos de una CPU.

Para aplicar OpenMP al algoritmo de Smith-Waterman:

- Se paraleliza el cálculo de la matriz diagonal por diagonal.
- En cada iteración sobre una diagonal, se calcula cada celda $M[i, j]$ de forma independiente, ya que las dependencias necesarias ya han sido resueltas en la iteración anterior.
- Se usa la directiva `#pragma omp parallel for` para distribuir la carga de trabajo entre hilos.

Paralelización en GPU con CUDA

CUDA (Compute Unified Device Architecture) es una plataforma desarrollada por NVIDIA para el cómputo en GPU. Permite ejecutar miles de hilos concurrentemente, aprovechando la arquitectura masivamente paralela de las tarjetas gráficas.

Un programa desarrollado con CUDA se basa en un modelo de ejecución heterogéneo, en el que la CPU (host) y la GPU (device) cooperan para resolver un problema computacional.

Primero, se define el kernel, que es una función especial que se ejecuta en la GPU. Esta función, marcada con la palabra clave `__global__`, será ejecutada por muchos hilos en paralelo, cada uno operando sobre un subconjunto de los datos.

A continuación, en la CPU (host), se inicializan las estructuras de datos y se reserva la memoria correspondiente. Luego, se asigna memoria en la GPU (device) mediante funciones como cudaMalloc. Una vez reservada la memoria, los datos se transfieren desde el host al device usando cudaMemcpy.

Por último, se lanza el kernel desde la CPU, especificando la cantidad de bloques e hilos que ejecutarán la función en la GPU. Una vez finalizado el cálculo en la GPU, los resultados se copian de vuelta a la memoria de la CPU. Finalmente, se libera la memoria utilizada tanto en el host como en el device.

El algoritmo Smith-Waterman se adapta a CUDA de forma similar a la implementación con OpenMP:

- Cada celda de la matriz puede ser computada por un hilo distinto, siempre y cuando se respete el orden de dependencias.
- Se procesan las celdas en bloques diagonales (también llamados "wavefronts").
- CUDA permite asignar un kernel por cada diagonal, donde cada hilo se encarga de una celda específica de esa diagonal.

Existen varias consideraciones importantes a tener en cuenta al desarrollar programas con CUDA. En primer lugar, es fundamental considerar la latencia de acceso a la memoria global de la GPU, la cual puede afectar significativamente el rendimiento si no se gestiona adecuadamente. Para mitigar este problema, hay que optimizar el uso de memoria compartida, que es mucho más rápida y permite la cooperación eficiente entre hilos dentro de un mismo bloque.

Otro aspecto relevante es la configuración del número de hilos y bloques. Esta debe ser cuidadosamente calculada en función de la arquitectura de la GPU, con el objetivo de maximizar la ocupación de los multiprocesadores y asegurar que la GPU esté completamente aprovechada durante la ejecución.

Finalmente, la transferencia de datos entre el host (CPU) y el device (GPU) representa un posible cuello de botella, debido al tiempo requerido para mover grandes volúmenes de datos a través del bus PCIe. Por tanto, debemos minimizar la frecuencia y el volumen de estas transferencias, manteniendo los datos el mayor tiempo posible en la memoria del device y realizando las operaciones directamente en la GPU siempre que sea factible.

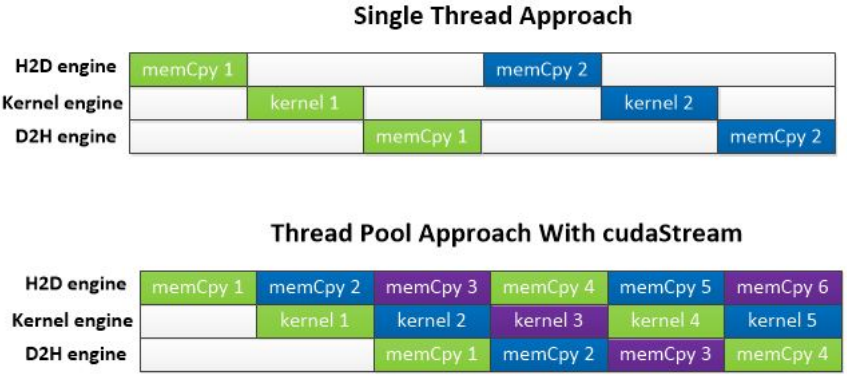

Figura 1.2: Flujo de ejecución CUDA.

1.3. Implementación

En este trabajo se desarrollaron tres versiones del algoritmo de Smith-Waterman utilizando el lenguaje C: una versión secuencial, una versión concurrente a nivel de CPU mediante OpenMP, y una versión paralelizada en GPU utilizando CUDA. Cada una de estas implementaciones calcula el alineamiento local óptimo entre dos secuencias de ADN, siguiendo la formulación clásica basada en programación dinámica.

Dado que el algoritmo requiere reservar matrices de puntuación de tamaño $\theta(mn)$, donde m y n representan las longitudes de las secuencias a alinear, el consumo de memoria crece rápidamente con el tamaño de entrada. Por este motivo, y considerando las limitaciones computacionales disponibles, se optó por trabajar exclusivamente con secuencias de ADN mitocondrial, cuya longitud (alrededor de 16.500 pares de bases) es suficientemente representativa para evaluar el comportamiento de los algoritmos sin comprometer la viabilidad del experimento.

Para comparar el rendimiento de cada aproximación, se ejecuta cada implementación varias veces sobre el mismo par de secuencias: genoma mitocondrial de Homo sapiens (humano) y Pan troglodytes (chimpancé), registrando el tiempo de ejecución en cada repetición. A partir de estas mediciones, se calcula el tiempo medio de ejecución de cada versión del algoritmo. Finalmente, se representa gráficamente el tiempo medio de cada

implementación mediante un gráfico de barras, lo cual permite comparar de forma clara y directa la eficiencia relativa de cada enfoque.

El entorno de experimentación está conformado por un procesador Ryzen 7 5800HS, y una gráfica Nvidia GeForce RTX 3060 (versión portátil). Como mencionado anteriormente, el lenguaje de programación usado es C, popular en bioinformática por su alto rendimiento a bajo nivel.

1.3.1. Aproximación secuencial

Para establecer una línea base en el análisis comparativo del rendimiento, se desarrolló primero una versión secuencial del algoritmo de Smith-Waterman.

Las secuencias de entrada, codificadas en archivos FASTA, se leen y almacenan en buffers de tamaño fijo. Dado que se trata de genomas mitocondriales, su longitud es limitada y manejable en memoria. Se emplea una rutina de lectura que omite las líneas de encabezado (>) y concatena las secuencias en memoria.

El algoritmo utiliza dos matrices principales: una matriz de puntuaciones H, que almacena los valores de alineamiento local en cada posición, y una matriz *traceback*, que guarda la dirección desde la cual se obtuvo el valor óptimo para poder reconstruir el alineamiento una vez completado el cálculo. Ambas matrices se inicializan con valores cero, representando que inicialmente no existe alineamiento.

El núcleo del algoritmo está en el cálculo de la matriz de puntuaciones. Para cada par de posiciones (i, j), correspondientes a los nucleótidos de las dos secuencias, se calcula el mejor valor posible considerando las opciones de la función 1.1.

Este valor se asigna a la celda $H_{i,j}$, y se almacena la dirección correspondiente en la matriz de trazado. El cálculo se realiza de forma iterativa sobre todas las filas y columnas:

```
int diag = H[(i - 1) * (n + 1) + (j - 1)] + s;
int up   = H[(i - 1) * (n + 1) + j] + GAP;
int left = H[i * (n + 1) + (j - 1)] + GAP;
int val  = max(0, diag, up, left);
H[i * (n + 1) + j] = val;
```

Figura 1.3: Cálculo de la puntuación máxima en la celda $H_{i,j}$ de la matriz de alineamiento, considerando las tres posibles direcciones y un reinicio en cero.

Donde s representa la puntuación por *match* o *mismatch*, extraída de la matriz de similitud 1.1.

Una vez rellenada la matriz, se busca la celda con el valor máximo, que indica el final del mejor alineamiento local. A partir de esa posición, se recorre la matriz traceback en sentido inverso hasta llegar a una celda con valor cero, reconstruyendo así las dos secuencias alineadas.

Durante la ejecución, se mide tanto el tiempo real como el tiempo de CPU con ayuda

de las funciones `omp_get_wtime()` y `clock()` respectivamente, permitiendo más adelante comparar el rendimiento con versiones paralelas.

1.3.2. Aproximación concurrente a nivel de CPU

La versión concurrente del algoritmo de Smith-Waterman mantiene intactos muchos de los componentes de la versión secuencial, incluyendo la lectura de secuencias en formato FASTA, la inicialización de la matriz de puntuaciones H y la definición de la matriz de similitud. Sin embargo, el cálculo de puntuaciones en la matriz se ha adaptado para explotar paralelismo a nivel de CPU utilizando OpenMP.

La principal modificación se introduce en el bucle que calcula las puntuaciones de la matriz H. En lugar de recorrer la matriz de forma fila por fila, se procesan diagonales antidiagonales—también conocidas como *wavefronts*, lo que permite que las celdas dentro de una misma diagonal se calculen de forma independiente y concurrente.

El siguiente fragmento muestra cómo se paraleliza cada diagonal mediante una directiva de OpenMP:

```
for (int diag = 2; diag <= m + n; diag++) {
    #pragma omp parallel for reduction(max:max_score)
    for (int i = 1; i <= m; i++) {
        int j = diag - i;
        if (j >= 1 && j <= n) {
            // calculo del valor en H[i][j]
        }
    }
}
```

Figura 1.4: Recorrido *wavefront* con una directiva de OpenMP.

En este esquema, cada iteración del bucle interno evalúa una celda $H_{i,j}$ cuya posición está determinada por la diagonal actual. Al recorrer las diagonales en orden creciente, se garantiza que las dependencias necesarias (celdas a la izquierda, arriba y en diagonal) ya han sido calculadas previamente.

Dentro de cada celda, el valor se calcula del mismo modo que en la versión secuencial 1.3, considerando las tres direcciones posibles y tomando el máximo con cero para permitir reinicio de alineamientos locales.

1.3.3. Aproximación concurrente a nivel de GPU

Esta aproximación difiere sustancialmente tanto de la versión secuencial como de la versión paralela con OpenMP, no solo en el modelo de ejecución, sino también en la organización de la memoria y en la forma de recorrer la matriz de puntuaciones H.

A diferencia de la versión paralela en CPU, que utilizaba directivas `#pragma omp parallel for` para distribuir las iteraciones de cada diagonal entre núcleos, en CUDA se lanza un kernel por cada diagonal de la matriz.

El kernel `smith_waterman_kernel` accede a los datos de entrada desde memoria global de la GPU y emplea memoria constante para almacenar la matriz de similitud entre pares de bases, lo que reduce significativamente la latencia de acceso en comparación con variables regulares. La matriz H se aloja también en memoria global y se inicializa en cero. Para cada celda, se calcula el valor máximo entre los tres posibles movimientos de igual forma que las aproximaciones anteriores.

```
__global__ void smith_waterman_kernel(
    char *seq1, char *seq2, int *H, int m, int n, int diag,
    int start_i, int end_i)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int i = start_i + idx;
    if (i <= end_i) {
        int s = 0;
        int j = diag - i + 1;
        int ind1 = caracter(seq1[i - 1]);
        int ind2 = caracter(seq2[j - 1]);
        if (ind1 >= 0 && ind2 >= 0) {
            s = d_similitud[[ind1 * 4 + ind2]];
        }

        int diag = H[(i - 1) * (n + 1) + (j - 1)] + s;
        int up = H[(i - 1) * (n + 1) + j] + GAP;
        int left = H[i * (n + 1) + (j - 1)] + GAP;
        int val = max(0, diag, up, left);
        H[i * (n + 1) + j] = val;
    }
}
```

Figura 1.5: Kernel de CUDA para el cálculo de puntuaciones en el algoritmo Smith-Waterman.

Una diferencia crucial con respecto a la versión en CPU es el uso explícito de funciones de sincronización (`cudaDeviceSynchronize()`) tras cada kernel, garantizando que una diagonal esté completamente procesada antes de iniciar la siguiente.

Un aspecto importante a la hora de usar CUDA, es el cálculo del tamaño por bloque óptimo. Si el bloque es demasiado pequeño, no se generan suficientes hilos para mantener ocupados los multiprocesadores, y la latencia de memoria no se oculta de forma efectiva. Por otro lado, si el bloque es demasiado grande, puede saturarse la cantidad de registros

o memoria compartida por bloque, lo que lleva a una menor concurrencia y, en algunos casos, incluso a errores de ejecución.

En esta implementación, se permite ajustar el tamaño de bloque desde la línea de comandos, forzando que sea múltiplo de 32. Esto permite experimentar y optimizar el rendimiento en función del hardware específico, el tamaño de las secuencias y el número de celdas que se procesan por diagonal. Con un *script* sencillo de bash 1.6, podemos automatizar la ejecución del programa en un bucle para medir el rendimiento con cada tamaño de bloque (*Véase la figura 1.7*).

```
#!/bin/bash

for size in 32 64 128 256 512 1024; do
    echo "Times for $size:" >> tiempos.txt
    for i in $(seq 1 3); do
        ./Smith_Waterman_Cuda.exe
        resources/Homo_sapiens.fasta     # argumento 1
        resources/Pan_troglodytes.fasta # argumento 2
        $size                            # argumento 3
        >> tiempos.txt # fichero con los tiempos de ejecucion
    done
    echo "" >> tiempos.txt # salto de linea
done
```

Figura 1.6: *Script* de bash para la ejecución automática del programa. Para cada tamaño de bloque múltiplo de 32 (desde 32 hasta 1024) se lanzan 3 ejecuciones, y se guardan los tiempos de ejecución.

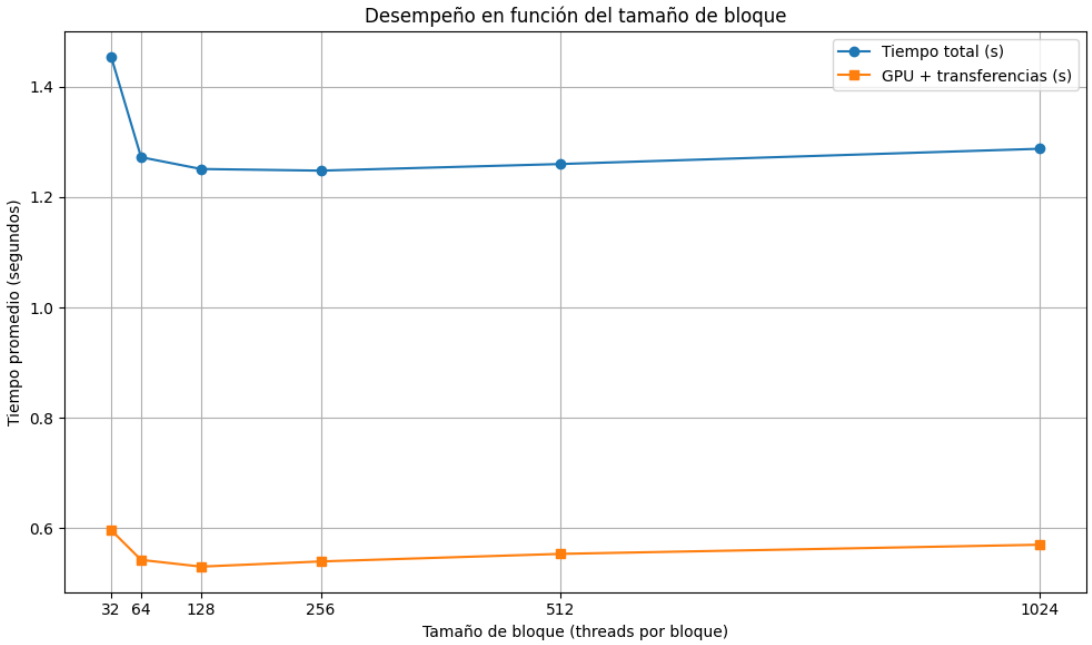

Figura 1.7: Tiempos de ejecución de Smith-Waterman con CUDA en función del tamaño de bloque. El tamaño de bloque óptimo para estas condiciones es de 128 hilos.

1.4. Análisis del rendimiento

En la gráfica 1.8 se muestra el tiempo medio de ejecución, en segundos, para las tres versiones del algoritmo de Smith-Waterman: Secuencial, OpenMP (paralelización en CPU) y CUDA (paralelización en GPU). Los resultados permiten extraer varias conclusiones relevantes sobre la eficiencia computacional de cada aproximación.

En primer lugar, la implementación más rápida es la versión en CUDA, con un tiempo medio significativamente menor que el de las otras dos variantes. Esta mejora se debe a la alta capacidad de paralelización masiva de las GPUs, que permite distribuir el cálculo de la matriz de puntuaciones entre cientos o miles de hilos concurrentes. Aunque el tiempo total incluye la transferencia de datos entre CPU y GPU, el beneficio computacional supera ampliamente este coste.

En contraste, la versión OpenMP resulta ser la más lenta de las tres, incluso peor que la versión secuencial. Esto es inesperado en principio, ya que OpenMP está diseñado para explotar el paralelismo en arquitecturas multinúcleo. Algunos motivos de peso son:

- Sobrecarga de sincronización: En OpenMP, las dependencias entre iteraciones (por ejemplo, en el cálculo diagonalizado de la matriz) pueden requerir sincronizaciones frecuentes que anulan el beneficio del paralelismo.

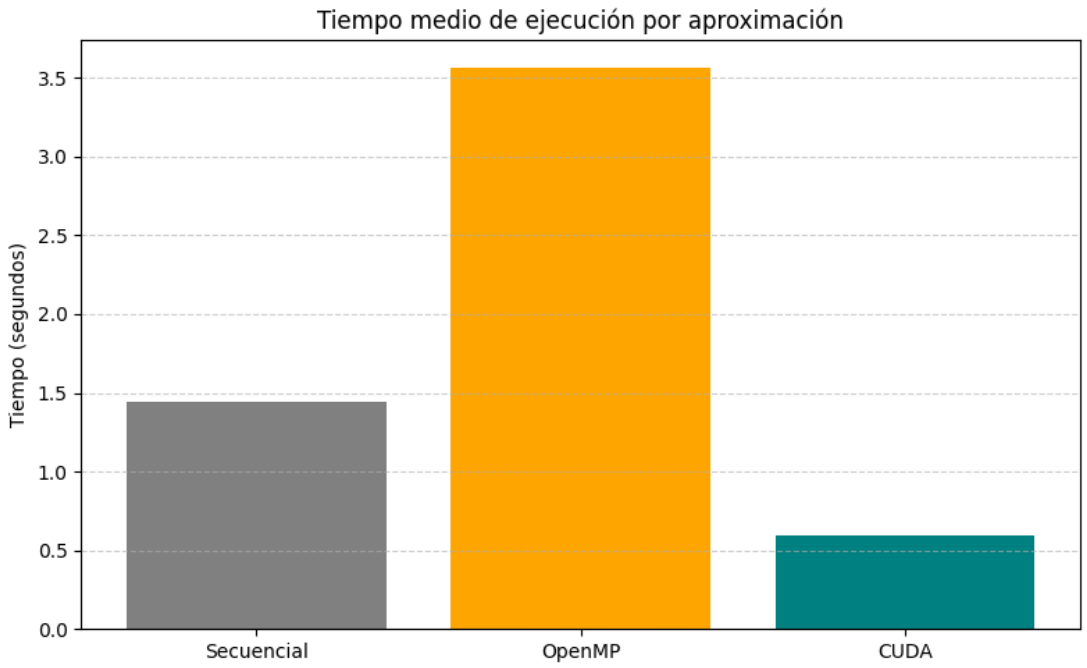

Figura 1.8: Rendimiento de las tres aproximaciones para una media de 10 ejecuciones.

- Mala distribución de carga: Es posible que algunos hilos tengan mucho trabajo y otros casi ninguno.
- Contenido cacheable y uso de memoria: Si el algoritmo requiere mucho acceso a memoria compartida puede generar un cuello de botella, que en este caso no compensa los beneficios (como sí pasa con CUDA).

La versión secuencial, aunque conceptualmente más simple, logra mejores resultados que OpenMP al evitar la sobrecarga de gestión de hilos y sincronización, lo que resalta que no toda paralelización en CPU garantiza mejor rendimiento.

1.5. Conclusión

El alineamiento de secuencias es una técnica central en bioinformática, utilizada como herramienta multiusos aplicable a muchos ámbitos. En la actualidad, con el crecimiento exponencial de los datos biológicos —como genomas completos y bases de datos masivas—, se hace cada vez más necesario recurrir a algoritmos optimizados y enfoques paralelos que permitan realizar las comparaciones en tiempos razonables.

En este proyecto se ha abordado el problema mediante la implementación del algoritmo de Smith-Waterman en tres variantes: una versión secuencial, una versión paralela en

CPU utilizando OpenMP, y una versión en GPU mediante CUDA. Los resultados obtenidos muestran diferencias significativas en cuanto al rendimiento. La versión secuencial proporciona una línea base estable, con tiempos de ejecución favorables. Sorprendentemente, la versión paralela en CPU con OpenMP ha resultado ser la más lenta, lo cual se explica probablemente por la sobrecarga asociada a la gestión de múltiples hilos y una posible distribución ineficiente de la carga de trabajo. En contraste, la versión CUDA ha demostrado ser la más eficiente, logrando tiempos de ejecución considerablemente menores. Esto evidencia que, para algoritmos que involucran estructuras de datos altamente paralelizables como las matrices de alineamiento, la arquitectura de las GPUs resulta especialmente adecuada.

En conclusión, el uso de cómputo paralelo en GPU parece muy viable para resolver problemas bioinformáticos a gran escala. Por otro lado, hay que tener cuidado con la paralelización en CPU, ya que no siempre conduce a mejoras si no se gestiona adecuadamente el equilibrio de carga y la sincronización entre hilos. En definitiva, este estudio confirma que para seguir el ritmo del crecimiento de los datos genómicos actuales, es imprescindible adoptar técnicas computacionales modernas, y que la elección adecuada de la tecnología puede marcar una diferencia sustancial en el rendimiento de los algoritmos empleados en bioinformática.

Bibliografía

de J. Pérez Jiménez, M. (2014). Alineamiento de pares de secuencias de genes/proteínas. Técnicas Inteligentes en Bioinformática. Máster Universitario en Lógica, Computación e Inteligencia Artificial, Universidad de Sevilla.

Koerkamp, R. G. (2025). A history of pairwise alignment.
URL: *https://curiouscoding.nl/posts/pairwise-alignment/*

Likic, V. (n.d.). The needleman-wunsch algorithm forsequence alignment. The University of Melbourne.

Okada, D., Ino, F. and Hagihara, K. (2015). Accelerating the smith-waterman algorithm with interpair pruning and band optimization for the all-pairs comparison of base sequences, *BMC Bioinformatics* .

Rosenberg, M. S. (n.d.). Sequence alignment. Arizona State University.

Wikipedia (n.d.). Algoritmo smith-waterman.
URL: *https://es.wikipedia.org/wiki/Algoritmo*$smith - Waterman$