Universidad Católica San Pablo

Informe del laboratorio 1

Computer Science — Biología Molecular Computacional

Harold Alejandro Villanueva Borda

---

1. Introducción

El algoritmo Needleman-Wunsch es un método utilizado para realizar alineamiento de secuencias, comúnmente empleado en bioinformática para comparar secuencias de ADN, ARN o proteínas. El objetivo principal es encontrar la mejor manera de alinear dos secuencias, maximizando las coincidencias y minimizando las inserciones, eliminaciones o sustituciones (rupturas en la secuencia).

2. Funcionamiento

2.1. Inicialización

Se crea una matriz que representa las posibles alineaciones entre dos secuencias. La primera fila y la primera columna se llenan de valores basados en penalizaciones por la inserción de huecos (gaps).

2.2. Recurrencia

Cada posición en la matriz se llena comparando el valor de tres posibles caminos: el valor proveniente de la diagonal (coincidencia o sustitución), el de arriba (inserción) y el de la izquierda (eliminación). La idea es elegir el valor que maximice el *score*.

2.3. Backtracking

Después de llenar la matriz, se recorre desde la esquina inferior derecha hacia la superior izquierda, reconstruyendo el alineamiento óptimo.

3. Paradigma de Programación

El algoritmo utiliza programación dinámica, un paradigma que resuelve problemas dividiéndolos en subproblemas más pequeños y almacenando los resultados de estos para evitar cálculos repetidos. La matriz bidimensional almacena soluciones parciales que luego se usan para reconstruir la solución completa.

3.1. Mejoras con Programación Dinámica

Gracias a la programación dinámica, el algoritmo Needleman-Wunsch reduce la complejidad de un enfoque ingenuo (exponencial) a una complejidad $O(n \times m)$, donde n y m son las longitudes de las secuencias. Esto permite manejar de manera eficiente el alineamiento de secuencias largas.

4. Análisis del Código Proporcionado

El código proporcionado es una implementación paralela del algoritmo Needleman-Wunsch utilizando CUDA. Aquí se detallan las partes clave del código:

4.1. Lógica General

- Inicialización de la matriz de puntuación (M) y la matriz de dirección (D), que almacenan el puntaje de alineamiento y el camino seguido (diagonal, arriba, izquierda) respectivamente.
- Uso de CUDA: Se distribuyen las operaciones para el cálculo de cada celda de la matriz en múltiples hilos de GPU mediante bloques y subbloques, lo que permite un procesamiento más rápido y en paralelo.
- Reconstrucción del alineamiento: A partir de la matriz de dirección (D), se realiza un backtracking desde la esquina inferior derecha de la matriz para reconstruir las dos secuencias alineadas.

4.2. CUDA

El código utiliza el siguiente enfoque con CUDA:

- **Kernel**: El kernel `needlemanWunschKernel` distribuye el cálculo de las celdas de la matriz de puntuación y dirección en múltiples hilos. Cada hilo de CUDA se encarga de calcular una celda específica de la matriz.
- **Paralelización**: Se paraleliza el proceso de llenado de la matriz mediante bloques y *threads*, distribuyendo los cálculos de cada fila y columna entre distintos hilos.

- **Optimización del Tiempo**: Se mide el tiempo de ejecución utilizando `cudaEventRecord` para calcular el tiempo que toma el proceso completo de alineamiento.

5. Detección de la Solución con Menos Rupturas

Para detectar la solución que ofrece menos rupturas, el algoritmo utiliza la matriz de dirección (D). La dirección diagonal indica una coincidencia o una sustitución mínima, lo cual sugiere que no hay rupturas. Al hacer el backtracking, se priorizan los movimientos diagonales sobre los movimientos hacia arriba o hacia la izquierda, ya que estos últimos indican la inserción de huecos (rupturas).

6. Idea de la Matriz de Puntos

La Matriz de Puntos visualiza las coincidencias entre dos secuencias. En el código, esto se implementa mostrando un asterisco (*) cuando los caracteres de ambas secuencias coinciden, y un punto (.) cuando no lo hacen. Esta matriz es útil para identificar regiones con alto grado de similitud y observar la distribución de coincidencias a lo largo de las secuencias.

7. Lógica del Algoritmo Needleman-Wunsch

El algoritmo Needleman-Wunsch realiza un alineamiento global entre dos secuencias (como ADN, ARN o proteínas) maximizando las coincidencias y minimizando las inserciones y eliminaciones (gaps). La lógica detrás del algoritmo es la siguiente:

7.1. Inicialización:

Se crea una matriz de puntuación (M) de tamaño $(n + 1) \times (m + 1)$, donde n y m son las longitudes de las dos secuencias s y t. Las primeras filas y columnas de la matriz se rellenan con penalizaciones de gaps: cada celda de la primera fila ($M[0][j]$) es $j \times gap$, y cada celda de la primera columna ($M[i][0]$) es $i \times gap$.

7.2. Relleno de la Matriz:

Para cada celda en la matriz $M[i][j]$, se compara el puntaje de tres posibles caminos:

- **Diagonal:** Viene de la celda $M[i-1][j-1]$ si los caracteres coinciden (se le suma el puntaje de coincidencia), o el puntaje de sustitución si no coinciden.
- **Arriba:** Viene de $M[i-1][j]$ y penaliza una inserción.

- **Izquierda:** Viene de $M[i][j-1]$ y penaliza una eliminación.

El valor máximo entre estos tres posibles caminos se guarda en $M[i][j]$ y se registra la dirección en la matriz D (para la posterior reconstrucción del alineamiento).

7.3. Reconstrucción del Alineamiento:

Una vez que la matriz está completa, se realiza un *backtracking* desde la última celda $M[n][m]$ para reconstruir el alineamiento óptimo. El *backtracking* sigue las direcciones registradas en D, priorizando las coincidencias (movimientos diagonales), las inserciones (movimientos hacia arriba) o las eliminaciones (movimientos hacia la izquierda).

8. Lógica de la Paralelización usando CUDA

La paralelización del algoritmo Needleman-Wunsch mediante CUDA divide el cálculo de las celdas de la matriz entre múltiples hilos de procesamiento de la GPU. Aquí está la lógica de cómo funciona:

8.1. Distribución de Tareas:

El cálculo de cada celda de la matriz $M[i][j]$ es independiente de los demás (excepto por las dependencias previas en los valores de las celdas adyacentes), lo que lo hace un problema altamente paralelizable. Cada hilo de CUDA se encarga de calcular el valor de una celda en la matriz de puntuación $M[i][j]$ y su correspondiente dirección en la matriz $D[i][j]$.

8.2. Grids y Blocks:

La GPU organiza los hilos en bloques y grids. Cada bloque contiene un número fijo de *threads* (en este caso 16x16). Cada bloque se asigna a una subregión de la matriz M, de modo que cada *thread* dentro del bloque calcula una celda. La función `dim3 threadsPerBlock(16, 16)` establece que cada bloque tiene un grid de 16x16 hilos. La cantidad de bloques se calcula en función de las dimensiones de las secuencias $(n+1)$ y $(m+1)$.

8.3. Kernel CUDA:

El kernel `needlemanWunschKernel` se ejecuta en la GPU y realiza el cálculo para cada celda de la matriz de puntuación y dirección. Los hilos calculan simultáneamente las celdas de $M[i][j]$ y $D[i][j]$, acelerando enormemente el tiempo de procesamiento en comparación con la ejecución secuencial en CPU.

8.4. Sincronización y Cálculo Completo:

El kernel se lanza con el comando `needlemanWunschKernel«<blocks, threadsPerBlock»>`, y la función `cudaDeviceSynchronize()` asegura que todos los hilos terminen antes de continuar. Una vez finalizado el cálculo en la GPU, los resultados (las matrices M y D) se copian de vuelta al host (CPU) con `cudaMemcpy()`.

9. Paradigma de Programación

El paradigma principal utilizado en este código es programación paralela combinada con programación dinámica:

9.1. Programación Dinámica:

El algoritmo Needleman-Wunsch se basa en la técnica de programación dinámica, donde se construye una solución global a partir de soluciones óptimas de subproblemas. Cada celda de la matriz $M[i][j]$ se calcula utilizando los valores ya computados en las celdas adyacentes.

9.2. Programación Paralela:

CUDA permite paralelizar el cálculo de las celdas de la matriz $M[i][j]$ y $D[i][j]$. Los hilos de CUDA se asignan a diferentes celdas, y estas se calculan en paralelo, lo que reduce considerablemente el tiempo de ejecución.

10. Técnicas de Programación Utilizadas

10.1. Memoria de GPU:

El código usa `cudaMalloc` para reservar memoria en la GPU para las matrices M y D, así como para las secuencias s y t. Luego, las secuencias se copian desde el host (CPU) a la GPU con `cudaMemcpy`.

10.2. Paralelismo a nivel de datos:

Cada hilo de CUDA trabaja de manera independiente en una celda de la matriz. Esta técnica explota el paralelismo a nivel de datos, dado que los cálculos de las celdas se pueden realizar simultáneamente sin interferencia.

10.3. Sincronización:

El uso de `cudaDeviceSynchronize` asegura que todos los hilos finalicen su trabajo antes de que los resultados se utilicen en el host, evitando condiciones de carrera y asegurando la correcta ejecución.

10.4. Medición del Tiempo:

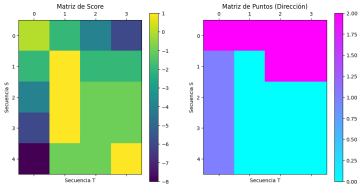
Se utiliza `cudaEventRecord` y `cudaEventElapsedTime` para medir el tiempo de ejecución del kernel en la GPU, lo que permite evaluar el rendimiento de la paralelización en comparación con una versión secuencial.

11. Resultados al Compilar

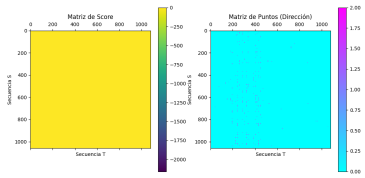
Los tiempos de ejecución obtenidos muestran que el código es eficiente al manejar distintas secuencias biológicas:

- bacteria-influenza: 0.585568 ms
- bacteria-SARS-CoV: 0.657344 ms
- influenza-SARS-CoV: 0.593472 ms

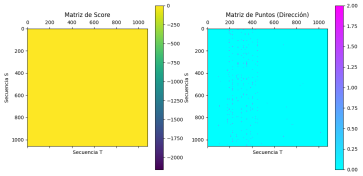
El uso de CUDA acelera significativamente el proceso de alineamiento, permitiendo comparar secuencias complejas en milisegundos.



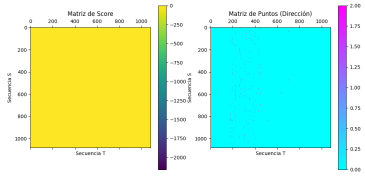
1: Ejemplo en clase.



2: Bacteria Vs Influenza.



3: Bacteria Vs Sars-cov.



4: Influenza Vs Sars-cov.

Figura 1: Resultados de alineamiento usando Needleman-Wunsch.

12. Análisis de Alineamientos

1. **Bacteria vs. Influenza** (Esquina superior izquierda)
   - **Matriz de Score:** Muestra una variedad en las puntuaciones, desde verde oscuro (baja) hasta amarillo (alta), indicando diversidad en coincidencias y desajustes.
   - **Matriz de Puntos (Dirección):** Colores variados sugieren alineamiento óptimo con coincidencias, inserciones y eliminaciones.
2. **Bacteria vs. SARS-CoV** (Esquina superior derecha)
   - **Matriz de Score:** Mayormente amarilla, indicando puntuaciones bajas y poca coincidencia entre secuencias.
   - **Matriz de Puntos (Dirección):** Uniforme, sugiere un alineamiento directo con pocas modificaciones.
3. **Influenza vs. SARS-CoV** (Esquina inferior derecha)
   - **Matriz de Score:** Similar a Bacteria vs. SARS-CoV, con puntuaciones bajas y muchas diferencias.
   - **Matriz de Puntos (Dirección):** Uniforme, indicando un alineamiento simple con pocas inserciones o eliminaciones.
4. **Influenza vs. SARS-CoV** (Esquina inferior izquierda)
   - **Matriz de Score:** Mayormente amarilla, con puntuaciones negativas, sugiriendo una alta disimilitud.
   - **Matriz de Puntos (Dirección):** Uniforme, con un alineamiento directo y pocas modificaciones.

13. Implementación en Github

El código fuente del análisis se encuentra disponible en GitHub: GitHub.