Proyecto Final: Análisis de Rendimiento de Dotplot Secuencial vs. Paralelización

Dany Orlando Guancha Tarapues
Jhair Alexander Peña
Jefferson Henao Cano

Introducción

La comparación de secuencias de ADN o proteínas es esencial en bioinformática para comprender su estructura y función. Un método comúnmente utilizado para este propósito es el dotplot, que permite visualizar la similitud entre dos secuencias de manera gráfica .

En la **implementación secuencial**, se desarrollará un algoritmo que realiza el dotplot de forma lineal, procesando uno a uno los elementos de las secuencias. Aunque sencillo, este enfoque puede ser lento cuando se trabaja con secuencias grandes.

En la **versión paralela con multiprocessing**, se aprovechará la capacidad de ejecutar múltiples procesos simultáneamente en sistemas con varios núcleos de CPU. Dividiendo el trabajo en tareas más pequeñas y distribuyéndolas entre los procesos, se acelerará el cálculo del dotplot, logrando un procesamiento más rápido y un uso eficiente de los recursos.

En la **versión paralela con mpi4py**, se utilizará la biblioteca mpi4py, que se basa en el estándar Message Passing Interface (MPI) para la programación paralela. Se distribuirán las tareas de cálculo del dotplot entre los procesos MPI, permitiendo un rendimiento aún mayor en comparación con la versión paralela con multiprocessing .

En la **versión paralela con PyCUDA**, se aprovechará la capacidad de las GPUs para realizar cálculos masivos en paralelo. Usando la biblioteca PyCUDA, se implementará el dotplot aprovechando el paralelismo masivo que ofrecen las tarjetas gráficas, lo que puede resultar en una aceleración significativa del proceso.

Al comparar estas cuatro formas de realizar un dotplot, se evaluará la eficiencia y el rendimiento de cada enfoque. Esto proporcionará información valiosa sobre las ventajas y desventajas de los enfoques secuenciales y paralelos, así como la comparación entre las bibliotecas multiprocessing, mpi4py y PyCUDA, para tareas de bioinformática.

Estos resultados podrían ser útiles para optimizar futuros proyectos de análisis comparativo de secuencias y mejorar la eficiencia en el procesamiento de grandes volúmenes de datos genómicos y proteómicos. Además, se busca contribuir al análisis de las secuencias incluyendo una función de filtrado la cual indica la similitud de cada carácter de las secuencias empleadas, esto con el propósito de ayudar visualmente a la evaluación por parte del personal especializado .

Es por esto que el proyecto tiene como objetivo implementar y analizar el rendimiento de cuatro enfoques diferentes para realizar un dotplot: una versión secuencial, una versión paralela utilizando la biblioteca multiprocessing de Python, una versión paralela utilizando mpi4py, y una versión paralela utilizando PyCUDA.

Materiales y funciones.

A. Librerías

Se utilizó el lenguaje de programación Python en su versión 3.12 para implementar el algoritmo. Se emplearon varias bibliotecas clave, como Numpy, Matplotlib, Time, mpi4py, opencv, Multiprocessing y PyCUDA, las cuales son fundamentales para manipular matrices, generar gráficos, leer archivos, detectar diagonales mediante filtros, medir tiempos, obtener parámetros de línea de comandos y trabajar con múltiples procesadores y GPU. En la Tabla 1 se detallan las versiones de cada uno de los software y bibliotecas utilizados en la investigación.

Paquete	Versión
Python	3.12
Matplotlib	3.7.1
Numpy	1.24.2
mpi4py	3.1.4
Time	3.7
Multiprocessing	3.7
PyCUDA	2023.1.1
OpenCV	4.7.0.72

B. Paralelización

El proceso de paralelización se realizó por medio de las librerías multiprocessing, mpi4py y PyCUDA de Python. Para este proceso, como entradas se cargaron dos secuencias que se querían alinear gráficamente, organizándose en una matriz NxM (donde N y M son las longitudes de las secuencias respectivamente).

- Multiprocessing: Se utilizó la librería Pool para generar los procesos de acuerdo con la cantidad de threads que se recibían por parámetro, y por cada pool se manejaba la función map para recorrer cada índice de la primera secuencia por cada índice de la segunda secuencia y realizar la comparación de estos para guardar en una lista un número de representación de color, que sería la representación gráfica que queremos obtener.

- mpi4py: Se utilizó la estrategia de Chunks, para dividir la primera secuencia en matrices más pequeñas y se creó otra matriz donde se guardará la solución de la implementación. En cada recorrido de los Chunks contra las posiciones de la segunda secuencia, se efectúa la misma comparación de la implementación con multiprocessing, para darle un valor de color a la posición de la solución, luego se unen las soluciones generadas.

- PyCUDA: Se aprovechó la capacidad de las GPUs para realizar cálculos masivos en paralelo. Utilizando PyCUDA, se implementó el dotplot distribuyendo las operaciones

de comparación entre los núcleos de la GPU, acelerando significativamente el procesamiento en comparación con CPU.

C. Datos de experimentación

Para realizar las pruebas de rendimiento, se utilizaron dos archivos FASTA, el de Salmonella y E. Coli. Estos archivos contienen alrededor de 4 millones de bases nitrogenadas. Se ejecutó el algoritmo con el objetivo de comparar los tiempos de ejecución al aplicar estrategias paralelas como multiprocessing, mpi4py y PyCUDA, en contraste con los tiempos de ejecución obtenidos en secuencial. De manera similar, se evaluó la eficiencia.

D. Arquitectura computacional

Las pruebas se ejecutaron en un computador con las siguientes especificaciones:

- La primera máquina tiene 4 procesadores, es un AMD RYZEN 7 3700U con 8GB de RAM.
- La segunda máquina tiene 4 procesadores, es un Intel Core i7 de séptima generación con 4GB de RAM.
- La tercera máquina tiene 4 procesadores, es un AMD RYZEN 5 3500U con 4GB de RAM.

Las tres máquinas tienen como sistema operativo Windows 11.

E. Disponibilidad del algoritmo

La herramienta presentada en este trabajo es de acceso libre. El código fuente y la guía de uso e instalación están disponibles en: PCD-project.

III. RESULTADOS

![Error para asignar memoria a las matrices con toda la secuencia](fig1.png)
Error en la asignación del espacio de memoria, ya que no se cuenta con la suficiente para representar toda la secuencia, lo que no permitía siquiera arrancar a procesar los datos. Por lo tanto, se efectuaron pruebas hasta encontrar el máximo de datos de una matriz con el cual cada máquina mencionada anteriormente podía implementar la solución. Al final de las pruebas se encontró que una matriz conjunta con la secuencia uno y dos de 16,000 x 16,000 datos era la matriz evaluada que se podía ejecutar en una de las máquinas.

![Tiempo secuencial](fig2.png)

![Dotplot para la solución de la matriz evaluada](fig3.png)
Con un dotplot solución, en el que se evidencia la diagonal principal que servirá para el análisis de las secuencias.

![Dotplot filtrado para la solución de la matriz evaluada](fig4.png)
Terminado el proceso secuencial, se vuelve a correr el archivo, pero con la implementación paralela de multiprocessing, la cual presenta los siguientes datos:

En la imagen anterior se observa la escalabilidad de la implementación. Se utilizaron 8 hilos; el último tuvo un tiempo de ejecución de 1,2 minutos, lo cual se puede evidenciar en la aceleración y la eficacia obtenidas en la ejecución.

Al observar las gráficas de comparación entre tiempo de ejecución, aceleración y eficacia, podemos observar que para un hilo la ejecución tomaba más tiempo en reproducir la solución. A medida que se incrementaban los hilos, el tiempo de ejecución mejoraba. Sin embargo, a mayor cantidad de hilos, por el overhead, los tiempos van en aumento, la aceleración disminuye y la eficacia se pierde.

En la imagen anterior vemos los valores que nos arroja la ejecución del algoritmo con diferentes números de núcleos y el aumento o disminución de la aceleración y la eficiencia.

En la gráfica vemos que hasta cierto punto, a medida que aumentamos el número de núcleos, la ejecución del algoritmo se hace más rápida. Pero entre los 3 y 4 núcleos, la aceleración disminuye y su eficiencia también.

IV. REFERENCIAS

1. J. S. Piña, S. Orozco-Arias, N. Tobón-Orozco, M. S. Candamil-Cortés, R. Tabares-Soto y R. Guyot, "Alineamiento gráfico de secuencias a través de programación paralela: un enfoque desde la era postgenómica", Revista Ingeniería Biomédica, vol. 13, n.º 26, pp. 37–45, 2019. Accedido el 7 de junio de 2023. [En línea]. Disponible: https://revistapostgrado.eia.edu.co/index.php/BME/article/view/1404/1330
2. G. Zaccone, Python Parallel Programming Cookbook. Packt Publishing, Limited, 2015.
3. ”Función de convolución—ArcMap —Documentación”. https://desktop.arcgis.com/es/arcmap/latest/manage-data/raster-and-images/convolution-function.htm (accedido el 7 de junio de 2023).