Análisis de Rendimiento de Dotplot Secuencial vs. Paralelización

Daniel Antonio Giraldo Quintero

Programación Concurrente y Distribuida, Ingeniería de Sistemas y Computación
daniel.1701214080@ucaldas.edu.co

I. Introducción

La comparación de secuencias de ADN o proteínas es fundamental en bioinformática para comprender su estructura y función. Un método comúnmente utilizado para este propósito es el dotplot, que permite visualizar gráficamente la similitud entre dos secuencias .[1]

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

En la versión paralela con multiprocessing, se aprovechará la capacidad de ejecutar múltiples procesos simultáneamente en sistemas con varios núcleos de CPU. Al dividir el trabajo en tareas más pequeñas y distribuirlas 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, basada en el estándar Message Passing Interface (MPI) para la programación paralela. Las tareas de cálculo del dotplot se distribuirán entre los procesos MPI, lo que permitirá un rendimiento aún mayor en comparación con la versión paralela con multiprocessing.[2]

Al comparar estas tres 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 métodos secuenciales y paralelos, así como una comparación entre las bibliotecas multiprocessing y mpi4py para tareas de bioinformática. Los resultados podrían ser útiles para optimizar futuros proyectos de análisis comparativo de secuencias, mejorando la eficiencia en el procesamiento de volúmenes de datos genómicos y proteómicos.

Se busca contribuir al análisis de las secuencias adicionando una función de filtrado la cual indica la similitud de cada carácter de las secuencias empleadas, esto para ayudar visualmente a la evaluación por parte del personal especializado.[3]

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

II. Materiales y Métodos

A. *Librerías:*

El algoritmo se implementó utilizando el lenguaje de programación Python en su versión 3.11.7. Se emplearon varias bibliotecas clave para diversas funciones, incluyendo la manipulación de matrices, generación de gráficos, lectura de archivos, detección de diagonales mediante filtros, medición de tiempos, obtención de parámetros de línea de comandos y trabajo con múltiples procesadores. Las bibliotecas utilizadas son Numpy, Matplotlib, Time, mpi4py, OpenCV y Multiprocessing. En la Tabla 1 se detallan las versiones de cada uno de los software y bibliotecas utilizadas

Paquete	versión
Python	3.11.7
Matplotlib	3.8.0
Numpy	1.26.4
mpi4py	3.1.6
Time	3.11
Multiprocessing	3.11.7
Opencv	4.10.0.82

B. *Paralelización:*

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

Pool para generar los procesos de acuerdo con la cantidad de threads que se recibían por parametro, y por cada pool´ se manejaba la funcion map para recorrer cada índice de la primera secuencia por cada ´ındice de la segunda secuencia y realizando la comparacion de estos para guardar en una lista un numero de representacion de color, que sería la representacion gráfica que queremos obtener. Con mpi4py se utilizó la estrategia de Chucks, para dividir la primera secuencia en matrices mas pequeñas y se crea otra matriz donde se guardara la solución de la implementación, en cada recorridó de los Chuck contra las posiciones de la segunda secuencia, se efectua la misma comparación de la implementación con multiprocessing, para darle un valor de color a la posicion de la solucion, luego se unen las soluciones generadas.

C. Datos de experimentación:

Se realizaron pruebas de rendimiento utilizando archivos FASTA de Salmonella y E. coli, cada uno con aproximadamente 4 millones de bases nitrogenadas. Comparé los tiempos de ejecución entre las implementaciones secuencial, multiprocessing y mpi4py para evaluar la eficiencia de cada enfoque. Estos resultados son fundamentales para determinar qué método ofrece el procesamiento más rápido y eficiente, especialmente en análisis bioinformáticos con grandes conjuntos de datos genómicos.

D. Arquitectura computacional:

Se ejecuto en una computadora que contienen las siguientes especificaciones: Intel Core i5 con 8gb de Ram, con sistema operativo sonoma

E. Disponibilidad del algoritmo:

El software desarrollado en este proyecto está disponible de forma gratuita. Tanto el código fuente como las instrucciones para instalar y utilizar la herramienta están accesibles en: https://github.com/Danielgiraldo2010/ProjectConcurrent.git

III. RESULTADOS:

matriz conjunta con la secuencia uno y dos de 16.000 x 16.000 datos, era la matriz evaluada que se podía ejecutar

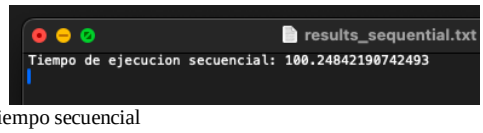

Fig. 1. Tiempo secuencial

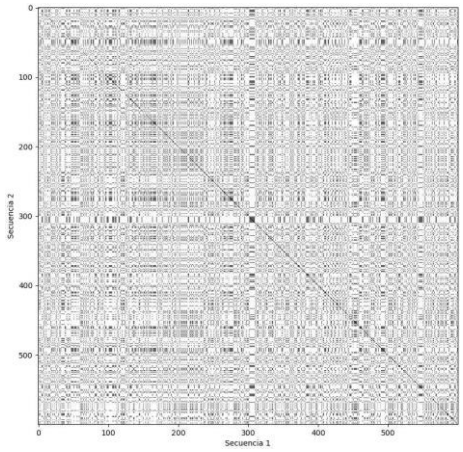

Fig. 2. Dotplot para la solución de la matriz evaluada

Con un dotplot resultante, se puede observar la diagonal principal, la cual es fundamental para el análisis comparativo de las secuencias

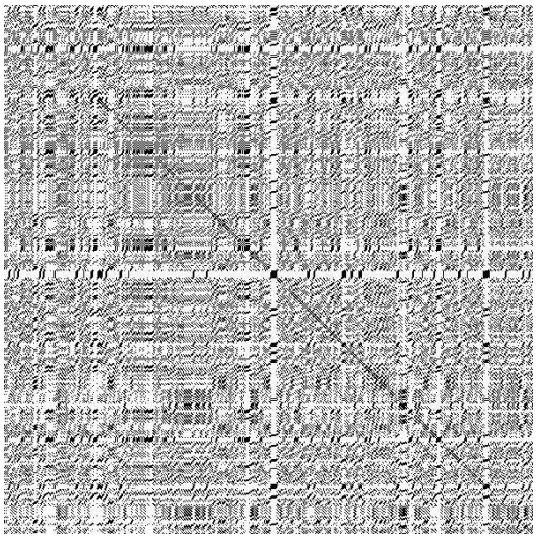

Fig. 3. Dotplot filtrado para la solución de la matriz evaluada

Al finalizar el proceso secuencial, se ejecuta nuevamente el archivo utilizando la implementación paralela con multiprocessing, lo cual arroja los siguientes datos.

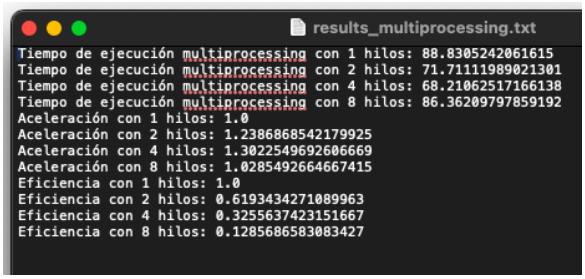

Fig. 4. Tiempos para la Escalabilidad, Aceleración y Eficiencia Multiprocessing

En la imagen anterior se puede ver cómo se escaló la implementación. Se emplearon 8 hilos, y el último hilo tuvo un tiempo de ejecución lo cual se refleja en la aceleración y la eficacia logradas durante la ejecución.

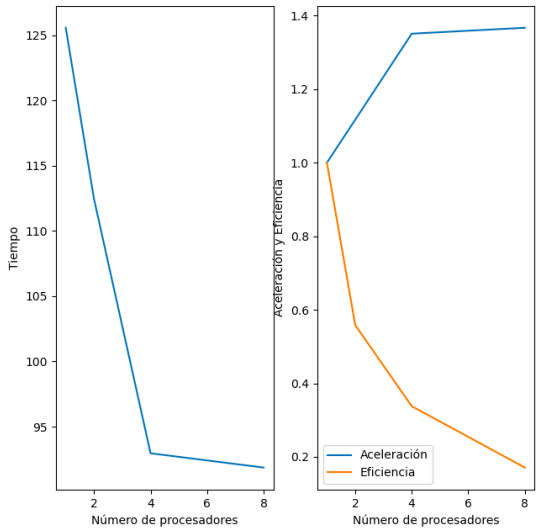

Fig 5 Gráficas de aceleración y aceleración vs eficiencia Multiprocessing

Al analizar las gráficas de comparación entre el tiempo de ejecución, la aceleración y la eficiencia, se puede observar que con un solo hilo la ejecución tomaba más tiempo en generar la solución. A medida que se incrementa el número de hilos, el tiempo de ejecución mejora. Sin embargo, cuando se utilizan demasiados hilos, debido al overhead, los tiempos vuelven a aumentar, la aceleración disminuye y la eficiencia se reduce.

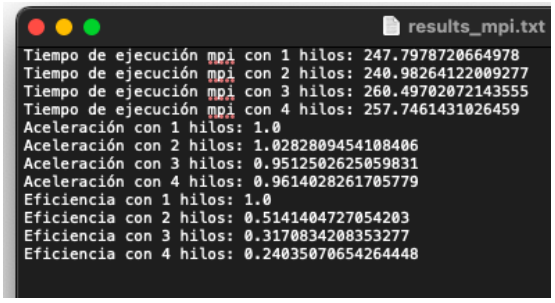

Fig 6 Tiempos para la escalabilidad, aceleración y eficiencia de MPI

La imagen anterior muestra los resultados de la ejecución del algoritmo con distintos números de núcleos, destacando las variaciones en aceleración y eficiencia.

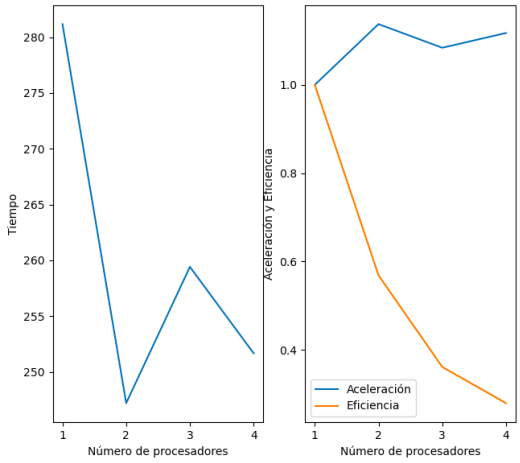

Fig 7 Gráficas de aceleración vs eficiencia MPI

En la gráfica se observa que, hasta cierto punto, el aumento en el número de núcleos mejora la velocidad de ejecución del algoritmo. Sin embargo, entre los 3 y 4 núcleos, la aceleración disminuye y la eficiencia también se reduce.

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", Evolutionary Bioinformatics, volumen 19: 1-10.º 26, pp. 37–45, 2019. Accedido el 12 de junio de 2024. [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 12 de junio de 2024).