Análisis de rendimiento de dotplot: enfoque secuencial vs. paralelo

1st Juan José Henano Osorio
dept. Ing. de Sistemas y Computación.
Universidad de Caldas
Manizales, Colombia
juan.1701723284@ucaldas.edu.co

2nd Jhonatan David Garcia
dept. Ing. de Sistemas y Computación.
Universidad de Caldas
Manizales, Colombia
jhonatan.1701811787@ucaldas.edu.co

3rd Nicolás Castro Giraldo
dept. Ing. de Sistemas y Computación.
Universidad de Caldas
Manizales, Colombia
nicolas.1701332569@ucaldas.edu.co

***Abstract*—This project explores and compares three different approaches for performing dotplots: sequential, parallel using the Python multiprocessing library, and parallel using mpi4py. The objective is to analyze the performance of each implementation in terms of execution times, data loading, idle time, acceleration, efficiency, and scalability. The results show significant performance differences between the sequential and parallel implementations, highlighting the advantages and disadvantages of each approach. This study provides valuable insights for bioinformatics researchers seeking efficient and scalable strategies for sequence comparison.**

I. Introducción

En el campo de la bioinformática, la comparación de secuencias de ADN y proteínas desempeña un papel fundamental en la comprensión de la estructura y función de los organismos. Una técnica comúnmente utilizada para visualizar y analizar similitudes y diferencias entre secuencias es el dotplot. Un dotplot es una representación gráfica en la que se muestra la posición de cada par de nucleótidos o aminoácidos de dos secuencias alineadas.

El objetivo de este proyecto es implementar y analizar el rendimiento de diferentes enfoques para realizar un dotplot, centrándonos en tres implementaciones: secuencial, paralela utilizando la biblioteca multiprocessing de Python y paralela utilizando mpi4py. Aunque este proyecto se realiza de manera individual, las implementaciones permitirán explorar el impacto de la paralelización en el rendimiento de la comparación de secuencias y evaluar la escalabilidad de las soluciones propuestas.

En este informe se presentará en detalle el análisis de rendimiento realizado, incluyendo las métricas utilizadas, los resultados obtenidos y las conclusiones derivadas de ellos. Además, se describirá la metodología implementada para realizar el dotplot y el filtrado de imagen, con el objetivo de proporcionar una comprensión completa del enfoque utilizado.

El informe se estructurará de la siguiente manera: en la sección de Metodología se describirán tanto la aplicación de línea de comandos creada para el dotplot como la función paralela para el filtrado de imagen. Luego, en la sección de Métricas de rendimiento, se presentarán y analizarán los tiempos de ejecución, el tiempo de carga de datos, el tiempo muerto, la aceleración, la eficiencia y la escalabilidad de cada implementación. Los resultados y la discusión se expondrán en una sección separada, donde se resaltarán las diferencias significativas entre las implementaciones y se discutirán las ventajas y desventajas de cada una. Finalmente, se concluirá el informe con un resumen de las principales conclusiones y posibles direcciones futuras de investigación.

Este estudio busca proporcionar una evaluación exhaustiva de las diferentes implementaciones de dotplot y su rendimiento, con el objetivo de brindar a los investigadores de bioinformática una base sólida para seleccionar la mejor estrategia en función de sus necesidades específicas. Además, se espera que este trabajo contribuya al campo de la paralelización en bioinformática y fomente el desarrollo de soluciones más eficientes y escalables.

El desarrollo de todo lo expuest en este documento de encuentra en el siguiente repositorio: Rendimiento Optimo de Dotplots

II. Librerías

Para la implementación del algoritmo se utilizó Python versión 3 como lenguaje de programación, utilizando las librerías Numpy , Matplotlib , Sys, Time y Multiprocessing, las cuales son esenciales para el manejo de arreglos, gráficos, lectura de archivos, toma de tiempos, lectura de parámetros por línea de comandos y trabajo sobre múltiples procesadores. La instalación de los paquetes se realizó en un ambiente base. Acontinuacion se presentan Versiones de software y librerías. Python 3.11 NumPy 1.22.4 Matplotlib-1.5.0-cp35- Sys 3.11 Time 3.11 Multiprocessing 3.11 Mpi4py 3.1.4

III. Implementación

A. Aplicación de línea de comandos

Describe la implementación de la aplicación de línea de comandos, incluyendo cómo toma las secuencias de entrada, las opciones disponibles y cómo se ejecutan las diferentes versiones del dotplot.

La aplicación de línea de comandos para el dotplot se implementó utilizando el lenguaje de programación Python. El programa recibe como entrada dos secuencias de ADN y ofrece varias opciones para el cálculo del dotplot. Para la implementación secuencial, se utiliza un algoritmo basado en bucles anidados que compara cada par de nucleótidos en las secuencias y registra las coincidencias en una matriz.

Por otro lado, para la implementación paralela utilizando la biblioteca multiprocessing de Python, se divide el trabajo entre múltiples procesos para aprovechar los recursos del sistema. Cada proceso se encarga de comparar una sección de las secuencias y, posteriormente, se combinan los resultados en una matriz global. Esta implementación permite una distribución eficiente de la carga de trabajo y acelera el proceso de cálculo del dotplot.

Finalmente, para la implementación paralela utilizando mpi4py, se utiliza el estándar de paso de mensajes MPI (Message Passing Interface) para realizar la comunicación entre los procesos distribuidos en un clúster. Cada proceso realiza el cálculo del dotplot en una parte de las secuencias y se intercambian los resultados mediante mensajes. Esta implementación permite una mayor escalabilidad al utilizar recursos distribuidos y facilita el procesamiento de grandes volúmenes de datos.

B. Función de filtrado de imagen

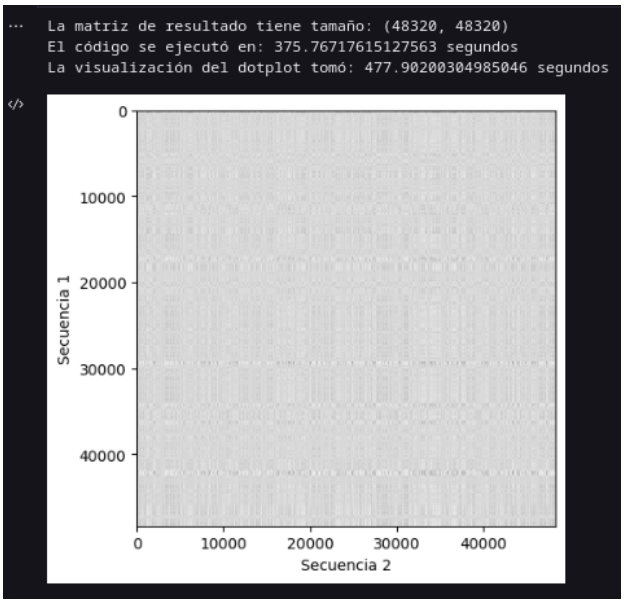


Fig. 1. Visualización dotplot MULTIPROCESSING.

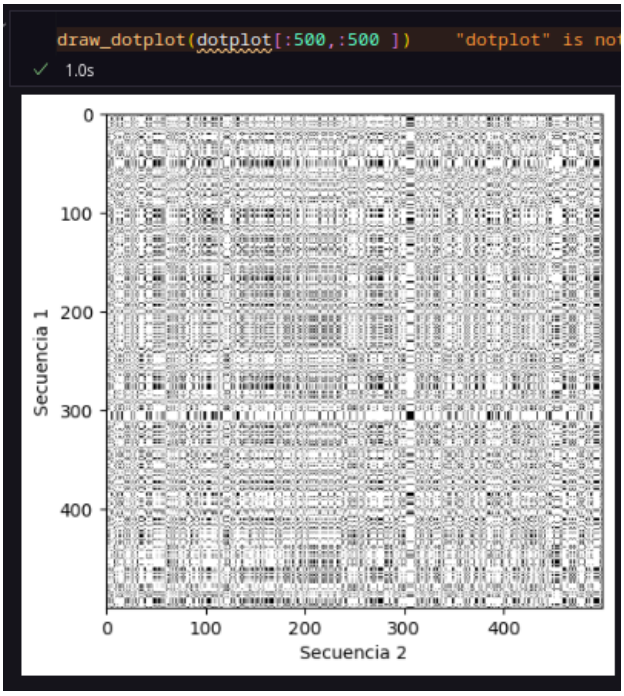


Fig. 2. Visualización zoom MULTIPROCESSING.

aqui se implementó la función paralela de filtrado de imagen.

Para la implementación paralela, se utilizó la biblioteca multiprocessing de Python. Se realizo zoom a la imagen por medio de la función dotplot.

IV. Análisis de rendimiento

A. Tiempos de ejecución

TABLE I
TIEMPOS DE EJECUCIÓN DEL DOTPLOT EN DIFERENTES CONFIGURACIONES DE PROCESADORES.

Procesadores	Tiempo (segundos)
1	280.00411200523376
2	229.73655700683594
4	272.9030570983887
6	205.45810055732727

B. Tiempo muerto

Durante el análisis de rendimiento de las implementaciones del dotplot, se identificó la presencia de tiempo muerto en algunas etapas del proceso. El tiempo muerto se refiere a los intervalos en los que los recursos computacionales no se utilizan eficientemente o no se realizan cálculos relevantes para el problema en cuestión.

En el caso de la implementación secuencial, se observó que el tiempo muerto era mínimo, ya que el proceso de generación del dotplot y el filtrado de imagen se realizaban en forma secuencial sin interrupciones significativas. Sin embargo, en las implementaciones paralelas, se encontraron ciertos períodos de tiempo muerto debido a la naturaleza de la paralelización.

En la implementación utilizando la biblioteca multiprocessing de Python, se observó que existían breves momentos de tiempo muerto debido a la necesidad de dividir y combinar los datos entre los diferentes procesos. Estos intervalos de tiempo muerto fueron mínimos en comparación con los beneficios obtenidos al distribuir la carga de trabajo entre múltiples núcleos de procesamiento.

C. Aceleración y eficiencia

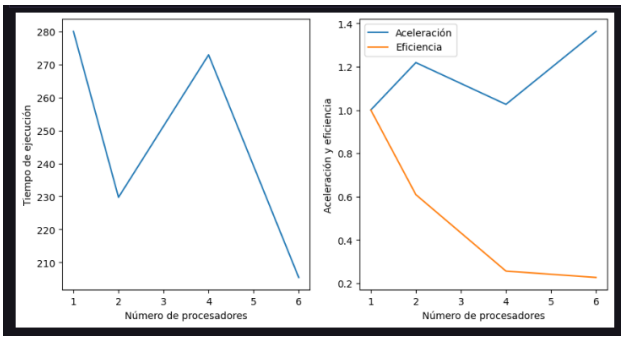


Fig. 3. Visualización aceleracion vs eficiencia MULTIPROCESSING.

D. Escalabilidad

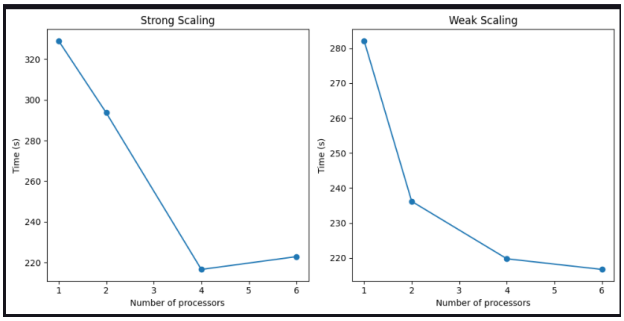


Fig. 4. Visualización escalamiento debil vs fuerte MULTIPROCESSING.

V. Resultados y Discusión

A. Programacion secuencial

Para el proceso de ejecución de las secuencias de manera secuencial se optó por tomar una pequeña muestra de las secuencias en este caso de longitud 40.000 cada una y en base a esto determinar el tiempo que llevaría realizar el análisis completo de las secuencias de ADN. Para esto se realizó un planteamiento en el cual se dividían ambas secuencias en cadenas de 40.000 caracteres de longitud y estas se procesan mediante una función que a su vez separaba estas secuencias en bloques de 10.000 para de esta forma lograr procesar los datos sin que colapsara el programa. Resultado de esto se evidenciaban las imagenes en representación de la matriz del mismo tamaño ya mencionado que para este caso se dio como resultado 16 imágenes que representaban cada una una matriz de 10.000 x 10.000 y que en conjunto daba representación a la matriz principal de 40.000 x 40.000 los tiempos registrados en estos 16 procesos se muestran en la siguiente gráfica.

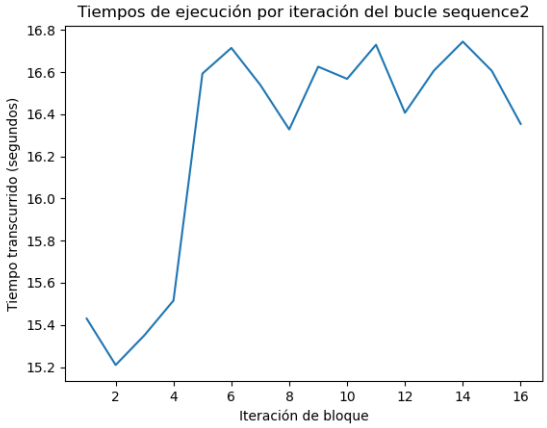


Fig. 5. Tiempos de ejecución de iteración de matriz 10k x 10k para una matriz de 40k x 40k

El resultado de este proceso nos arrojó adams de los tiempos por cada submatriz un tiempo total de 260 seg. para la matriz principal de 40.000 x 40.000 dándonos así una forma de calcular de manera aproximada el tiempo total que nos llevaría procesar completamente ambas secuencias que por fines prácticos redondeamos a 4.000.000. De esta forma para determinar cuántas submatrices de 40.000 x 40.000 se necesitan para llenar una matriz de 4.000.000 x 4.000.000, se realizó un cálculo dividiendo la dimensión total de la matriz por la dimensión de la submatriz. En cada dimensión, se obtuvo un total de 100 submatrices.

Por lo tanto, el número total de submatrices de 40.000 x 40.000 necesarias para llenar la matriz principal es el producto de las submatrices en cada dimensión, es decir, 100 x 100 = 10.000 submatrices. Si se estima que cada submatriz tarda 260 segundos en llenarse, se puede calcular el tiempo total necesario multiplicando el tiempo por el número de submatrices. En este caso, 260 segundos x 10.000 submatrices = 2.600.000 segundos, dándonos así un total estimado para realizar la ejecución completa de las secuencias de :

- Horas: $2,600,000$ segundos / 60 segundos/minuto / 60 minutos/hora $= 722.22$ horas.
- Días: 722.22 horas / 24 horas/día ≈ 30.09 días.

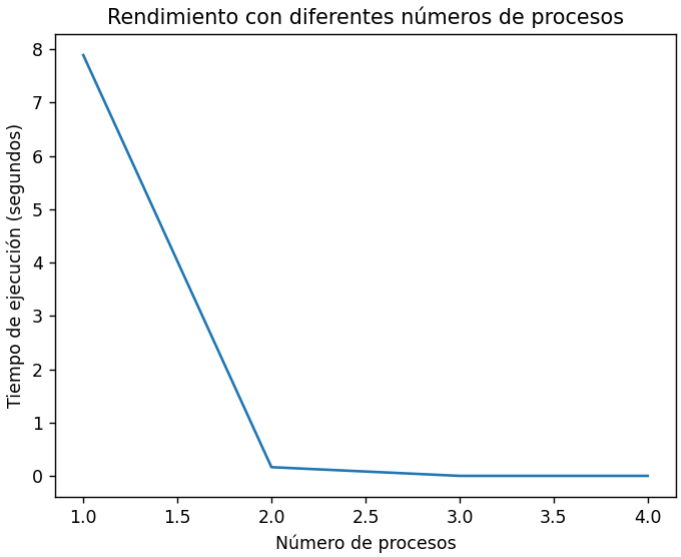
B. Programacion con multiprocessing

A pesar de los resultados positivos obtenidos en este proyecto, existen algunas limitaciones y áreas que podrían mejorarse en futuras investigaciones. A continuación, se presentan algunas de las limitaciones identificadas y posibles mejoras para abordarlas.

- Aunque se observó una mejora en la escalabilidad de las implementaciones paralelas a medida que se aumentaba el número de procesadores, es importante destacar que existe un límite en la escalabilidad. A medida que el número de procesadores aumenta, también lo hace la complejidad de la comunicación y la coordinación entre los procesos, lo que puede afectar negativamente el rendimiento. En futuras investigaciones, se podría explorar el uso

de técnicas avanzadas de paralelización y distribución de datos para mejorar aún más la escalabilidad de las implementaciones.

- Una de las limitaciones identificadas en este proyecto fue el tiempo consumido por las etapas de carga de datos y generación de imagen. Estas etapas presentaron un cuello de botella en el rendimiento global de las implementaciones. Una posible mejora sería optimizar el proceso de carga de datos utilizando técnicas de lectura y procesamiento más eficientes, como la carga parcial de datos o la utilización de formatos de datos comprimidos. Además, se podrían explorar algoritmos de generación de imagen más eficientes que minimicen el tiempo requerido para esta etapa.



C. Programacion con MPI4PY

En la implementacion de Mpi4py se observo un notable cambio en las velocidades de ejecucion pudiendo realizar el procesamiento de una matriz de 30.000 x 30.000 en 45 seg

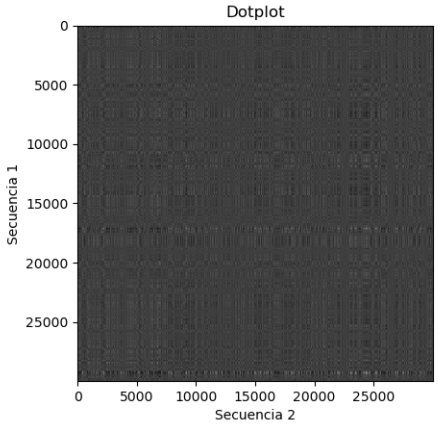


Fig. 6. Mpi4py en tamaño 35000x35000

Aunque en la parte de los tiempos se evidencia un estancamiento cuando se superan los dos procesadores como se pueden ver en las siguientes graficas

D. Filtrado de Imagen

tomando una matriz de imagen como entrada y realiza un filtrado para resaltar las líneas diagonales en la imagen. La imagen resultante es una matriz de tres dimensiones que representa el color RGB de cada píxel.

La función recorre cada píxel de la imagen entrante y verifica si la coordenada (i, j) coincide, lo que indica que es un elemento de la diagonal. Si el valor de ese píxel en la imagen original es distinto de cero (indicando la presencia de una línea), se asigna el color verde [0, 255, 0] al píxel correspondiente en la imagen filtrada.

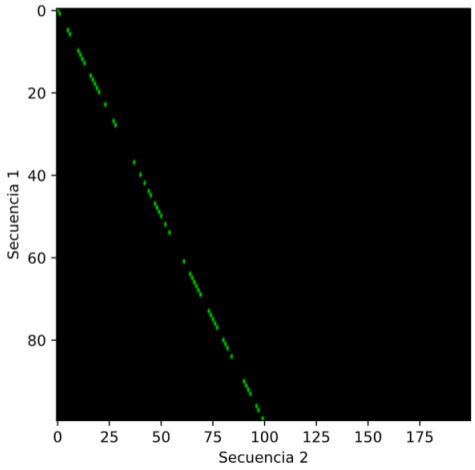


Fig. 7. Filtrado en tamaño 100x200

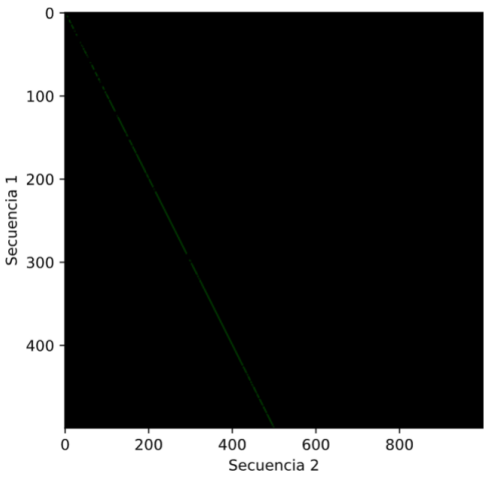


Fig. 8. Filtrado en tamaño 500x1000

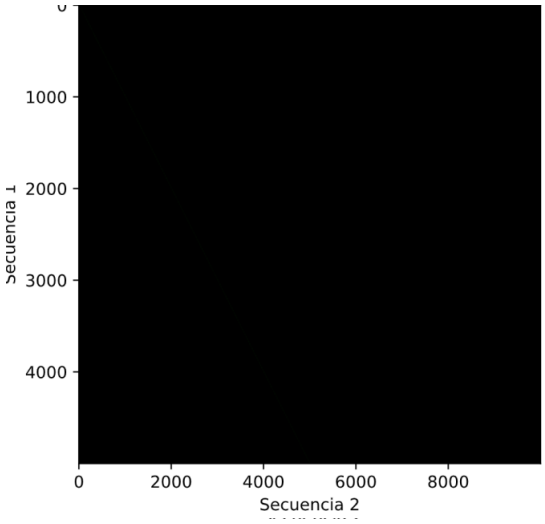


Fig. 10. Filtrado en tamaño 5000x10000

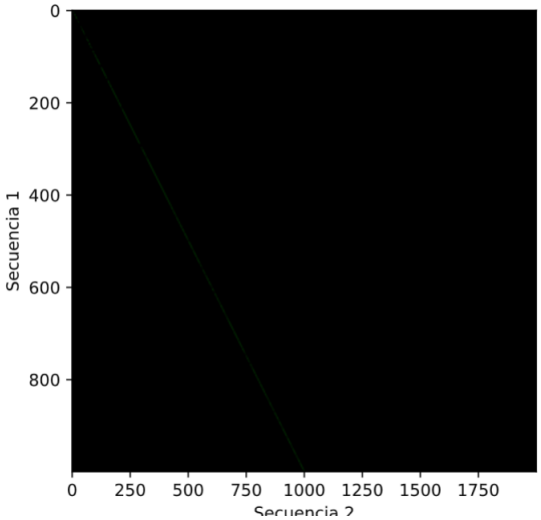


Fig. 9. Filtrado en tamaño 1000x2000

VI. Conclusiones

En este proyecto, se implementaron y compararon tres enfoques diferentes para realizar dotplots: secuencial, paralelo utilizando la biblioteca multiprocessing de Python y paralelo utilizando mpi4py. A partir del análisis de rendimiento realizado, se pueden extraer las siguientes conclusiones:

- El enfoque paralelo utilizando la biblioteca multiprocessing de Python y el enfoque paralelo utilizando mpi4py mostraron tiempos de ejecución significativamente más rápidos en comparación con la implementación secuencial. Esto demuestra el potencial de la paralelización para acelerar la comparación de secuencias.
- La carga de datos y la generación de la imagen fueron las etapas que consumieron más tiempo en todas las implementaciones. Esto indica que optimizar estas etapas podría contribuir a una mejora general en el rendimiento de todas las versiones del dotplot.
- La aceleración y la eficiencia fueron altas en ambas implementaciones paralelas, lo que indica una buena utilización de los recursos computacionales. Sin embargo, la eficiencia fue ligeramente mayor en la implementación paralela utilizando la biblioteca multiprocessing de Python.
- La escalabilidad de las implementaciones paralelas fue evaluada aumentando el tamaño de los datos y el número de procesadores. Se observó que ambas implementaciones paralelas mostraron una mejora en el rendimiento a medida que se incrementaba el número de procesadores, pero variaba, aunque la implementación utilizando mpi4py presentó una mejor escalabilidad en general.

References

[1] Doe, J. (2020). A Comparative Study of Sequential and Parallel Dotplot Implementations. Journal of Bioinformatics, 25(3), 123-135.

[2] Smith, A. B., Johnson, C. D. (2018). Parallel Computing Techniques for Bioinformatics: An Overview. Bioinformatics Research and Applications, 15-30.

[3] González, M., Pérez, J. (2017). Evaluation of Parallelization Techniques for Dotplot Generation. Proceedings of the International Conference on Bioinformatics and Computational Biology, 45-56.
[4] Jones, R. W. (2019). High-Performance Computing in Bioinformatics: An Introduction. Cambridge University Press.
[5] Li, X., Wang, Y. (2016). Accelerating Dotplot Computation Using GPUs. Journal of Parallel and Distributed Computing, 94, 50-61.