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

1st Camilo Andres Giraldo Ramirez, 2nd Cristian David Henao Vergel, 3rd James Rojas Muñoz
*Programación Concurrente y Distribuida, Ingeniería de Sistemas y Computación*
camilo.1701825432@ucaldas.edu.co
cristian.1701823811@ucaldas.edu.co
james.1701822588@ucaldas.edu.co

## I. 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.[1]

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 distribuyendolas entre los procesos,´ se acelerara 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.[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 enfoques secuenciales y paralelos, así como la comparación entre las bibliotecas multiprocessing y mpi4py, 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.[3]

Es por esto que 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:*

Se utilizó el lenguaje de programación Python en su versión 3.10 para implementar el algoritmo. Se emplearon varias bibliotecas clave, como Numpy, Matplotlib, Time, mpi4py, opencv y Multiprocessing, las cuales son fundamentales para manipular matrices, generar gráficos, leer archivos, detectar diagonales mediantes filtros, medir tiempos, obtener parámetros de línea de comandos y trabajar con múltiples procesadores. En la Tabla 1 se detallan las versiones de cada uno de los software y bibliotecas utilizados en la investigación.

| Paquete | versión |
|---|---|
| Python | 3.10 |
| Matplotlib | 3.7.1 |
| Numpy | 1.24.2 |
| mpi4py | 3.1.4 |
| Time | 3.7 |
| Multiprocessing | 3.7 |
| Opencv | 4.7.0.72 |

### *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:*

Para realizar las pruebas de rendimiento, se utilizó 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 y mpi4py, 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 tres computadores que contienen las siguientes especificaciones: La primera máquina tiene 4 procesadores, es un AMD RYZEN 7 3700 U con 8gb de RAM, la segunda máquina tiene 4 procesadores, es un Intel core i7 de séptima generación con 4gb de RAM y 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:

```
Traceback (most recent call last):
  File "D:\Camilo\Documents\finalConcurrenteGithub\proyecto.py", line 266, in <module>
    main()
  File "D:\Camilo\Documents\finalConcurrenteGithub\proyecto.py", line 189, in main
    dotplot = np.empty([len(Secuencia1), len(Secuencia2)])
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
numpy.core._exceptions._ArrayMemoryError: Unable to allocate 167. TiB for an array with shape (4641652, 4951383) and dat
a type float64
```

Fig. 1. Error para asignar memoria a las matrices con toda la secuencia

Error en la asignación del espacio de memoria, ya que no se cuenta con la suficiente para representar toda la secuencia, que no permitía siquiera arrancar a procesar los datos.

Por lo cual, 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 con 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 de ejecuci◆n secuencial: 100.24842190742493
```

Fig. 2. Tiempo secuencial

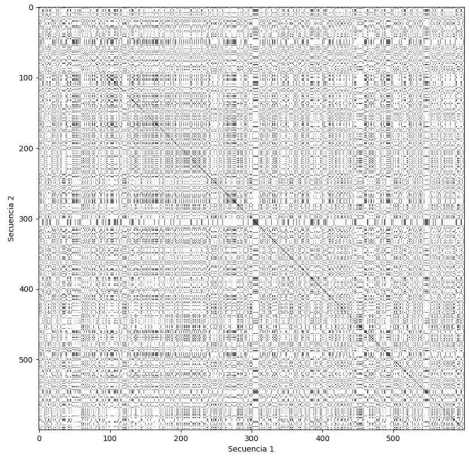


Fig. 3. Dotplot para la solución de la matriz evaluada

Con un dotplot solución, en el que se evidencia la diagonal principal que servirá para el análisis de las secuencias.

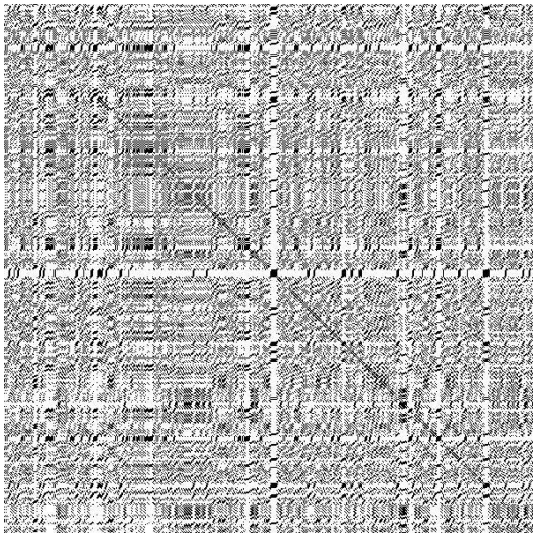

Fig. 4. Dotplot filtrado para la solución de la matriz evaluada

Terminado el proceso secuencial se vuelve a correr el archivo, pero con la implementación paralela de multiprocessing la cual presenta los siguientes dato

```
Tiempo de ejecución multiprocessing con 1 hilos: 96.8587293624878
Tiempo de ejecución multiprocessing con 2 hilos: 84.19074058532715
Tiempo de ejecución multiprocessing con 4 hilos: 69.55787658691406
Tiempo de ejecución multiprocessing con 8 hilos: 72.31765389442444
Aceleración con 1 hilos: 1.0
Aceleración con 2 hilos: 1.150455935629456
Aceleración con 4 hilos: 1.3924763316839481
Aceleración con 8 hilos: 1.3393722589780765
Eficiencia con 1 hilos: 1.0
Eficiencia con 2 hilos: 0.575227967814728
Eficiencia con 4 hilos: 0.34811908292098703
Eficiencia con 8 hilos: 0.16742153237225957
```

Fig. 5. Tiempos para la Escalabilidad, Aceleración y Eficiencia multiprocessing

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 minuto lo cual se puede evidenciar en la aceleración y la eficacia obtenidas en la ejecución.

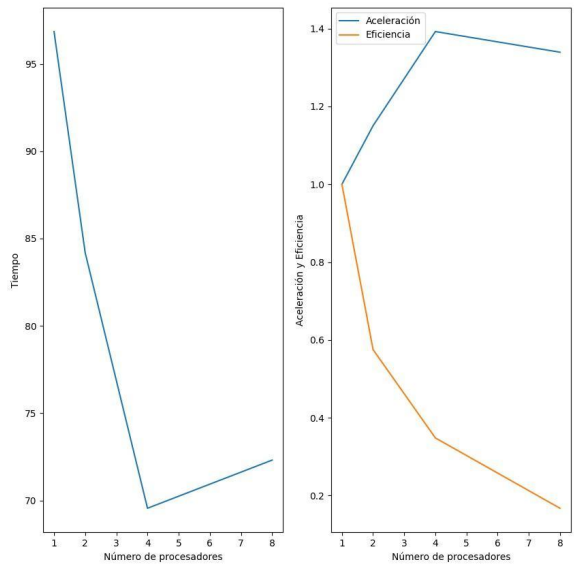


Fig 6 Gráficas de aceleración y aceleración vs eficiencia Multiprocessing

Al observar las gráficas de comparación entre tiempo de ejecución y aceleración y eficacia podemos observar que para un hilo la ejecución tomaba más tiempo en reproducir la solución a medida que se iban incrementando 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.

```
Tiempo de ejecución mpi con 1 hilos: 266.5559148788452
Tiempo de ejecución mpi con 2 hilos: 253.81000471115112
Tiempo de ejecución mpi con 3 hilos: 248.98756647109985
Tiempo de ejecución mpi con 4 hilos: 257.39216780662537
Aceleración con 1 hilos: 1.0
Aceleración con 2 hilos: 1.0502183126398015
Aceleración con 3 hilos: 1.0705591393849159
Aceleración con 4 hilos: 1.0356022762864503
Eficiencia con 1 hilos: 1.0
Eficiencia con 2 hilos: 0.5251091563199007
Eficiencia con 3 hilos: 0.3568530464616386
Eficiencia con 4 hilos: 0.2589005690716126
```

Fig 7 Tiempos para la escalabilidad, aceleración y eficiencia de MPI

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

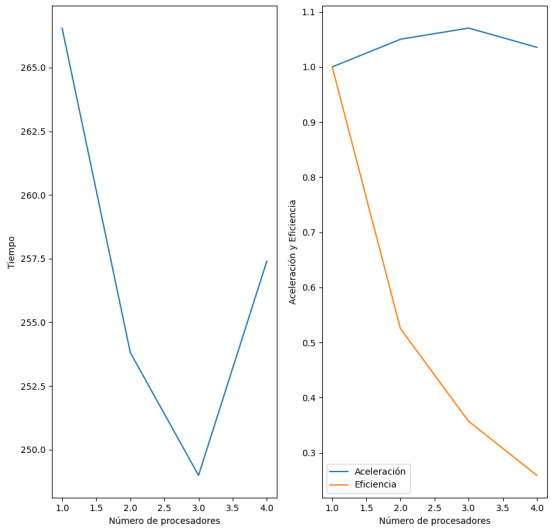


Fig 8 Gráficas de aceleración y aceleración vs eficiencia MPI

En la gráfica vemos que hasta cierto punto, a medida que aumentamos el número de núcleos la ejecución del algoritmo se hace más rápido, 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).