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

Morales S. Juan Camilo, Ospina G. Juan Felipe

***Abstract*—El objetivo de este proyecto es implementar y analizar el rendimiento de tres formas de realizar un dotplot, una técnica comúnmente utilizada en bioinformática para comparar secuencias de ADN o proteínas.**

Esto se realizó a través de una versión secuencial, una versión paralela utilizando la biblioteca multiprocessing de Python, y una versión paralela utilizando mpi4py. Compararemos el rendimiento de estas tres implementaciones utilizando varias métricas y visualizarán sus resultados en un informe estilo artículo científico en formato IEEE.

***Index Terms*—Multiprocessing, MPI4py, python, Numpy, dotplot.**

I. Introducción

La matriz que muestra la similitud entre dos cadenas de ADN diferentes se llama matriz de similitud o matriz de puntuación. Esta matriz asigna un valor numérico a cada posible par de nucleótidos (A, T, C, G - Figura 1) para cuantificar la similitud o diferencia entre ellos. Los valores en la matriz representan la puntuación o penalización asignada a cada par de nucleótidos en función de su similitud o diferencia. Esta matriz se utiliza en algoritmos de alineación de secuencias para encontrar la mejor coincidencia entre las secuencias de ADN y determinar su grado de similitud. Un ejemplo común de matriz de similitud es la matriz de sustitución de BLOSUM (Figura 2) o PAM (Figura 3) utilizada en la alineación de secuencias de proteínas.

Se creó entonces una aplicación que toma dos secuencias (en formato de archivos FASTA) como entrada y producir un dotplot de las secuencias. La aplicación debe ser capaz de ejecutar el dotplot de tres maneras: secuencialmente, utilizando multiprocessing, y utilizando mpi4py.

II. Implementaciones

A. Consideraciones

Para el desarrollo de este proyecto, se tuvieron en cuenta aspectos que consideramos relevantes, como lo son el tamaño de los valores a comparar, la capacidad de procesamiento y el tiempo a invertir. Aspectos que iremos mencionando en el transcurso de este documento iremos explicando más detenidamente.

Resaltaremos entonces que el objetivo principal del proyecto es encontrar las similitudes entre dos cadenas de ADN mediante una gráfica de matriz que se forma por la comparación de las bases (Figura 1) en donde estas sean iguales, es decir, cuando se encuentre A - A en una misma posición, en la matriz se debería mostrar como un espacio lleno de color negro.

Símbolo[1]	Descripción	Bases representadas					Complemento
A	Adenina	A					T
C	Citosina		C				G
G	Guanina			G		1	C
T	Timina				T		A
U	Uracilo				U		A

Figura 1. Tabla de Bases ADN

	C	S	T	A	G	P	D	E	Q	N	H	R	K	M	I	L	V	W	Y	F	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
A	0	1	0	4																	A
G	-3	0	-2	0	6																G
P	-3	-1	-1	-1	-2	7															P
D	-3	0	-1	-2	-1	-1	6														D
E	-4	0	-1	-1	-2	-1	2	5													E
Q	-3	0	-1	-1	-2	-1	0	2	5												Q
N	-3	1	0	-2	0	-2	1	0	0	6											N
H	-3	-1	-2	-2	-2	-2	-1	0	0	1	8										H
R	-3	-1	-1	-1	-2	-2	-2	0	1	0	0	5									R
K	-3	0	-1	-1	-2	-1	-1	1	1	0	-1	2	5								K
M	-1	-1	-1	-1	-3	-2	-3	-2	0	-2	-2	-1	-1	5							M
I	-1	-2	-1	-1	-4	-3	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-1	-4	-3	-4	-3	-2	-3	-3	-2	-2	2	2	4					L
V	-1	-2	0	0	-3	-2	-3	-2	-2	-3	-3	-3	-2	1	3	1	4				V
W	-2	-3	-2	-3	-2	-4	-4	-3	-2	-4	-2	-3	-3	-1	-3	-2	-3	11			W
Y	-2	-2	-2	-2	-3	-3	-3	-2	-1	-2	2	-2	-2	-1	-1	-1	-1	2	7		Y
F	-2	-2	-2	-2	-3	-4	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	1	3	6	F
	C	S	T	A	G	P	D	E	Q	N	H	R	K	M	I	L	V	W	Y	F	

Figura 2 - Matriz Blosum

Original amino acid

Replacement amino acid	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	Sum
A	9867	2	9	10	3	8	17	21	2	6	4	2	6	2	22	35	32	0	2	18	1.007
R	1	9913	1	0	1	10	0	0	10	3	1	19	4	1	4	6	1	8	0	1	0.998
N	4	1	9822	36	0	4	6	6	21	3	1	13	0	1	2	20	9	1	4	1	0.996
D	6	0	42	9859	0	6	53	6	4	1	0	3	0	0	1	5	3	0	0	1	0.999
C	1	1	0	0	9973	0	0	0	1	1	0	0	0	0	1	5	1	0	3	2	0.999
Q	3	9	4	5	0	9876	27	1	23	1	2	6	4	0	6	2	2	0	0	1	0.997
E	10	0	7	56	0	35	9865	4	2	3	1	4	1	0	3	4	2	0	1	2	1.000
G	21	1	12	11	1	3	7	9935	1	0	1	2	1	1	3	21	3	0	0	5	1.003
H	1	8	18	3	1	20	1	0	9912	0	1	1	0	2	3	1	1	1	4	1	0.998
I	2	2	3	1	2	1	2	0	0	9872	9	2	12	7	0	1	7	0	1	33	0.996
L	3	1	3	0	0	6	1	1	4	22	9947	2	45	13	3	1	3	4	2	15	1.008
K	2	37	25	6	0	12	7	2	2	4	1	9926	20	0	3	8	11	0	1	1	1.007
M	1	1	0	0	0	2	0	0	0	5	8	4	9874	1	0	1	2	0	0	4	0.990
F	1	1	1	0	0	0	0	1	2	8	6	0	4	9946	0	2	1	3	28	0	1.000
P	13	5	2	1	1	8	3	2	5	1	2	2	1	1	9926	12	4	0	0	2	0.999
S	28	11	34	7	11	4	6	16	2	2	1	7	4	3	17	9840	38	5	2	2	1.004
T	22	2	13	4	1	3	2	2	1	11	2	8	6	1	5	32	9871	0	2	9	1.000
W	0	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	9976	1	0	0.998
Y	1	0	3	0	3	0	1	0	4	1	1	0	0	21	0	1	1	2	9945	1	0.999
V	13	2	1	1	3	2	2	3	3	57	11	1	17	1	3	2	10	0	2	9901	1.004
Sum	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	

Figura 3 - Matriz PAM

Las secuencias de ADN con las que se trabajaron para el desarrollo de las soluciones secuencial y paralelas fueron E-Coli y la Salmonella. Que son básicamente archivos FASTA (extensión .fna) de 4,7 MB de tamaño en promedio.

Camilo Morales, (email:juan.1701622573@ucaldas.edu.co).
Juan Ospina, (email: juan.1701720807@ucaldas.edu.co).

B. Lenguaje y Librerías

Python es un lenguaje de programación poderoso y versátil, el módulo multiprocessing permite la programación paralela y concurrente en Python, y MPI4py es una biblioteca que implementa el estándar MPI para desarrollar aplicaciones paralelas y distribuidas en Python. Estas herramientas son fundamentales para aprovechar el poder de procesamiento de sistemas multiprocesador y clústeres de computadoras, mejorando el rendimiento y la eficiencia.

1) Python: Es un lenguaje de programación de alto nivel y fácil de aprender. Es ampliamente utilizado en diversos campos, incluyendo desarrollo web, análisis de datos, inteligencia artificial y computación científica. Python es conocido por su sintaxis legible y su amplia gama de bibliotecas y frameworks.

2) Multiprocessing: Es un módulo incorporado en Python que proporciona funcionalidades para la programación paralela y concurrente. Permite la ejecución de tareas en paralelo utilizando múltiples procesos en lugar de un solo proceso. Esto es especialmente útil para tareas intensivas en CPU que se pueden dividir en subprocesos independientes.

3) MPI4py: Es una biblioteca de Python que implementa el estándar Message Passing Interface (MPI). MPI es un modelo de programación utilizado para desarrollar aplicaciones paralelas y distribuidas en sistemas de memoria distribuida. Permite la comunicación y la coordinación entre múltiples procesos que se ejecutan en diferentes nodos de un clúster de computadoras.

Algunas de las otras librerías que utilizamos son las siguientes:

- NumPy: Biblioteca para computación numérica en Python, con estructuras eficientes y funciones matemáticas de alto nivel.
- Matplotlib.pyplot: Biblioteca de visualización en Python para crear gráficos y visualizaciones.
- Time: Módulo para trabajar con el tiempo, medir el tiempo de ejecución y manipular fechas y horas.
- Bio import SeqIO: Parte de Biopython, permite leer, escribir y manipular datos de secuencias biológicas.
- CV2: Biblioteca de visión por computadora (OpenCV) para procesamiento de imágenes y análisis de video.
- Argparse: Módulo para analizar argumentos de línea de comandos en programas Python.
- Tqdm import tqdm: Biblioteca para mostrar una barra de progreso en bucles iterativos en Python.
- Numba: Biblioteca para compilar funciones y bucles numéricos en Python para obtener un rendimiento más rápido.

C. Práctica

Para la ejecución de la aplicación se solicitaba que fuera a través de una línea de comandos, en el que se pudiera seleccionar la cantidad de procesadores que se soliciten.

Después de empezar a implementar las cadenas de ADN de prueba, nos enfrentamos al problema de que la memoria de las máquinas utilizadas para el procesamiento se llenaba y colapsaba por la cantidad de información procesada. Por lo que también se hizo necesario poder seleccionar el porcentaje total de información que se va a procesar.

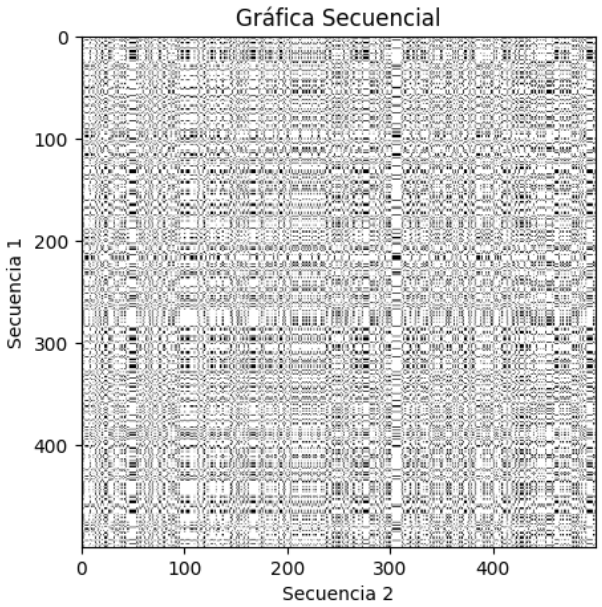
Una vez realizadas todas las configuraciones en el código necesarias para cumplir con estos requerimientos, la instrucción que se debe ejecutar luce así:

```
!python appDotplot.py --file1=Salmonella.fna --file2=E_coli.fna --ps 0.2 --threads 2 4 8 --mpi
```

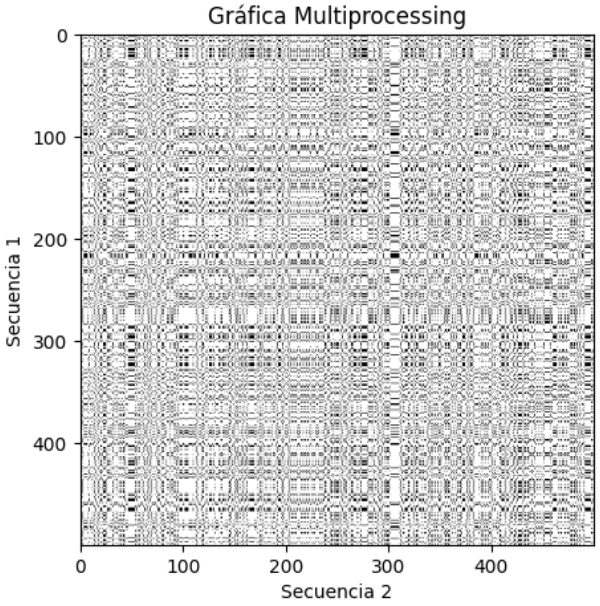
Por efectos prácticos, en nuestro caso decidimos usar el 20% de todos los datos, y hacer un zoom en la matriz resultante con un tamaño de 500x500, esto con el fin de agilizar los procesos y poder visualizar los cambios.

Encontramos entonces, que efectivamente las gráficas que generan cada una de las librerías no cambia o no presentan alteraciones como lo vamos a ver a continuación, sin embargo, la diferencia fundamental rádica el la cantidad de tiempo que se toma cada proceso en realizar las operaciones.

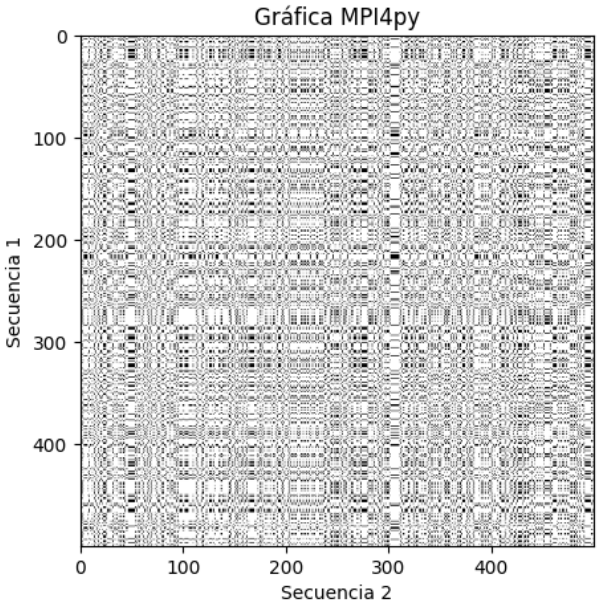
1) Secuencial: En este caso, corresponde a realizar los procesamientos a pura fuerza brutal y sin paralelización.



2) Multiprocessing: En el caso de multiprocessing, corresponde a una de las librerías comentadas anteriormente, que básicamente divide las tareas por la cantidad de procesadores ingresados.

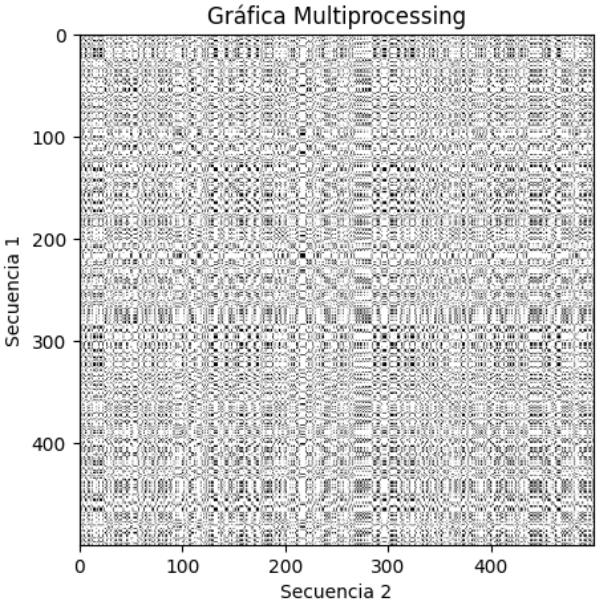


3) MPI4py: En el caso de MPI4py, esta librería permite la comunicación y la coordinación entre múltiples procesos que se ejecutan en diferentes nodos.

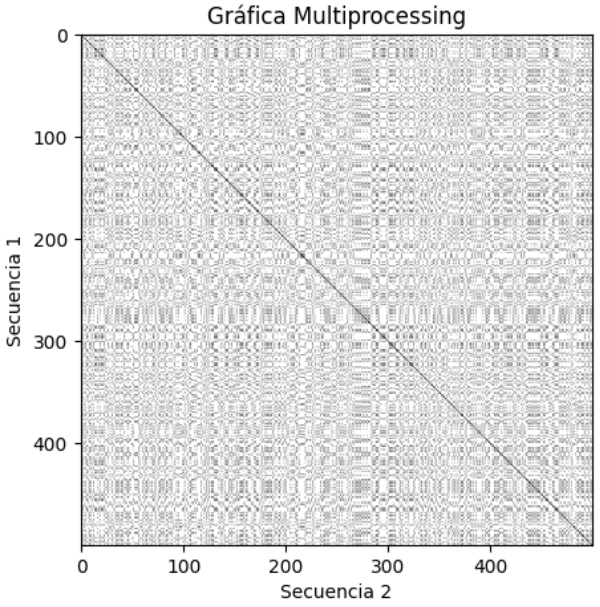


4) Filtro: Dado el caso se encontrara dos secuencias parecidas en ciertos puntos, se requería implementar un filtro, que cambiara los valores en la matriz, si el valor en la matriz era 0 dejarlo en blanco, si había coincidencia y era diagonal se pintará de negro, si no que se pintará de gris.

Este sería el resultado sin filtrar:



Y este sería el resultado filtrado como lo especificado:

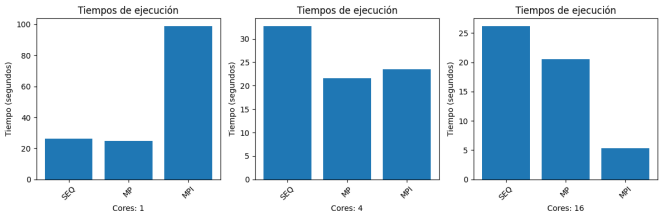


III. Tiempos y Aceleración

A continuación analizaremos los tiempos, la aceleración y eficiencia total de los tiempos de ejecución dependiendo la cantidad de procesadores ingresados.

A. Tiempos

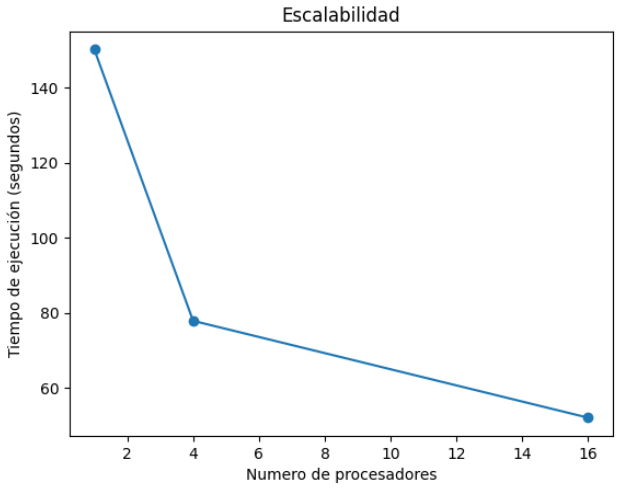
En esta sección revisaremos la gráfica de tiempos que está divida por la cantidad de procesadores en las cuales está representado la librería utilizada y el tiempo total utilizado. Para la muestra hemos utilizado 2, 4 y 16 cores.



Como podemos observar el claro ganador a mayor número de procesadores es MPI, pero vemos que presenta una falencia grande al momento de presentar pocos procesadores.

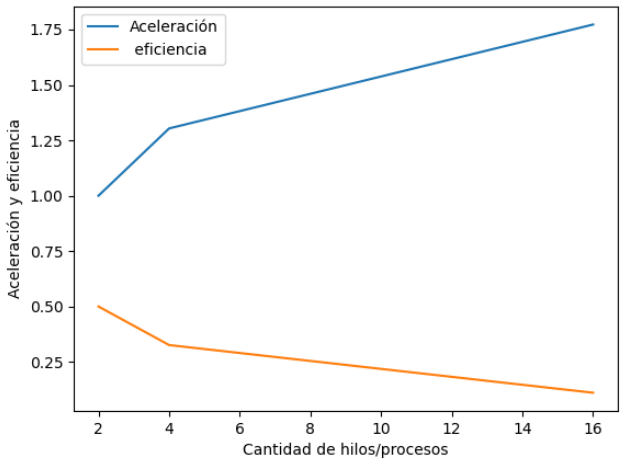
B. Rendimiento

Python puede presentar limitaciones en aceleración y escalabilidad debido a su naturaleza de alto nivel. Multiprocessing puede proporcionar cierta aceleración al aprovechar múltiples núcleos, pero la escalabilidad puede verse limitada. MPI4py ofrece una mejor aceleración y escalabilidad al aprovechar sistemas de memoria distribuida y comunicación eficiente entre procesos.



Como vemos al aumentar la cantidad de cores el tiempo tiende a disminuir, hasta llegar a un punto en el que el tiempo por más que se quisiera no podría disminuir.

La elección de la biblioteca de paralelización dependerá de los requisitos y características específicas del proyecto, así como de la arquitectura del sistema en el que se ejecutará y la cantidad de código que sea posible paralelizar.



IV. Conclusión

En resumen, si lo que se busca es aceleración y escalabilidad y se tiene la capacidad para aumentar el número de núcleos, las bibliotecas especializadas como multiprocessing y MPI4py suelen ser más efectivas que el uso directo de Python. Multiprocessing es útil para aprovechar múltiples núcleos en un solo sistema, mientras que MPI4py es más adecuado para la programación paralela y distribuida en sistemas de memoria distribuida. La elección de la biblioteca depende de la naturaleza y los requisitos de tu proyecto, así como de la arquitectura del sistema en el que se ejecutará.

Agradecimientos

Gracias al profesor Reinel Tabares Soto por la materia impartida. A Google por prestarnos sus equipos. Y a OpenAI por despejarnos dudas.