Informe Proyecto Final HPC

Integrantes:

Jhon Eduard Rivas Mancilla
Alejandro Salazar Gonzalez

Profesor:
Reinel Tabares

Revisor:
Johan Piña

Facultad de Ingenierías, Universidad de Caldas

01 de Diciembre de 2024

Manizales

Tabla de Contenido

1. Introducción

A través del presente documento se comparten las evidencias del análisis e implementación de tres versiones de un DotPlot [1] (Secuencial, Multiprocessing y con MPI4PY) con el fin de identificar regiones similares o patrones repetitivos entre las secuencias.Un dotplot es una representación visual que muestra las similitudes y diferencias entre las dos secuencias, donde cada punto indica una coincidencia de caracteres en las posiciones correspondientes de ambas secuencias. Este DotPlot se construyó a partir de dos secuencias de ADN previamente compartidas y fueron cargadas en formato FASTA mediante línea de comandos.

Para el presente proyecto el análisis de rendimiento de los métodos de paralelización se realizaron bajo las métricas de:

1. Tiempos de ejecución totales y parciales (porción paralelizable).
2. Tiempo de carga de los datos y de generación de la imagen.
3. Tiempo muerto (tiempo no empleado en la ejecución del problema).
4. Aceleración y eficiencia.
5. Escalabilidad.

2. Estrategia de Paralelización

El objetivo de este proyecto fue 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. Se desarrolló un proyecto en Jupyter Notebook llamado **proyectoFinal_HPC_Entregable.ipynb** que explora estas variantes de implementación:

1. **Implementación Secuencial:** Un único hilo de ejecución.
2. **Paralelización con multiprocessing:** Uso de múltiples procesos en el mismo nodo.
3. **Paralelización con mpi4py:** Distribución de la tarea entre múltiples nodos y procesos usando MPI.

Cabe resaltar que independientemente de cualquiera de las tres variantes de implementación, se genera un Dotplot de tamaño superior a 136.000 x 136.0000, pero por temas computacionales se trabajó con una matriz de 20.000 x 20.000 para realizar de manera rápida y eficiente los respectivos análisis.

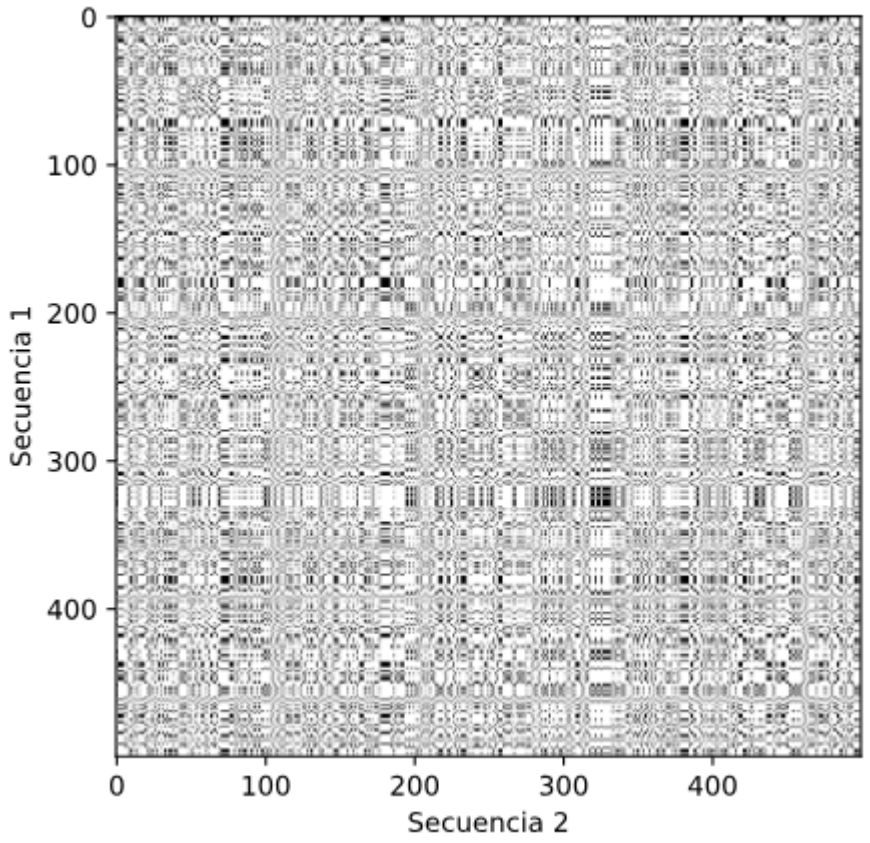

Imagen 1. Región inicial del Dotplot General(Se tomó un fragmento indexado de 500 x 500)

1. Dotplot: una herramienta visual utilizada principalmente en bioinformática y análisis de secuencias para comparar dos secuencias biológicas, como ADN, ARN o proteínas. Su objetivo es identificar regiones similares o patrones repetitivos entre las secuencias.

Con el fin de realizar de manera óptima la paralelización se aplicaron algunos ajustes en el código los cuales aceleraron la ejecución:

1. Se realizó un mapeo de las secuencias y se convirtieron a numericas para trabajar con arrays de tipo entero.
   Nota: Se realizó una asignación numérica para cada una de las bases nitrogenadas.
   {'A': 0, 'C': 1, 'G': 2, 'T': 3, 'N' : 4}

2. Se realizó la normalización de las secuencias para que operaran con enteros sin signo de 8 bits ya que con este mañana se logra dar cobertura al universo de números dentro de las secuencias. De otro modo, la asignación de memoria superaría los 174 GB de memoria RAM. Después de esta implementación, 100000 datos ocuparían 10 GB de memoria RAM.

3. Evidencias

A partir de construcción secuencial, multiprocessing y mpi4py, se logró tener la siguiente información.

3.1. Rendimiento

3.1.1 Secuencial

[SECUENCIAL] Con un procesador el código se ejecutó en: 101.23174715042114 segundos

3.1.2 Multiprocessing

[MULTIPROCESSING]Con: 1 procesadores, el tiempo de Ejecución es: {57.43933892250061} segundos
[MULTIPROCESSING]Con: 2 procesadores, el tiempo de Ejecución es: {47.5102162361145} segundos
[MULTIPROCESSING]Con: 4 procesadores, el tiempo de Ejecución es: {38.63143491744995} segundos
[MULTIPROCESSING]Con: 6 procesadores, el tiempo de Ejecución es: {35.207391023635864} segundos

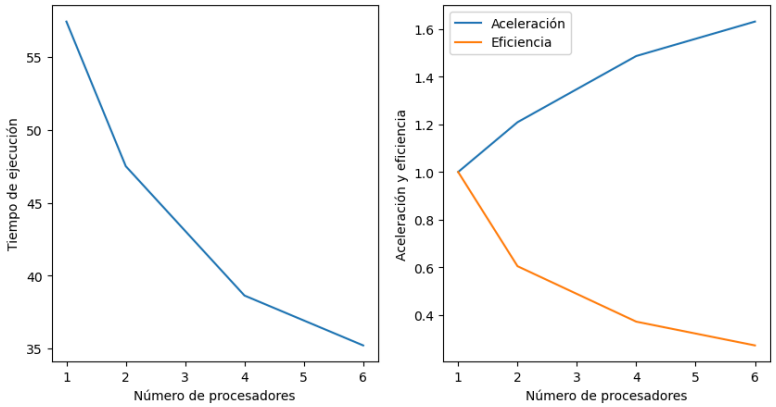

Imagen 2. Rendimiento con Multiprocessing

3.1.3 Mpi4py

MPI4PY]Con: 1 Nodo, el tiempo de Ejecución es: {220.87083458900452} segundos
[MPI4PY]Con: 2 Nodo(s), el tiempo de Ejecución es: {205.64088129997253} segundos
[MPI4PY]Con: 4 Nodo(s), el tiempo de Ejecución es: {119.33210253715515} segundos
[MPI4PY]Con: 6 Nodo(s), el tiempo de Ejecución es: {110.66779136657715} segundos

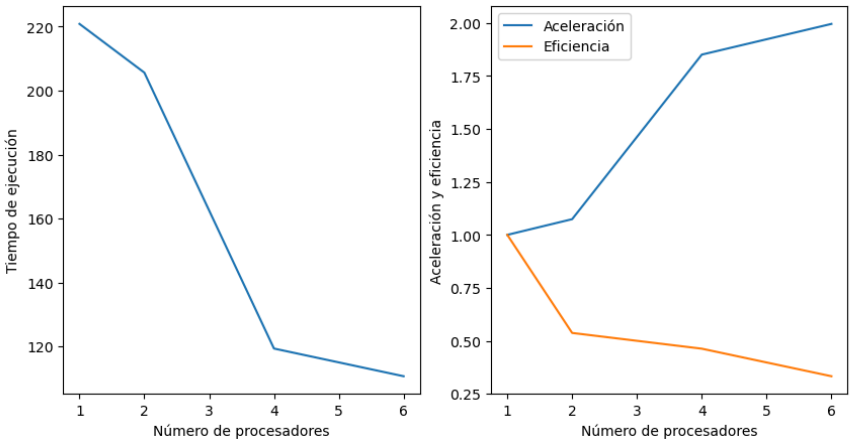

Imagen 3. Rendimiento con MPI4PY

3.2. Escalamiento

Es importante señalar que en el entorno de pruebas, se utilizaron secuencias con 20,000 datos cada una para la ejecución de las pruebas.

3.2.1 **Fuerte**:

3.2.1.1 Multiprocessing

[MULTIPROCESSING]Con: 1 procesadores, el tiempo de Ejecución es: {56.069985151290894} segundos
[MULTIPROCESSING]Con: 2 procesadores, el tiempo de Ejecución es: {42.375622510910034} segundos
[MULTIPROCESSING]Con: 4 procesadores, el tiempo de Ejecución es: {34.6992769241333} segundos
[MULTIPROCESSING]Con: 6 procesadores, el tiempo de Ejecución es: {36.71133279800415} segundos

3.2.1.2 Mpi4py

MPI4PY]Con: 1 Nodo, el tiempo de Ejecución es: {220.87083458900452} segundos
[MPI4PY]Con: 2 Nodo(s), el tiempo de Ejecución es: {205.64088129997253} segundos
[MPI4PY]Con: 4 Nodo(s), el tiempo de Ejecución es: {119.33210253715515} segundos
[MPI4PY]Con: 6 Nodo(s), el tiempo de Ejecución es: {110.66779136657715} segundos

3.2.2 Débil:

3.2.2.1 Multiprocessing

[MULTIPROCESSING]Con: 1 procesadores, el tiempo de Ejecución es: {54.876006841659546} segundos
[MULTIPROCESSING]Con: 2 procesadores, el tiempo de Ejecución es: {40.78957414627075} segundos
[MULTIPROCESSING]Con: 4 procesadores, el tiempo de Ejecución es: {35.12255024909973} segundos
[MULTIPROCESSING]Con: 6 procesadores, el tiempo de Ejecución es: {35.251624584198} segundos

3.2.2.2 Mpi4py

```
[MPI4PY]Con:  1 Nodo, el tiempo de Ejecución es:  {743.0506873130798} segundos
[MPI4PY]Con:  2 Nodo, el tiempo de Ejecución es:  {179.15939474105835} segundos
[MPI4PY]Con:  4 Nodo, el tiempo de Ejecución es:  {126.75746655464172} segundos
[MPI4PY]Con:  6 Nodo, el tiempo de Ejecución es:  {98.34345865249634} segundos
```

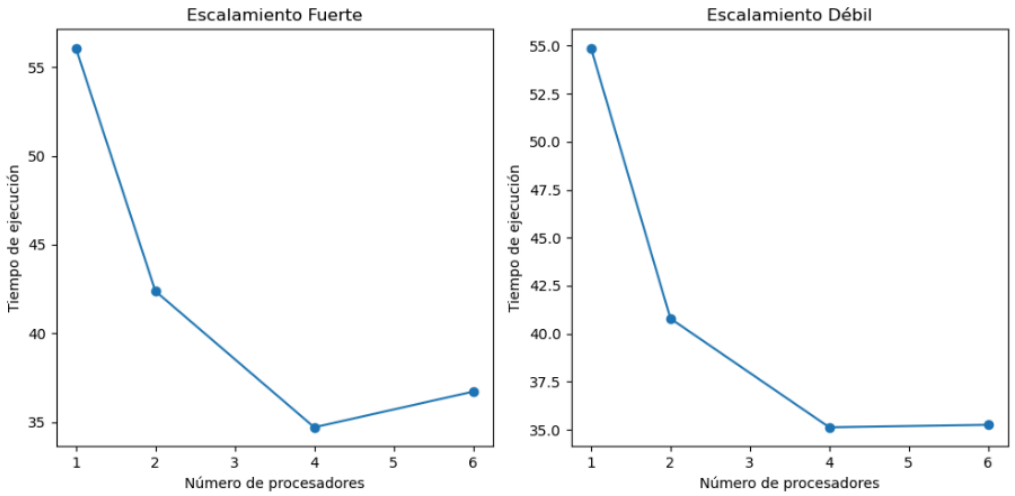

Imagen 4. Escalamiento con Multiprocessing

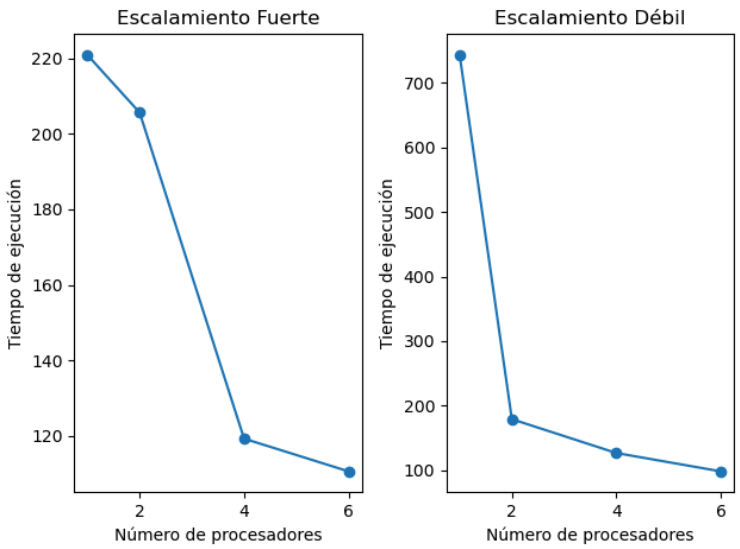

Imagen 5. Escalamiento con MPI4PY

4. Conclusión

Paralelización Secuencial: Es la forma más simple y directa de calcular el dotplot, pero con tiempos de ejecución elevados para grandes secuencias.

Multiprocessing: La implementación con multiprocessing mostró mejoras considerables en la velocidad de procesamiento y una mejor escalabilidad al usar múltiples núcleos de CPU.

Eficiencia: Aunque la aceleración mejoró al aumentar los núcleos, la eficiencia disminuyó a medida que se aumentó el número de procesos, lo que indica que el problema no se escala linealmente con el número de núcleos.

MPI4PY: El enfoque con mpi4py permite distribuir la carga de trabajo en múltiples nodos de un clúster, pero en este caso, la ejecución con un solo nodo fue más lenta que con multiprocessing.

Escalabilidad: Se observó que la solución con MPI es efectiva para escenarios de computación distribuida, pero se necesita un clúster de computación más grande para ver beneficios de rendimiento más marcados.

5. Insumos

Tanto la Data, el informe y el código fuente se comparten en un repositorio de GitHub.
Enlace: https://github.com/asalazargo/HPC_FINAL.

6. Bibliografía

Piña, J. S., Orozco-Arias, S., Tobón-Orozco, N., Camargo-Forero, L., Tabares-Soto, R., & Guyo, R. (2023). G-SAIP: Graphical sequence alignment through parallel programming in the post-genomic era. Evolutionary Bioinformatics, 19. https://pmc.ncbi.nlm.nih.gov/articles/PMC9871978/pdf/10.1177_11769343221150585.pdf