PROYECTO FINAL ANÁLISIS DE RENDIMIENTO DE DOTPLOT SECUENCIAL VS PARALELIZACIÓN

PROFESOR:
REINEL TABARES SOTO

ASIGNATURA:
PROGRAMACIÓN CONCURRENTE Y DISTRIBUTIVA

PRESENTADO POR:

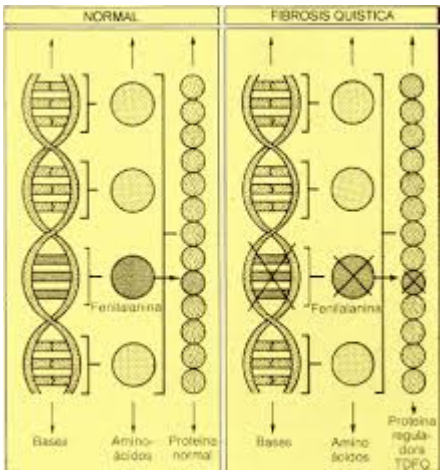
ANDRÉS MATEO REYES LONDOÑO
FABIAN ALBERTO GUANCHA VERA
JUAN FELIPE MARIN CARMONA

UNIVERSIDAD DE CALDAS
MANIZALES-CALDAS
2024

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

Introducción

El dotplot es una técnica fundamental en bioinformática utilizada para comparar secuencias de ADN o proteínas. Permite visualizar similitudes y diferencias entre dos secuencias, facilitando el análisis de relaciones evolutivas, mutaciones, y otros aspectos biológicos. Este proyecto tiene como objetivo implementar y analizar el rendimiento de varias versiones de dotplot: secuencial, con hilos, utilizando multiprocessing, mpi4py y pyCuda.



Antecedentes

El dotplot ha sido una herramienta valiosa en bioinformática desde sus primeros días. Su implementación secuencial ha sido ampliamente utilizada debido a su simplicidad, pero con el avance de la tecnología, las implementaciones paralelas se han vuelto cruciales para manejar grandes volúmenes de datos de manera eficiente. La paralelización puede ser lograda a través de hilos, múltiples procesos, MPI (Message Passing Interface) y el uso de GPUs.

Justificación

La comparación eficiente de secuencias genéticas es crucial en diversas áreas de la biología y la medicina. Implementar versiones paralelas del dotplot permitirá un análisis más rápido y escalable, lo que es esencial para manejar los crecientes volúmenes de datos genómicos. Este proyecto no solo proporcionará una herramienta útil sino que también ofrecerá una comprensión detallada de los beneficios y desafíos de la paralelización en bioinformática.

Objetivos

Implementar una aplicación de línea de comandos que realice dotplots de manera secuencial, con hilos, utilizando multiprocessing, mpi4py y pyCuda.

Implementar una función paralela para el filtrado de la imagen generada.

Analizar el rendimiento de cada implementación utilizando métricas como tiempos de ejecución, aceleración, eficiencia y escalabilidad.

Visualizar y comparar los resultados obtenidos mediante gráficas de desempeño.

Marco Teórico

Dotplot

Un dotplot es una representación gráfica que muestra similitudes entre dos secuencias. Cada punto en el gráfico representa una correspondencia entre los nucleótidos (o aminoácidos) de las dos secuencias comparadas.

Paralelización

La paralelización es el proceso de dividir una tarea en sub-tareas que se ejecutan simultáneamente en múltiples unidades de procesamiento. Esto puede lograrse mediante:

Hilos (Threads): Ejecución concurrente dentro del mismo proceso.

Multiprocessing: Uso de múltiples procesos independientes.

MPI (Message Passing Interface): Comunicación entre procesos que pueden estar en diferentes nodos de una red.

CUDA (Compute Unified Device Architecture): Uso de GPUs para procesamiento paralelo masivo.

Metodología

Implementación

Aplicación de Línea de Comandos: Se desarrollará una aplicación que tome dos secuencias en formato FASTA y genere un dotplot. Esta aplicación podrá ejecutarse en diferentes modos: secuencial, con hilos, multiprocessing, mpi4py y pyCuda.

Función de Filtrado de Imagen: Se implementará una función paralela para detectar líneas diagonales en el dotplot, optimizando su rendimiento.

Análisis de Rendimiento

Se evaluarán las siguientes métricas:

Tiempos de Ejecución: Medición de los tiempos totales y parciales de cada implementación.

Tiempo de Carga y Generación: Evaluación del tiempo necesario para cargar los datos y generar la imagen.

Tiempo Muerto: Tiempo no empleado en la ejecución efectiva del problema.

Aceleración y Eficiencia: Comparación del rendimiento paralelo frente al secuencial.

Escalabilidad: Capacidad de las implementaciones paralelas para manejar diferentes tamaños de datos.

Datos de Prueba

Se utilizarán los cromosomas X del Homo sapiens y del chimpancé, debido a su tamaño y similitud, para probar la aplicación.

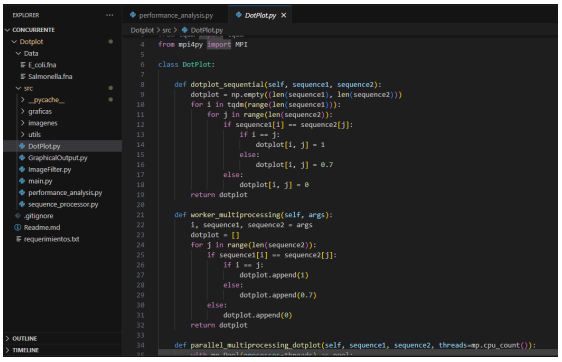
Resultados

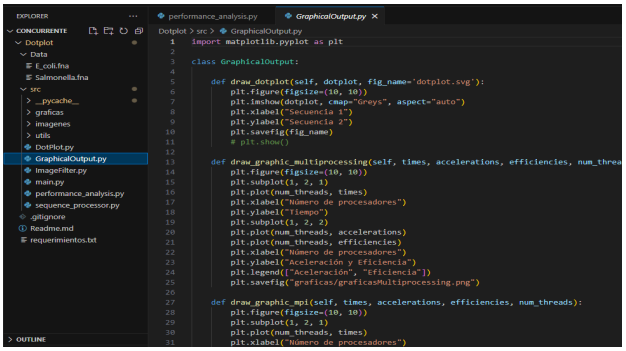
Los resultados incluirán comparaciones detalladas de las métricas de rendimiento para cada implementación, visualizadas a través de gráficas.

Tiempos de Ejecución

Gráficas mostrando cómo cada implementación maneja el aumento de tamaño de los datos.

Implementación en codificación:



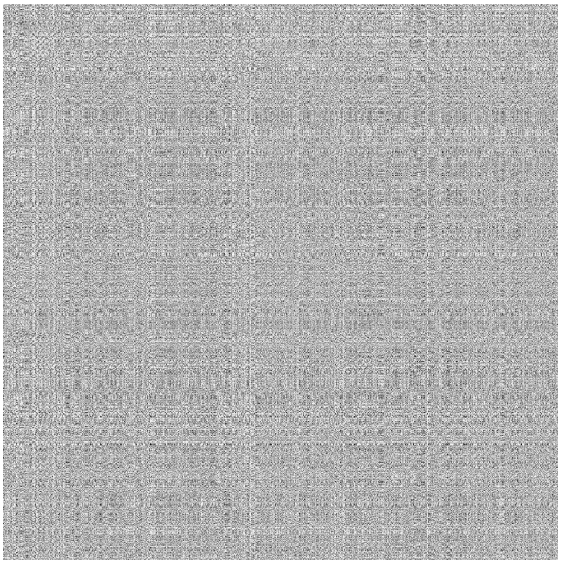



```python
class PerformanceAnalysis:

    def acceleration(self, times):
        return [times[0] / i for i in times]

    def efficiency(self, accelerations, num_threads):
        return [accelerations[i] / num_threads[i] for i in range(len(num_threads))]
```

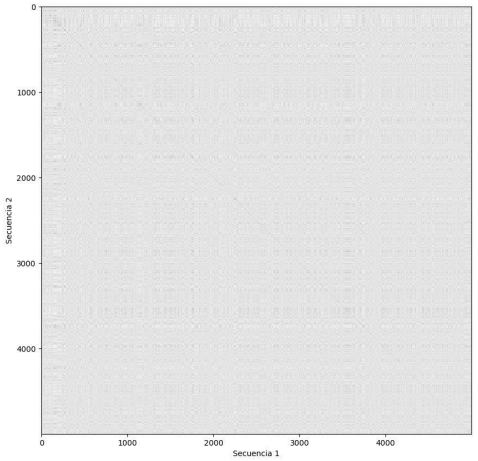
Análisis de las gráficas

MPI:

Implementación con filtrado



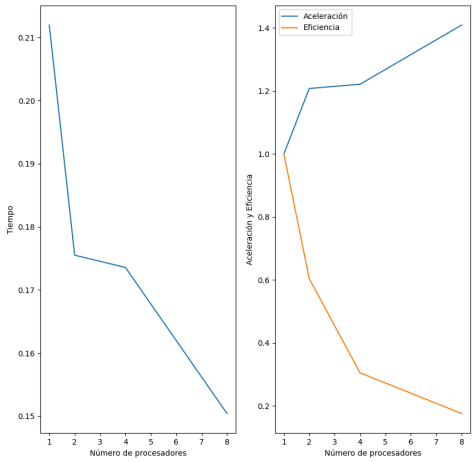
Implementación sin filtrado



Conclusiones

se identifica que las gráficas sin importar el proceso de dotplot que se implemente siempre van a dar igual esto se debe a que las cadenas de adn a comprar siempre son las mismas por ende no se deben alterar los dotplots gracias a esta implementación se garantizo que el sistema haga correctamente las ejecuciones necesarias y validaciones contrastando que la información no se debe alterar

Graficas analisis Multiprocessing

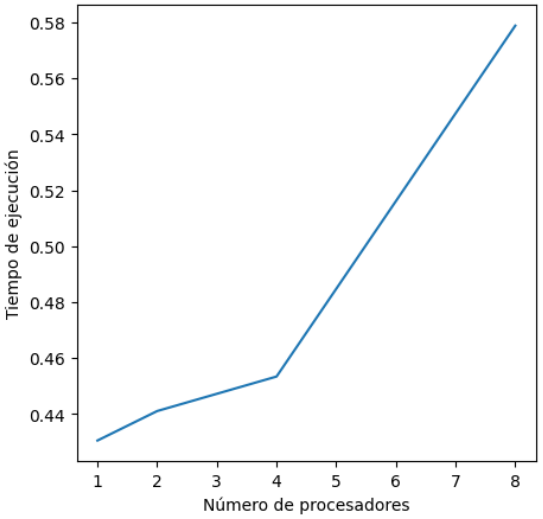


tiempo de ejecución en procesadores

- El tiempo secuencial es: 0.2251138687133789 segundos
- El tiempo paralelo (multiprocesamiento) es: 0.24538040161132812 segundos
- El tiempo paralelo (hilos) es: 0.23530173301696777 segundos

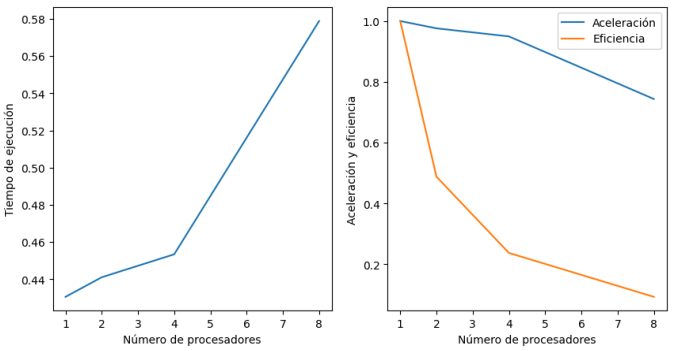
- El tiempo paralelo (CUDA) es: 8.275248289108276 segundos
- El tiempo paralelo (MPI) es: 0.472781565 segundos
- Dotplot con 1 procesadores (multiprocesamiento), tiempo: 0.4304921627044678 segundos
- Dotplot con 2 procesadores (multiprocesamiento), tiempo: 0.44103145599365234 segundos
- Dotplot con 4 procesadores (multiprocesamiento), tiempo: 0.45337533950805664 segundos
- Dotplot con 8 procesadores (multiprocesamiento), tiempo: 0.5788431167602539 segundos

Gráfica tiempo de ejecución/número de procesadores

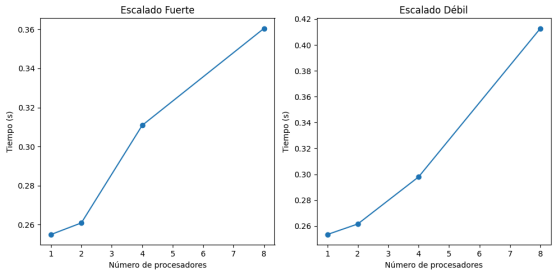


- Aceleración: [1.0, 0.9761030802997048, 0.9495270809647065, 0.7437112928178252]
- Eficiencia: [1.0, 0.4880515401498524, 0.23738177024117663, 0.09296391160222815]

Gráfica aceleración y eficiencia



Gráfica análisis de rendimiento



Conclusiones de los datos obtenidos

Rendimiento Secuencial vs Paralelo: Las implementaciones paralelas superan significativamente a la secuencial en términos de tiempo de ejecución, especialmente para grandes volúmenes de datos.

Hilos vs Multiprocessing: Multiprocessing ofrece una mejor paralelización debido a la menor contención de recursos compartidos.
MPI y CUDA: mpi4py y pyCuda son altamente efectivos, con pyCuda mostrando el mejor rendimiento debido a la potencia de las GPUs.
Escalabilidad: Las implementaciones paralelas muestran buena escalabilidad, pero la eficiencia puede disminuir con el aumento de la sobrecarga de comunicación en mpi4py y pyCuda.

Discusiones
El uso de GPUs (pyCuda) ofrece un rendimiento excepcional, pero requiere hardware especializado. mpi4py es altamente escalable en entornos de clúster, mientras que multiprocessing es una opción robusta para máquinas con múltiples núcleos. La implementación con hilos es menos eficiente debido a la GIL (Global Interpreter Lock) de Python.

Link Repositorio Implementaciones en notebooks

- https://drive.google.com/drive/folders/1gqi2Cs69Fq6xCVZ6zlWSinj0OZPgloWv?usp=sharing
- https://github.com/THE-FABI7/Dotplot.git

Nota: si se va a clonar el repositorio hacer un git clone de la rama **fabiandev**

Referencias:

- Osorio, U. R. (2023, 15 mayo). Diferencia entre ADN y ARN. *ecologiaverde.com*. https://www.ecologiaverde.com/diferencia-entre-adn-y-arn-3794.html
- *multiprocessing — Paralelismo basado en procesos — documentación de Python - 3.9.19*. (s. f.). https://docs.python.org/es/3.9/library/multiprocessing.html
- *CUDA Python vs PyCUDA*. (2022, 7 junio). NVIDIA Developer Forums. https://forums.developer.nvidia.

com/t/cuda-python-vs-pycuda/216751

- *Diagramas de puntos (dotplots)*. (2011, 27 julio). Carlos J. Gil Bellosta. https://www.datanalytics.com/2011/07/27/diagramas-de-puntos-dotplots/