Comparación de secuencias de ADN o Proteínas mediante DotPlot

Alejandro Mesa Rodríguez
Ingeniería de Sistemas y Computación
Universidad de Caldas
Manizales, Colombia
alejandro.1701922112@ucaldas.edu.co

Julián Rivera Castaño
Ingeniería de SiStemas y Computación
Universidad de Caldas
Manizales, Caldas
julian.1701910633@ucaldas.edu.co

Juan David Díaz Castaño
Ingeniería de SiStemas y Computación
Universidad de Caldas
Manizales, Colombia
juan.1701920068@ucaldas.edu.co

INTRODUCCIÓN

En el ámbito de la bioinformática, la comparación de secuencias de ADN o proteínas es fundamental para comprender la estructura, función y evolución de biomoléculas. Una técnica ampliamente utilizada para visualizar y analizar similitudes entre secuencias es el dotplot, que permite identificar regiones de homología y patrones repetitivos.

El objetivo de este proyecto es implementar y analizar el rendimiento de diferentes enfoques para la construcción de dotplots. Estos enfoques incluyen una implementación secuencial, una versión paralela utilizando la biblioteca multiprocessing de Python, una versión paralela utilizando mpi4py para procesamiento distribuido, y una implementación usando pyCuda para aprovechar la potencia de cómputo de GPUs.

JUSTIFICACIÓN Y RELEVANCIA

La construcción eficiente de dotplots es crucial en bioinformática debido al creciente volumen de datos genómicos y proteómicos generados por las tecnologías modernas de secuenciación. La capacidad de realizar comparaciones rápidas y precisas entre secuencias es fundamental para identificar homologías, investigar relaciones evolutivas y descubrir estructuras conservadas.

El análisis comparativo de las distintas implementaciones no solo proporcionará información sobre la eficiencia y escalabilidad de cada enfoque, sino que también permitirá identificar el contexto adecuado para su aplicación. La evaluación de métricas de rendimiento como el tiempo de ejecución, la utilización de recursos y la escalabilidad bajo diferentes cargas de trabajo proporcionará una visión detallada de las fortalezas y limitaciones de cada método.

Este trabajo fue realizado con fines académicos para la materia: Programación concurrente y distribuída. Universidad de Caldas, Manizales, Colombia.

DESCRIPCIÓN DE LOS INSUMOS Y RECURSOS

Antes de explicar cómo funciona cada una de las implementaciones es preciso exponer qué necesitamos para poder desarrollar cada una de ellas y con qué equipos disponemos para ejecutarlas.

Usaremos el lenguaje de programación Python en su versión 3.11. Utilizaremos el entorno local para la implementación secuencial y las de múltiples procesos a nivel de CPU. Ejecutaremos cada una de las implementaciones en un portátil con 16 GB de memoria RAM versión DDR5, un procesador Intel Core i7-12700H con 14 núcleos y 20 procesadores lógicos, y el sistema operativo Windows 11. Para la implementación en PyCuda (GPU), usaremos Google Colab, ya que nos ofrece una gráfica potente de manera gratuita, lo cual ayuda en la reducción de tiempos de ejecución.

Para el correcto funcionamiento del proyecto necesitaremos instalar las siguientes librerías:

- **Numpy:** es una biblioteca fundamental en Python para computación científica y análisis de datos. Proporciona estructuras de datos eficientes, como arreglos multidimensionales (ndarrays), que son cruciales para el procesamiento numérico rápido y eficiente de datos en bioinformática. En el proyecto, numpy facilita operaciones matemáticas complejas y manipulación de matrices, permitiendo la implementación eficiente de algoritmos para la construcción y análisis de dotplots entre secuencias biológicas.
- **Matplotlib:** es una biblioteca de visualización en Python que permite crear gráficos de alta calidad para la presentación de datos. Es especialmente útil en bioinformática para visualizar resultados de análisis como dotplots, histogramas de distribuciones de datos, y otras representaciones gráficas necesarias para interpretar y comunicar hallazgos científicos de manera efectiva. En el proyecto, matplotlib se utiliza para generar visualizaciones claras e informativas de los resultados obtenidos a partir de las comparaciones de secuencias.

- **Mpi4Py:** MPI (Message Passing Interface) es un estándar fundamental para la comunicación entre procesos en entornos distribuidos y paralelos, facilitando la sincronización y el intercambio de datos entre múltiples nodos de computación. En el contexto de Python, MPI4PY sirve como una interfaz robusta que permite a los desarrolladores crear programas paralelos y distribuidos de manera efectiva. Esta biblioteca no solo ofrece una amplia gama de funciones y utilidades para la comunicación entre procesos, sino que también brinda soporte para diversas implementaciones de MPI, asegurando flexibilidad y compatibilidad en entornos de alta performance computing (HPC). En el contexto del proyecto actual, MPI4PY desempeña un papel crucial al proporcionar las herramientas necesarias para implementar una versión paralela del dotplot, permitiendo así explorar y analizar eficazmente el rendimiento de este algoritmo bajo condiciones distribuidas, mejorando significativamente la escalabilidad y el tiempo de ejecución de las comparaciones de secuencias.
- **Biopython:** es una biblioteca especializada en bioinformática que ofrece una amplia gama de herramientas y algoritmos para el manejo, análisis y manipulación de datos biológicos. Facilita tareas como la lectura y escritura de formatos de archivos biológicos (como FASTA y GenBank), el cálculo de alineamientos de secuencias, la extracción de información de bases de datos biológicas, entre otros. En el proyecto, Biopython es esencial para el preprocesamiento de secuencias biológicas y la comparación detallada requerida para la construcción de dotplots.
- **Opencv-python:** OpenCV (Open Source Computer Vision Library) es una biblioteca de visión por computadora y procesamiento de imágenes ampliamente utilizada. En bioinformática, opencv-python proporciona capacidades avanzadas para el análisis de imágenes biológicas, como la detección de patrones, segmentación de imágenes, y extracción de características. En el proyecto, opencv-python podría emplearse para análisis avanzados de estructuras biológicas en secuencias, mejorando la precisión y detalle en la construcción de dotplots y la interpretación de resultados.
- **Multiprocessing:** La biblioteca multiprocessing en Python facilita la creación de procesos múltiples para ejecutar tareas concurrentes en sistemas multi-core y distribuidos. Es especialmente útil en bioinformática para mejorar el rendimiento mediante la ejecución paralela de algoritmos intensivos en CPU. En el proyecto, multiprocessing será crucial para implementar versiones paralelas de algoritmos de construcción de dotplots, aprovechando la capacidad de cómputo de múltiples núcleos de CPU para acelerar el procesamiento de grandes volúmenes de datos de secuencias. Esto permitirá explorar y comparar el rendimiento de las implementaciones paralelas con respecto a la versión secuencial, mejorando significativamente la eficiencia y el tiempo de respuesta del sistema.
- **Tqdm:** es una biblioteca de Python que proporciona barras de progreso interactivas y otras utilidades para monitorear el avance de bucles y operaciones largas. Es útil en proyectos de bioinformática que implican procesamiento intensivo de datos, ya que permite a los desarrolladores y usuarios seguir el progreso de tareas computacionalmente intensivas. En el proyecto, tqdm se utiliza para mejorar la experiencia del usuario al ejecutar algoritmos de comparación de secuencias y construcción de dotplots, proporcionando retroalimentación visual sobre el avance de las operaciones y el tiempo restante estimado.
- **Time:** La biblioteca time en Python proporciona funciones para medir el tiempo de ejecución y manejar operaciones relacionadas con el tiempo. Es esencial en bioinformática para evaluar el rendimiento de algoritmos y procesos computacionales, permitiendo la medición precisa de tiempos de ejecución y la optimización de algoritmos. En el proyecto, time se utilizará para calcular y comparar los tiempos de ejecución de diferentes implementaciones de dotplots, proporcionando datos cruciales para evaluar la eficiencia y escalabilidad de cada enfoque.

IMPLEMENTACIONES DEL DOTPLOT

Secuencial

Mediante un doble ciclo for se comparan elemento a elemento de ambas secuencias. Si coinciden ambos caracteres, se evalúa si están en la misma posición dentro de la secuencia; de ser así, se asigna una intensidad de 2 y se agrega este valor en la matriz que almacenará los datos del DotPlot para luego graficarlos. En caso contrario, se les asigna un valor de 1, y si no coinciden, se asigna 0.

Para ejecutar el programa secuencial, utilice el siguiente comando:

```
python Main.py --file1=./data/E_coli.fna --
    file2=./data/Salmonella.fna --sequential
```

o

```
python Main.py --file1=./data/E_coli.fna --
    file2=./data/Salmonella.fna --maxLen=20000
     --sequential
```

Donde se indica el archivo a ejecutar, las direcciones de las secuencias a comparar en formato fna, el tamaño máximo del DotPlot (opcional) y el tipo de ejecución, en este caso, secuencial.

En cuanto a los tiempos de ejecución tenemos que el tiempo de ejecución secuencial fue 45.49750733375549 segundos, y el tiempo de ejecución en bloque secuencial fue 45.56563186645508 segundos.

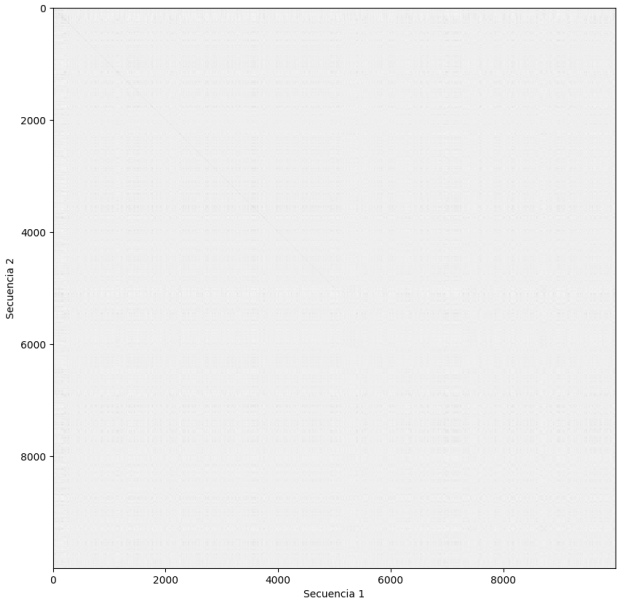

Fig. 1. DotPlot realizado en secuencial

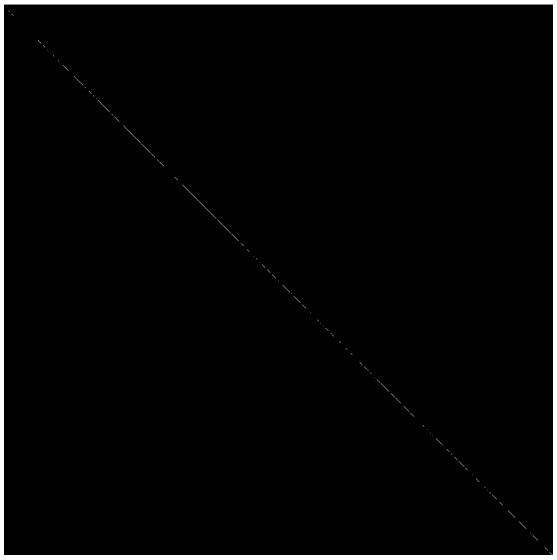

Fig. 2. Zoom de la diagonal usando filtro de convolución

Paralelo con Multiprocessing

Se emplea un worker para realizar las comparaciones entre las secuencias de manera similar al método secuencial, almacenando los resultados en un arreglo. Posteriormente, utilizando el pool de procesos que se distribuye automáticamente en función del número de procesos indicados, para nuestro proyecto lo haremos con varios números de procesos para evuluar diversas métricas, se generan diversas gráficas, incluyendo la aceleración y eficiencia del proceso.

Para ejecutar el programa con multiprocessing, use el siguiente comando:

```
python Main.py --file1=./data/E_coli.fna --
    file2=./data/Salmonella.fna --
    multiprocessing
```

o

```
python Main.py --file1=./data/E_coli.fna --
    file2=./data/Salmonella.fna --maxLen=20000
     --multiprocessing
```

Donde se indica el archivo a ejecutar, las direcciones de las secuencias a comparar en formato fna, el tamaño máximo del DotPlot (opcional) y el tipo de ejecución, en este caso, paralelo con la librería multiprocessing.

Tiempos de Ejecución Parcial

1 procesadores: 42.58053207397461
2 procesadores: 22.82190465927124
4 procesadores: 13.978235006332397
8 procesadores: 11.155134201049805
16 procesadores: 12.050483465194702

Aceleración

1 procesadores: 1.0
2 procesadores: 1.8657746892600642
4 procesadores: 3.0462023320315366
8 procesadores: 3.8171241427079714
16 procesadores: 3.5335123438789453

Eficiencia

1 procesadores: 1.0
2 procesadores: 0.9328873446300321
4 procesadores: 0.7615505830078841
8 procesadores: 0.4771405178384964
16 procesadores: 0.22084452149243408

Tiempos de Ejecución en Bloque

1 procesadores: 42.651254415512085
2 procesadores: 22.892627000808716
4 procesadores: 14.048957347869873
8 procesadores: 11.22585654258728
16 procesadores: 12.121205806732178

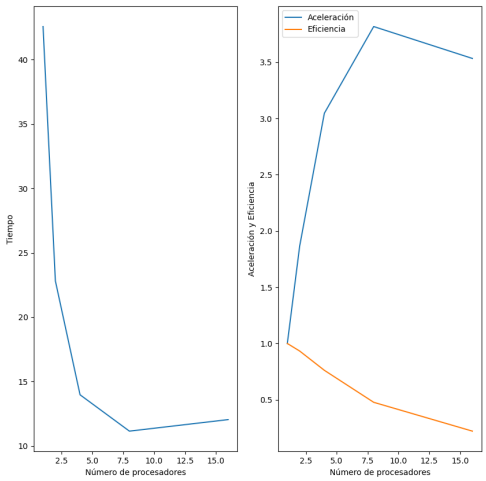

Fig. 3. Gráficas de aceleración y eficiencia para multiprocessing

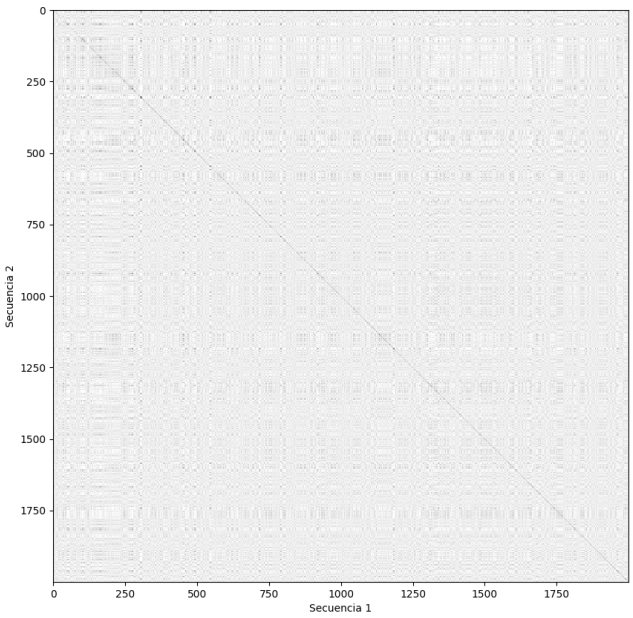

Fig. 4. DotPlot realizado en paralelo con multiporcessing

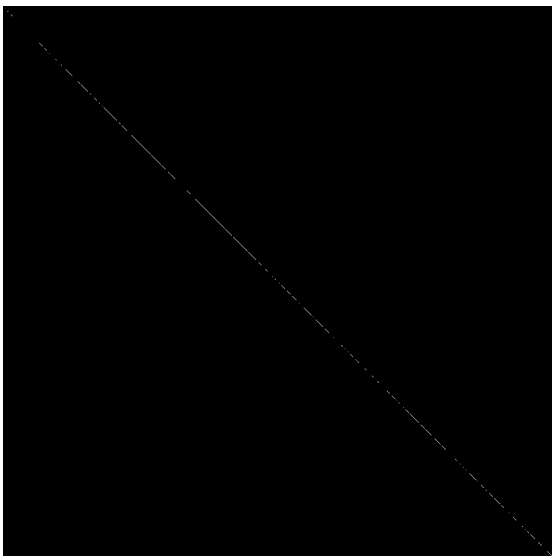

Fig. 5. Zoom de la diagonal usando filtro de convolución

Paralelo con MPI4PY

Para MPI, se establece la conexión inicial y se definen los chunks en función del número de procesos disponibles. Cada proceso realiza la comparación elemento a elemento de manera similar al método secuencial, pero en múltiples secciones simultáneamente, lo que reduce el tiempo de ejecución. Luego, utilizando funciones como vstack y gather, se unifican los resultados de cada proceso para generar el DotPlot. Finalmente, se generan gráficas para evaluar la aceleración y eficiencia del proceso.

Para ejecutar el programa con MPI, utilice el siguiente comando:

```
python Main.py --num_processes 1 2 4 8 --file1
    =./data/E_coli.fna --file2=./data/
    Salmonella.fna --mpi
```

o

```
python Main.py --num_processes 1 2 4 8 --file1
    =./data/E_coli.fna --file2=./data/
    Salmonella.fna --maxLen=20000 --mpi
```

Donde se indica el archivo a ejecutar, la cantidad de procesos a utilizar, las direcciones de las secuencias a comparar en formato fna, el tamaño máximo del DotPlot (opcional) y el tipo de ejecución, en este caso, paralelo con MPI4PY.

Tiempos de Ejecución

1 procesador: 54.880202293395996
2 procesadores: 55.88390898704529
4 procesadores: 55.87148690223694
8 procesadores: 55.883705615997314

Aceleración

1 procesador: 1.0
2 procesadores: 0.9820394329630383
4 procesadores: 0.9822577729034583
8 procesadores: 0.9820430067845384

Eficiencia

1 procesador: 1.0
2 procesadores: 0.49101971648151915
4 procesadores: 0.24556444322586457
8 procesadores: 0.1227553758480673

Tiempos de Ejecución en Bloque

1 procesador: 54.9443085193634
2 procesadores: 55.948015213012695
4 procesadores: 55.935593128204346
8 procesadores: 55.94781184196472

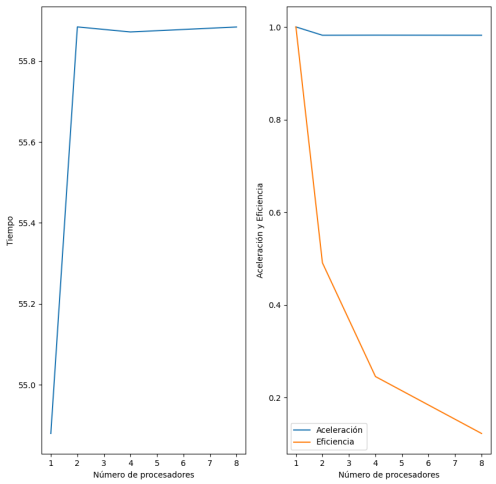

Fig. 6. Gráficas de aceleración y eficiencia para Mpi4Py

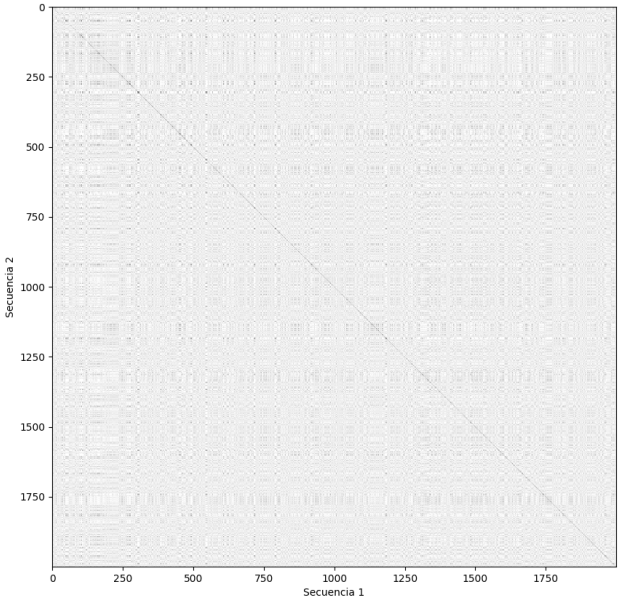

Fig. 7. DotPlot realizado en paralelo con Mpi4Py

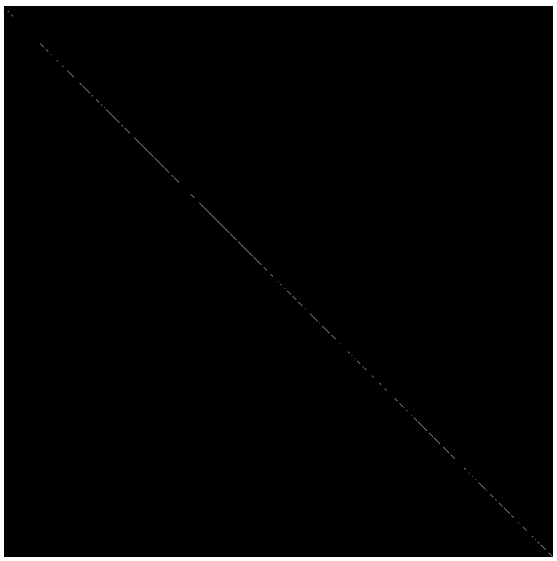

Fig. 8. Zoom de la diagonal usando filtro de convolución

Paralelo con PyCuda

Tenemos 3 parámetros iniciales:

max-length: Longitud de la cadena procesar.

window-size: Este parámetro determina la longitud de la subsecuencia que se compara entre las dos secuencias. Una ventana más grande significa que se buscan coincidencias de subsecuencias más largas, lo que puede ser útil para identificar patrones más largos y menos ruidosos, pero puede perder coincidencias más cortas.

block-size: CUDA divide las tareas en bloques y cada bloque se procesa en paralelo en los multiprocesadores de la GPU. block-size especifica el número de hilos por bloque en cada dimensión (por ejemplo, si block-size es 16, habrá 16x16=256 hilos por bloque).

Tiempos de Carga de Datos

Salmonella.fna: 0.03 segundos
E-coli.fna: 0.03 segundos
Tiempo de carga de datos total: 0.06 segundos

Longitudes de Secuencias

Secuencia 1: 10000
Secuencia 2: 10000

Tiempos de Ejecución

Secuencial: 31 segundos
Ejecución del kernel CUDA: 0.00 segundos
Cálculo para generar el dotplot: 0.67 segundos
Generar y guardar la imagen: 2.31 segundos
Ejecución del programa: 3.04 segundos
Ejecución total: 3.04 segundos

Otros Tiempos

Generación de la imagen: 2.31 segundos
Tiempo muerto: 3.04 segundos

Aceleración y Eficiencia

Aceleración: 10.20
Eficiencia: 3.36

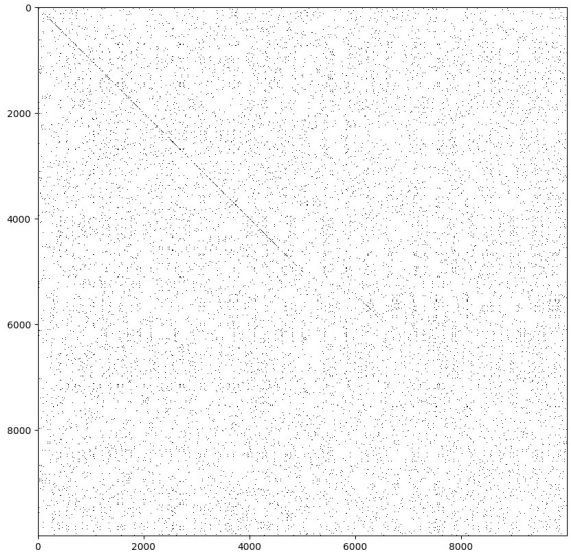

Fig. 9. DotPlot realizado en paralelo con PyCuda

DISCUSIÓN

Eficiencia y Rendimiento de las Implementaciones

En este estudio, exploramos varias metodologías para la construcción de DotPlots: secuencial, multiprocessing, MPI4PY y PyCuda para GPU. Cada una de estas implementaciones tiene implicaciones significativas en términos de eficiencia y rendimiento computacional.

Secuencial vs. Paralelo (Multiprocessing y MPI4PY): Comparando el tiempo de ejecución entre la implementación secuencial y las versiones paralelas utilizando multiprocessing y MPI4PY, observamos una clara reducción en el tiempo de ejecución al emplear técnicas paralelas, destacando la capacidad de estos métodos para manejar grandes volúmenes de datos de secuencias de manera más eficiente.

MPI4PY y Escalabilidad: La escalabilidad de MPI4PY fue evaluada aumentando el número de procesos, observando cómo el tiempo de ejecución varió con el número de nodos de procesamiento. La evaluación de métricas como aceleración y eficiencia mostró cómo MPI4PY permite distribuir eficazmente la carga de trabajo en sistemas distribuidos, mejorando la capacidad de procesamiento conforme se añaden más recursos computacionales.

GPU Acceleration (PyCuda): Analizamos los beneficios y limitaciones de utilizar GPUs para acelerar la construcción de DotPlots. Encontramos una mejora significativa en el rendimiento computacional al aprovechar la potencia de cálculo masiva de las GPUs, especialmente notable en comparación con las implementaciones CPU-bound.

Aplicaciones y Significado en Bioinformática

Impacto en la Investigación Biomolecular: Los DotPlots son cruciales en bioinformática para visualizar similitudes estructurales y funcionales en secuencias de ADN o proteínas. Estas herramientas son fundamentales para descubrir homologías, investigar la evolución molecular y descubrir regiones conservadas entre especies.

Relevancia en Grandes Volúmenes de Datos: La capacidad de las implementaciones paralelas y aceleradas por GPU es esencial para manejar eficientemente los grandes conjuntos de datos generados por tecnologías modernas de secuenciación. La escalabilidad y la eficiencia de estas técnicas son críticas para avanzar en el campo de la genómica comparativa y la bioinformática estructural.

Limitaciones y Consideraciones Futuras

Desafíos en Implementaciones Paralelas: Las implementaciones paralelas presentan desafíos, como la sincronización de datos y la gestión de recursos en entornos distribuidos. Es necesario optimizar algoritmos y técnicas de paralelización para maximizar el rendimiento y la escalabilidad en futuros desarrollos.

Exploración de Nuevas Tecnologías: Áreas para investigación futura incluyen el uso de técnicas de aprendizaje automático para mejorar la precisión en la comparación de secuencias o la integración de herramientas de visión por computadora para análisis más avanzados de estructuras biológicas.

Conclusiones y Aportaciones

Resumen de Resultados: Los principales hallazgos y conclusiones de este estudio muestran que las diferentes implementaciones ofrecen ventajas específicas según el contexto de aplicación, proporcionando a los investigadores herramientas flexibles y escalables para estudios bioinformáticos avanzados.

Contribución al Campo: Esta investigación contribuye al campo de la bioinformática al ofrecer una evaluación exhaustiva y comparativa de métodos para la construcción de DotPlots. Los resultados pueden influir en prácticas futuras y en el desarrollo de herramientas computacionales más eficaces para análisis genómico y proteómico.