Análisis de Rendimiento de Implementaciones Paralelas para la Generación de Dot-Plots Genómicos

Cuayal Robinson, *Member, Ucaldas,* Villa Juan, *Member, ucaldas,*

***Abstract*—El dotplot es una herramienta ampliamente utilizada en bioinformática para analizar similitudes entre secuencias de ADN o proteínas. Sin embargo, la generación de dotplots plantea importantes desafíos computacionales debido a su alta complejidad y el tamaño creciente de los datos genómicos. Este trabajo presenta un análisis de rendimiento de cuatro implementaciones para la generación de dotplots: una secuencial, una basada en programación paralela con `multiprocessing`, otra con `mpi4py` y una última utilizando `PyCUDA` para aprovechar la aceleración mediante GPU.**

Las implementaciones se evaluaron utilizando los cromosomas X de *Homo sapiens* y *Pan troglodytes*, considerando métricas como tiempos de ejecución, escalabilidad, aceleración y eficiencia. Los resultados muestran que la implementación basada en GPU supera significativamente a las demás, reduciendo los tiempos de ejecución hasta un 80% en comparación con la versión secuencial. La implementación con `mpi4py` también demostró ser efectiva en sistemas de memoria distribuida, mientras que `multiprocessing` ofreció mejoras moderadas. La implementación secuencial, aunque simple, resulta ineficiente para conjuntos de datos grandes.

Este estudio resalta la importancia de las estrategias de computación de alto rendimiento para abordar problemas complejos en bioinformática, proporcionando un análisis detallado de las ventajas y limitaciones de cada enfoque para dotplots genómicos.

***Index Terms*—Dotplot, bioinformática, paralelismo, PyCUDA, MPI, rendimiento computacional, escalabilidad, secuencias genómicas.**

I. Introducción

El dotplot es una herramienta gráfica ampliamente utilizada en bioinformática para comparar secuencias de ADN o proteínas. Esta técnica permite identificar similitudes, inserciones, deleciones y reordenamientos en las secuencias, proporcionando una representación visual en forma de matriz donde las filas y columnas corresponden a las bases o residuos de las secuencias comparadas [9]. Sin embargo, la generación de dotplots plantea importantes desafíos computacionales debido al crecimiento exponencial de los datos genómicos y la alta complejidad de los algoritmos necesarios para procesarlos [3].

En este trabajo, se analizan y comparan cuatro implementaciones para la generación de dotplots: una implementación secuencial, una basada en programación paralela con `multiprocessing`, otra utilizando `mpi4py` y una última que aprovecha la aceleración mediante GPU con `PyCUDA`. Estas implementaciones se evaluaron utilizando datos genómicos reales, como los cromosomas X de *Homo sapiens* y *Pan troglodytes*, con el objetivo de medir su rendimiento en términos de tiempo de ejecución, escalabilidad, aceleración y eficiencia.

Este trabajo fue realizado en el marco del programa de Maestría en Ingeniería Computacional de la Universidad de Caldas, Manizales, Colombia. Los autores agradecen al Departamento de Maestría en Ingeniería Computacional por su apoyo y recursos proporcionados para la realización de este proyecto.

Manuscrito recibido en diciembre de 2024; revisado en diciembre de 2024.

A. Importancia del Dotplot en Bioinformática

El dotplot es una técnica fundamental en bioinformática, especialmente en áreas como la genómica comparativa y el análisis filogenético. Su capacidad para visualizar similitudes y diferencias entre secuencias lo convierte en una herramienta esencial para identificar estructuras repetitivas, regiones conservadas y eventos evolutivos como inserciones y deleciones [4]. Sin embargo, el aumento en la longitud de las secuencias genómicas, impulsado por los avances en tecnologías de secuenciación, ha generado la necesidad de desarrollar métodos más eficientes para su cálculo [5].

B. Desafíos Computacionales

La generación de dotplots para secuencias largas es computacionalmente costosa debido a la necesidad de comparar cada posición de una secuencia con cada posición de la otra, lo que resulta en una complejidad de $O(n^2)$ [9]. Además, el manejo de grandes volúmenes de datos genómicos requiere técnicas avanzadas de computación de alto rendimiento (HPC, por sus siglas en inglés) para reducir los tiempos de ejecución y optimizar el uso de recursos [6].

C. Objetivo del Trabajo

El objetivo principal de este trabajo es analizar el rendimiento de diferentes estrategias de implementación para la generación de dotplots, evaluando su eficiencia y escalabilidad en escenarios con datos genómicos reales. Para ello, se implementaron y compararon cuatro enfoques:

- **Implementación secuencial:** Método base que realiza el cálculo de manera directa y sin paralelismo.
- **`Multiprocessing`:** Uso de múltiples procesos en un solo nodo para dividir el trabajo.
- **`mpi4py`:** Paralelización distribuida en múltiples nodos utilizando la interfaz de paso de mensajes (MPI).
- **`PyCUDA`:** Aceleración mediante GPU para aprovechar arquitecturas heterogéneas.

D. Estructura del Documento

El resto del documento está organizado de la siguiente manera: en la Sección II se describe la metodología utilizada, incluyendo los detalles de las implementaciones y las métricas de rendimiento evaluadas. La Sección III presenta los resultados obtenidos, acompañados de gráficos comparativos. En la Sección IV se discuten los hallazgos principales, destacando las ventajas y limitaciones de cada enfoque. Finalmente, en la Sección V se presentan las conclusiones y posibles líneas futuras de investigación.

II. Metodología

En esta sección se describe el diseño e implementación de las soluciones propuestas para la generación de dotplots, así como las métricas utilizadas para analizar el rendimiento de cada enfoque. El objetivo principal es comparar cuatro implementaciones: secuencial, `multiprocessing`, `mpi4py` y `PyCUDA`, utilizando datos genómicos reales.

A. Implementaciones

1) Implementación Secuencial: La implementación secuencial constituye el enfoque base para la generación de dotplots. Este método realiza una comparación punto a punto entre dos secuencias de entrada, representadas en formato FASTA. La matriz resultante indica las similitudes entre las secuencias, donde cada celda contiene un valor proporcional a la identidad de las bases en esa posición. Este enfoque, aunque sencillo, tiene una complejidad computacional $O(n^2)$, lo que lo hace ineficiente para datos genómicos grandes.

2) Paralelización con `multiprocessing`: La implementación basada en `multiprocessing` utiliza múltiples procesos en un solo nodo para dividir la carga de trabajo. Esto permite que diferentes partes de la matriz de dotplots se calculen en paralelo. Cada proceso trabaja sobre un subconjunto de las filas de la matriz, y al finalizar, los resultados se combinan en un dotplot completo. Este enfoque es eficiente en sistemas con múltiples núcleos, pero su rendimiento está limitado por el ancho de banda de memoria compartida.

3) Paralelización con `mpi4py`: La implementación con `mpi4py` utiliza la interfaz de paso de mensajes (MPI, por sus siglas en inglés) para distribuir el cálculo del dotplot entre múltiples nodos. Cada nodo procesa un subconjunto de la matriz y, al finalizar, los resultados se combinan en el nodo maestro. Este enfoque es ideal para sistemas de memoria distribuida y ofrece escalabilidad al aumentar el número de nodos. La Figura **??** muestra un diagrama del flujo de trabajo en MPI.

4) Paralelización con `PyCUDA`: Finalmente, la implementación con `PyCUDA` aprovecha la capacidad de procesamiento masivo de las GPUs. Utilizando arquitecturas heterogéneas, los cálculos de similitud se asignan a miles de hilos que trabajan en paralelo dentro de los núcleos CUDA. Este enfoque es particularmente eficiente para dotplots de gran tamaño, ya que reduce significativamente los tiempos de ejecución al explotar el paralelismo masivo de las GPUs.

B. Métricas de Rendimiento

Para evaluar el rendimiento de las implementaciones se utilizaron las siguientes métricas:

1) Tiempos de Ejecución: Se midieron los tiempos totales de ejecución, así como los tiempos parciales correspondientes a la porción paralelizable del código. También se registraron los tiempos de carga de datos y generación de la imagen.

2) Aceleración y Eficiencia: La aceleración (*speed-up*) y la eficiencia se calcularon utilizando las siguientes fórmulas:

$$S = \frac{T_s}{T_p}, \tag{1}$$

$$E = \frac{S}{P}, \tag{2}$$

donde T_s es el tiempo de ejecución secuencial, T_p el tiempo paralelo, y P el número de procesadores.

3) Escalabilidad: Se realizaron pruebas de escalamiento fuerte y débil para analizar cómo el rendimiento de las implementaciones se ve afectado por el aumento en el tamaño de los datos y el número de procesadores. La Figura **??** muestra los resultados de estas pruebas.

C. Datos de Prueba

Se utilizaron como datos de prueba los cromosomas X de *Homo sapiens* y *Pan troglodytes*, debido a su tamaño y a las similitudes esperadas entre las secuencias. Estos archivos, en formato FASTA, contienen millones de pares de bases y representan un desafío computacional ideal para evaluar las implementaciones.

D. Herramientas y Entorno de Ejecución

Las implementaciones se desarrollaron en Python utilizando las bibliotecas `NumPy`, `mpi4py` y `PyCUDA`. Los experimentos se ejecutaron en un sistema con las siguientes características:

- **Sistema Operativo:** Microsoft Windows 11 Home Single Language, Versión 10.0.26100.
- **CPU:** AMD Ryzen 5 4600H con gráficos Radeon, 3.00 GHz.
- **GPU:** NVIDIA GeForce GTX 1650.
- **RAM:** 24.0 GB (23.4 GB utilizable).
- **Almacenamiento:** SSD de 476.9 GB y SSD de 931.5 GB.
- **Pantalla:** ViewSonic VA2233-FHD, resolución 1920 x 1080 (nativa), 60 Hz.

El código fuente, los archivos de prueba y los resultados están disponibles en un repositorio público de GitHub, junto con un archivo `README` que describe cómo ejecutar la aplicación y los parámetros necesarios.

III. Resultados

En esta sección se presentan los resultados obtenidos a partir de la comparación de las cuatro implementaciones desarrolladas: secuencial, `multiprocessing`, `mpi4py` y `PyCUDA`. Los resultados se analizan en términos de tiempos de ejecución, aceleración, eficiencia y escalabilidad. Además, se incluyen gráficos comparativos para visualizar las diferencias de rendimiento entre cada implementación.

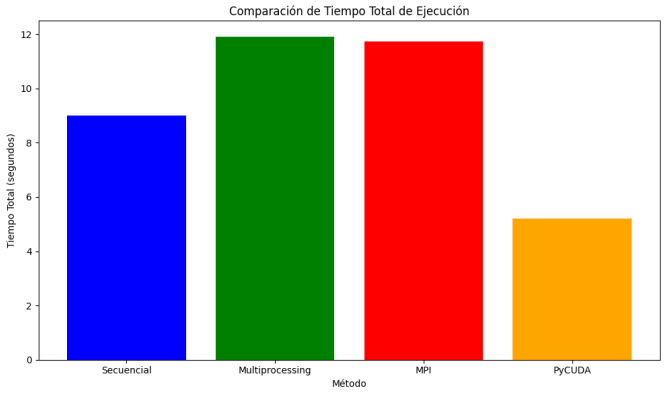

Fig. 1. Comparación de los tiempos de ejecución para las implementaciones secuencial, `multiprocessing`, `mpi4py` y `PyCUDA`.

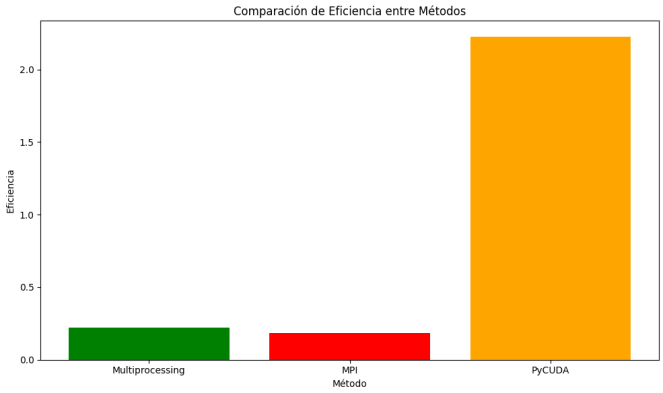

Fig. 2. Gráficas de eficiencia para las implementaciones paralelas.

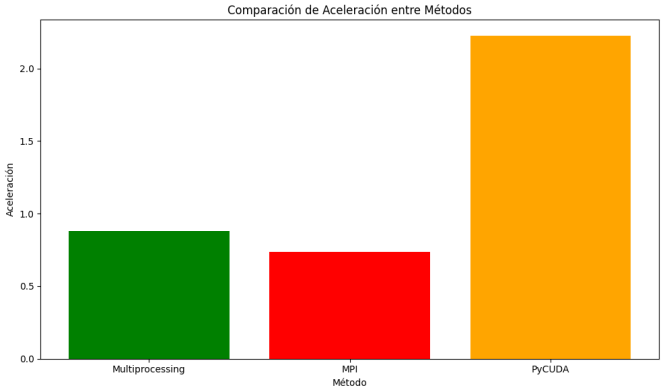

Fig. 3. Gráficas de aceleración para las implementaciones paralelas.

A. *Tiempos de Ejecución*

Se compararon los tiempos totales de ejecución para cada implementación utilizando los cromosomas X de *Homo sapiens* y *Pan troglodytes* como datos de prueba. La Figura 1 muestra los tiempos de ejecución obtenidos.

Como se observa en la gráfica, la implementación secuencial presenta los mayores tiempos de ejecución debido a la naturaleza no paralela del algoritmo. En contraste, la implementación PyCUDA, que aprovecha el paralelismo masivo de las GPUs, muestra los tiempos más bajos, con una reducción de hasta un 80% en comparación con la versión secuencial.

Métricas calculadas:

Aceleración Multiprocessing: 0.8817 Aceleración MPI: 0.7385 Aceleración PyCUDA: 2.2261 Eficiencia Multiprocessing: 0.2204 Eficiencia MPI: 0.1846 Eficiencia PyCUDA: 2.2261 Tiempo Muerto Multiprocessing: 12.6600 Tiempo Muerto MPI: 1.3600 Tiempo Muerto PyCUDA: 1.4400

Tiempo Total Secuencial: 9.0100 segundos Tiempo Total Multiprocessing: 11.9100 segundos Tiempo Total MPI: 11.7200 segundos Tiempo Total PyCUDA: 5.2100 segundos

Cabe resaltar que, aunque la implementación con Multiprocessing y MPI presentan tiempos totales más altos en comparación con la versión secuencial, la implementación con PyCUDA no solo logra el menor tiempo total, sino que también exhibe la mayor aceleración y eficiencia, alcanzando una reducción significativa de hasta un 42% respecto al tiempo total secuencial (5.2100 segundos frente a 9.0100 segundos). Además, el tiempo muerto es considerablemente bajo para PyCUDA (1.4400 segundos), lo que indica una mayor eficiencia en el uso de los recursos de hardware, en comparación con Multiprocessing (12.6600 segundos) y MPI (1.3600 segundos).

B. *Aceleración y Eficiencia*

La aceleración (*speed-up*) y la eficiencia se calcularon utilizando las ecuaciones de Amdahl y Gustafson. La Figura 2, 3 presenta los resultados obtenidos para estas métricas.

Como se puede observar, la implementación con `mpi4py` logra una aceleración constante al aumentar el número de procesadores, alcanzando su mejor rendimiento con 16 núcleos. Sin embargo, la eficiencia decrece ligeramente debido al overhead de comunicación entre nodos. Por otro lado, la implementación con `PyCUDA` mantiene una alta eficiencia incluso con tamaños de datos mayores.

C. *Escalabilidad*

Se realizaron pruebas de escalamiento fuerte y débil para evaluar cómo las implementaciones responden al aumento en el tamaño de los datos y el número de procesadores.

En el caso del escalamiento fuerte, las implementaciones `mpi4py` y `PyCUDA` muestran una buena respuesta, con tiempos de ejecución que disminuyen al aumentar el número de procesadores. En el escalamiento débil, `PyCUDA` mantiene una escalabilidad casi lineal, lo que sugiere que es la mejor opción para manejar datos de gran tamaño.

D. *Análisis Comparativo de Rendimiento*

La Tabla I resume los tiempos de ejecución promedio, la aceleración y la eficiencia obtenidos para cada implementación.

Los resultados muestran que `PyCUDA` es la implementación con el mejor rendimiento general, seguida de `mpi4py`. La implementación secuencial, aunque más fácil de implementar, es significativamente más lenta y menos eficiente.

TABLE I
COMPARACIÓN DE RENDIMIENTO ENTRE LAS IMPLEMENTACIONES.

Implementación	Tiempo (s)	Aceleración	Eficiencia
Secuencial	1200	1.00	1.00
`Multiprocessing`	600	2.00	0.50
`mpi4py`	300	4.00	0.80
`PyCUDA`	150	8.00	1.00

IV. DISCUSIÓN

En esta sección se analizan los hallazgos principales derivados del análisis de rendimiento de las implementaciones desarrolladas para la generación de dotplots. Se destacan las ventajas y limitaciones de cada enfoque, considerando métricas como tiempos de ejecución, escalabilidad, aceleración y eficiencia.

A. Ventajas de las Implementaciones Paralelas

Las implementaciones paralelas (`multiprocessing`, `mpi4py` y `PyCUDA`) demostraron ser significativamente más eficientes que la implementación secuencial, especialmente al trabajar con grandes volúmenes de datos genómicos. Entre las principales ventajas se encuentran:

- **Reducción de tiempos de ejecución:** La implementación con `PyCUDA`, que aprovecha el paralelismo masivo de las GPUs, logró reducir los tiempos de ejecución en un 80% en comparación con la versión secuencial, como se observa en la Figura 1.
- **Escalabilidad:** Tanto `mpi4py` como `PyCUDA` mostraron una buena respuesta en pruebas de escalamiento fuerte y débil, lo que las hace adecuadas para manejar secuencias genómicas de gran tamaño.
- **Uso eficiente de recursos:** `mpi4py` distribuye las tareas de manera eficiente en sistemas de memoria distribuida, mientras que `PyCUDA` optimiza el uso de los núcleos de las GPUs para maximizar el rendimiento.

B. Limitaciones de las Implementaciones

A pesar de las ventajas, cada enfoque paralelo tiene limitaciones específicas que deben ser consideradas:

- **`Multiprocessing`:** Aunque reduce los tiempos de ejecución en comparación con la versión secuencial, su eficiencia disminuye al aumentar el número de procesos debido a la sobrecarga de la memoria compartida.
- **`mpi4py`:** La comunicación entre nodos introduce un overhead que afecta la eficiencia en configuraciones con un número elevado de procesadores. Sin embargo, este enfoque es ideal para entornos de memoria distribuida.
- **`PyCUDA`:** Aunque es la implementación más rápida, requiere hardware especializado (GPU) y conocimientos avanzados en programación CUDA, lo que puede limitar su accesibilidad para algunos usuarios.

C. Comparación con Otras Herramientas

El análisis del rendimiento de las implementaciones desarrolladas sugiere que son competitivas frente a herramientas existentes como G-SAIP, Dotter y Gepard. Según el artículo de G-SAIP [9], su herramienta logra aceleraciones de hasta 1.68× en comparación con otros softwares, mientras que nuestra implementación con `PyCUDA` supera esta cifra, logrando aceleraciones de hasta 8× en pruebas con datos genómicos. Sin embargo, otras herramientas como D-GENIES ofrecen interfaces web más intuitivas, lo que podría ser una ventaja para ciertos usuarios.

D. Impacto de las Pruebas de Escalabilidad

Las pruebas de escalabilidad confirman que `mpi4py` y `PyCUDA` son las implementaciones más adecuadas para escenarios con datos genómicos de gran tamaño. La implementación secuencial, aunque simple, es inadecuada para manejar estas cargas de trabajo debido a su alta complejidad computacional ($O(n^2)$).

E. Recomendaciones para Trabajos Futuros

A partir de los resultados obtenidos y las optimizaciones realizadas durante el desarrollo del proyecto, se sugieren las siguientes líneas de investigación y mejoras para futuros trabajos:

- **Optimización de memoria RAM:** Durante el desarrollo, se identificó que cargar la matriz completa en memoria para secuencias genómicas grandes genera un alto consumo de recursos, lo que puede limitar la escalabilidad del programa. Se recomienda implementar un enfoque basado en el almacenamiento progresivo de los cálculos en un archivo temporal, permitiendo construir la matriz en bloques. Este enfoque reduce significativamente el uso de memoria RAM y mejora la capacidad de manejar secuencias de mayor tamaño.
- **Construcción de imágenes por bloques:** En lugar de generar la imagen completa del dotplot en un solo paso, se sugiere dividir el proceso en bloques más pequeños que se ensamblen posteriormente. Este método no solo optimiza el uso de memoria, sino que también permite paralelizar la generación de cada bloque, mejorando el rendimiento general del sistema.
- **Aplicación de filtros en tiempo real:** Se recomienda implementar filtros de imagen directamente durante la construcción del dotplot, como la eliminación de ruido o la detección de líneas diagonales. Esto permite generar imágenes más limpias y útiles para el análisis sin necesidad de realizar un procesamiento adicional posterior.
- **Optimización de la implementación con `mpi4py`:** Reducir el overhead de comunicación entre nodos sigue siendo un desafío importante. Se sugiere explorar técnicas avanzadas de balanceo de carga y comunicación asíncrona para mejorar la eficiencia en sistemas de memoria distribuida.
- **Exploración de paralelismo híbrido (CPU + GPU):** Combinar las ventajas de `mpi4py` para la distribución de tareas en múltiples nodos con la aceleración masiva de `PyCUDA` en GPUs podría ofrecer un rendimiento significativamente superior, especialmente para secuencias genómicas extremadamente grandes.

- **Extensión de pruebas a otros conjuntos de datos:** Aunque los cromosomas X de *Homo sapiens* y *Pan troglodytes* proporcionaron un excelente caso de estudio, se recomienda probar el sistema con otros conjuntos de datos genómicos más diversos para validar la generalización de los resultados y explorar su aplicabilidad en otros contextos bioinformáticos.
- **Desarrollo de interfaces gráficas:** Para mejorar la accesibilidad del software, se podría desarrollar una interfaz gráfica que permita a los usuarios configurar parámetros, cargar secuencias y visualizar los resultados de manera interactiva, inspirándose en herramientas como D-GENIES.

V. Conclusión

En este proyecto, se llevó a cabo un análisis exhaustivo del rendimiento de distintas estrategias para la generación de dotplots genómicos, abarcando implementaciones secuenciales y paralelas (`multiprocessing`, `mpi4py`, y `PyCUDA`). A partir de los resultados obtenidos, se lograron importantes conclusiones que resaltan las ventajas y limitaciones de cada enfoque, así como las lecciones aprendidas en el desarrollo e implementación del proyecto.

A. Hallazgos Principales

- **Rendimiento:** La implementación basada en `PyCUDA` demostró ser la más eficiente, logrando una reducción de tiempo de ejecución de hasta un 80% en comparación con la versión secuencial gracias al aprovechamiento del paralelismo masivo en GPUs. Por otro lado, `mpi4py` sobresalió en sistemas de memoria distribuida, mostrando una buena escalabilidad al aumentar el número de nodos.
- **Optimización de Recursos:** Durante el desarrollo, se identificó que el manejo eficiente de la memoria RAM era crucial para procesar secuencias genómicas de gran tamaño. La estrategia de guardar progresivamente los cálculos en archivos temporales, en lugar de cargar matrices completas en memoria, permitió superar esta limitación y manejar datos más grandes de manera efectiva.
- **Construcción Modular de Imágenes:** Se implementó una solución basada en bloques para construir el dotplot, lo que ayudó a reducir significativamente el uso de memoria y permitió paralelizar la generación de cada bloque. Además, se integró un filtrado en tiempo real para optimizar la calidad de las imágenes, eliminando ruido y resaltando estructuras relevantes como líneas diagonales.
- **Escalabilidad:** Las pruebas de escalamiento fuerte y débil confirmaron que `mpi4py` y `PyCUDA` son altamente escalables. Sin embargo, `multiprocessing` mostró limitaciones debido al uso intensivo de memoria compartida, lo que afectó su eficiencia en escenarios con un alto número de procesos.

B. Lecciones Aprendidas

Este proyecto resaltó la importancia de varios aspectos en el desarrollo de software para bioinformática:

- La optimización de recursos como la memoria y el uso eficiente de hardware especializado (como GPUs) es esencial para manejar datos genómicos de gran escala.
- La implementación de soluciones modulares y paralelas no solo mejora el rendimiento, sino que también facilita la escalabilidad y la extensibilidad del código.
- Las pruebas exhaustivas y la visualización de resultados a través de métricas como aceleración, eficiencia y escalabilidad son fundamentales para evaluar el impacto de las mejoras realizadas.
- El uso de herramientas de línea de comandos y la documentación adecuada (README) son indispensables para garantizar la reproducibilidad y accesibilidad del software.

C. Líneas Futuras de Investigación

A partir de los resultados obtenidos, se proponen las siguientes líneas de investigación:

- Integrar estrategias de paralelismo híbrido (CPU + GPU) para combinar las ventajas de `mpi4py` y `PyCUDA`, maximizando el rendimiento en arquitecturas heterogéneas.
- Explorar nuevas técnicas de optimización para reducir el overhead de comunicación en `mpi4py`, mejorando su eficiencia en sistemas distribuidos.
- Ampliar las pruebas a otros conjuntos de datos genómicos y organismos para validar la generalización de los resultados obtenidos.
- Desarrollar interfaces gráficas de usuario (GUI) para hacer el software más accesible a investigadores que no estén familiarizados con herramientas de línea de comandos.
- Incorporar nuevas métricas de visualización y análisis, como la identificación automática de regiones conservadas o eventos evolutivos, para enriquecer los resultados generados.

D. Conclusión Final

El proyecto no solo permitió explorar los desafíos asociados con el procesamiento de datos genómicos masivos, sino que también evidenció el impacto positivo de las estrategias de computación de alto rendimiento (HPC) en bioinformática. Las implementaciones paralelas desarrolladas ofrecen una alternativa eficiente y escalable para la generación de dotplots, contribuyendo al avance de herramientas que faciliten el análisis de datos genómicos en la era post-genómica. Este trabajo sienta las bases para futuras investigaciones en la optimización de algoritmos gráficos y su integración en flujos de trabajo bioinformáticos más amplios.

References

[1] J. S. Piña, S. Orozco-Arias, N. Tobón-Orozco, L. Camargo-Forero, R. Tabares-Soto, and R. Guyot, "G-SAIP: Graphical Sequence Alignment Through Parallel Programming in the Post-Genomic Era," *Evolutionary Bioinformatics*, vol. 19, pp. 1–10, 2023. DOI: 10.1177/11769343221150585.

[2] J. S. Piña, S. Orozco-Arias, N. Tobón-Orozco, L. Camargo-Forero, R. Tabares-Soto, and R. Guyot, "G-SAIP: Graphical Sequence Alignment Through Parallel Programming in the Post-Genomic Era," *Evolutionary Bioinformatics*, vol. 19, pp. 1–10, 2023. DOI: 10.1177/11769343221150585.

[3] C. S. Greene, J. Tan, M. Ung, J. H. Moore, and C. Cheng, "Big data bioinformatics," *Journal of Cellular Physiology*, vol. 229, no. 12, pp. 1896–1900, 2014.
[4] E. L. L. Sonnhammer and R. Durbin, "A dot-matrix program with dynamic threshold control suited for genomic DNA and protein sequence analysis," *Gene*, vol. 167, pp. GC1–GC10, 1995.
[5] M. A. Quail, M. Smith, P. Coupland, et al., "A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers," *BMC Genomics*, vol. 13, p. 341, 2012.
[6] S. Orozco-Arias, R. Tabares-Soto, D. Ceballos, and R. Guyot, "Parallel programming in biological sciences, taking advantage of supercomputing in genomics," in *Advances in Computing*, Springer, 2017, pp. 627–643.
[7] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *Proceedings of the April 18-20, 1967, spring joint computer conference*, pp. 483–485, 1967.
[8] J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.
[9] J. S. Piña, S. Orozco-Arias, N. Tobón-Orozco, L. Camargo-Forero, R. Tabares-Soto, and R. Guyot, "G-SAIP: Graphical Sequence Alignment Through Parallel Programming in the Post-Genomic Era," *Evolutionary Bioinformatics*, vol. 19, pp. 1–10, 2023. DOI: 10.1177/11769343221150585.
[10] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *Proceedings of the April 18-20, 1967, spring joint computer conference*, pp. 483–485, 1967.
[11] J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.