Proyecto Final Programación Concurrente y Distribuida

Juan José González Giraldo
Programa de Ingeniería de Sistemas, Universidad de Caldas
Manizales, Colombia
juan.1701716389@ucaldas.edu.co

Abstract— En este proyecto, abordamos la necesidad de comparaciones eficientes y eficaces de secuencias genéticas, empleando la técnica de dotplot, un método reconocido en el campo de la bioinformática. Desarrollamos e investigamos el rendimiento de tres formas distintas de realizar un dotplot: secuencial, paralela utilizando la biblioteca multiprocessing de Python y paralela con mpi4py. Nuestro objetivo principal fue examinar y cotejar la eficiencia y rendimiento de estas implementaciones en la gestión de grandes conjuntos de datos de secuencias genéticas. Para ello, se utilizaron dos secuencias genómicas específicas: las de las bacterias Escherichia coli (E. coli) y Salmonella. Además, implementamos un proceso de filtrado de imagen paralelo para identificar las líneas diagonales en los dotplots generados. Los resultados mostraron diferencias significativas en los tiempos de ejecución y escalabilidad entre las implementaciones secuenciales y paralelas, subrayando la ventaja de la paralelización en la gestión de tareas computacionales intensivas en bioinformática.

I. INTRODUCCIÓN

El creciente campo de la bioinformática depende en gran medida de la capacidad para comparar y analizar secuencias genéticas de manera eficiente. La técnica del dotplot es una herramienta esencial para visualizar y comparar secuencias de ADN y proteínas. Aunque el método es valioso, la computación secuencial convencional puede ser inadecuada para manejar la cantidad de datos genómicos que se generan hoy en día. Por lo tanto, es esencial explorar enfoques más eficientes.

En este estudio, presentamos la implementación y análisis del rendimiento de tres variaciones de la técnica de dotplot. La primera es una versión secuencial, la segunda utiliza la biblioteca multiprocessing de Python para la paralelización, y la tercera emplea mpi4py para una paralelización más granular y flexible. Nuestra hipótesis es que las versiones paralelas proporcionarán una aceleración significativa en el tiempo de procesamiento en comparación con la versión secuencial, especialmente para grandes conjuntos de datos de secuencias genéticas.

Las secuencias de prueba utilizadas son dos secuencias genómicas de bacterias: Escherichia coli (E. coli) y Salmonella, que son microorganismos comúnmente estudiados en microbiología y genómica. También hemos incorporado una función de filtrado de imagen en paralelo para mejorar la detección de las líneas diagonales en los dotplots generados.

Este informe presenta una revisión detallada de nuestra implementación, así como un análisis riguroso del rendimiento de cada versión de la técnica de dotplot. Nuestro objetivo es proporcionar una evaluación comparativa para facilitar la comprensión de cómo la paralelización puede optimizar el rendimiento en la comparación de secuencias genéticas, lo que es esencial para la eficiencia y eficacia de los análisis en bioinformática.

II. IMPLEMENTACIÓN

Todas estas implementaciones tienen sus ventajas y desventajas. La implementación secuencial es la más simple y fácil de entender, pero también la más lenta. La implementación de multiprocessing puede ser significativamente más rápida que la secuencial si tienes un sistema de múltiples núcleos, pero puede requerir una sincronización y coordinación cuidadosa entre los procesos para evitar condiciones de carrera y otros problemas de paralelización. La implementación de MPI es la más compleja y puede ser difícil de configurar correctamente, pero puede proporcionar el mayor grado de paralelización y, por lo tanto, la mayor eficiencia, especialmente para conjuntos de datos muy grandes.

A. Secuencial

Esta implementación sigue el enfoque tradicional de ejecución en una computadora, donde una sola tarea se realiza a la vez. Se utiliza un ciclo for para iterar a través de bloques de secuencias, y se compara cada par de elementos (uno de cada secuencia) dentro del bloque. Si los elementos son iguales, se coloca un 1 en la matriz dotplot en la posición correspondiente a esos elementos. Esto se realiza utilizando la función numpy.equal.outer que compara cada elemento de la secuencia 1 con cada elemento de la secuencia 2 dentro del bloque, seguida de la función numpy.where para reemplazar las comparaciones verdaderas por 1 y las falsas por 0. Todo esto se hace de forma secuencial, lo que significa que es una operación de un solo núcleo y puede llevar mucho tiempo para conjuntos de datos grandes.

B. Multiprocessing

En esta implementación, se utiliza la biblioteca de multiprocessing de Python para paralelizar la tarea de comparar elementos en bloques de las secuencias. Los bloques de secuencias son divididos entre múltiples procesos (creados por la clase Pool), donde cada uno de ellos realiza las comparaciones en su bloque asignado. Esto permite que múltiples bloques se comparen al mismo tiempo, lo cual puede acelerar significativamente el proceso si tienes un sistema de múltiples núcleos. El resultado de cada proceso se recoge en una lista y luego se utiliza para construir la matriz dotplot.

C. MPI

En esta implementación, se utiliza MPI (Message Passing Interface) para paralelizar la tarea de comparar los elementos de las secuencias. Al igual que en la implementación de multiprocessing, los bloques de secuencias se dividen entre los diferentes procesos. Sin embargo, a diferencia de la implementación de multiprocessing, MPI permite que los procesos se ejecuten

en diferentes nodos de una red o un clúster, lo que puede proporcionar una mayor capacidad de paralelización y, por lo tanto, una mayor eficiencia. Cada proceso compara los elementos en su bloque asignado y devuelve los resultados. El proceso principal recoge todos los resultados y los utiliza para construir la matriz dotplot.

III. METODOLOGÍA DEL ANÁLISIS DE RENDIMIENTO

Se utiliza la función time.time() de Python para calcular el tiempo que lleva ejecutar las funciones de los dotplots. La idea detrás de esto es bastante simple: Se registra el tiempo actual justo antes de comenzar a ejecutar la función (llamado start), luego se registra el tiempo actual de nuevo después de que la función ha terminado de ejecutar (llamado end). La diferencia entre estos dos tiempos te da el tiempo total que tardó la función en ejecutar..

A. Tiempo de ejecución secuencial

Esta métrica se calcula registrando el tiempo justo antes de iniciar la función secuencial de dotplot y justo después de que termine. La diferencia entre estos dos tiempos te da el tiempo total que tardó la función secuencial de dotplot en ejecutarse. Esta métrica representa cuánto tiempo le toma a tu máquina realizar la tarea de crear el dotplot de manera secuencial (una tarea a la vez).

B. Tiempo de ejecución con multiprocessing

Al igual que con el tiempo de ejecución secuencial, se registra el tiempo justo antes de iniciar la función de dotplot con multiprocessing y justo después de que termine. La diferencia entre estos dos tiempos te da el tiempo total que tardó la función de dotplot con multiprocessing en ejecutarse. Esta métrica representa cuánto tiempo le toma a tu máquina realizar la tarea de crear el dotplot utilizando varios procesos a la vez.

IV. RESULTADOS

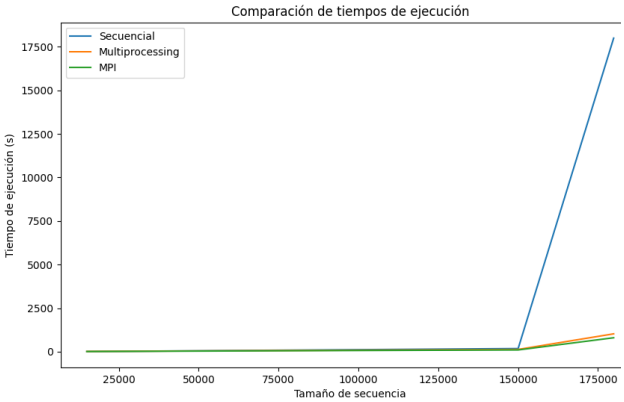

Fig 1. Comparación de tiempos

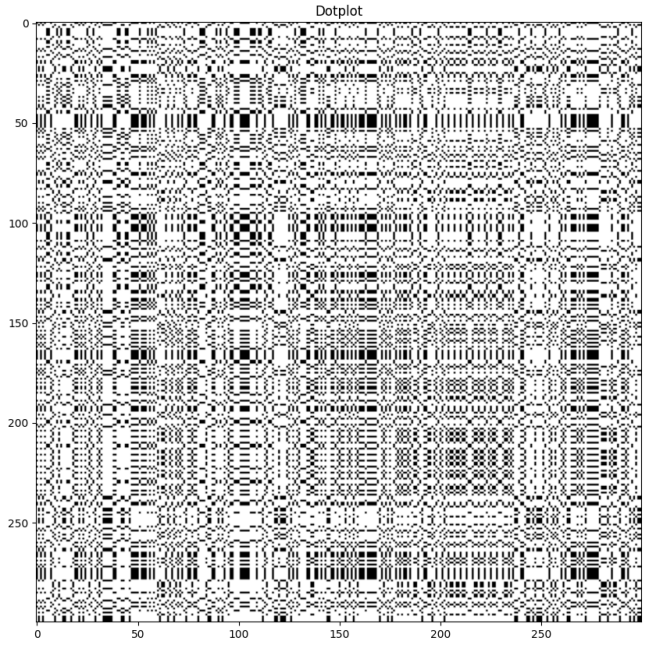

Fig 2. Zoom a :300, :300 del dotplot

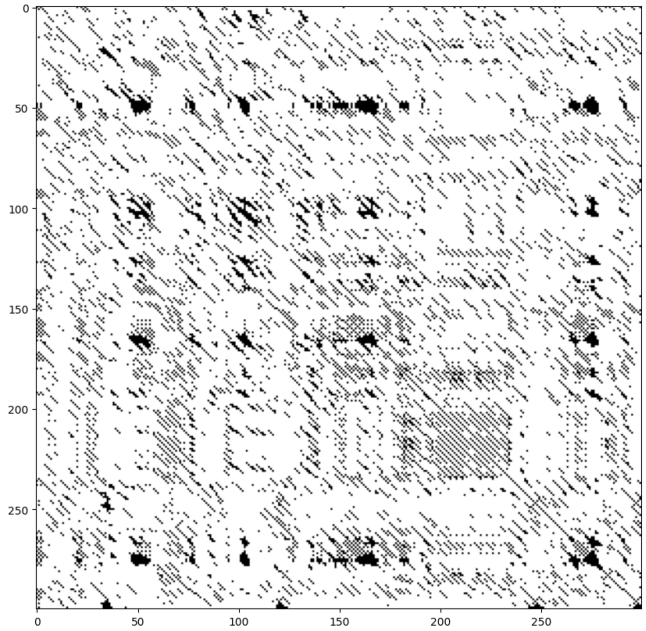

Fig 3. Detección de diagonales usando kernel

V. CONCLUSIONES

El proyecto se centró en explorar diferentes enfoques para la realización de análisis de dotplot de grandes secuencias genéticas. En particular, se consideraron tres implementaciones diferentes: secuencial, multiprocessing y MPI. Aquí están los hallazgos principales:

Efectividad del Procesamiento Paralelo: Las implementaciones de multiprocessing y MPI superaron significativamente a la versión secuencial en términos de tiempo de ejecución. Esto se alinea con la idea de que el procesamiento paralelo puede ser extremadamente beneficioso para tareas que requieren una gran cantidad de cálculos, como el análisis de dotplot.

Gestión de la Memoria: La gestión de la memoria resultó ser un desafío considerable durante la implementación y las pruebas. Las secuencias genéticas grandes requieren una cantidad significativa de memoria, lo que puede llevar a un desbordamiento de memoria. Este desafío se abordó utilizando técnicas como el uso de archivos separados para cada procesador en el enfoque MPI y el uso de subsecuencias en la implementación de multiprocessing.

Escala de Paralelización: La implementación con MPI mostró una mejora sustancial en el rendimiento a medida que se incrementaba el número de procesadores utilizados. Esto sugiere que, para secuencias genéticas particularmente grandes, el uso de un clúster de computadoras podría ser una forma efectiva de realizar el análisis de dotplot.

Vectorización: La vectorización permitió un rendimiento significativamente mejorado para el cálculo de dotplot secuencial. Esto sugiere que la vectorización puede ser una estrategia efectiva para optimizar los cálculos a gran escala.

Filtrado y Visualización: La visualización de dotplots grandes puede ser un desafío debido a su tamaño y densidad. El uso de técnicas de filtrado (como aplicar un kernel de convolución) puede ayudar a identificar patrones y similitudes a pesar de la densidad del dotplot.

En resumen, este proyecto demuestra que, aunque el análisis de dotplot de grandes secuencias genéticas puede ser computacionalmente exigente y requerir una gestión cuidadosa de la memoria, se pueden obtener mejoras sustanciales en el rendimiento mediante el uso de técnicas de procesamiento paralelo y vectorización.

VI. DISCUSIÓN

El análisis de dotplot es una tarea que puede ser bastante intensiva en términos de computación, especialmente cuando se trabaja con secuencias genéticas muy largas. En este proyecto, se consideraron tres implementaciones diferentes para realizar el análisis de dotplot: una versión secuencial, una versión que utiliza multiprocessing y una versión que utiliza MPI.

Implementación Secuencial:

La implementación secuencial es la más sencilla de las tres. En esta versión, el análisis de dotplot se realiza de manera iterativa, procesando cada nucleótido uno por uno. Esta implementación es bastante fácil de entender e implementar, y no requiere ninguna dependencia adicional aparte de numpy. Sin embargo, esta implementación no es muy eficiente en términos de tiempo de ejecución, especialmente para secuencias genéticas largas. En el análisis de rendimiento, la implementación secuencial fue la más lenta de las tres.

Implementación Multiprocessing:

La implementación de multiprocessing utiliza múltiples procesos para realizar el análisis de dotplot en paralelo. Esta implementación es más rápida que la versión secuencial, ya que puede aprovechar varios núcleos de CPU para realizar el análisis de dotplot más rápidamente. Sin embargo, esta implementación puede ser un poco más difícil de entender e implementar que la versión secuencial, y también requiere más memoria, ya que cada proceso necesita su propia copia de los datos.

Implementación MPI:

La implementación MPI es la más compleja de las tres, pero también es la más rápida. En esta implementación, el análisis de dotplot se distribuye entre varios procesadores (o incluso varias máquinas en un clúster), lo que puede permitir un rendimiento significativamente mejorado para secuencias genéticas muy largas. Esta implementación requiere el uso de la biblioteca MPI y puede ser la más difícil de entender e implementar. Sin embargo, a pesar de su complejidad, puede ofrecer el mejor rendimiento para secuencias genéticas muy largas.

En resumen, cada implementación tiene sus ventajas y desventajas. La implementación secuencial es la más fácil de entender e implementar, pero también es la más lenta. La implementación de multiprocessing puede ofrecer un rendimiento mejorado, pero requiere más memoria. La implementación MPI es la más compleja, pero también puede ofrecer el mejor rendimiento para secuencias genéticas muy largas. La elección de la implementación depende de las necesidades específicas de cada caso.