PROYECTO FINAL

IRENE GONZÁLEZ GONZÁLEZ
MARTA GORRAIZ

2025-2026

ÍNDICE

1. INTRODUCCIÓN

Este proyecto se centra en la implementación y análisis de la multiplicación de una matriz dispersa por un vector, una operación común en muchos ámbitos, como por ejemplo en ingeniería.

El objetivo del proyecto es resolver esta operación de forma eficiente, implementando una versión secuencial y dos paralelas, una versión con OpenMP y otra con MPI. Para este análisis, se trabajó con diferentes tamaños de entrada (matrices pequeñas, medianas y grandes), y en el caso de la versión MPI se probó también con diferentes configuraciones de número de núcleos: 1, 2, 4, 8 y 16. El propósito final es evaluar el speedup, la escalabilidad y la eficiencia de cada enfoque según el tipo de entrada y la cantidad de recursos paralelos utilizados.

2. RESULTADOS OBTENIDOS

A continuación se van a mostrar, en las siguientes imágenes, los resultados obtenidos al ejecutar las distintas versiones del programa: secuencial, paralela con OpenMP y paralela con MPI.

Las pruebas se realizaron con tres matrices de diferentes tamaños y se midieron los tiempos de ejecución usando distintos números de hilos o núcleos. Los gráficos permiten comparar el comportamiento de cada versión y analizar cómo afecta el tamaño de la matriz y el grado de paralelismo al rendimiento general del algoritmo.

Metodo	N_filas	Hilos	Tiempo_segundos
Secuencial	4	1	0.000026
Secuencial	500	1	0.002908
Secuencial	1391	1	0.004925

Metodo	N_filas	Hilos	Tiempo_segundos
OpenMP	4	8	0.000366
OpenMP	500	8	0.002629
OpenMP	1391	8	0.000398

Metodo	N_filas	Núcleos	Tiempo_segundos
MPI	4	1	0.000005
MPI	4	2	0.000029
MPI	4	4	0.000016
MPI	4	8	0.000421
MPI	4	16	0.001171
MPI	5	1	0.000004
MPI	500	1	0.000021
MPI	500	2	0.000017
MPI	500	4	0.000020
MPI	500	8	0.000202
MPI	500	16	0.000448
MPI	1391	1	0.000027
MPI	1391	2	0.000056
MPI	1391	4	0.000330
MPI	1391	8	0.000478
MPI	1391	16	0.002159

2.1 COMPARACIÓN SECUENCIAL VS PARALELO

Para comprobar el rendimiento de las diferentes versiones del programa (secuencial, OpenMP y MPI), se hicieron pruebas utilizando matrices de tres tamaños distintos: una muy pequeña de 4 filas, una intermedia de 500 y otra más grande de 1391 filas.

En el caso de la matriz pequeña, la versión secuencial fue la más rápida, con un tiempo de ejecución de apenas 0.000026 segundos. Esto puede deberse a que en problemas tan pequeños el coste que supone usar paralelismo (el llamado "overhead") es mayor que el beneficio que aporta.

Pero cuando el tamaño de la matriz empieza a crecer, las versiones paralelas comienzan a rendir mejor. Por ejemplo, con la matriz de 1391 filas, los resultados fueron muy distintos:

- Secuencial: 0.004925 segundos
- OpenMP con 8 hilos: 0.000398 segundos
- MPI con 16 núcleos: 0.002159 segundos

Esto deja claro que, cuando el problema es más grande, usar paralelismo realmente marca la diferencia y ayuda a reducir mucho el tiempo de ejecución.

2.2 COMPARACIÓN EN PARALELO CON DIFERENTES TAMAÑOS

En este apartado se analiza cómo afecta el tamaño del problema al rendimiento de las versiones paralelas. Para ello, se mantuvo constante el número de hilos o núcleos utilizados y se compararon los tiempos de ejecución al utilizar matrices de diferentes tamaños: 4, 500 y 1391 filas.

- **OpenMP con distintos tamaños**

En el caso de OpenMP, se utilizó una configuración con 8 hilos. Los resultados obtenidos muestran cómo cambia el rendimiento al aumentar el tamaño de la matriz:

- 4 filas: 0.000366 segundos
- 500 filas: 0.002629 segundos
- 1391 filas: 0.000398 segundos

A simple vista, puede sorprender que el tiempo con 1391 filas sea menor que con 500. Esto puede deberse a factores como la forma en que están distribuidos los elementos no nulos en la matriz o el comportamiento del acceso a memoria. En este caso, queda claro que, en general, el paralelismo escala mejor con matrices más grandes, aunque esa mejora no siempre sea lineal ni totalmente predecible. Hay muchos factores internos del sistema que influyen en el rendimiento final.

- **MPI con distintos tamaños**

En el caso de MPI, se usaron 16 núcleos para todas las pruebas. Al igual que en OpenMP, se evaluó cómo varía el tiempo de ejecución según el tamaño de la matriz:

- 4 filas: 0.001171 segundos
- 500 filas: 0.000448 segundos
- 1391 filas: 0.002159 segundos

Lo primero que llama la atención es que, aunque al aumentar el número de núcleos el tiempo debería reducirse, en la práctica eso no ocurre de forma indefinida. A partir de los 4 núcleos, el rendimiento empieza a empeorar en lugar de mejorar. Esto se puede deber a dos razones:

Por un lado, está el overhead de comunicación. En MPI, cada proceso funciona por separado y necesita intercambiar mensajes para coordinarse. Cuantos más procesos hay, más comunicación es necesaria, y eso introduce cierta latencia. Si el tamaño del problema no es lo suficientemente grande, ese coste adicional puede superar el beneficio de repartir el trabajo.

Por otro lado, está la fragmentación de la carga. Cuando se divide el trabajo entre muchos procesos, puede pasar que algunos de ellos apenas tengan trabajo real que hacer. Eso genera tiempos muertos y desperdicio de recursos. Si el problema no escala bien con el número de núcleos, se produce lo que se conoce como pérdida de escalabilidad.

2.3 MPI CON DIFERENTES NÚCLEOS

En este apartado se analiza el rendimiento de la versión MPI al variar el número de núcleos utilizados, manteniendo constante el tamaño del problema. Para ello, se utilizó la matriz más grande del conjunto de pruebas, con 1391 filas, y se midieron los tiempos de ejecución con 1, 2, 4, 8 y 16 núcleos. Los resultados fueron los siguientes:

- 1 núcleo: 0.000027 segundos
- 2 núcleos: 0.000056 segundos
- 4 núcleos: 0.000330 segundos
- 8 núcleos: 0.000478 segundos
- 16 núcleos: 0.002159 segundos

Aunque lo esperado sería que al aumentar el número de núcleos el tiempo de ejecución se redujera, en la práctica no ocurre así de forma indefinida. A partir de los 4 núcleos, el rendimiento empieza a empeorar en lugar de mejorar. Esto se debe, principalmente, a dos factores que afectan al paralelismo distribuido:

Por un lado, está el overhead de comunicación. En MPI, cada proceso trabaja en su propio espacio de memoria y necesita enviar mensajes a otros procesos para coordinarse. A medida que se suman más procesos, aumenta la cantidad de comunicación necesaria, lo que introduce latencias que pueden acabar siendo más costosas que el propio cálculo.

Por otro lado, aparece la fragmentación de la carga. Al dividir el trabajo entre muchos procesos, puede pasar que algunos tengan muy poco que hacer. Esto genera tiempos muertos, desequilibrios y, en consecuencia, una pérdida de eficiencia. Si el problema no escala proporcionalmente con el número de núcleos, se produce lo que se conoce como pérdida de escalabilidad.

En resumen, aumentar el número de núcleos no siempre mejora el rendimiento. De hecho, puede llegar a empeorar si el problema no es lo suficientemente grande o si no se reparte el trabajo de forma equilibrada entre los procesos.