Multiplicación de Matrices

Andrés Felipe Salgado Ramírez, Brighitte Vélez Castro, Juan David Céspedes Mendoza y

Karen Melissa Sepúlveda Parra

Facultad de Ingenierías, Universidad Tecnológica de Pereira

IS9D3: HPC (High Performance Computing)

Ramiro Andrés Barrios Valencia

19 de septiembre de 2025

Tabla de Contenido

RESUMEN.....4

INTRODUCCIÓN.....5

MARCO CONCEPTUAL.....6

1. High Performance Computing (HPC).....6

2. Multiplicación de Matrices.....6

3. Computación Secuencial.....6

4. Paralelismo.....7

5. Complejidad Computacional.....7

6. Hilos y Procesos.....7

7. Medición de Tiempos.....8

8. Speedup.....8

MARCO CONTEXTUAL.....10

1. Características de la Máquina.....10

2. Desarrollo.....10

2.1 *Multiplicación Secuencial*.....11

2.2 *Multiplicación con Hilos*.....11

2.3 *Multiplicación con Procesos e Hilos*.....11

2.4 *Automatización y Análisis de Resultados*.....12

PRUEBAS.....14

CONCLUSIONES.....21

BIBLIOGRAFÍA.....23

RESUMEN

En este trabajo se implementan y comparan distintos métodos para la multiplicación de matrices cuadradas, con el objetivo de analizar su rendimiento y eficiencia. Se desarrollaron versiones secuenciales y paralelas mediante hilos y procesos, evaluando su comportamiento en diferentes dimensiones de matrices.

La experimentación se automatizó a través de scripts que permiten compilar, ejecutar y registrar los resultados de manera sistemática. Posteriormente, los datos fueron procesados para generar tablas y gráficas que facilitan la interpretación de los tiempos de ejecución y el impacto del paralelismo.

Este estudio busca evidenciar las ventajas y limitaciones de cada enfoque, proporcionando una visión general sobre el aprovechamiento de los recursos computacionales en tareas de cálculo intensivo.

INTRODUCCIÓN

La multiplicación de matrices es una de las operaciones fundamentales en el ámbito del cómputo científico y la ingeniería, debido a su amplia aplicación en áreas como la simulación numérica, la estadística, la inteligencia artificial, el procesamiento de imágenes y la computación de alto rendimiento (HPC). Sin embargo, se trata de una tarea intensiva en el uso de recursos computacionales, especialmente cuando el tamaño de las matrices es grande, lo que convierte su optimización en un desafío relevante.

En este trabajo se aborda la implementación de diferentes estrategias para la multiplicación de matrices cuadradas, con el objetivo de analizar el impacto del paralelismo en la reducción de los tiempos de ejecución. Se desarrollaron versiones secuenciales, basadas en hilos (multithreading) y basadas en procesos (multiprocessing), permitiendo así comparar los beneficios y limitaciones de cada enfoque.

Para garantizar la validez de los resultados, se construyó un sistema de automatización que permite ejecutar múltiples pruebas sobre distintos tamaños de matrices, registrar métricas de tiempo y generar visualizaciones en forma de gráficos y tablas. De esta manera, se facilita la comparación del rendimiento de los distintos métodos.

El presente documento se estructura en varias secciones: inicialmente se presentan los fundamentos teóricos, luego se describe la metodología aplicada, se exponen los resultados experimentales obtenidos y finalmente se discuten las conclusiones más relevantes del estudio.

MARCO CONCEPTUAL

1. High Performance Computing (HPC)

La Computación de Alto Rendimiento (HPC) se refiere al uso de sistemas y técnicas que permiten ejecutar cálculos complejos en el menor tiempo posible. Su objetivo es aprovechar al máximo los recursos de hardware (procesadores, memoria, redes) mediante estrategias de paralelismo y optimización. HPC es fundamental en áreas como la simulación científica, inteligencia artificial, análisis de grandes volúmenes de datos y modelado de fenómenos físicos.

2. Multiplicación de Matrices

La multiplicación de matrices es una operación fundamental en matemáticas y computación, utilizada en álgebra lineal, gráficos por computadora, inteligencia artificial y simulaciones. Si A es una matriz de dimensión $m \times n$ y B una matriz $n \times p$, entonces el producto $C = A \cdot B$ será una matriz de dimensión $m \times p$, donde cada elemento se calcula como:

$$C_{ij} = \sum_{n=1}^{k} A_{in} * B_{nj}$$

3. Computación Secuencial

La computación secuencial consiste en ejecutar instrucciones una tras otra, en un único flujo. Este enfoque es simple pero limitado, ya que utiliza un solo procesador o núcleo, lo que genera tiempos de ejecución altos para problemas grandes.

Ejemplo: multiplicar matrices de forma secuencial implica que un único proceso realiza todas las operaciones de cálculo y almacenamiento de resultados.

4. Paralelismo

El paralelismo busca dividir un problema en partes más pequeñas que se ejecutan simultáneamente en múltiples núcleos o procesadores.

Tipos principales:

- **Paralelismo a nivel de datos:** dividir los datos para que distintos procesadores realicen la misma operación en subconjuntos diferentes.
- **Paralelismo a nivel de tareas:** diferentes procesos ejecutan operaciones distintas en paralelo.

5. Complejidad Computacional

La complejidad computacional mide el costo de un algoritmo en términos de tiempo y recursos (generalmente memoria). Para la multiplicación de matrices:

- **Método clásico:** $O(N^3)$.
- **Métodos avanzados (como Strassen):** $O(N^{2,81})$ o menores.

En HPC, reducir la complejidad práctica mediante paralelismo y optimización es clave.

6. Hilos y Procesos

- Proceso: unidad de ejecución independiente con su propio espacio de memoria.

- Hilo: unidad de ejecución más ligera que comparte la memoria del proceso principal.

En la multiplicación de matrices paralela:

Se pueden lanzar procesos distintos para manejar bloques de la matriz o bien usar hilos para que cada núcleo calcule filas/columnas en simultáneo.

7. Medición de Tiempos

La medición de los tiempos de ejecución se llevó a cabo mediante una función personalizada que ofrece diversas métricas de desempeño, utilizando time.h con clock_gettime y timespec para determinar el tiempo transcurrido, junto con sys/resource.h y sys/time.h con getrusage para obtener el tiempo de usuario y sistema a través de rusage y timeval. Este enfoque garantiza una estimación precisa de la carga computacional del algoritmo durante su ejecución.

8. Speedup

El **speedup** es una métrica fundamental en el análisis de algoritmos paralelos y de alto desempeño. Representa la mejora en el tiempo de ejecución que se obtiene al utilizar múltiples procesadores, hilos o procesos en comparación con la ejecución secuencial del mismo problema.

Matemáticamente se define como:

$$\text{Speedup} = \frac{T_s(n)}{T_p(n,p)}$$

Donde:

- $T_s(n)$: tiempo de ejecución secuencial.

- $T_p(n,p)$: tiempo de ejecución en paralelo.

Y con la **eficiencia (E):**

$$\text{Efficiency} = \frac{\text{Speedup}}{p} = \frac{T_s(n)}{p\ T_p(n,p)}$$

donde p es el número de procesadores.

MARCO CONTEXTUAL

1. Características de la Máquina

Las pruebas de rendimiento fueron ejecutadas en un computador portátil con especificaciones de gama media, adecuadas para tareas de programación, simulaciones y procesamiento de datos.

- Procesador: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40 GHz
- Cantidad de núcleos: 4
- Cantidad de subprocesos/hilos: 8 (4 núcleos × 2 hilos por núcleo)
- Frecuencia básica del procesador: 2.40 GHz
- Frecuencia Turbo máxima: 4.20 GHz
- RAM: 8GB (2 módulos de 4GB “dual channel”) DDR4 @ 3200 MHz
- SSD: INTEL SSDPEKNW512G8 512GB - R: 1500 MB/s - W: 1000 MB/s
- Sistema operativo: Ubuntu 24.04.3 LTS

2. Desarrollo

Para llevar a cabo el desarrollo del proyecto se implementaron diferentes estrategias de multiplicación de matrices cuadradas con el fin de analizar y comparar su desempeño. Las tres variantes consideradas fueron: ejecución secuencial, ejecución paralela con hilos y ejecución paralela combinando procesos e hilos.

El primer paso consistió en la generación de matrices cuadradas con valores aleatorios. Para ello se hizo uso de funciones auxiliares implementadas en la librería matriz_utils.h, que permiten reservar memoria dinámicamente, llenar las matrices con números aleatorios y guardar los resultados en archivos de texto. Los tamaños considerados en la experimentación fueron 100, 200, 400, 800, 1600 y 3200.

2.1 Multiplicación Secuencial

La implementación secuencial se desarrolló en C utilizando estructuras básicas del lenguaje. Se reservaron matrices dinámicas para A, B y C, y la multiplicación se realizó mediante tres bucles anidados que recorren filas y columnas para acumular el producto escalar de cada posición. Para la medición del tiempo de ejecución se usaron las funciones de la librería time.h en conjunto con getrusage (a través de iniciar_temporizador y fin_medicion).

La salida de este programa incluye tanto las matrices multiplicadas como el tiempo total de ejecución, el cual es almacenado en un archivo de texto (resultado_sec.txt).

2.2 Multiplicación con Hilos

En la segunda implementación se introdujo paralelismo utilizando la librería pthread.h. Cada hilo se encarga de un subconjunto de filas de la matriz A, que multiplica con la matriz B para construir la matriz resultante C.

El programa recibe como parámetros el tamaño de la matriz y el número de hilos. El reparto de las filas se realiza de forma equitativa entre los hilos, ajustando los casos donde el número de filas no es divisible de manera exacta. Posteriormente, cada hilo ejecuta la función multiply_matrices, donde se efectúa la operación de multiplicación en su rango asignado.

Al igual que en la versión secuencial, se mide el tiempo de ejecución total desde la creación de los hilos hasta la finalización de todos con pthread_join. Los resultados son escritos en el archivo resultado_hilos.txt.

2.3 Multiplicación con Procesos e Hilos

La tercera implementación combina procesos (usando fork()) y hilos (usando pthread.h) para incrementar el grado de paralelismo. Para coordinar la escritura en la matriz

resultante se utiliza memoria compartida mediante shmget y shmat, lo que permite que todos los procesos accedan a la misma matriz C.

En este caso, se dividen las filas entre distintos procesos y, dentro de cada proceso, estas filas son a su vez subdivididas entre hilos. De esta forma, se logra una jerarquía de paralelismo donde múltiples procesos cooperan y dentro de cada proceso, múltiples hilos realizan los cálculos. Una vez finalizada la multiplicación, se recolectan los resultados y se almacenan en el archivo resultado_proc_hilos.txt.

Para medir el tiempo en los procesos se usa el parámetro RUSAGE_CHILDREN en la función iniciar_temporizador y fin_medicion, que entrega la suma del tiempo total ejecutado entre todos los procesos.

2.4 Automatización y Análisis de Resultados

Finalmente, para facilitar la ejecución repetitiva de las pruebas y la recopilación de resultados, se desarrolló un script en Bash denominado automatizacion.sh.

Este script realiza las siguientes tareas:

1. Compilación automática de los tres programas (secuencial, hilos y procesos+hilos).
2. Ejecución repetida de cada programa un número definido de iteraciones (en este caso 10).
3. Variación de parámetros, como el tamaño de la matriz, número de hilos y número de procesos.
4. Registro de resultados en un archivo resultados.csv, donde cada línea contiene el tamaño de la matriz, el tiempo en milisegundos y el tipo de implementación utilizada.

5. Generación de gráficos y tablas de comparación mediante scripts en Python (grafico.py y tablas.py), lo que permite visualizar y analizar métricas como tiempo de ejecución y speedup.

Con este esquema, fue posible comparar objetivamente el desempeño de las tres variantes de multiplicación de matrices en distintos escenarios, validando la importancia del paralelismo y observando cómo los tiempos de ejecución varían según el tamaño de la matriz y la cantidad de hilos o procesos empleados.

Además, con el fin de garantizar la transparencia y la posibilidad de replicar los experimentos, toda la implementación del informe se encuentra disponible en un repositorio de **GitHub**: https://github.com/bvelez-Timo-EngSys/HPC1. Allí se incluyen los programas en C, las librerías auxiliares, el script de automatización y los códigos en Python para la generación de gráficos y tablas. Este repositorio permite a cualquier interesado acceder al código fuente, ejecutarlo en su propio entorno y verificar los resultados obtenidos, fomentando así la reproducibilidad y la colaboración académica.

PRUEBAS

1. **Resultados de Ejecutar el Algoritmo en Forma Secuencial (Utilizando un Solo Core del Procesador):** Estos resultados corresponden a la ejecución base, donde no se aplican técnicas de paralelismo. Sirven como punto de comparación para evaluar la mejora lograda con hilos y procesos. Los tiempos se presentan en ms.

Tabla 1. *Resultados de la Ejecución Secuencial.*

Secuencial						
Tamaño matriz / Iteración	100	200	400	800	1600	3200
1	10	40	210	1740	19230	194320
2	10	20	200	1790	19510	190930
3	10	40	190	1730	18250	194530
4	10	40	210	1860	18810	192410
5	0	40	210	1840	19290	190100
6	10	20	190	1780	19440	191140
7	0	40	190	1750	18940	191320
8	0	40	200	1770	19480	191260
9	10	40	200	1780	19410	194280
10	0	20	210	1840	19240	197930
Promedio	6	34	201	1788	19160	192822

Nota. *Diseño propio de los Autores.*

2. **Resultados de Ejecutar el Algoritmo con Dos Hilos:** Al emplear dos hilos, se aprecia una disminución en los tiempos de ejecución frente a la versión secuencial, especialmente en tamaños de problema más grandes. Esto muestra un primer nivel de aprovechamiento del paralelismo.

Tabla 2. *Resultados de la Ejecución con 2 Hilos.*

2 Hilos						
Tamaño matriz / Iteración	100	200	400	800	1600	3200
1	0	20	140	910	8280	97070
2	0	20	120	960	8450	86160
3	0	20	120	900	8230	81990
4	0	10	80	710	7090	73440
5	0	10	80	690	7150	79920
6	0	10	100	940	7840	73460
7	0	10	80	700	7070	73840
8	0	10	80	760	7030	72960
9	0	10	80	690	8260	78630
10	0	10	80	700	7070	81040
Promedio	0	13	96	796	7647	79851
Speedup	0	2,615384615	2,09375	2,24623115	2,505557735	2,414772514

Nota. *Diseño propio de los Autores.*

3. **Resultados de Ejecutar el Algoritmo con Cuatro Hilos:** Con cuatro hilos se logra un mayor grado de paralelización, lo cual se refleja en una reducción más marcada de los tiempos respecto a la versión secuencial y la de dos hilos.

Tabla 3. *Resultados de la Ejecución con 4 Hilos.*

4 Hilos						
Tamaño matriz / Iteración	100	200	400	800	1600	3200
1	0	10	40	440	4210	44490
2	0	10	40	570	4390	46180
3	0	0	40	370	3770	40940
4	0	0	40	430	4400	41290
5	0	10	40	580	4400	45700
6	0	10	50	490	4210	45680

7	0	10	40	430	3890	41730
8	0	0	40	380	4110	44450
9	0	0	70	460	4670	41250
10	0	10	70	460	4080	51400
Promedio	0	6	47	461	4213	44311
Speedup	0	5,666666667	4,276595745	3,878524946	4,547828151	4,351560561

Nota. *Diseño propio de los Autores.*

4. **Resultados de Ejecutar el Algoritmo con Ocho Hilos:** La ejecución con ocho hilos permite analizar cómo el incremento del paralelismo sigue reduciendo los tiempos, aunque también se observa el efecto de la sobrecarga que genera la coordinación entre más hilos.

Tabla 4. *Resultados de la Ejecución con 8 Hilos.*

8 Hilos						
Tamaño matriz / Iteración	100	200	400	800	1600	3200
1	0	10	50	430	5040	47640
2	0	10	50	420	4960	48310
3	0	10	50	390	4890	47960
4	0	10	40	380	4950	47940
5	0	10	60	440	5030	48210
6	0	0	40	390	4970	48410
7	0	10	40	390	4910	51580
8	0	10	50	440	5120	48350
9	0	10	50	380	5060	51410
10	0	10	50	390	5170	50430
Promedio	0	9	48	405	5010	49024
SpeedUp	0	3,777777778	4,1875	4,41481481	3,824351297	3,933216384

Nota. *Diseño propio de los Autores.*

5. **Resultados de Ejecutar el Algoritmo con Múltiples Procesos (2, 4 y 8 Procesos – 1 Hilo cada Uno):** En este escenario, en lugar de dividir el trabajo en hilos dentro de un mismo proceso, se distribuye en procesos independientes.

Tabla 5. *Resultados de la Ejecución con 2 Procesos y 1 Hilo.*

2 Procesos - 1 Hilo						
Tamaño matriz / Iteración	100	200	400	800	1600	3200
1	0	20	100	1010	8040	73550
2	0	10	80	740	7530	74390
3	0	10	80	710	7340	72350
4	0	10	80	790	7170	75390
5	0	10	80	760	7520	74540
6	0	10	80	720	7090	72270
7	0	10	80	780	7440	72890
8	0	10	80	670	6580	74040
9	0	10	80	690	7000	69340
10	0	10	80	720	7370	68390
Promedio	0	11	82	759	7308	72715
Speedup	0	3,09090909	2,45121951	2,355731225	2,621784346	2,651749983

Nota. *Diseño propio de los Autores.*

Tabla 6. *Resultados de la Ejecución con 4 Procesos y 1 Hilo.*

4 Procesos - 1 Hilo						
Tamaño matriz / Iteración	100	200	400	800	1600	3200
1	0	10	40	360	3370	37650
2	0	10	40	360	3700	37570
3	0	10	40	350	3740	37520
4	0	10	40	360	3710	37490
5	0	10	40	360	3710	37580
6	0	10	40	360	3670	39710
7	0	10	40	360	3750	39000
8	0	10	40	360	3730	37680
9	0	10	40	380	3730	37510
10	0	10	40	360	3330	37710
Promedio	0	10	40	361	3644	37942
SpeedUp	0	3,4	5,025	4,952908587	5,257958288	5,082019925

Nota. *Diseño propio de los Autores.*

Tabla 7. *Resultados de la Ejecución con 8 Procesos y 1 Hilo.*

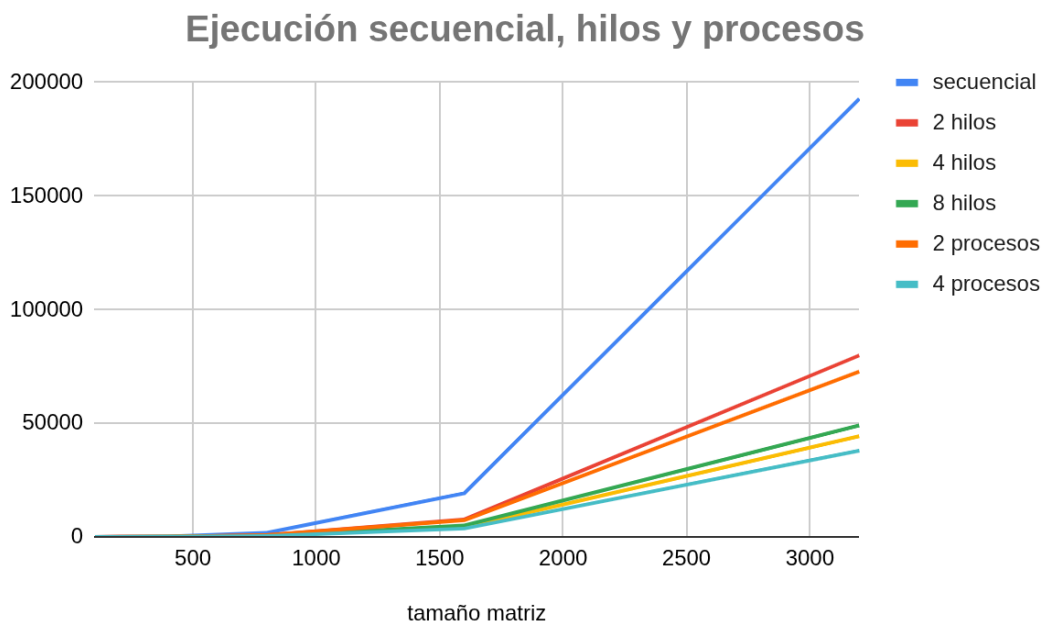
8 Procesos - 1 Hilo						
Tamaño matriz / Iteración	100	200	400	800	1600	3200
1	0	10	40	350	4650	46890
2	0	10	40	350	4620	47460
3	0	10	40	350	4960	47130
4	0	10	40	350	5080	46270
5	0	10	90	430	5200	52350
6	0	10	80	420	5330	50850
7	0	10	70	430	5490	51040
8	0	10	60	430	5300	50540
9	0	10	70	420	5370	51110
10	0	10	100	450	5380	49030
Promedio	0	10	63	398	5138	49267

Speedup	0	3,4	3,1904761	4,492462312	3,729077462	3,913816551

Nota. *Diseño propio de los Autores.*

Para poder analizar de mejor manera el rendimiento de todos los métodos, se realizó una gráfica de líneas comparando el promedio obtenido.

Gráfica 1. *Comparación de Tiempos de Ejecución.*



Nota. *Elaboración propia con python.*

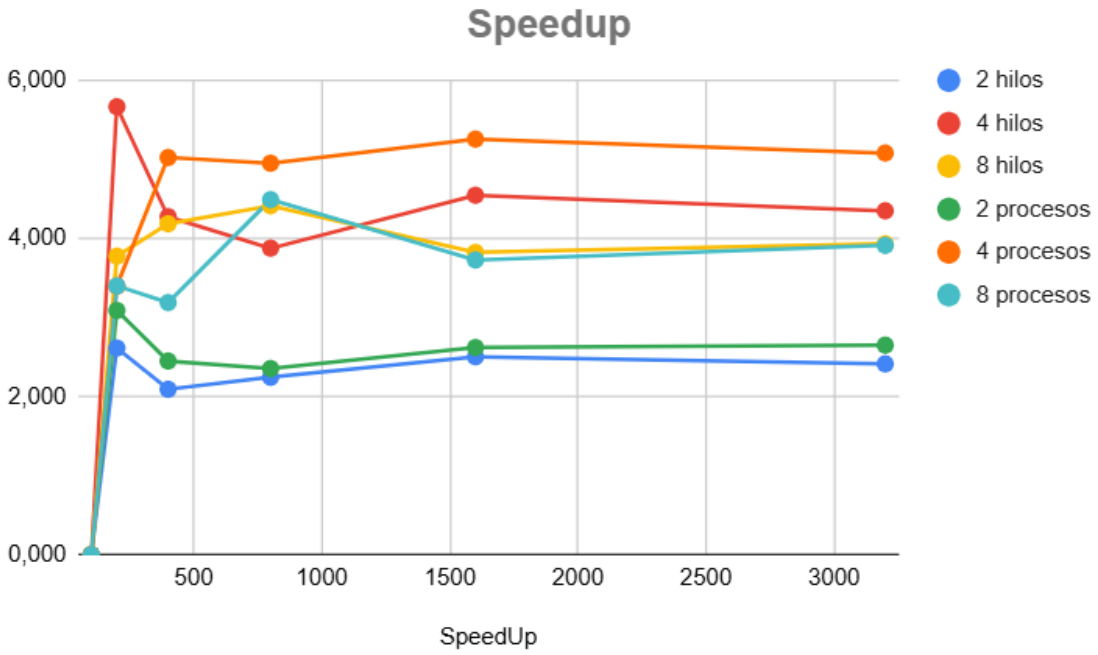
Tabla 8. *Resultados del Speedup.*

Relación Speedup						
Tamaño matriz	100	200	400	800	1600	3200
2 hilos	0	2,61538462	2,09375	2,24623115	2,50555774	2,41477251
4 hilos	0	5,66666667	4,27659575	3,87852495	4,54782815	4,35156056
8 hilos	0	3,77777778	4,1875	4,41481482	3,8243513	3,93321638
2 procesos – 1 hilo	0	3,09090909	2,45121951	2,35573123	2,62178435	2,65174998

4 procesos – 1 hilo	0	3,4	5,025	4,95290859	5,25795829	5,08201993
8 procesos – 1 hilo	0	3,4	3,1904761	4,49246231	3,72907746	3,91381655

Nota. *Diseño propio de los Autores.*

Gráfica 2. *Comparación del Speedup.*



Nota. *Diseño propio de los Autores.*

CONCLUSIONES

- La implementación secuencial permitió establecer una línea base de comparación, mostrando cómo el tiempo de ejecución crece exponencialmente con el tamaño de las matrices debido a la complejidad $O(N^3)$.
- El uso de hilos demostró una mejora significativa en los tiempos de ejecución frente a la versión secuencial, especialmente a medida que el tamaño de la matriz aumenta (+1600). La distribución equitativa de filas entre hilos evitó condiciones de carrera y permitió un aprovechamiento más eficiente de los núcleos del procesador.
- La estrategia con procesos e hilos evidenció un mayor grado de complejidad y de consumo de recursos, pero también permitió explotar el paralelismo jerárquico. El uso de memoria compartida garantizó la coherencia de los resultados entre procesos, aunque la comunicación interproceso implicó un overhead adicional. Esta variante fue más efectiva en tamaños de matriz grandes, donde el trabajo se reparte suficientemente para justificar el coste de la creación de procesos.
- La automatización mediante Bash y Python fue fundamental para garantizar la reproducibilidad de los experimentos, ya que permitió ejecutar múltiples iteraciones, recopilar resultados en formato estructurado (CSV) y generar gráficas y tablas comparativas. Esto facilitó el análisis objetivo del desempeño de cada enfoque.
- Analizando los gráficos de promedios, se observa que el comportamiento de los procesos y los hilos es muy similar en dimensiones pequeñas, pero a partir de cierto tamaño los procesos mantienen una ventaja, lo que los convierte en una mejor opción para problemas de mayor escala.
- Se decidió trabajar con un máximo de 8 hilos, en correspondencia con los 4 núcleos físicos del equipo utilizado. Consideramos que un mayor número de hilos no aportaría

mejoras significativas, ya que el hardware se vería limitado por la capacidad de procesamiento disponible. En ese escenario, el exceso de hilos podría incluso generar sobrecarga por cambio de contexto, reduciendo la eficiencia en lugar de incrementarla.

- Gracias a lo evidenciado en la gráfica del speedup, se observa que la mejor aceleración se obtiene al utilizar 4 procesos. Esto suponemos que puede explicarse ya que el equipo de prueba cuenta con 4 núcleos físicos, lo que permite que cada proceso se ejecute de manera independiente sobre un núcleo. De esta forma, se logra un aprovechamiento más eficiente del hardware, reduciendo la sobrecarga y maximizando el rendimiento.
- La construcción de tablas y gráficas representó un apoyo visual fundamental, ya que permitió interpretar de manera más clara los resultados obtenidos y comparar de forma objetiva el rendimiento de cada estrategia de ejecución.
- En términos generales, los resultados evidencian que el paralelismo es beneficioso cuando la carga de trabajo es suficientemente grande, mientras que para problemas pequeños la versión secuencial puede ser más eficiente por su bajo overhead. La correcta selección entre secuencial, hilos o procesos dependerá del tamaño de la matriz y de los recursos de hardware disponibles.

BIBLIOGRAFÍA

[1] (S/f). Purdue.edu. Recuperado el 10 de septiembre de 2025, de https://engineering.purdue.edu/~smidkiff/ece563/slides/speedupBasic.pdf

[2] *Too big efficiency - parallel computing*. (2014, June 28). Stack Overflow. https://stackoverflow.com/questions/24468566/too-big-efficiency-parallel-computing

[3] *Programación paralela*. (s. f.). https://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoria/index.html

[4] GeeksforGeeks. (2024b, mayo 20). *Multiplication of Matrix using threads*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/multiplication-of-matrix-using-threads/

[5] Parallel Programming: Speedups, & Law, A. (s/f). *Definition of Speedup*. Oregonstate.edu. Recuperado el 10 septiembre de 2025, de https://web.engr.oregonstate.edu/~mjb/cs575/Handouts/speedups.and.amdahls.law.6pp.pdf

[6] (S/f). Baeldung.com. Recuperado el 10 de septiembre de 2025, de https://www.baeldung.com/cs/matrix-multiplication-algorithms

[7] Schryen, G. (2024). Speedup and efficiency of computational parallelization: A unifying approach and asymptotic analysis. *Journal of Parallel and Distributed Computing*, *187*, 104835. https://www.researchgate.net/publication/274960405_Parallel_programming_definitions_mechanisms_and_trouble