Evaluación del Rendimiento en la Multiplicación de Matrices utilizando OpenMP: Un Estudio Experimental

Juan David Avendaño y Sebastián Franco

***Abstract*—La multiplicación de matrices de gran tamaño es computacionalmente intensiva y puede resultar en largos tiempos de ejecución. Para mejorar la eficiencia, proponemos optimizar esta operación mediante la paralelización con OpenMP. OpenMP permite la ejecución paralela en una única máquina utilizando múltiples hilos. Esta técnica busca maximizar la utilización de recursos disponibles. Los resultados preliminares indican una mejora significativa en el rendimiento y la eficiencia comparado con implementaciones secuenciales. Los detalles y métricas específicas se discutirán en las siguientes secciones.**

***Index Terms*—Multiplicación de matrices, computación de alto desempeño, paralelización, OpenMP, rendimiento, computación distribuida, experimentación, tiempo de ejecución.**

I. Introducción

La multiplicación de matrices es una operación fundamental en numerosos campos científicos y de ingeniería, destacándose en la física, la estadística, la informática, y de manera particular en la Inteligencia Artificial. Dada su importancia en estos ámbitos, es esencial que esta operación se realice con la mayor eficiencia posible, un desafío notable cuando se abordan matrices de gran tamaño. La naturaleza computacionalmente intensiva de la multiplicación de matrices, con su complejidad generalmente cúbica, puede conducir a tiempos de ejecución prolongados, lo cual es una preocupación central en investigaciones y aplicaciones prácticas.

En este contexto, la Computación de Alto Desempeño (HPC) surge como una solución crucial, ofreciendo herramientas y métodos para maximizar la utilización de los recursos computacionales disponibles. En la era actual, caracterizada por avances significativos en inteligencia artificial, análisis de grandes volúmenes de datos y simulaciones complejas, la demanda de soluciones HPC ha aumentado exponencialmente. Esta tendencia se observa en una amplia gama de aplicaciones, abarcando desde la investigación científica hasta desarrollos en la industria, donde la rapidez y la eficiencia en el procesamiento de datos son vitales para el progreso y la innovación.

En el marco de las estrategias de HPC, la paralelización se destaca como un enfoque esencial. El presente trabajo se enfoca en la utilización de OpenMP, una API diseñada para la programación paralela en una sola máquina, utilizando múltiples hilos para la división de tareas. El objetivo es caracterizar y comparar el rendimiento de la multiplicación de matrices bajo diferentes configuraciones de paralelización con OpenMP, buscando identificar las ventajas y limitaciones de este método. Se prestará especial atención a las variaciones en la eficiencia y el rendimiento según distintas representaciones de matrices y configuraciones de paralelización.

Maestría de Inteligencia Artificial, Pontificia Universidad Javeriana
Documento entregado Noviembre 13, 2023.

II. Estado del arte

La multiplicación de matrices es una tarea computacionalmente demandante, y ha sido importante en extensas investigaciones que buscan mejorar su eficiencia a través de técnicas de paralelización. Particularmente, el uso de OpenMP, una herramienta para la memoria compartida en sistemas multiproceso, ha sido una estrategia clave en la mejora de esta operación en el campo de la computación de alto rendimiento.

Un estudio destacado en este campo es el realizado por Ryan LaRose (2018), que investiga el uso de OpenMP para la simulación de circuitos cuánticos en el superordenador MSU Laconia Top500, demuestra cómo la paralelización efectiva mediante OpenMP puede llevar a simulaciones más eficientes y escalables, resaltando la utilidad de esta técnica en operaciones complejas de cálculo matricial.[1]

De manera relacionada, el artículo de Tigran M. Galstyan examina el desempeño de varios algoritmos de multiplicación de matrices utilizando OpenMP. Este estudio destaca la popularidad creciente de OpenMP en campos matemáticos y computacionales, resaltando su eficacia en mejorar la eficiencia de la paralelización de la multiplicación de matrices. [2]

Estos estudios, en conjunto, indican el potencial de OpenMP para mejorar el rendimiento en la multiplicación de matrices, un aspecto esencial para muchas aplicaciones en ciencias e ingeniería. Este conjunto de investigaciones sienta las bases para el presente estudio, que se enfoca en la multiplicación de matrices NxN utilizando OpenMP, concentrándose en la caracterización y experimentación con diferentes configuraciones de paralelización.

III. Metodología

A. Descripción del Problema

La multiplicación de matrices es una operación esencial en diversos campos como la ciencia de datos, física e ingeniería. Dada la complejidad cuadrática y cúbica en el tiempo de ejecución de los algoritmos de multiplicación de matrices convencionales, esta operación se vuelve intensiva y costosa en tiempo para matrices de gran tamaño. Por ende, optimizar este proceso es una área de interés significativo.

B. Enfoque Propuesto

Para abordar el problema de la multiplicación de matrices de gran tamaño, este estudio se centra en la paralelización utilizando OpenMP, una API para programación multiproceso en C y C++. Se exploran dos enfoques de paralelización:

- Experimento 1: OpenMP con Matrices 2D
  En este experimento, se paraleliza la multiplicación de matrices utilizando OpenMP, donde cada hilo se encarga de una porción de las filas de la matriz resultante. Este enfoque busca mejorar el tiempo de ejecución al permitir cálculos simultáneos.
- Experimento 2: OpenMP con Matrices como Vector 1D
  En este enfoque, las matrices se representan como un vector unidimensional para mejorar la localidad de los datos y minimizar la sobrecarga de gestión de memoria. Se mantiene la paralelización con OpenMP para evaluar la eficiencia en comparación con la representación bidimensional.

C. Metodología Experimental

Se diseñaron experimentos utilizando matrices de tamaños 100x100, 200x200, 400x400, 600x600 y 800x800. Los experimentos se ejecutaron utilizando diferentes cantidades de hilos (1, 2, 4, 8, 10, 14, 16, 20), para evaluar cómo escala el rendimiento con el tamaño de la matriz y los recursos computacionales disponibles. Los experimentos se realizaron en los clústeres proporcionados por la universidad.

- Parámetros de los Experimentos: Se realizaron pruebas para cada combinación de tamaño de matriz y número de hilos. Estos parámetros permiten evaluar la escalabilidad del rendimiento con respecto al tamaño del problema y los recursos computacionales.
- Métricas de Rendimiento: Las métricas clave incluyen el tiempo de ejecución y la eficiencia de paralelización (Speedup), calculada como la relación entre el tiempo de ejecución con un solo hilo y el tiempo de ejecución con múltiples hilos, ajustado por el número de hilos.
- Procedimiento Experimental: La multiplicación de matrices se implementó en C utilizando OpenMP para ambos enfoques de paralelización (2D y Vector 1D). Se registraron las métricas de rendimiento para cada configuración de tamaño de matriz y número de hilos.
- Análisis de Datos: Los datos obtenidos de los experimentos fueron analizados utilizando Python para identificar tendencias y patrones en el rendimiento y la escalabilidad de los diferentes enfoques.

D. Descripción Detallada de los Algoritmos

En este estudio, se implementaron dos algoritmos de multiplicación de matrices utilizando la programación paralela con OpenMP. Los algoritmos se diferencian en su enfoque de acceso a los datos de las matrices, uno centrado en filas y el otro en columnas. Ambos están diseñados para trabajar con matrices de tamaño NxN y están codificados en C.

1) Multiplicación de Matrices por Columnas: Desarrollado por J. Avendaño y S. Franco, este algoritmo optimiza el acceso a los datos al recorrer las matrices columna por columna. Emplea una estructura de datos lineal `MEM_CHUNK` para almacenar las matrices A, B y C. La paralelización se logra mediante la directiva `#pragma omp parallel`, asignando a cada hilo un conjunto específico de filas de la matriz resultante para realizar múltiples operaciones de multiplicación de manera simultánea.

2) Multiplicación de Matrices por Filas: Este enfoque, desarrollado por J. Avendaño y S. Franco, se enfoca en el acceso por filas para mejorar la localidad de los datos y reducir la sobrecarga de gestión de memoria. Utiliza la misma estructura de almacenamiento y esquema de paralelización que el algoritmo anterior. La diferencia clave radica en cómo se ejecuta la multiplicación: para cada fila de la matriz A, se recorre cada columna de la matriz B, acumulando el resultado en la matriz C.

E. Estructura del Proyecto y Herramientas de Medición

Los algoritmos se compilan utilizando un `Makefile` que especifica opciones de optimización y paralelización para el compilador GCC. El rendimiento se mide a través de `Otime.c`, ubicado en la carpeta *sample*. Este archivo mide el tiempo de ejecución de cada hilo usando `gettimeofday`. Los resultados se almacenan en las carpetas *Soluciones* que se encuentran en: *EXP-RendimientoMfilas* y *EXP-RendimientoMcolumnas*, según el tipo de acceso a las matrices.

F. Análisis Estadístico y de Datos en Python

El análisis de los datos obtenidos de los experimentos de OpenMP se llevó a cabo utilizando Python, un lenguaje de programación versátil y potente, ideal para el procesamiento y análisis de datos. A continuación, se detallan los aspectos técnicos clave y las etapas del análisis implementado en Python:

1) Entorno de Desarrollo en Python: El análisis se realizó en un entorno de Python, aprovechando su amplia gama de bibliotecas para el manejo de datos y la visualización. Este entorno proporcionó las herramientas necesarias para un procesamiento eficaz y un análisis detallado de los resultados experimentales.

2) Extracción y Preparación de Datos: Utilizando Python, se extrajeron los datos de los archivos de origen y se prepararon para el análisis. Esta etapa incluyó la limpieza de datos, su transformación y estructuración en DataFrames de Pandas, lo cual facilitó su manipulación y análisis posterior.

3) Cálculo del Speedup en Python: Se implementó una función en Python para calcular el Speedup de cada experimento. Este cálculo fue esencial para evaluar la eficiencia del procesamiento paralelo, comparando los tiempos de ejecución en configuraciones de un solo core y de múltiples cores.

4) Análisis de Punto Óptimo: Se utilizó Python para analizar los cambios en el rendimiento y determinar el número óptimo de cores. Este análisis involucró cálculos estadísticos y lógica de programación para identificar el punto en el que el

incremento en el número de cores ya no resultaba en mejoras significativas de rendimiento.

5) Visualización Gráfica con Matplotlib: Para la visualización de los resultados, se empleó la biblioteca Matplotlib de Python. Se generaron gráficos claros y detallados que mostraron cómo el tiempo de ejecución y el Speedup variaron con diferentes números de cores y tamaños de tarea, proporcionando una comprensión visual inmediata de los datos.

6) Análisis de Variabilidad del Rendimiento: Se calcularon las diferencias de tiempo entre el core más rápido y el más lento en cada experimento para evaluar la variabilidad del rendimiento. Este análisis proporcionó una perspectiva sobre la uniformidad en la distribución del trabajo entre los cores.

La metodología y las herramientas empleadas en Python permitieron un análisis exhaustivo y riguroso de los experimentos de OpenMP. La flexibilidad y potencia de Python, combinadas con su capacidad para manejar grandes volúmenes de datos y realizar cálculos complejos, lo hicieron ideal para esta tarea analítica.

IV. Resultados

A. Introducción a los Resultados

En esta sección, se presentan y analizan los resultados obtenidos de dos series de experimentos de OpenMP: uno con matrices cuadradas y otro con matrices convertidas a vectores. Se enfoca en comparar los tiempos de ejecución, medir el Speedup y determinar el punto óptimo de uso de cores en relación con el tamaño de la tarea.

B. Transformación y Preparación de Datos

Los datos de ambos experimentos se sometieron a un proceso de transformación y preparación, incluyendo la limpieza y estructuración en DataFrames de Pandas. Esto nos permitió realizar un seguimiento detallado de cada core y cada iteración del experimento, clasificando los datos por tamaño de tarea y número de cores.

Para analizar los resultados de manera efectiva, se creó un DataFrame que consolidó todos los experimentos, conteniendo las siguientes columnas:

- **No. Core:** Identificador del core individual dentro de un experimento específico.
- **Valor:** Tiempo de ejecución en microsegundos para cada core.
- **Archivo:** Nombre del archivo de datos del experimento correspondiente.
- **Size:** Tamaño de la matriz cuadrada utilizada en el experimento.
- **Core:** Número total de cores empleados en el experimento.

La función *agregar_numero_iteracion* añadió una columna *Iteración* al DataFrame, agrupando los resultados de acuerdo con la cantidad de cores usados. Esta columna fue esencial para diferenciar entre las distintas fases de cada experimento y simplificar el análisis de rendimiento.

Dado que se observaron variaciones en los tiempos de ejecución entre los diferentes cores en cada ejecución, se seleccionó el tiempo máximo de ejecución de todos los cores como un valor representativo de cada ejecución para simplificar el análisis. Además, se calculó la diferencia de tiempo entre el core más rápido y el más lento en el mismo experimento, proporcionando una perspectiva sobre la variabilidad del rendimiento dentro de cada ejecución.

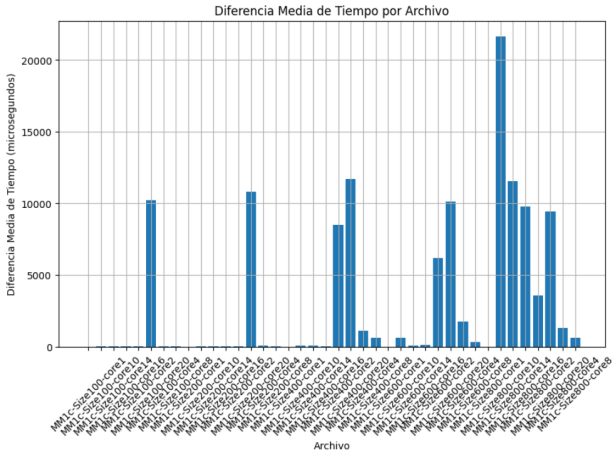

Fig. 1. Diferencia de tiempo entre el core más rápido y el más lento en matrices cuadradas.

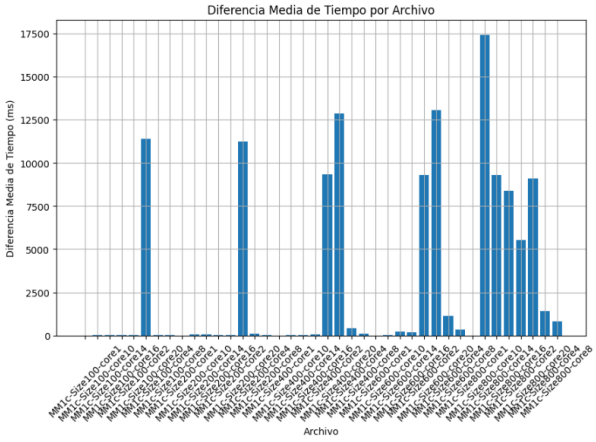

Fig. 2. Diferencia de tiempo entre el core más rápido y el más lento en matrices convertidas a vectores.

Se puede observar, que aunque hay algunos outliers, en general los datos tienden a ser del orden de los microsegundos de diferencia, por ello se concluye que no hay gran diferencia entre tomar el máximo o tomar el promedio, por esta razón se decide tomar el valor máximo como el representativo del experimento, de esta manera será el momento exacto en que todo el experimento haya finalizado para dicho proceso.

Posteriormente, se evaluó la consistencia del rendimiento con respecto al número de cores, esto con el propósito de ver en tiempo de ejecución cual era la mejora para cada tamaño de matriz de agregar un nuevo core. Se obtuvo:

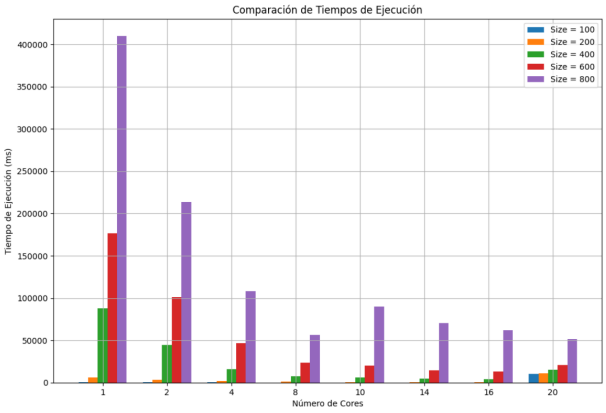

Fig. 3. Tiempo de ejecución en función del número de cores en matrices cuadradas.

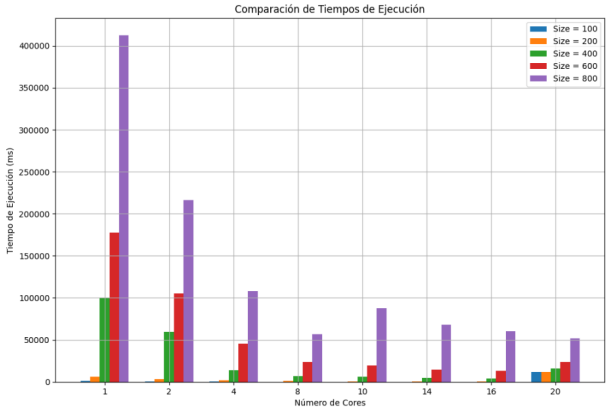

Fig. 4. Tiempo de ejecución en función del número de cores en matrices convertidas a vectores..

Finalmente, se calculó el Speedup para cada configuración y se añadió como una columna adicional al DataFrame, completando así el conjunto de datos necesario para el análisis detallado de los resultados. El Speedup se calculó comparando los tiempos de ejecución con múltiples cores frente al tiempo con un solo core.

C. Análisis de Rendimiento

En este apartado se presenta un análisis detallado del rendimiento obtenido en los experimentos. Se observó una mejora en el rendimiento con el aumento de cores, pero hasta un punto donde las mejoras se estabilizaron o disminuyeron. Este patrón es una manifestación de la ley de rendimientos decrecientes en computación paralela.

Para ambos experimentos se encontró el tiempo de ejecución versus el número de cores, se encontró que dependiendo del tamaño de la matriz el punto óptimo para la cantidad de cores difería:

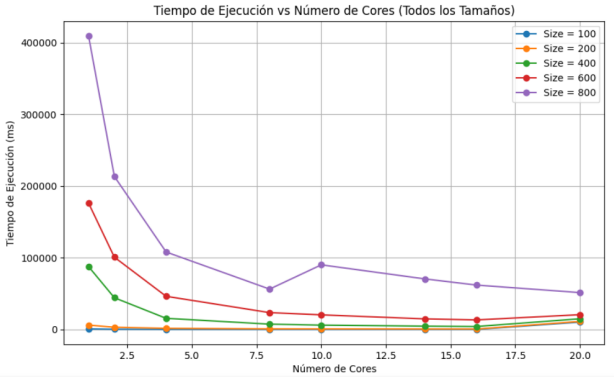

Fig. 5. Tiempo de ejecución en función del número de cores en matrices cuadradas.

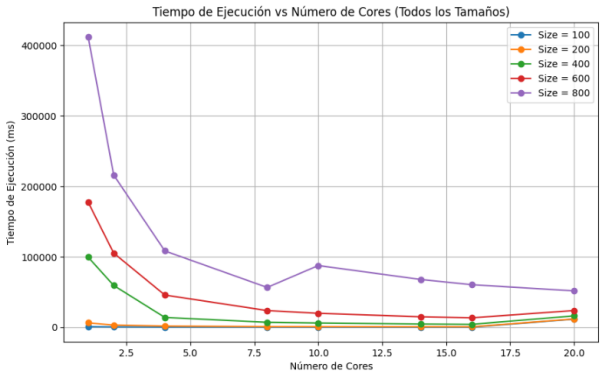

Fig. 6. Tiempo de ejecución en función del número de cores con matrices convertidas a vectores.

Posteriormente, se encontró la diferencia entre los speedups, con esta gráfica fue mucho más simple encontrar el punto de quiebre de forma visual, para saber cuándo era preferible dejar de aumentar la cantidad de cores.

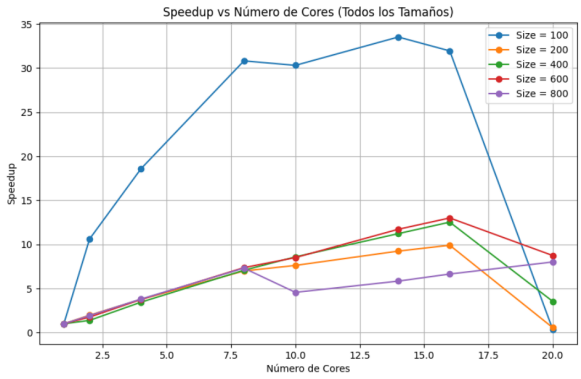

Fig. 7. Variación del Speedup con diferentes números de cores y tamaños de tarea en matrices cuadradas.

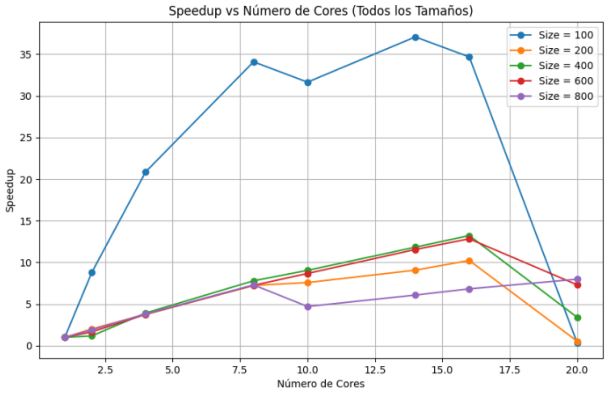

Fig. 8. Variación del Speedup en el experimento con matrices convertidas a vectores.

1) Encontrar el Punto Óptimo: Un aspecto crucial del análisis fue determinar el punto óptimo entre mejorar la métrica de rendimiento y adicionar más cores. Este punto óptimo se define como el momento en el que el beneficio adicional de rendimiento comienza a disminuir significativamente con la adición de más cores. En términos prácticos, se identifica el número de cores en el que el aumento de rendimiento, ya sea en términos de Speedup o reducción en tiempo de ejecución, comienza a disminuir. Agregar más cores más allá de este punto proporciona mejoras marginales, lo que puede llevar a un uso ineficiente de los recursos computacionales. Este análisis ayuda a equilibrar eficiencia y rendimiento, optimizando el uso de recursos sin incurrir en costos adicionales innecesarios o enfrentar rendimientos decrecientes. A continuación, se presentan las gráficas de dichos puntos óptimos para el tamaño de la matriz:

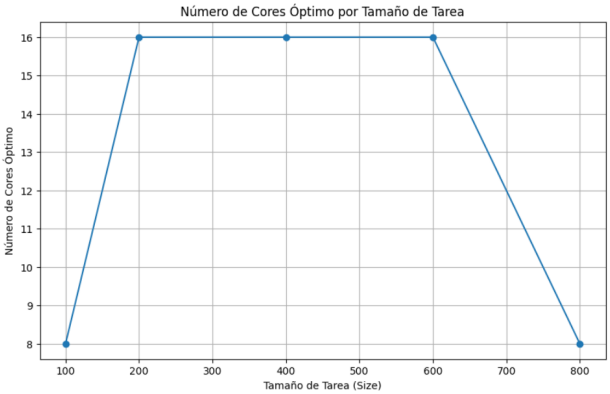

Fig. 9. Número óptimo de cores para cada tamaño de tarea en matrices cuadradas.

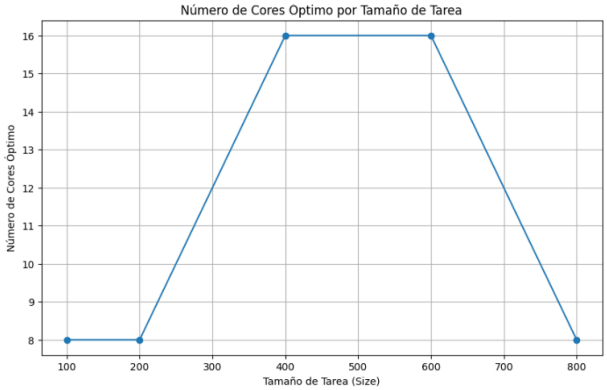

Fig. 10. Número óptimo de cores en el experimento de matrices convertidas a vectores.

Finalmente, se hace una comparación de ambos experimentos para ambas métricas:

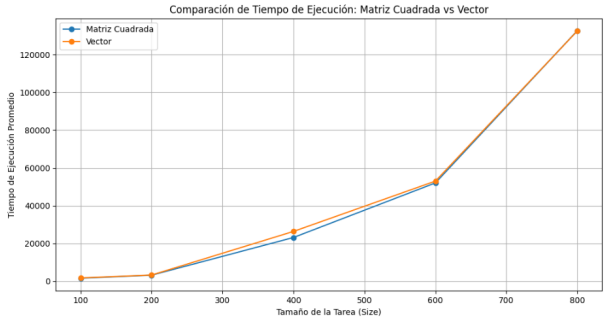

Fig. 11. Comparación de tiempos de ejecución.

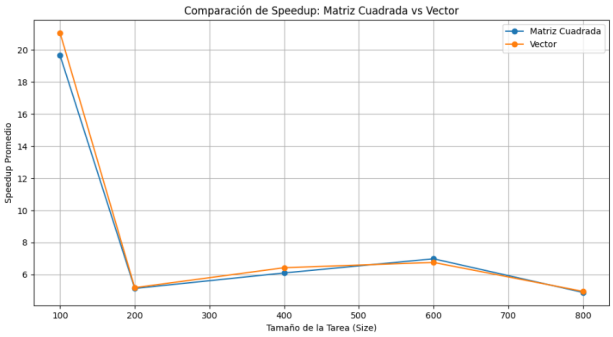

Fig. 12. Comparación de speedups.

D. Discusión de Resultados

En ambos experimentos, observamos un patrón común: el rendimiento mejora con un mayor número de cores, pero solo hasta cierto punto, en línea con la teoría de la ley de rendimientos decrecientes en la computación paralela. Es importante destacar que no se observó una mejora significativa entre un experimento con respecto al otro, dieron datos muy similares.

E. Conclusiones de los Resultados

Los resultados sugieren que, si bien el paralelismo mejora el rendimiento, existe un límite en la eficiencia alcanzable.

La selección cuidadosa del número de cores es crucial para optimizar el rendimiento en tareas de computación paralela.

F. Recomendaciones y Futuras Investigaciones

Recomendamos una evaluación detallada del número de cores en función del tamaño y la naturaleza de la tarea. Futuros estudios podrían enfocarse en explorar el impacto de diferentes algoritmos de paralelización y configuraciones de hardware en el rendimiento, al igual que la inclusión de MPI en los experimentos.

V. Discusión

Esta investigación ha demostrado que el uso de OpenMP en la computación paralela de matrices ofrece beneficios significativos en términos de rendimiento, pero también resalta la importancia de una selección cuidadosa del número de cores para optimizar dicho rendimiento. A lo largo del estudio, se observó que mientras el aumento del número de cores mejora generalmente el rendimiento, existe un punto óptimo más allá del cual las mejoras son marginales.

Estos resultados son consistentes con la teoría de rendimientos decrecientes en computación paralela y resaltan la complejidad inherente en la paralelización eficiente de tareas computacionales. La similitud en los resultados entre el uso de matrices cuadradas y matrices convertidas a vectores sugiere que la eficiencia de la paralelización puede depender más de factores como el tamaño de la tarea y la arquitectura del sistema que de la estructura de datos.

Además, el estudio resalta la importancia de considerar la variabilidad en el rendimiento entre diferentes cores, lo que puede tener implicaciones significativas en el diseño y la optimización de sistemas de computación paralela. La identificación del punto óptimo de cores es crucial no solo para maximizar el rendimiento, sino también para evitar el uso excesivo de recursos computacionales, lo cual es especialmente relevante en entornos donde los recursos son limitados o costosos.

VI. Conclusiones

El presente estudio proporciona una visión detallada de cómo el paralelismo, implementado a través de OpenMP, afecta el rendimiento en tareas de multiplicación de matrices. Se concluye que el paralelismo mejora significativamente el rendimiento, pero esta mejora sigue la ley de rendimientos decrecientes. Por tanto, es fundamental identificar el número óptimo de cores para cada tarea específica para aprovechar al máximo los recursos computacionales disponibles.

Los experimentos mostraron que no hay una diferencia significativa en el rendimiento al comparar la paralelización de matrices cuadradas con matrices convertidas a vectores, lo que indica que la elección del enfoque puede basarse en consideraciones específicas del problema más que en diferencias inherentes de rendimiento entre estos dos métodos.

En resumen, este estudio subraya la importancia de una comprensión detallada de la computación paralela y la necesidad de equilibrar el número de cores y el tamaño de la tarea para lograr una eficiencia óptima. Estos hallazgos pueden guiar decisiones futuras en el diseño y la implementación de sistemas computacionales paralelos, especialmente en entornos donde la eficiencia y el rendimiento son de suma importancia.

References

[1] R. LaRose, *Distributed Memory Techniques for Classical Simulation of Quantum Circuits,* Department of Computational Mathematics, Science, and Engineering, Michigan State University, Jun. 25, 2018.

[2] T. M. Galstyan, *Performance Analysis of Matrix Multiplication Algorithms Using MPI and OpenMP,* Institute for Informatics and Automation Problems of NAS RA, 2018.