HPC

Caso de Estudio I - OpenMP y benchmark

Multiplicación de Matrices

Héctor Fabio Vanegas
Brayan Cataño Giraldo
Juan Miguel Aguirre

Presentado a:

Ramiro Andres Barrios

Universidad Tecnológica De Pereira
2025

Abstract

El presente documento analiza el rendimiento de la multiplicación de matrices utilizando OpenMP, comparando resultados del benchmark sysbench en 2 máquinas diferentes. Cada máquina va a correr multiplicaciones de matrices con diferentes parámetros; finalmente, se hará un análisis del resultado de sysbench vs el resultado de la multiplicación de matrices por cada máquina.

Tabla de contenido

1 Introducción **3**

2 Marco conceptual **3**

2.1 High Performance Computing 3

2.2 Multiplicación matricial 3

2.3 Complejidad Computacional 4

2.4 Programación paralela 5

2.5 Hilos 5

2.6 Speedup 5

3 Marco contextual **5**

3.1 Especificaciones de los equipos de prueba 7

4 Desarrollo **7**

4.0.1 Sysbench máquina 1 7

4.0.2 Sysbench máquina 2 9

4.1 Pruebas multiplicación de matrices 11

4.1.1 Tiempo vs Tamaño de matrices 11

4.1.2 Speedup vs Número de hilos 12

4.1.3 Máquina 1 vs Máquina 2 14

5 Conclusiones **14**

1. INTRODUCCIÓN

La multiplicación de matrices es una operación fundamental en muchas aplicaciones científicas e ingenieriles, donde el rendimiento computacional es crítico. En el contexto de High Performance Computing (HPC), el hardware con el que se hacen estas operaicones es crucial para tener un desempeño, involcrando CPU y memoria principalmente (aunque con un papel más importante de la última).

Este reporte se centra en comparar 2 máquinas con diferentes especificaciones en pruebas de multiplicación de matrices utilizando la librería OpenMP en C:

Mediante una serie de benchmarks y pruebas en un entorno controlado, se evaluó el impacto de correr el algoritmo en 2 computadoras con diferentes especificaciones y arquitecturas.

2. MARCO CONCEPTUAL

2.1. High Performance Computing

Cuando hablamos de HPC por sus siglas en inglés, o Computación de alto rendimiento en español, "hacemos referencia a un campo de la computación actual que da solución a problemas tecnológicos muy complejos y que involucran un gran volumen de cálculos o de coste computacional." (López, 2017) Para nuestro caso, al aumentar la dimensión de las matrices a multiplicar, esto supone un alto costo computacional, el cual, según el análisis, se ve disminuido gracias al uso de herramientas que permiten, precisamente, disminuir el tiempo de ejecución.

2.2. Multiplicación matricial

Para implementar el algoritmo en el lenguaje C es necesario comprender previamente algunos conceptos, entre ellos, como es el proceso de multiplicar matrices de forma manual, el cual podemos aprender fácilmente gracias a vídeos explicativos y a diversos sitios web, tal como Producto de matrices.

Otros sitios en internet nos dan una idea más general de la programación de la multi-

plicación matricial, como por ejemplo el que se expone a continuación, que ha sido extraído del blog Análisis y diseño de algoritmos.

Dadas dos matrices A y B, tales que el número de columnas de la matriz A es igual al número de filas de la matriz B, es decir:

$$A := (a_{ij})_{m \times n}$$
$$B := (b_{ij})_{n \times p}$$

La multiplicación de A por B genera una nueva matriz resultado de la forma:

$$C = AB := (c_{ij})_{m \times p}$$

donde cada elemento c_{ij} está definido por:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Esta definición permite acercarse más a una idea del código, ya que de entrada plantea las tres variables a iterar para cada una de las celdas de la matriz resultado.

2.3. Complejidad Computacional

"La Teoría de la Complejidad estudia la eficiencia de los algoritmos en función de los recursos que utiliza en un sistema computacional (usualmente espacio y tiempo)" (Vásquez, 2004). Para este estudio, la complejidad computacional permite observar que en la multiplicación de matrices se gastan bastantes recursos de procesamiento en cuanto a tiempo, es decir, para matrices de dimensiones muy grandes, el procesador estará ocupado bastante tiempo resolviendo la multiplicación, lo que motiva aún más a programar de manera paralela el algoritmo, aprovechando al máximo los recursos del procesador.

2.4. Programación paralela

"La computación paralela es el uso de múltiples recursos computacionales para resolver un problema. Se distingue de la computación secuencial en que varias operaciones pueden ocurrir simultáneamente." (Ferestrepoca., 2019) En nuestro caso de estudio se ejecutará el código de multiplicación de matrices de forma secuencial y de forma paralela, esto con el fin de comparar los resultados obtenidos por cada ejecución y de esta manera poder analizar la eficiencia de la programación paralela.

2.5. Hilos

"La diferencia fundamental entre hilos y procesos es que un proceso dispone de un espacio de memoria separado, mientras que un hilo no. De esta forma los hilos pueden trabajar directamente con las variables creadas por el proceso padre, mientras que los procesos hijos no pueden hacerlo." (Antunez, 2011) Esta información nos facilita la comprensión del principio aplicado para esta implementación.

2.6. Speedup

El speedup representa la ganancia que se obtiene en la versión paralela de un programa con respecto a su versión secuencial. Para este estudio se ha tomado el speedup como:

$$S = \frac{T^*(n)}{T_p(n)}$$

donde $T^*(n)$ hace referencia al tiempo de ejecución secuencial y $T_p(n)$ hace referencia al tiempo de ejecución paralelo.

3. MARCO CONTEXTUAL

Los benchmarks preliminares para comparar las 2 máquinas fueron los siguientes, todas con una configuración de –time=60 en sysbench:

- Número de threads vs Eventos por segundo.

Para la máquina 1:

  - 1 hasta 6 threads.

Para la máquina 2:

  - 1 hasta 32 threads.

- Número de threads vs Transferencia (MiB/sec).
  Para la máquina 1:

  - 1 hasta 6 threads, memoria total=6G y tamaño de bloque=1M.

  Para la máquina 2:

  - 1 hasta 32 threads, memoria total=6G y tamaño de bloque=1M.
  - 1 hasta 32 threads, memoria total=124G y tamaño de bloque=1M.

Para evaluar el rendimiento del algoritmo, se realizaron pruebas con OpenMP, librería para utilizar hilos de forma fácil y eficiente. Se realizaron pruebas en las 2 computadoras con las mismas configuraciones: compilando utilizando la flag -O3 y simulando recibir una matriz transpuesta para optimizar el acceso a la caché. En la computadora 1 (con menos recursos), se realizaron pruebas con tamaños: **20x20**, **200x200**, **400x400**, **800x800**, **1600x1600**, **3200x3200** y **6400x6400**; en la computadora 2 (con más recursos) se realizaron pruebas con tamaños: **20x20**, **200x200**, **400x400**, **800x800**, **1600x1600**, **3200x3200**, **6400x6400**, **12800x12800** y **25600x25600**.

Para la máquina 1 se realizó la siguiente prueba:

- Para cada tamaño de matrices, se hizo una ejecución con 1, 2, 3, 4, 5 y 6 threads, en total se hicieron 42 pruebas (6 hilos * 7 tamaños de matrices)

Para la máquina 2 se realizó la siguiente prueba:

- Para cada tamaño de matrices, se hizo una ejecución con 1, 2, 3,..., 32 threads, en total se hicieron 288 pruebas (32 hilos * 9 tamaños de matrices)

3.1. Especificaciones de los equipos de prueba

Las pruebas fueron ejecutadas en dos equipos con las siguientes características:

Máquina 1:

- **Procesador:** AMD Ryzen 5 4500U (6 núcleos)
- **Memoria RAM:** 7,1Gi
- **Sistema operativo:** Ubuntu 22.04.5 LTS
- **Compilador utilizado:** gcc

Máquina 2:

- **Procesador:** 13th Gen Intel(R) Core(TM) i9-13900KF (24 núcleos - 32 threads)
- **Memoria RAM:** 125Gi
- **Sistema operativo:** Ubuntu 24.04.2 LTS
- **Compilador utilizado:** gcc

4. DESARROLLO

En esta sección se describe el proceso de implementación de los benchmark de sysbench para las 2 máquinas, tanto las pruebas realizadas en el caso de estudio.

- **Implementación con hilos (multithreading):** Se distribuye la carga de trabajo entre múltiples hilos para aprovechar el paralelismo a nivel de CPU utilizando la librería OpenMP.

4.0.1 Sysbench máquina 1

Sysbench CPU:
Para la máquina 1, se calcularon los eventos por segundo vs número de threads (máximo 6).

Llegando a 12000 eventos por segundo utilizando 6 hilos, comenzando en 2000 eventos con solo 1 hilo.

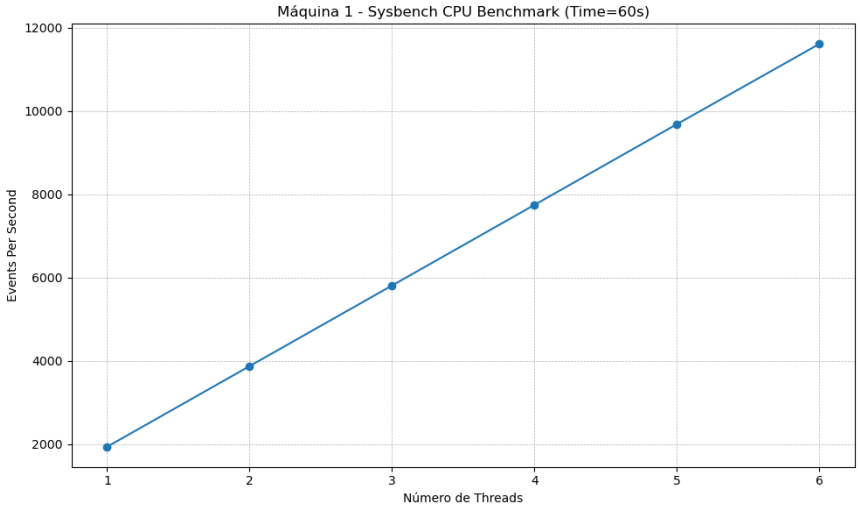

Figure 1: Hilos vs Eventos por Segundo (máquina 1).

Sysbench memory:

Para la máquina 1, se calculó la transferencia (MiB/sec) vs número de threads (máximo 6), con 6G de memoria total y 1M de bloque de memoria. En esta máquina podemos ver que es rentable aumentar los hilos para mejorar la transferencia, comenzando en 22500 MiB/sec con un hilo y finalizando con casi 40000 MiB/sec con 6 hilos.

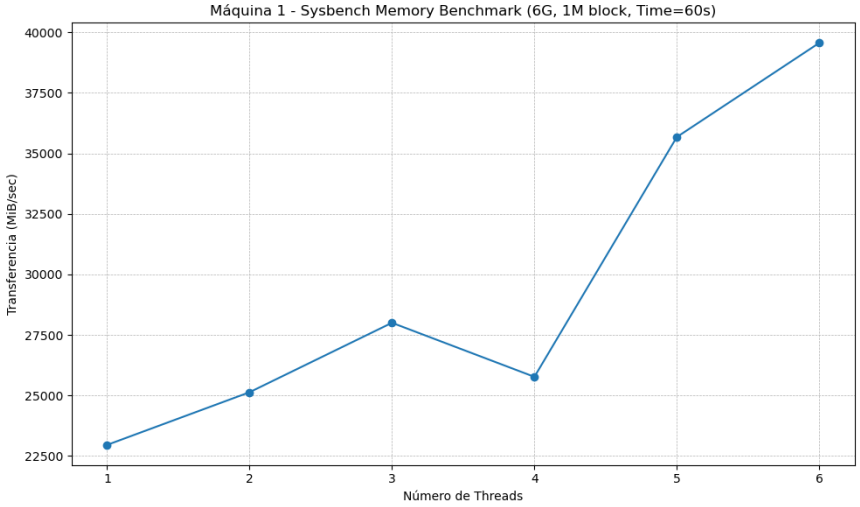

Figure 2: Hilos vs Transferencia (máquina 1).

4.0.2 Sysbench máquina 2

Sysbench CPU:
Para la máquina 2, se calcularon los eventos por segundo vs número de threads (máximo 32). Llegando a 100000 eventos por segundo utilizando 24 hilos, y desde 25 hilos en adelante los eventos por segundo se reducen o se mantienen iguales, esto es probablemente porque solo hay 24 núcleos físicos a pesar de haber 32 threads, con 6 hilos se llegó a casi 30000 eventos por segundo y con 1 solo hilo aproximadamente 5000.

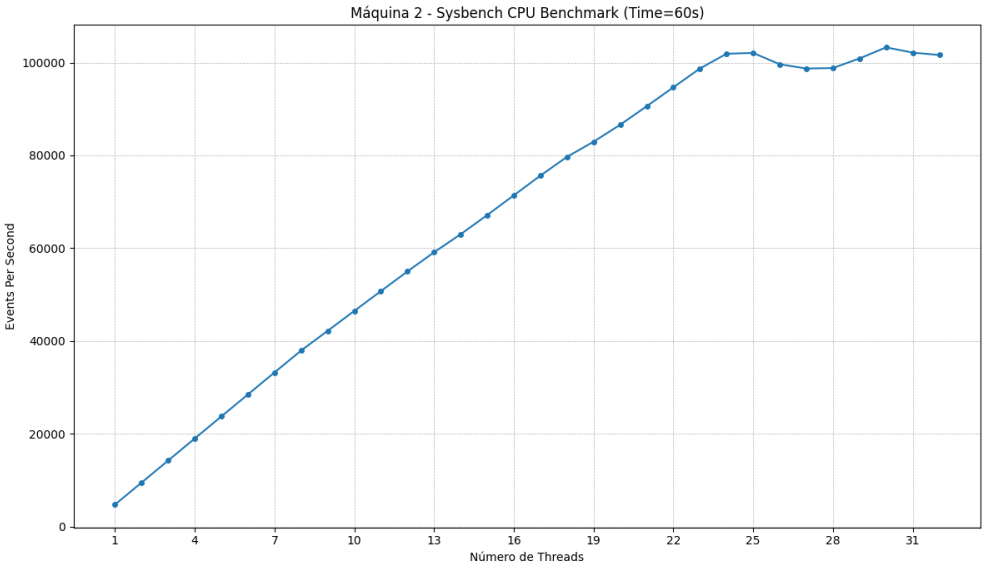

Figure 3: Hilos vs Eventos por segundo (máquina 2).

Sysbench memory:
Para la máquina 2, se calculó la transferencia (MiB/sec) vs número de threads (máximo 32), con 6G de memoria total y 1M de bloque de memoria. Donde vemos que el pico máximo en esta prueba fue utilizando 8 hilos, alcanzando 140000 MiB/sec, luego desde 9 hilos hasta 20 hilos vemos una taza de transferencia muy baja (menos de 40000 MiB/sec) peor que la máquina 1 con 6 hilos! y finalmente desde 21 hilos hasta 32 hilos el valor de MiB/sec aumenta entre los 50000 y 60000.

Se hizo esta prueba para hacer una comparación justa con **6G** de memoria total y solo comparando con los 6 primeros hilos, se ve una mejor transferencia en la segunda máquina (100000 MiB/sec) con 6 hilos a diferencia de la primera máquina (40000 MiB/sec) con 6 hilos.

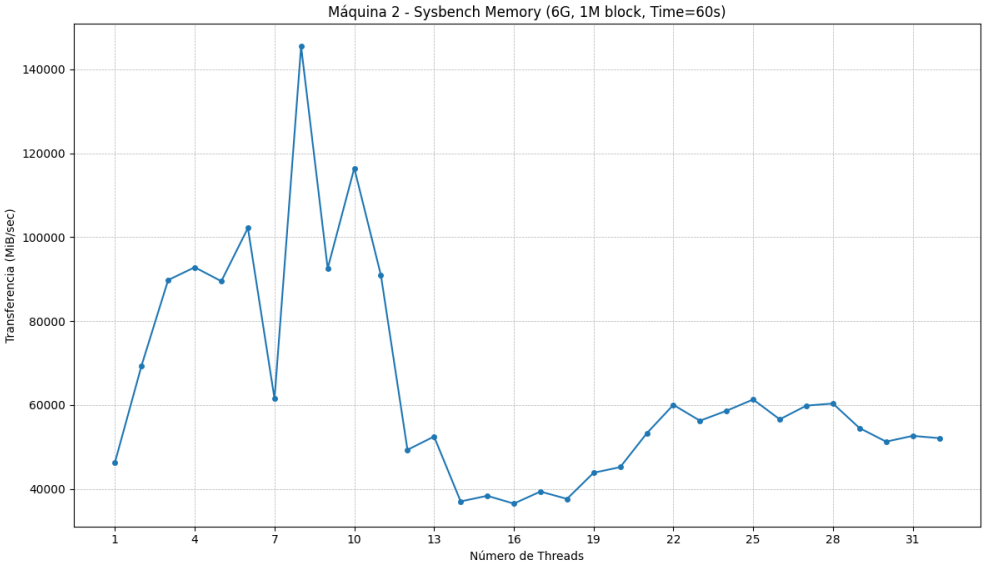

Figure 4: Hilos vs Transferencia 6G (máquina 2).

También se hizo el mismo cálculo pero con 124G de memoria total. Muy parecido con la anterior prueba pero con una curva mucho más suave y una transferencia máxima de 180000 MiB/sec utilizando 7 hilos.

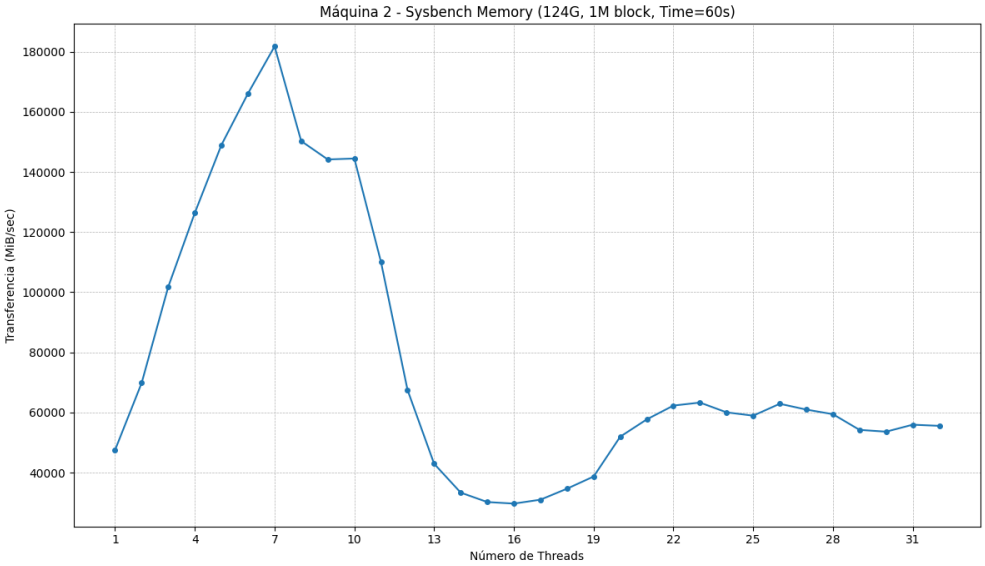

Figure 5: Hilos vs Transferencia 124G (máquina 2).

4.1. Pruebas multiplicación de matrices

A continuación se encuentran las gráficas del análisis de las pruebas de multiplicación de matrices en los 2 equipos.

4.1.1 Tiempo vs Tamaño de matrices

De las 2 gráficas a continuación podemos ver que la diferencia es bastante grande si comparamos las máquinas con la máxima configuración (6 hilos vs 32 hilos); con 6 hilos en la máquina 1 se puede llegar a 30.125 segundos para una matriz de **6400x6400**, en cambio, con 32 hilos en la máquina 2 llegamos hasta 6.825 segundos, pero el rendimiento no solo aparece cuando utilizamos la máxima configuración de la máquina 2, para hacer una comparación justa miramos la diferencia de las máquinas con los mismos hilos; donde en la máquina 2 con 6 hilos llegamos a realizar la multiplicación de matrices en solo 16.176 segundos, casi la mitad del tiempo que se demoró la máquina 1 tambien con 6 hilos!, aquí no solo se ve la diferencia en cuanto a cantidad de hilos, también en terminos de arquitectura del CPU.

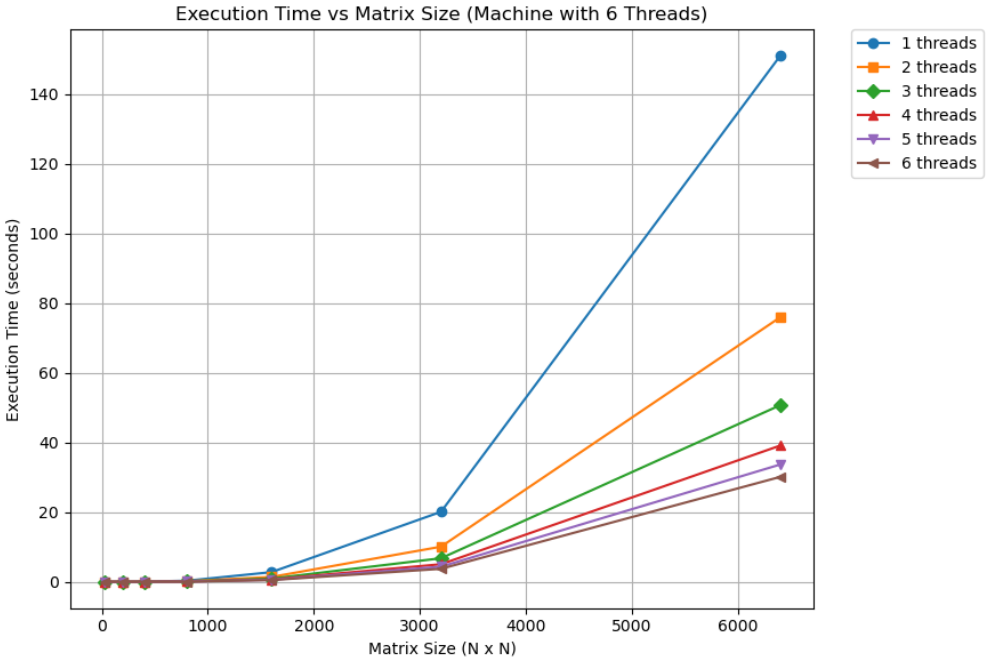

Figure 6: Tiempo vs Tamaño (máquina 1).

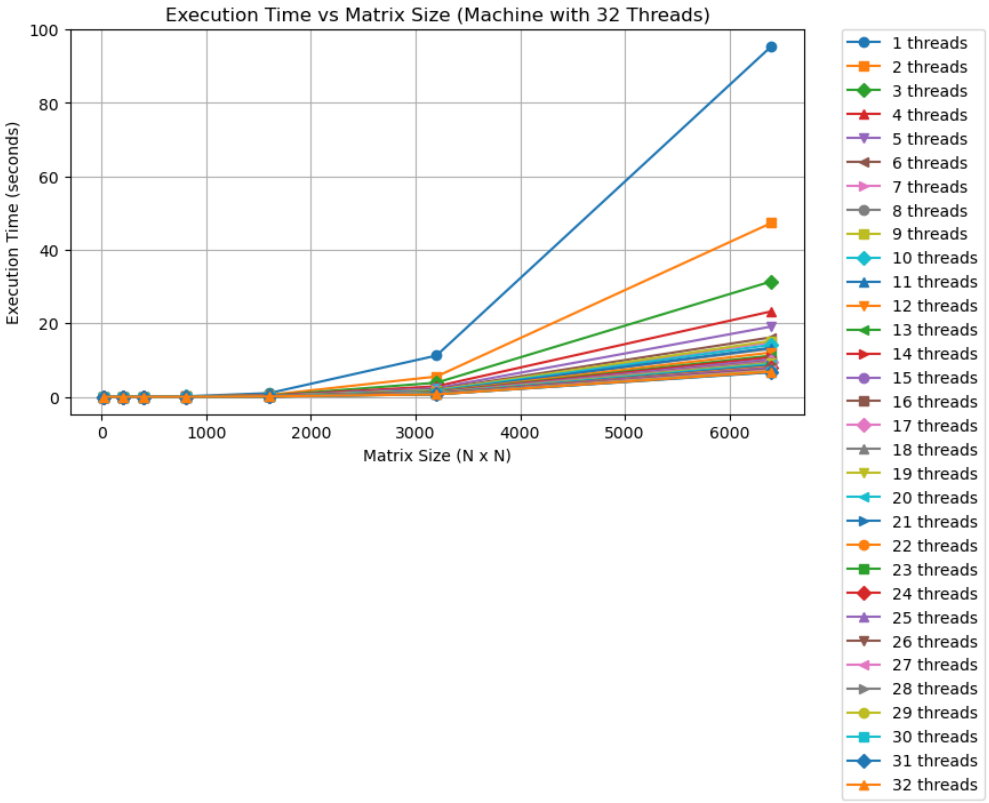

Figure 7: Tiempo vs Tamaño (máquina 2).

4.1.2 Speedup vs Número de hilos

A continuación, podemos ver las gráficas de speedup; como en el anterior informe, se nota un mejor speedup al usar más threads cuando las matrices son más grandes. No se ve una diferencia importante al comparar el speedup con los 6 primeros threads de cada máquina, pero al tener más threads en la segunda máquina, alcanzamos speedup más grandes, como el aproximadamente 17.5x en la multiplicación de matrices 3200x3200 utilizando 32 threads.

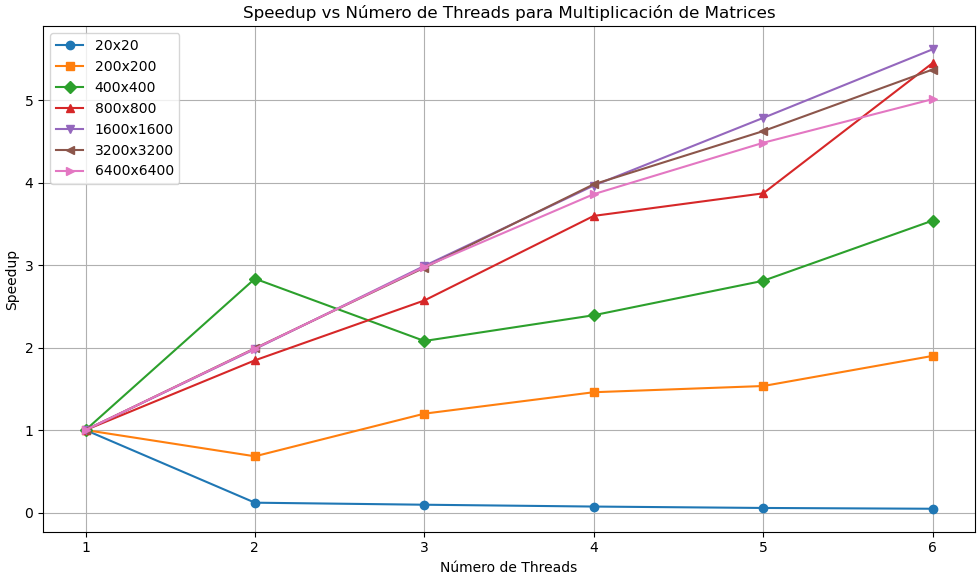

Figure 8: Threads vs Speedup (máquina 1).

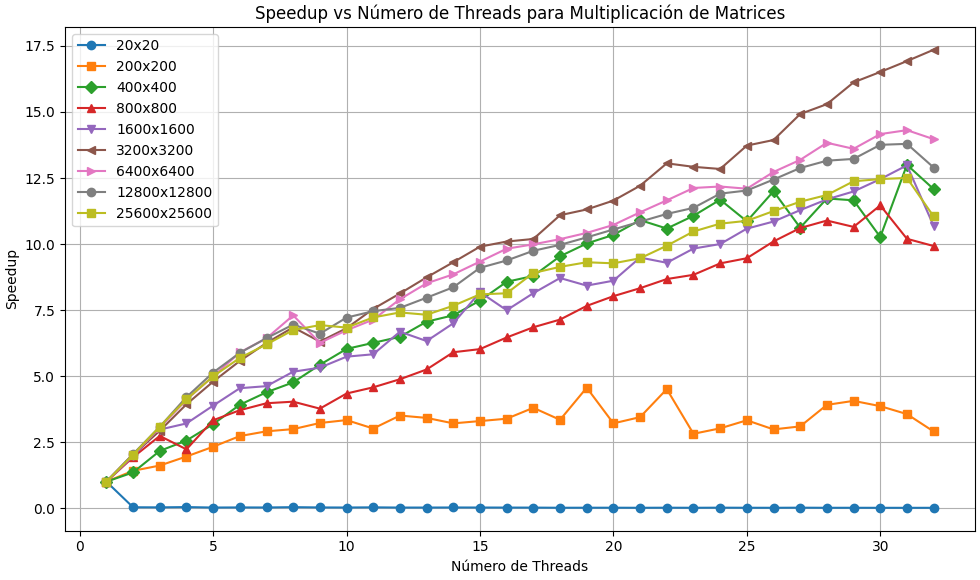

Figure 9: Threads vs Speedup (máquina 2).

4.1.3 Máquina 1 vs Máquina 2

Finalmente hay una comparación directa entre máquinas, del tiempo de ejecución vs el tamaño de la matriz y utilizando 6 hilos en cada máquina para una comparación justa; vemos que a medida que se van aumentando el tamaño de las matrices la diferencia entre máquinas va aumentando mucho más, esto es por la arquitectura del procesador y probablemente por mejor memoria y caché en la segunda máquina.

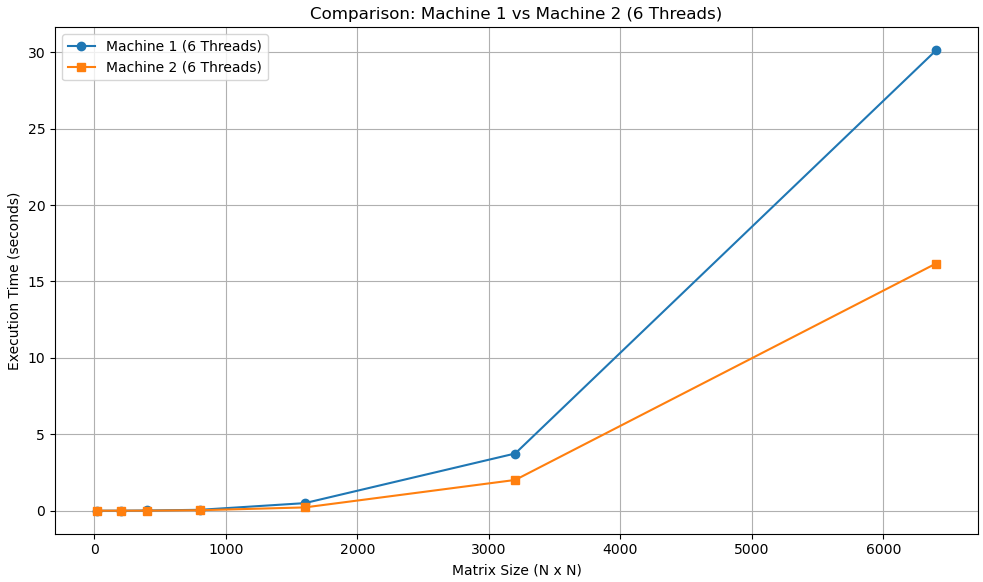

Figure 10: Máquina 1 vs Máquina 2.

5. CONCLUSIONES

El presente estudio evaluó comparativamente el rendimiento de la multiplicación de matrices utilizando OpenMP en dos máquinas con especificaciones distintas, complementando el análisis con benchmarks de CPU y memoria con sysbench. Se vió una gran diferencia en el rendimiento entre la Máquina 1 y la Máquina 2. Los benchmarks de sysbench mostraton una capacidad mayor de procesamiento de eventos por segundo y de transferencia de memoria (MiB/sec) en la Máquina 2, tendencia que se reflejó directamente en los tiempos de ejecución de la multiplicación de matrices. La Máquina 2 logró tiempos considerablemente menores, incluso al comparar ejecuciones con el mismo número de hilos (como con 6 hilos), diferencia que se vió al aumentar la dimensión de las matrices. Esta diferencia en el desempeño es por las diferencias en las especificaciones de hardware, destacando el mayor

número de núcleos/hilos del procesador (24/32 vs 6), la mayor cantidad de memoria RAM (125Gi vs 7.1Gi), diferencias en la arquitectura del procesador y la velocidad de memoria (incluyendo caché) de la Máquina 2. OpenMP demostró ser efectiva para paralelizar la carga de trabajo y reducir los tiempos de ejecución en ambas máquinas. El speedup obtenido mostró una tendencia general creciente con el número de hilos y el tamaño de las matrices, aunque se observaron indicios de rendimientos decrecientes o estabilización, especialmente en la Máquina 2 al superar el número de núcleos físicos o al encontrar posibles cuellos de botella en el acceso a memoria, como se vió en algunas de las pruebas de sysbench. Finalmente, los resultados muestran la importancia de la selección del hardware en entornos de HPC y cómo las características del procesador y la memoria impactan directamente en la eficiencia de algoritmos paralelos como la multiplicación de matrices.

References

Antunez, R. R. (2011). *Parallel programming: Definitions, mechanisms and trouble*. ResearchGate.

Ferestrepoca. (2019). *Programación paralela*. Ferestrepoca.

López, A. D. R. (2017). *Introducción al high performance computing (hpc)*. Encamina.

Vásquez, A. C. (2004). *Teoría de la complejidad computacional y teoría de la computabilidad*. RISI.