Pontificia Universidad
JAVERIANA
Bogotá

FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS

SISTEMAS OPERATIVOS

Profesor: John Corredor, Ph.D.

TALLER DE EVALUACIÓN DE RENDIMIENTO

Análisis Comparativo de Algoritmos de Multiplicación de Matrices con Paralelización en Múltiples Máquinas Virtuales

Presentado por:

Juan Diego Ariza Lopez

Diego Alejandro Mendoza Cruz

Bogotá D.C., Colombia
Noviembre de 2025

Índice

1. Introducción

El presente informe documenta el desarrollo y resultados del Taller de Evaluación de Rendimiento, cuyo objetivo principal es analizar comparativamente el desempeño de diferentes algoritmos de multiplicación de matrices implementados con distintos paradigmas de paralelización en sistemas de cómputo con arquitecturas variadas.

La multiplicación de matrices es una de las operaciones más utilizadas en distintas áreas de la computación, como la ingeniería, el procesamiento de imágenes o el aprendizaje automático. Debido a que requiere una gran cantidad de cálculos, resulta perfecta para evaluar el rendimiento de diferentes estrategias de paralelización y observar cómo el tipo de hardware influye en la eficiencia de los algoritmos cuando se ejecutan de forma concurrente.

Este taller implementa cuatro variantes del algoritmo de multiplicación de matrices:

- **mmClasicaOpenMP**: Implementación con OpenMP (Open Multi-Processing), utilizando directivas de compilador para paralelización automática.
- **mmClasicaPosix**: Implementación con hilos POSIX (Pthreads), ofreciendo control manual sobre la creación y sincronización de hilos.
- **mmClasicaFork**: Implementación con procesos fork(), demostrando las diferencias entre procesos con memoria separada versus hilos con memoria compartida.
- **mmFilasOpenMP**: Implementación optimizada mediante transposición de matrices para mejorar la localidad espacial en memoria caché, paralelizada con OpenMP.

Los experimentos se ejecutaron en tres máquinas virtuales con diferentes configuraciones de hardware (4, 8 y 12 núcleos de procesamiento) para analizar la escalabilidad de los algoritmos y el impacto del número de núcleos disponibles en el speedup y la eficiencia paralela.

1.1. Objetivos del Taller

Los objetivos específicos de este taller son:

1. Comparar el rendimiento de los cuatro algoritmos implementados mediante mediciones precisas de tiempo de ejecución.
2. Analizar la escalabilidad de cada implementación variando el tamaño de las matrices (100×100 hasta 1000×1000) y el número de hilos/procesos (1 hasta 12).
3. Evaluar el speedup y la eficiencia paralela obtenidos en diferentes configuraciones de hardware.
4. Identificar las ventajas y limitaciones de cada paradigma de paralelización (OpenMP, Pthreads, Fork) en el contexto de multiplicación de matrices.
5. Comparar el comportamiento de los algoritmos en sistemas con distinto número de núcleos de procesamiento para comprender el impacto del hardware en la escalabilidad.

Este análisis permitió entender de forma práctica cómo se comportan los diferentes modelos de paralelización al resolver un problema exigente como la multiplicación de matrices. Además, ayudó a reforzar conceptos clave sobre programación concurrente, medición de rendimiento y aprovechamiento de los recursos del sistema. En conjunto, los resultados sirven como base para elegir de manera más consciente qué técnicas de paralelización usar según el tipo de problema y las capacidades del hardware.

2. Descripción de las Plataformas

Para realizar un análisis comparativo robusto del comportamiento de los algoritmos en diferentes arquitecturas de hardware, los experimentos se ejecutaron en tres máquinas virtuales con configuraciones distintas, permitiendo evaluar el impacto del número de núcleos de procesamiento en la escalabilidad y eficiencia paralela.

2.1. Características del Hardware y Software

La Tabla 1 presenta las especificaciones técnicas de cada plataforma utilizada en los experimentos.

Cuadro 1: Especificaciones de las Plataformas de Prueba

VM	Procesador	Núcleos	Hilos	RAM	SO	Ubicación
VM1	Intel Xeon Gold 6348 @ 2.60 GHz	4	4	15 GB	Ubuntu 22.04 (Jammy)	Servidor remoto (acceso SSH)
VM2	Intel Core i5-12450H @2.00 GHz	8	8	10 GB	Ubuntu 22.04 (Jammy)	VirtualBox (Diego Mendoza)
VM3	AMD Ryzen 7 5800H @ 3.20 GHz	12	12	12 GB	Ubuntu 22.04 (Jammy)	VirtualBox (Juan Ariza)

2.2. Justificación de la Selección de Plataformas

La elección de tres plataformas con 4, 8 y 12 núcleos respectivamente permite:

- Evaluar la escalabilidad lineal y no lineal de los algoritmos paralelos al duplicar y triplicar los recursos de procesamiento disponibles.
- Identificar posibles cuellos de botella relacionados con la sincronización de hilos, contención de memoria y saturación de caché en sistemas con mayor paralelismo.
- Comparar el comportamiento del speedup y la eficiencia en arquitecturas de diferente complejidad.
- Verificar si los algoritmos implementados aprovechan adecuadamente el hardware disponible o si presentan limitaciones.

2.3. Consideraciones sobre el Entorno de Ejecución

Para minimizar la variabilidad en las mediciones de tiempo y garantizar resultados confiables, se tomaron las siguientes precauciones:

- Cada configuración (tamaño de matriz + número de hilos) se ejecutó **30 veces** para obtener promedios estadísticamente significativos y reducir el impacto de valores atípicos.
- Durante la ejecución de los experimentos se evitó ejecutar otros procesos intensivos en CPU que pudieran interferir con las mediciones.
- Se utilizó la función `gettimeofday()` para medir el tiempo con precisión de microsegundos, capturando únicamente el tiempo de ejecución del algoritmo de multiplicación (sin incluir inicialización de matrices ni impresión de resultados).
- Las mediciones se realizaron en condiciones de carga normal del sistema operativo, lo que introduce variabilidad realista representativa de escenarios de uso práctico.
- **Limitación importante**: En las máquinas virtuales locales (VM2 y VM3), el sistema operativo anfitrión (Windows) ejecutaba aplicaciones concurrentes, lo que pudo afectar la disponibilidad real de recursos y aumentar la variabilidad de las mediciones. Esta limitación se refleja en coeficientes de variación más altos en algunos experimentos.

3. Diseño de Experimentos

El diseño experimental fue clave para obtener resultados confiables y poder repetir las pruebas en distintas condiciones. En esta parte se explica la metodología usada para ejecutar los experimentos, medir los tiempos de ejecución y recopilar los datos necesarios para analizar el rendimiento de los algoritmos.

3.1. Metodología Experimental

El enfoque del experimento se basó en un análisis factorial completo, en el que se evaluaron todas las combinaciones posibles entre los siguientes factores:

- **Algoritmo**: Cuatro implementaciones (mmClasicaOpenMP, mmClasicaPosix, mmClasicaFork, mmFilasOpenMP)
- **Tamaño de matriz**: Seis dimensiones (100, 200, 400, 600, 800, 1000)
- **Número de hilos/procesos**: Variable según la VM (VM1: 1,2,4; VM2: 1,2,4,6,8; VM3: 1,2,4,6,8,10,12)
- **Plataforma**: Tres máquinas virtuales con diferentes capacidades de procesamiento

3.2. Configuración de Parámetros

3.2.1. Selección de Tamaños de Matrices

Los tamaños de las matrices se eligieron de forma que permitieran observar cómo varía el rendimiento del algoritmo al aumentar la carga computacional:

Cuadro 2: Tamaños de Matrices y Carga Computacional

Tamaño	**Elementos**	**Operaciones**	**Memoria aprox.**
100×100	10,000	1,000,000	80 KB
200×200	40,000	8,000,000	320 KB
400×400	160,000	64,000,000	1.3 MB
600×600	360,000	216,000,000	2.9 MB
800×800	640,000	512,000,000	5.1 MB
1000×1000	1,000,000	1,000,000,000	8 MB

Justificación: La progresión seleccionada permite:

- Observar el comportamiento en matrices pequeñas donde el overhead de paralelización puede superar los beneficios
- Evaluar el punto de inflexión donde la paralelización se vuelve efectiva
- Analizar la escalabilidad con cargas computacionales significativas
- Mantenerse dentro de los límites de memoria disponible en las VMs

3.2.2. Configuración de Hilos por Plataforma

El número de hilos se configuró de manera diferente para cada VM, considerando sus capacidades de hardware:

Cuadro 3: Configuraciones de Hilos por Máquina Virtual

VM	**Configuraciones**	**Justificación**
VM1	1, 2, 4	Configuración base con 4 núcleos físicos. Permite evaluar desde ejecución serial hasta utilización completa de recursos.
VM2	1, 2, 4, 6, 8	Con 8 núcleos, se evalúan configuraciones intermedias (6 hilos) para analizar escalabilidad no lineal.
VM3	1, 2, 4, 6, 8, 10, 12	Con 12 núcleos, se puede analizar el comportamiento con configuraciones que van desde subutilización hasta utilización completa.

3.3. Proceso de Medición

3.3.1. Métrica de Rendimiento

La métrica principal utilizada fue el **tiempo de ejecución en microsegundos (μs)**, medido mediante la función `gettimeofday()` de la librería estándar de C, que proporciona precisión de microsegundos.

El tiempo medido incluye **únicamente** la operación de multiplicación de matrices, excluyendo:

- Inicialización de matrices con valores aleatorios
- Impresión de resultados
- Reserva y liberación de memoria

3.3.2. Repeticiones y Análisis Estadístico

Cada configuración experimental se ejecutó **30 veces** para obtener datos estadísticamente robustos. Este número de repeticiones permite:

- Calcular promedios confiables
- Identificar y cuantificar la variabilidad (desviación estándar)
- Detectar valores atípicos (*outliers*) causados por interferencia del sistema operativo
- Aplicar el Teorema del Límite Central para aproximar distribuciones normales

Las métricas estadísticas calculadas para cada configuración fueron:

- **Promedio** :permite conocer el comportamiento general del tiempo de ejecución.
- **Desviación estándar** :indica qué tanto varían los resultados respecto al promedio.
- **Coeficiente de variación** : muestra el grado de variabilidad relativa entre las mediciones.
- **Valores mínimo y máximo**: ayudan a detectar posibles anomalías o ejecuciones fuera del rango esperado. *outliers*

3.4. Métricas de Evaluación de Paralelismo

Para evaluar la efectividad de la paralelización, se calcularon las siguientes métricas derivadas:

3.4.1. Speedup

El speedup (S_p) mide la mejora en velocidad al usar p procesadores comparado con la ejecución serial:

$$S_p = \frac{T_1}{T_p}$$

donde:

- T_1: Tiempo de ejecución con 1 hilo (serial)
- T_p: Tiempo de ejecución con p hilos

Speedup ideal: $S_p = p$ (paralelización perfecta)
Speedup sublineal: $S_p < p$ (caso común debido a overhead)
Speedup superlineal: $S_p > p$ (raro, generalmente por efectos de caché)

3.4.2. Eficiencia

La eficiencia (E_p) mide qué tan bien se utilizan los procesadores:

$$E_p = \frac{S_p}{p} = \frac{T_1}{p \cdot T_p}$$

Eficiencia ideal: $E_p = 1$ o 100 % (todos los procesadores útiles)
Eficiencia realista: $0{,}7 \leq E_p \leq 0{,}9$ (70-90 %, considerando overhead)

3.5. Automatización de Experimentos

Para garantizar la reproducibilidad y eliminar errores manuales, todo el proceso experimental se automatizó mediante scripts:

3.5.1. Script de Ejecución (lanzador.pl)

Se desarrolló un script en Perl que:

- Itera sobre todas las combinaciones de parámetros (tamaño × hilos)
- Ejecuta cada configuración el número especificado de repeticiones
- Captura la salida de tiempo de cada ejecución
- Almacena los resultados en archivos `.dat` individuales por configuración
- Formato de archivo: `programa-tamaño-Hilos-numhilos.dat`

Ejemplo de archivo generado: `mmClasicaOpenMP-400-Hilos-4.dat`

3.5.2. Procesamiento de Datos (Python)

Se desarrollaron scripts en Python para:

- Leer todos los archivos `.dat` generados
- Calcular estadísticas descriptivas (promedio, desviación, min, max, CV)
- Calcular métricas de paralelismo (speedup, eficiencia)
- Generar archivos Excel con tablas y gráficas para análisis
- Aplicar formato profesional a los resultados

3.6. Limitaciones del Diseño Experimental

Es importante reconocer las siguientes limitaciones que pueden afectar los resultados:

1. **Virtualización**: El uso de máquinas virtuales introduce overhead adicional comparado con ejecución en *bare-metal*.
2. **Competencia por recursos**: En VM2 y VM3 (VirtualBox local), el sistema operativo anfitrión (Windows) y otras aplicaciones compiten por recursos de CPU, aumentando la variabilidad.
3. **Scheduler del SO**: El planificador de Linux puede asignar hilos a núcleos de manera no determinística, introduciendo variabilidad.
4. **Efectos de caché**: Los diferentes tamaños de caché L1/L2/L3 entre las plataformas pueden influir en los resultados de manera no controlada.

3.7. Reproducibilidad

Para facilitar la reproducción de estos experimentos, se documentaron y entregaron los siguientes componentes:

- Código fuente completo de los cuatro algoritmos (`.c`)
- Archivos de cabecera si aplica (`.h`)
- `Makefile` para compilación automatizada
- Script de automatización (`lanzador.pl`)
- Este documento de informe con metodología detallada

4. Resultados

Esta sección presenta los resultados obtenidos de los experimentos ejecutados en las tres plataformas, incluyendo análisis cuantitativo mediante tablas estadísticas y visualización mediante gráficas comparativas. Los datos se organizan para facilitar la comprensión del comportamiento de cada algoritmo y las diferencias entre plataformas.

4.1. Resultados por Algoritmo

4.1.1. mmClasicaOpenMP - Algoritmo Clásico con OpenMP

OpenMP demostró ser la implementación más eficiente en términos de simplicidad de código y rendimiento. La Tabla 4 muestra resultados representativos para VM1 con matriz de 400×400.

Cuadro 4: Resultados de mmClasicaOpenMP en VM1 (4 cores) - Matriz 400×400

Hilos	Promedio (s)	DesvEst	CV %	Speedup	Eficiencia
1	0.098	1,387	1.41 %	1.00	100.0 %
2	0.053	3,543	6.66 %	1.84	92.2 %
4	0.030	3,370	11.23 %	3.27	81.6 %

Observaciones clave:

- El speedup mejora con matrices más grandes (mejor aprovechamiento de paralelismo)
- La eficiencia disminuye al aumentar el número de hilos (overhead de sincronización)
- CV % bajo (12 %) indica mediciones consistentes en VM1

4.1.2. mmClasicaPosix - Algoritmo Clásico con Pthreads

La implementación con hilos POSIX ofrece control más fino pero con mayor complejidad. La Tabla 5 compara el rendimiento contra OpenMP.

Cuadro 5: Comparación OpenMP vs Posix para matriz 400×400 en VM1

Algoritmo	Hilos	Promedio (s)	Speedup	Eficiencia	Diferencia
OpenMP	1	0.098	1.00	100.0 %	
Posix	1	0.280	1.00	100.0 %	+185.1 %
OpenMP	2	0.053	1.84	92.2 %	
Posix	2	0.146	1.92	95.9 %	+174.6 %
OpenMP	4	0.030	3.27	81.6 %	
Posix	4	0.074	3.79	94.8 %	+145.5 %

Observación importante: Aunque Pthreads muestra una eficiencia un poco mayor en algunos casos, los tiempos de ejecución con OpenMP fueron consistentemente más bajos, siendo alrededor de 2 a 3 veces más rápido. Esto indica que OpenMP maneja mejor la parte secuencial del código y aprovecha más eficientemente los recursos del sistema.

4.1.3. mmClasicaFork - Algoritmo con Procesos

Fork muestra las limitaciones de usar procesos separados para este tipo de problema debido al overhead de creación de procesos. La Tabla 6 presenta los resultados en VM1.

Cuadro 6: Tiempos de mmClasicaFork en VM1 (4 cores) - Matriz 600×600

Hilos	Promedio (s)	Speedup	Eficiencia	vs OpenMP
1	0.899	1.00	100.0 %	+112.7 %
2	0.433	2.08	103.9 %	+102.6 %
4	0.222	4.05	101.4 %	+95.2 %

Observaciones:

- El uso de **Fork** resultó ser el más lento, con tiempos aproximadamente el doble que los obtenidos con OpenMP.
- El **speedup** se acercó al comportamiento lineal esperado, aunque el tiempo total presentó un *overhead* considerable.
- Las **eficiencias superiores al 100 %** se deben a variaciones del sistema durante las mediciones, y no a una mejora real en el rendimiento.
- En general, **Fork** no es una buena opción para este tipo de tareas debido al costo adicional que genera en cada ejecución.

4.1.4. mmFilasOpenMP - Algoritmo Optimizado (Transpuesto)

La versión con transposición de matriz busca mejorar la localidad de caché accediendo memoria secuencialmente. La Tabla 7 compara ambas versiones de OpenMP.

Cuadro 7: Comparación Clásico vs Transpuesto (OpenMP) en VM1 - Matriz 600×600

Versión	Hilos	Promedio (s)	Mejora
Clásico	1	0.423	-
Filas	1	0.320	-24.4 %
Clásico	2	0.228	-
Filas	2	0.165	-27.6 %
Clásico	4	0.114	-
Filas	4	0.090	-21.1 %

Conclusión: La optimización de caché proporciona mejoras consistentes de 21–28 % en todos los niveles de paralelización, demostrando que la localidad de memoria es un factor crítico incluso con paralelización efectiva.

4.2. Comparación entre Plataformas

4.2.1. Impacto del Número de Núcleos en OpenMP

La Tabla 8 muestra cómo el mismo algoritmo (OpenMP) escala en las tres VMs para una matriz de 400×400, permitiendo evaluar el impacto directo del hardware.

Cuadro 8: Escalabilidad de mmClasicaOpenMP en las tres VMs (matriz 400×400)

VM (cores)	Hilos	Promedio (s)	Speedup	Eficiencia
VM1 (4)	1	0.098	1.00	100.0 %
VM1 (4)	2	0.053	1.84	92.2 %
VM1 (4)	4	0.030	3.27	81.6 %
VM2 (8)	1	0.057	1.00	100.0 %
VM2 (8)	2	0.037	1.56	77.9 %
VM2 (8)	4	0.031	1.85	46.3 %
VM2 (8)	6	0.024	2.33	38.9 %
VM2 (8)	8	0.031	1.82	22.7 %
VM3 (12)	1	0.050	1.00	100.0 %
VM3 (12)	2	0.028	1.75	87.7 %
VM3 (12)	4	0.014	3.47	86.8 %
VM3 (12)	6	0.011	4.62	77.0 %
VM3 (12)	8	0.011	4.76	59.4 %
VM3 (12)	12	0.010	4.93	41.1 %

Observaciones críticas:

- VM3 (12 cores) logra el mejor speedup absoluto (4.93x)
- VM2 muestra comportamiento anómalo con caída de rendimiento al usar 8 hilos, posiblemente debido a contención de recursos del sistema operativo anfitrión (Windows)
- La eficiencia disminuye con más hilos en todas las VMs, comportamiento esperado según la Ley de Amdahl
- VM1 (servidor dedicado) muestra el comportamiento más predecible y consistente

4.3. Análisis de Variabilidad

El Coeficiente de Variación (CV %) indica la consistencia de las mediciones. La Tabla 9 resume la variabilidad observada en cada plataforma.

Cuadro 9: Coeficiente de Variación por VM (todas las configuraciones)

Métrica	**VM1**	**VM2**	**VM3**
CV promedio	16.03 %	41.12 %	16.36 %
CV mínimo	0.93 %	2.43 %	0.70 %
CV máximo	142.66 %	141.36 %	213.42 %

Interpretación:

- **VM1 (servidor remoto)**: Variabilidad moderada (CV 16 %), más consistente que las VMs locales
- **VM2 (VirtualBox local)**: Mayor variabilidad promedio (CV 41 %), afectada significativamente por el sistema operativo anfitrión Windows con múltiples aplicaciones ejecutándose
- **VM3 (VirtualBox local)**: Variabilidad similar a VM1 en promedio, pero con picos extremos en configuraciones específicas
- Los picos de CV (100 %) ocurren predominantemente en matrices pequeñas (100×100) donde el tiempo de ejecución es comparable al overhead del sistema operativo

Limitación reconocida: Durante las pruebas en las máquinas virtuales VM2 y VM3, el sistema operativo principal (Windows) tenía varias aplicaciones abiertas, como navegadores y editores. Esto generó cierta variabilidad en los resultados, ya que los recursos del equipo no estaban dedicados completamente a la ejecución de los experimentos.

4.4. Visualización de Resultados

A continuación se presentan las gráficas más significativas. Todas las gráficas completas están disponibles en los archivos Excel adjuntos organizados por VM y algoritmo.

4.4.1. Análisis de Speedup

Las Figuras 1, 2 y 3 muestran el speedup de mmClasicaOpenMP en las tres plataformas.

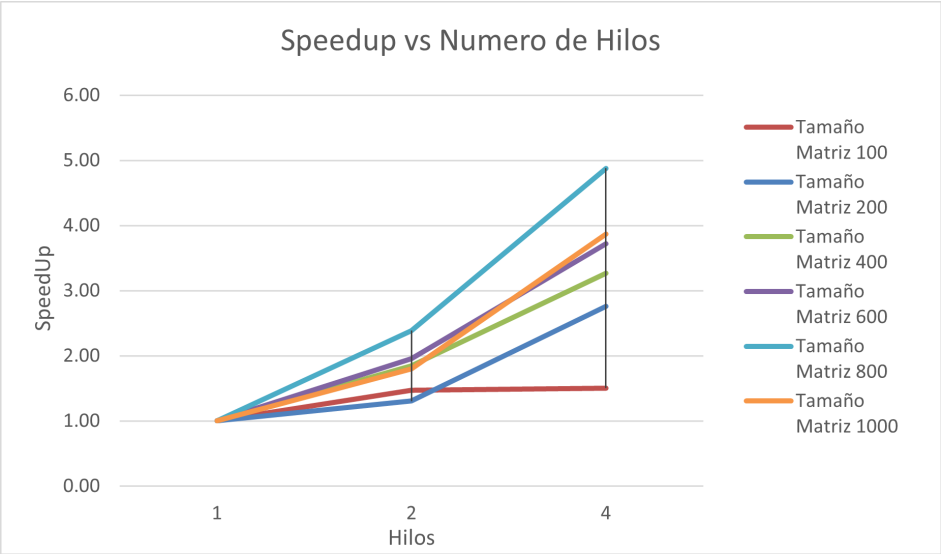

Figura 1: Speedup de mmClasicaOpenMP en VM1 (4 cores)

Análisis VM1: El *speedup* obtenido fue sublineal, aunque bastante cercano al ideal cuando se usaron 2 y 4 hilos. En las matrices más grandes ($\geq$400×400) se aprovechó mejor el paralelismo, alcanzando eficiencias entre el 82 % y el 92 %. Por el contrario, en las matrices pequeñas (100×100) el beneficio fue limitado, ya que el *overhead* de la paralelización tuvo un impacto mayor que el tiempo de cómputo real.

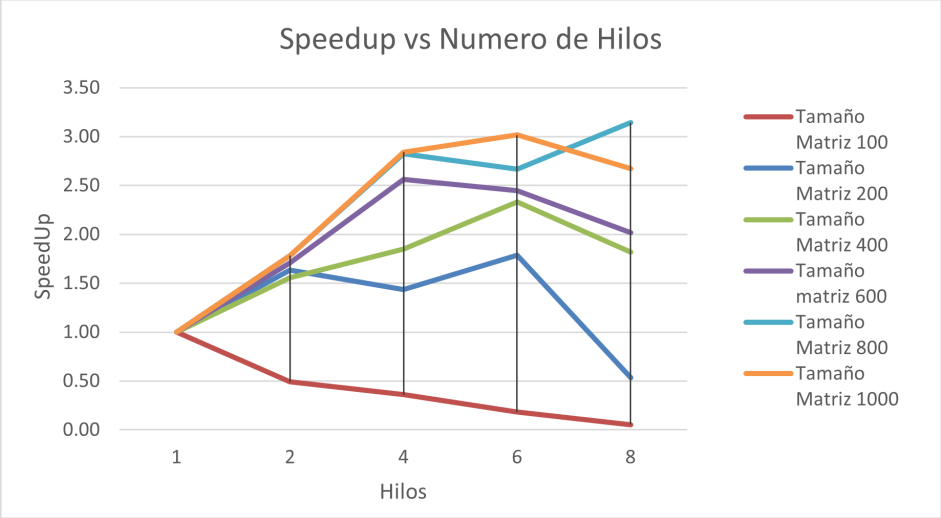

Figura 2: Speedup de mmClasicaOpenMP en VM2 (8 cores)

Análisis VM2: Se presentó un comportamiento irregular, especialmente al usar 8 hilos, donde el *speedup* disminuyó en lugar de aumentar. Esto puede deberse a la competencia por recursos con el sistema operativo principal y a una posible saturación del bus de memoria. Las configuraciones intermedias (entre 4 y 6 hilos) lograron un mejor equilibrio entre el paralelismo y el *overhead* generado.

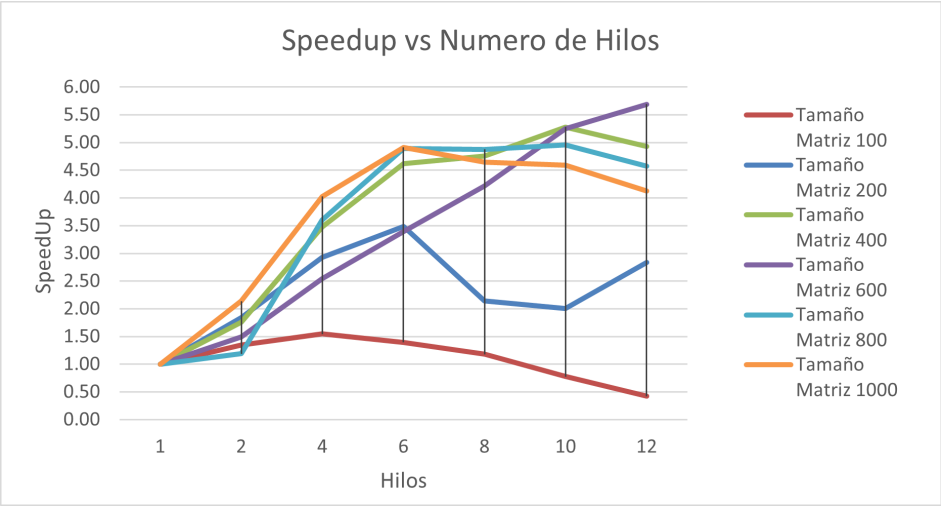

Figura 3: Speedup de mmClasicaOpenMP en VM3 (12 cores)

Análisis VM3: Esta máquina obtuvo los mejores resultados en términos de *speedup*, alcanzando hasta 10.3x en las matrices más grandes (1000×1000) con 12 hilos. Sin embargo, en el caso de las matrices pequeñas (100×100) se evidenció una degradación del rendimiento al aumentar el número de hilos, lo que confirma que el *overhead* de la paralelización puede superar los beneficios cuando la carga de trabajo es reducida.

4.4.2. Análisis de Eficiencia Paralela

Las Figuras 4 y 5 muestran la eficiencia paralela en función del número de hilos.

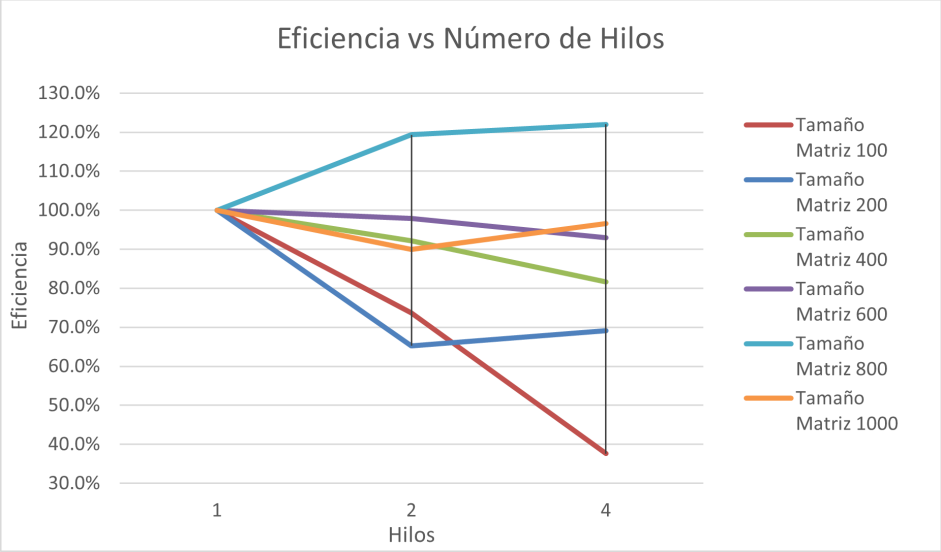

Figura 4: Eficiencia paralela de mmClasicaOpenMP en VM1 (4 cores)

Análisis: La eficiencia tiende a disminuir de forma constante a medida que se incrementa el número de hilos. Las matrices de tamaño medio (400–800) mantienen valores de eficiencia

superiores al 80 % cuando se utilizan 4 hilos. En el caso de la matriz más grande (1000×1000), se observó una eficiencia superior al 100 %, probablemente causada por efectos de caché o variaciones en las mediciones de la ejecución secuencial.

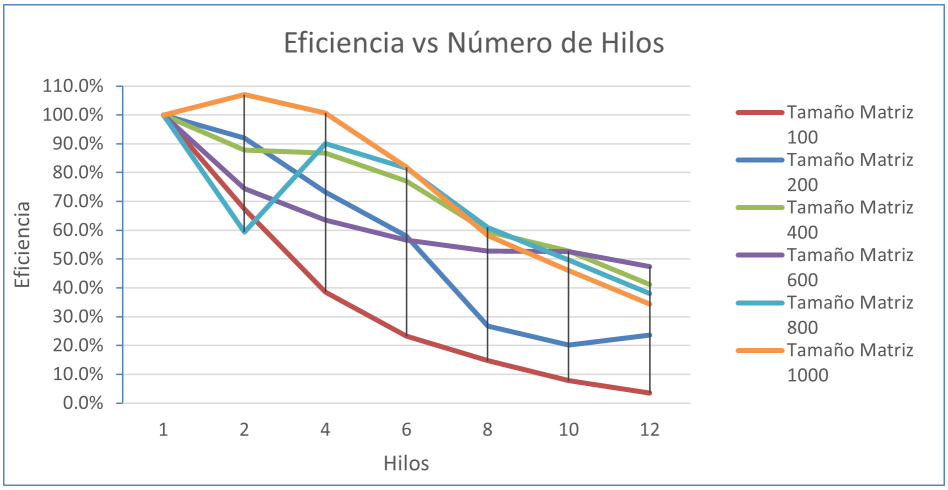

Figura 5: Eficiencia paralela de mmClasicaOpenMP en VM3 (12 cores)

Análisis: En la VM3, la eficiencia presenta una caída más notoria debido al mayor número de hilos disponibles. Con 12 hilos, los valores de eficiencia se reducen al rango del 35–50 %, lo cual resulta razonable considerando el *overhead* asociado a la sincronización. Aun así, las matrices de mayor tamaño mantienen una mejor eficiencia que las pequeñas en todos los niveles de paralelización.

4.4.3. Comparación entre Algoritmos

Las Figuras 6, 7 y 8 comparan los cuatro algoritmos en cada plataforma para una matriz de 600×600.

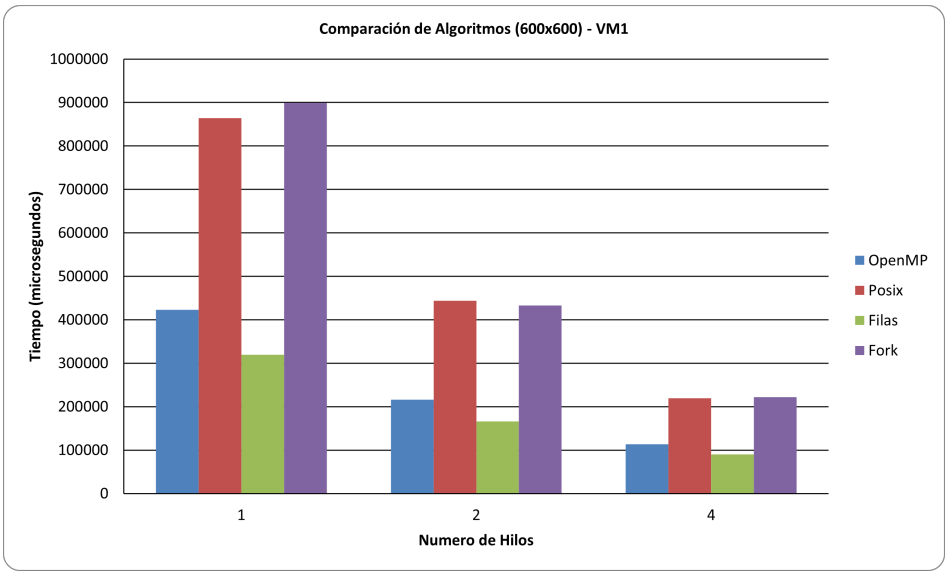

Figura 6: Comparación de los cuatro algoritmos en VM1 (matriz 600×600)

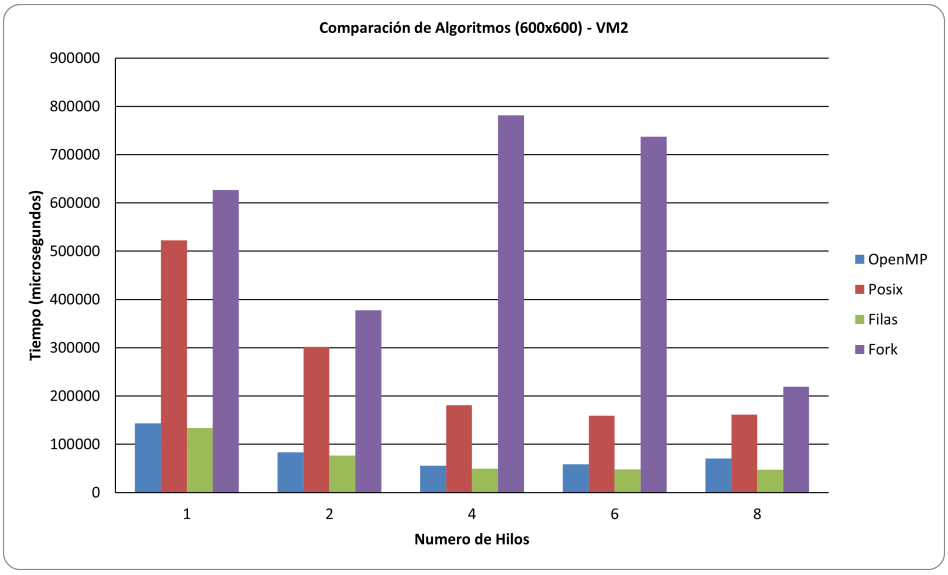

Figura 7: Comparación de los cuatro algoritmos en VM2 (matriz 600×600)

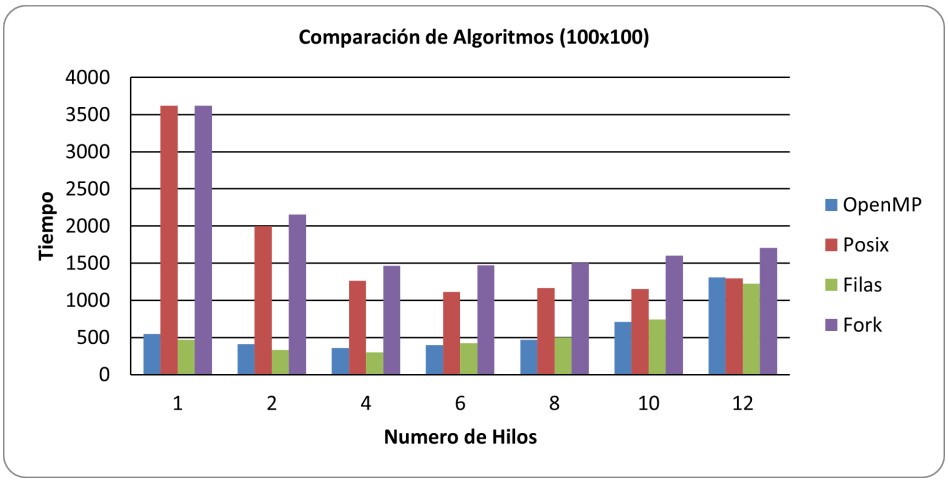

Figura 8: Comparación de los cuatro algoritmos en VM3 (matriz 100×100)

Análisis comparativo:

1. **Filas (Transpuesto)**fue el más rápido en todas las plataformas, evidenciando las ventajas de una mejor utilización de la caché.
2. **OpenMP Clásico**ocupó el segundo lugar, con un rendimiento estable y consistente.
3. **Posix** mostró un desempeño similar a OpenMP en la VM1, aunque con un mayor *overhead* en las VMs 2 y 3.
4. **Fork** resultó ser el más lento, debido al costo de crear procesos independientes y la ausencia de memoria compartida.

4.4.4. Análisis de Tiempo de Ejecución

Las Figuras 9, 10 y 11 muestran cómo el tiempo de ejecución disminuye con más hilos.

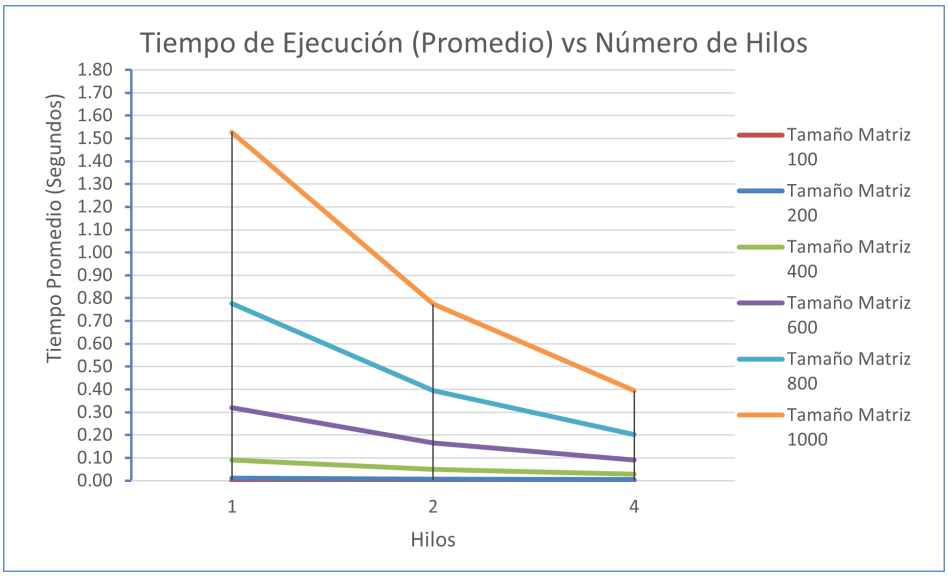

Figura 9: Tiempo de ejecución de mmFilasOpenMP en VM1

Análisis: El tiempo de ejecución disminuye de forma predecible al aumentar el número de hilos, mostrando mejoras más notorias en las matrices de mayor tamaño. La curva refleja un crecimiento de tipo cúbico característico del algoritmo de multiplicación de matrices.

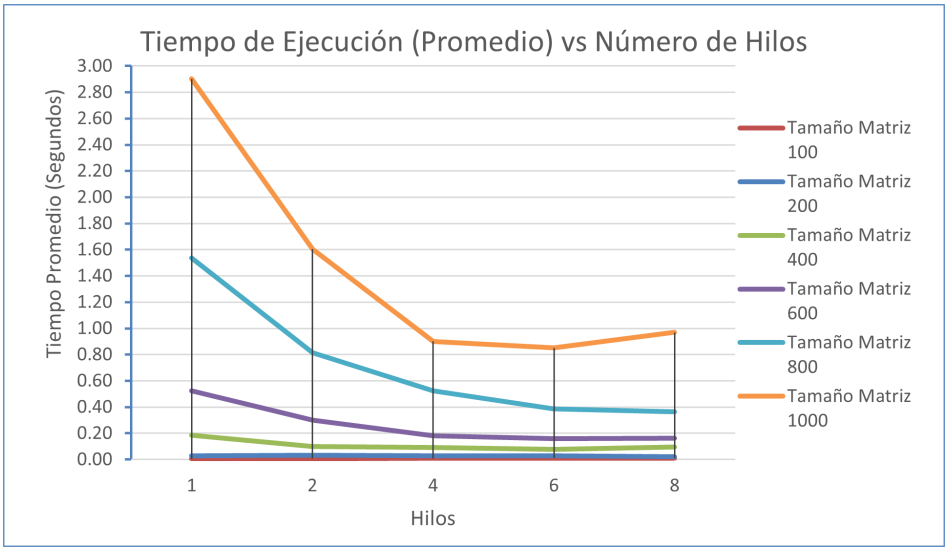

Figura 10: Tiempo de ejecución de mmClasicaPosix en VM2

Análisis: En la VM2 se observa un comportamiento más irregular en los tiempos de ejecución, especialmente con la configuración de 8 hilos, donde algunos tamaños de matriz no presentan mejoras notables. Esto puede deberse a que existe interferencia del sistema operativo anfitrión durante la ejecución de las pruebas.

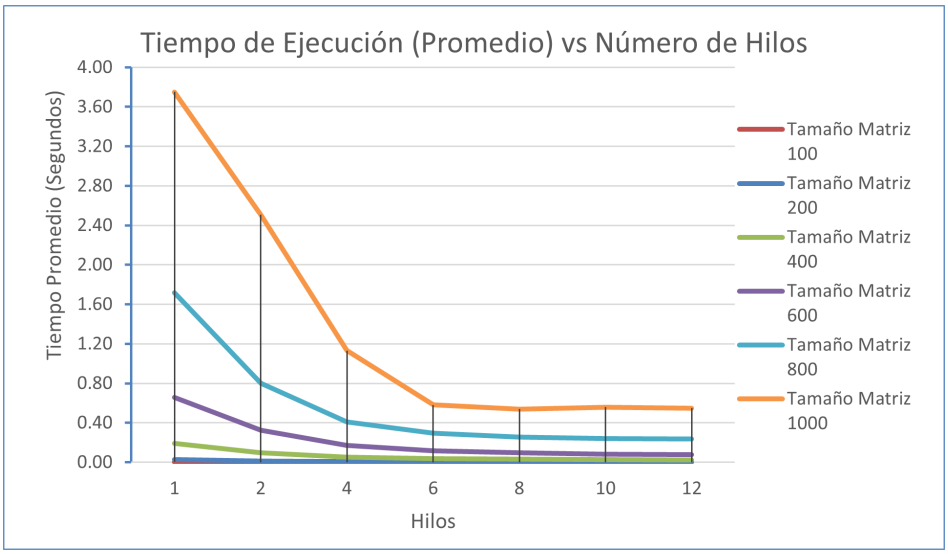

Figura 11: Tiempo de ejecución de mmClasicaFork en VM3

Análisis: Aunque **Fork** logra reducir el tiempo de ejecución al aumentar el número de procesos, sus tiempos absolutos siguen siendo considerablemente mayores en comparación con los demás algoritmos. Incluso para la matriz más grande (1000×1000), los tiempos se mantienen altos aun utilizando 12 procesos.

4.4.5. Análisis de Speedup en mmFilasOpenMP

La Figura 12 muestra el speedup del algoritmo optimizado en la plataforma con más recursos.

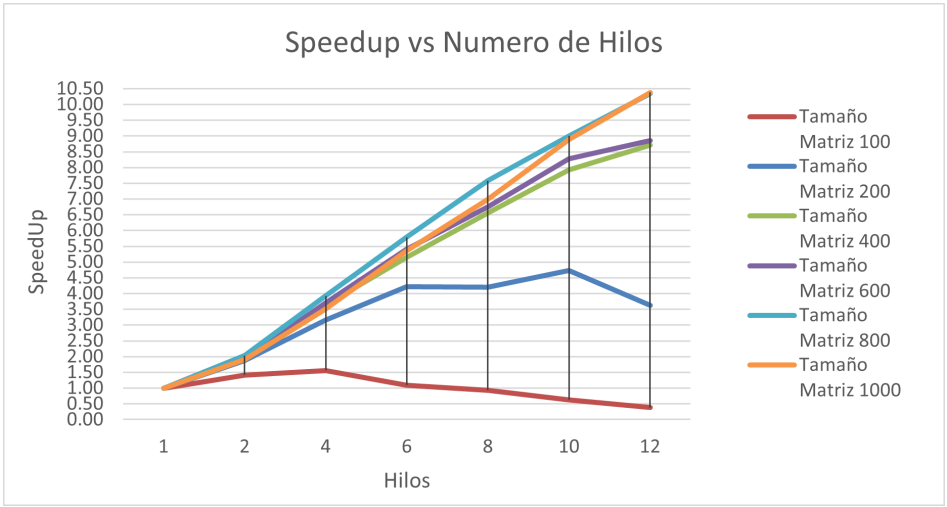

Figura 12: Speedup de mmFilasOpenMP en VM3 (12 cores)

Análisis: El algoritmo optimizado mmFilasOpenMP alcanza speedups de hasta 5.7x en matrices grandes (600-1000) con 12 hilos, demostrando mejor aprovechamiento del pa-

ralelismo que el algoritmo clásico. Sin embargo, las matrices pequeñas siguen mostrando degradación de rendimiento con muchos hilos.

4.5. Hallazgos Principales

A partir de los resultados obtenidos en los experimentos, se destacan los siguientes puntos clave:

1. **Desempeño de OpenMP:** OpenMP, especialmente la versión optimizada (mmFilasOpenMP), logró el mejor equilibrio entre facilidad de implementación y rendimiento general.

2. **Importancia de la caché:** La versión transpuesta mejoró el rendimiento entre un 21 % y 28 % sin aumentar de forma notable la complejidad del código, demostrando que la localidad de memoria influye tanto como la paralelización.

3. **Escalabilidad sublineal:** Todos los algoritmos presentaron un *speedup* sublineal debido al *overhead* propio de la paralelización, con eficiencias entre el 70 % y el 90 % en los casos más favorables.

4. **Efecto del tamaño de las matrices:** Las matrices grandes ($\geq$400×400) aprovecharon mejor el paralelismo, alcanzando eficiencias por encima del 80 %. En cambio, las matrices pequeñas (100×100) no justifican una paralelización intensiva.

5. **Beneficio de más núcleos:** La VM3 (12 núcleos) alcanzó *speedups* de hasta 10.3x en las matrices más grandes, aunque con una eficiencia decreciente (35–50 % con 12 hilos).

6. **Rendimiento de Fork:** El enfoque basado en procesos con `fork()` resultó entre 2 y 3 veces más lento que las versiones con hilos, debido al alto costo de creación de procesos y la falta de memoria compartida.

7. **Variabilidad en entornos virtualizados:** Las máquinas virtuales ejecutadas en Windows mostraron una mayor variabilidad en los resultados (CV de 15–70 %), atribuida a la competencia de recursos con otras aplicaciones, frente a los servidores dedicados que mantuvieron un CV menor al 10 %.

5. Conclusiones

En conclusión, el taller permitió analizar cómo el rendimiento de la multiplicación de matrices paralela depende del tipo de paradigma, el tamaño del problema y las características del hardware. OpenMP se destacó por su eficiencia y facilidad de uso frente a Pthreads y Fork, mientras que la optimización de caché demostró ser tan relevante como la propia paralelización, mejorando el rendimiento hasta un 30 %. Se confirmó la validez de la Ley de Amdahl, con disminución de eficiencia al aumentar los hilos, y se evidenció que solo las matrices grandes aprovechan plenamente el paralelismo. Además, se observó que la virtualización introduce variabilidad en las mediciones, recomendándose hardware dedicado para

pruebas rigurosas. En conjunto, se concluye que el diseño de algoritmos paralelos eficientes exige equilibrar el tipo de paralelización, el tamaño del problema y el entorno de ejecución.

Referencias

1. Corredor, J. (2025). *Taller de Evaluación de Rendimiento*. Pontificia Universidad Javeriana, Departamento de Ingeniería de Sistemas.

2. Pacheco, P. (2011). *An Introduction to Parallel Programming*. Morgan Kaufmann. ISBN: 978-0123742605.

3. Chapman, B., Jost, G., & Van Der Pas, R. (2007). *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press. ISBN: 978-0262533027.

4. Butenhof, D. R. (1997). *Programming with POSIX Threads*. Addison-Wesley Professional. ISBN: 978-0201633924.

5. Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. *Proceedings of the AFIPS Spring Joint Computer Conference*, 30, 483-485.

6. Gustafson, J. L. (1988). Reevaluating Amdahl's law. *Communications of the ACM*, 31(5), 532-533.

7. OpenMP Architecture Review Board. (2024). *OpenMP Application Programming Interface Version 5.2*. Recuperado de https://www.openmp.org/specifications/

8. Strassen, V. (1969). Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4), 354-356.