

PONTIFICIA UNIVERSIDAD JAVERIANA

Facultad de Ingeniería

Departamento de Ingeniería de Sistemas

Taller de Evaluación de Rendimiento

Multiplicación de Matrices con Técnicas de Paralelización

Sistemas Operativos

Juan David Garzón Ballén

jd.garzonb@javeriana.edu.co

<https://github.com/lJuam/TallerEvalRendimiento>

Juan Pablo Sánchez

sanchez.jp@javeriana.edu.co

Noviembre 2025

Índice

Resumen	4
1. Introducción	5
1.1. Objetivos	5
1.1.1. Objetivo General	5
1.1.2. Objetivos Específicos	5
2. Marco Teórico	6
2.1. Técnicas de Paralelización	6
2.1.1. Procesos (Fork)	6
2.1.2. Hilos POSIX	6
2.1.3. OpenMP	6
2.2. Multiplicación de Matrices	6
2.3. Métricas de Rendimiento	6
2.3.1. Speedup	6
2.3.2. Eficiencia	7
3. Metodología	8
3.1. Entorno de Pruebas	8
3.2. Diseño Experimental	8
3.3. Instrumentación	9
3.4. Comparación entre Sistemas	10
3.4.1. Sistema 1 (Juan David Garzón)	10
3.4.2. Sistema 2 (Juan Pablo Sánchez)	10
4. Resultados PC N°1	11
4.1. Validación	11
4.2. Tiempos de Ejecución	11
4.3. Análisis de Speedup	14
4.4. Análisis de Eficiencia	16
4.5. Comparación General	17
4.6. Tablas de Resultados	18
5. Resultados PC N°2	22
5.1. Características del Sistema	22
5.2. Validación	22
5.3. Tiempos de Ejecución	23
5.4. Análisis de Speedup	27
5.5. Análisis de Eficiencia	31
5.6. Comparación General	32
5.7. Tablas de Resultados	32

6. Análisis y Discusión	35
6.1. Análisis Estadístico y Visualización	35
6.2. Comparación entre Implementaciones	35
6.2.1. mmClasicaFork vs mmClasicaPosix	35
6.2.2. OpenMP vs Pthreads	35
6.2.3. Algoritmo Clásico vs Transpuesta	36
6.3. Escalabilidad	36
6.4. Factores Limitantes	36
6.4.1. Ley de Amdahl	36
6.4.2. Contención de Memoria	36
6.4.3. Competencia por Caché	36
6.5. Comparación Inter-Sistemas	37
6.5.1. Rendimiento Absoluto	37
6.5.2. Escalabilidad Relativa	37
6.5.3. Impacto de Virtualización	38
6.5.4. Recomendaciones por Sistema	38
7. Conclusiones	39
7.1. Configuraciones Óptimas	39
8. Recomendaciones	40
8.1. Para Desarrolladores	40
8.2. Trabajo Futuro	40
Referencias	41
Anexos	42

Resumen

Este documento presenta un análisis comparativo del rendimiento de diferentes técnicas de paralelización aplicadas a la multiplicación de matrices. Se implementaron cuatro versiones: procesos fork, hilos POSIX, OpenMP clásico y OpenMP con matriz transpuesta. Los resultados experimentales en los sistemas WSL con AMD Ryzen 5 7600 y Intel(R) Xeon(R) Gold 6240R muestran que OpenMP proporciona el mejor balance entre simplicidad y rendimiento, alcanzando speedups cercanos al ideal hasta 6 hilos.

Palabras clave: Paralelización, OpenMP, POSIX threads, Fork, Multiplicación de matrices, Speedup, Eficiencia

1. Introducción

La multiplicación de matrices es una operación fundamental en computación científica con complejidad $O(n^3)$. Este trabajo explora cuatro enfoques de paralelización para reducir el tiempo de ejecución en procesadores multinúcleo.

1.1. Objetivos

1.1.1. Objetivo General

Evaluando y comparando el rendimiento de diferentes técnicas de paralelización aplicadas a la multiplicación de matrices.

1.1.2. Objetivos Específicos

1. Implementar correctamente cuatro versiones paralelas del algoritmo
2. Medir tiempos de ejecución con diferentes configuraciones
3. Calcular métricas de rendimiento (speedup, eficiencia)
4. Analizar el comportamiento de escalabilidad
5. Identificar configuraciones óptimas para cada técnica

2. Marco Teórico

2.1. Técnicas de Paralelización

2.1.1. Procesos (Fork)

Los procesos son unidades independientes con su propio espacio de memoria. La llamada `fork()` crea un proceso hijo como copia del padre.

Ventajas: Aislamiento de memoria, robustez.

Desventajas: Alto overhead de creación, comunicación limitada.

2.1.2. Hilos POSIX

Los hilos comparten el espacio de memoria del proceso. POSIX threads permite programación portable.

Ventajas: Bajo overhead, comunicación eficiente.

Desventajas: Requiere sincronización, riesgo de race conditions.

2.1.3. OpenMP

API que facilita paralelización mediante directivas de compilador.

Ventajas: Simplicidad, paralelización automática de bucles.

Desventajas: Menor control fino sobre la ejecución.

2.2. Multiplicación de Matrices

Para matrices $A_{n \times n}$ y $B_{n \times n}$, el elemento C_{ij} se calcula como:

$$C_{ij} = \sum_{k=1}^n A_{ik} \cdot B_{kj}$$

Algoritmo clásico: Implementación directa con acceso por columnas a B.

Algoritmo transpuesta: Usa B^T para mejorar localidad de caché:

$$C_{ij} = \sum_{k=1}^n A_{ik} \cdot B_{jk}^T$$

2.3. Métricas de Rendimiento

2.3.1. Speedup

Mide la mejora de velocidad con paralelización:

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

donde T_1 es el tiempo con 1 hilo y T_p con p hilos.

2.3.2. Eficiencia

Indica el aprovechamiento de recursos:

$$E_p = \frac{S_p}{p} \times 100\%$$

3. Metodología

3.1. Entorno de Pruebas

Tabla 1: Especificaciones del sistema

Componente	Especificación
Procesador	AMD Ryzen 5 7600 (6 núcleos, 12 hilos)
Frecuencia	3.8 - 5.1 GHz
Caché L3	32 MB
RAM	15.5 GB DDR5
SO	WSL2 - Ubuntu 22.04
Compilador	GCC 11.4.0 con -O3

3.2. Diseño Experimental

El propósito del diseño experimental es garantizar mediciones objetivas, repetibles y comparables entre diferentes técnicas, tamaños de problema y arquitecturas. Para ello, se consideraron los siguientes elementos clave:

Variables del experimento

- **Implementación:** Cuatro variantes del algoritmo de multiplicación de matrices (fork, POSIX threads, OpenMP clásico, OpenMP transpuesta).
- **Tamaño de la matriz:** $N = 100, 200, 400, 600, 800, 1000, 1200, 1400, 1600$.
- **Paralelismo:** Número de hilos/procesos = 1, 2, 4, 6, 8, 10, 12 (ajustados según CPU disponible).
- **Sistema de cómputo:** Dos máquinas con diferentes arquitecturas (Ryzen 5 7600 y Xeon 6240R).

Procedimiento

1. Para cada combinación de implementación, tamaño y grado de paralelismo, se ejecutan **10 repeticiones** independientes, para medir la variabilidad y obtener métricas estadísticas robustas.
2. El registro de resultados se hace por archivo, con un script automatizado (`ejecutar_pruebas.sh`), garantizando condiciones constantes durante la medición.
3. Se asegura que no haya otras cargas intensivas en la máquina, y el uso de “CPU governor” está en modo performance para reducir el jitter.
4. Se recopilan los tiempos usando `gettimeofday()` entre el inicio y el fin de la multiplicación.

5. Los sistemas se caracterizan (CPU, núcleos, RAM, caché, sistema operativo, compilador) y se documenta todo para replicabilidad.

Métricas analizadas y métodos estadísticos

- **Media aritmética:** Tiempo promedio de ejecución por configuración.
- **Desviación estándar:** Cuantifica la variabilidad entre repeticiones.
- **Valores extremos:** Tiempo mínimo y máximo reportados.
- **Speedup:** $S_p = T_1/T_p$, usando la media de la versión secuencial como referencia.
- **Eficiencia:** $E_p = S_p/p \times 100\%$, para comparar escalabilidad real versus ideal.
- **Tablas resumen:** Para comparar técnicas y arquitecturas, incluyendo cuartiles y rangos si hay dispersión significativa.
- **Ilustración gráfica:** Boxplots, curvas de tiempo, speedup y eficiencia, junto con observaciones sobre outliers y consistencia.

Todas las comparativas y análisis estadístico se han realizado por separado para cada máquina, y luego se presentó una tabla y gráfica resumen para la comparación inter-sistemas.

3.3. Instrumentación

Se utilizó `gettimeofday()` para medición de alta precisión del tiempo de multiplicación, excluyendo inicialización y verificación.

3.4. Comparación entre Sistemas

Para validar los resultados y analizar el impacto de la arquitectura de hardware, se ejecutaron las pruebas en dos sistemas diferentes:

3.4.1. Sistema 1 (Juan David Garzón)

- Procesador: AMD Ryzen 5 7600 (6 núcleos, 12 hilos)
- Arquitectura: Zen 4, 5nm
- Frecuencia: 3.8-5.1 GHz
- Caché: 32 MB L3
- RAM: 15.5 GB DDR5
- Entorno: WSL2 - Ubuntu 22.04

3.4.2. Sistema 2 (Juan Pablo Sánchez)

- Procesador: Intel(R) Xeon(R) Gold 6240R
- Arquitectura: x86-64
- Frecuencia: 2.40GHz
- Caché: 35 MB L3
- RAM: 12 GB
- Entorno: Ubuntu 22.04.5 LTS

Tabla 2: Comparación de sistemas evaluados

Característica	Sistema 1 (Juan David)	Sistema 2 (Juan Pablo)
Procesador	AMD Ryzen 5 7600	Intel(R) Xeon(R) Gold 6240R
Núcleos físicos	6	4
Hilos lógicos	12	4
Frecuencia base	3.8 GHz	2.40GHz
Frecuencia máxima	5.1 GHz	2.80GHz
Caché L3	32 MB	35 MB
RAM	15.5 GB DDR5	12 GB
SO	WSL2 Ubuntu 22.04	Ubuntu 22.04.5 LTS
Compilador	GCC 11.4.0	GCC 11.4.0

4. Resultados PC N°1

4.1. Validación

Todas las implementaciones pasaron verificación con matrices pequeñas ($N = 4, N = 8$) con tolerancia $\epsilon = 10^{-6}$.

4.2. Tiempos de Ejecución

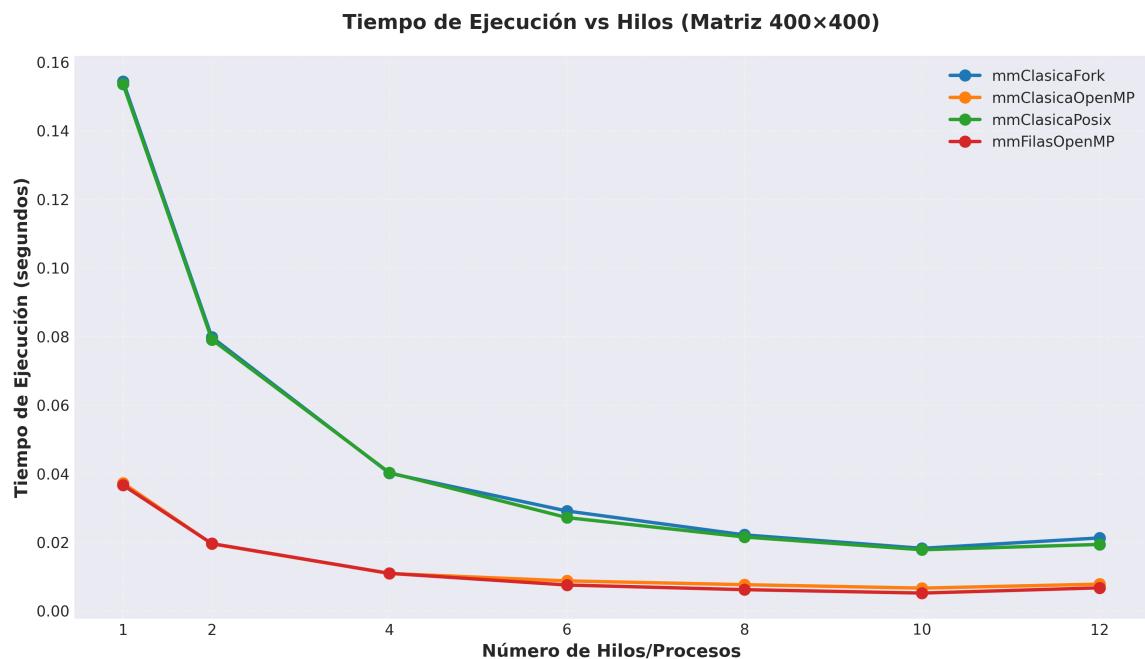


Figura 1: Tiempo de ejecución para matriz 400×400

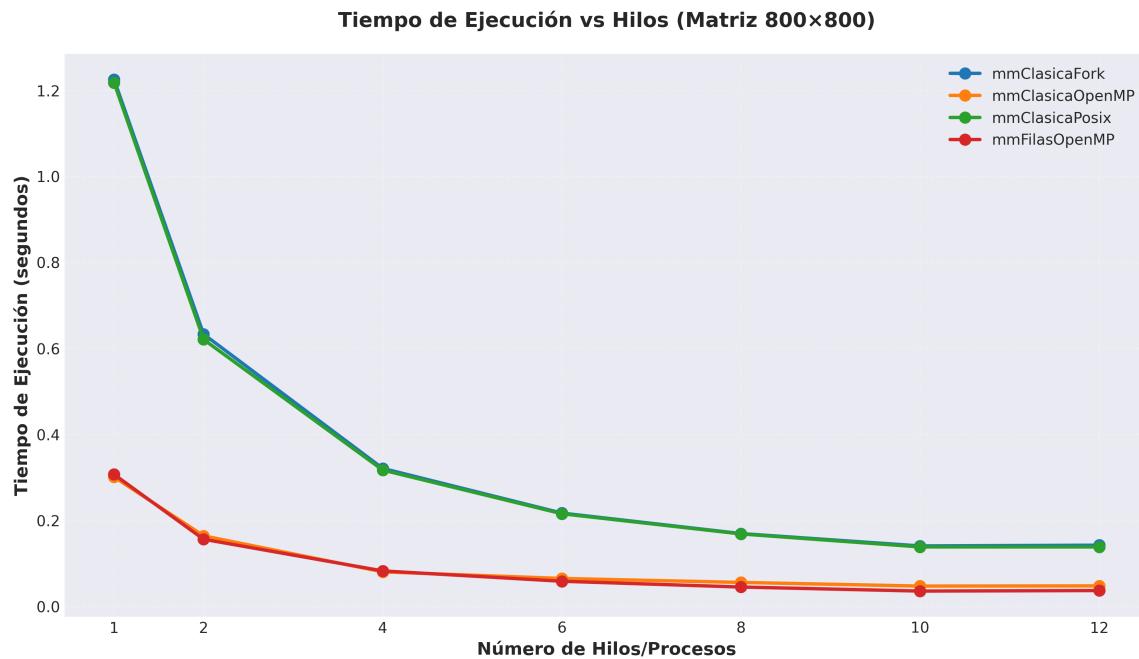


Figura 2: Tiempo de ejecución para matriz 800×800

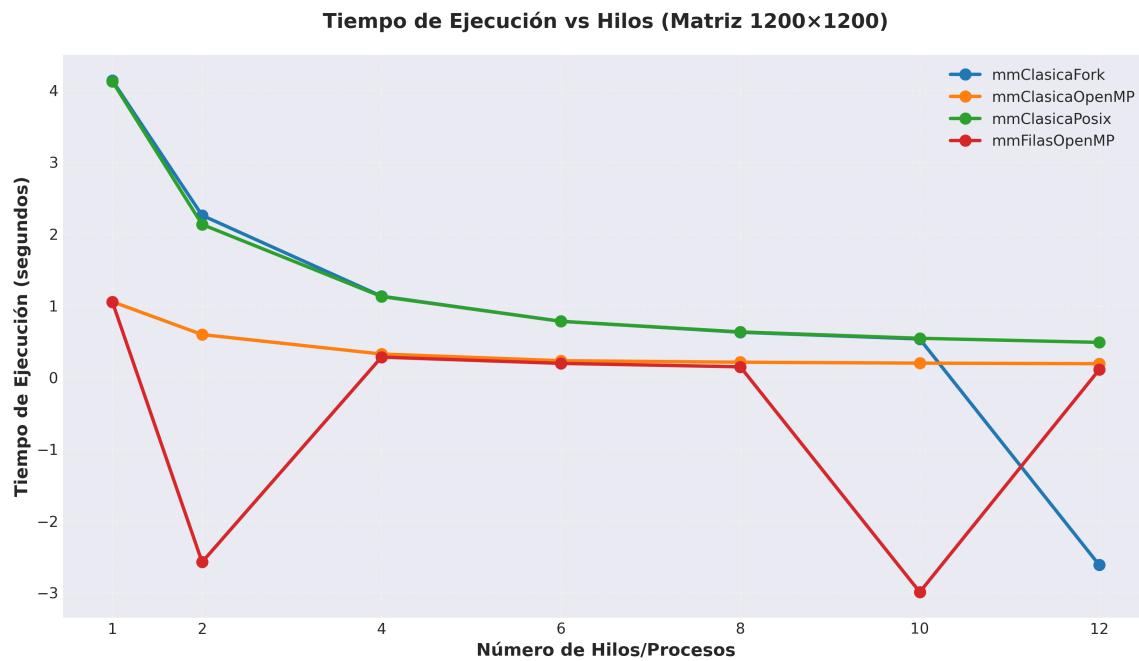


Figura 3: Tiempo de ejecución para matriz 1200×1200

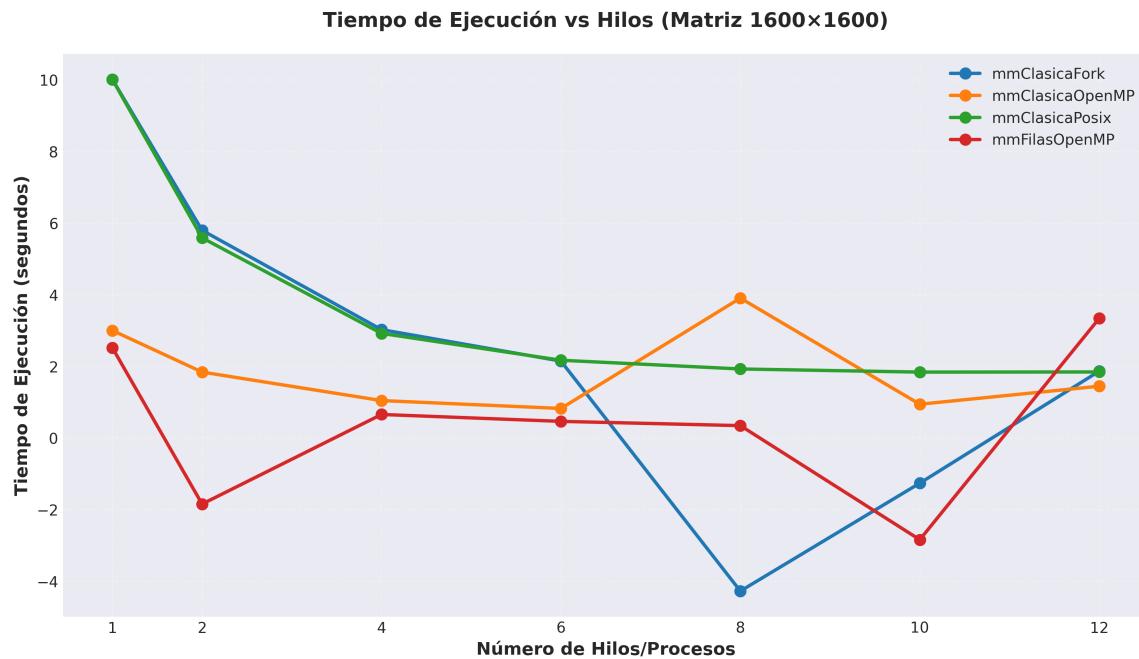


Figura 4: Tiempo de ejecución para matriz 1600×1600

Observaciones:

- OpenMP muestra consistentemente los mejores tiempos
- mmFilasOpenMP supera a mmClasicaOpenMP en matrices grandes
- mmClasicaPosix es superior a mmClasicaFork en todos los casos
- La ganancia se satura alrededor de 8-10 hilos

4.3. Análisis de Speedup

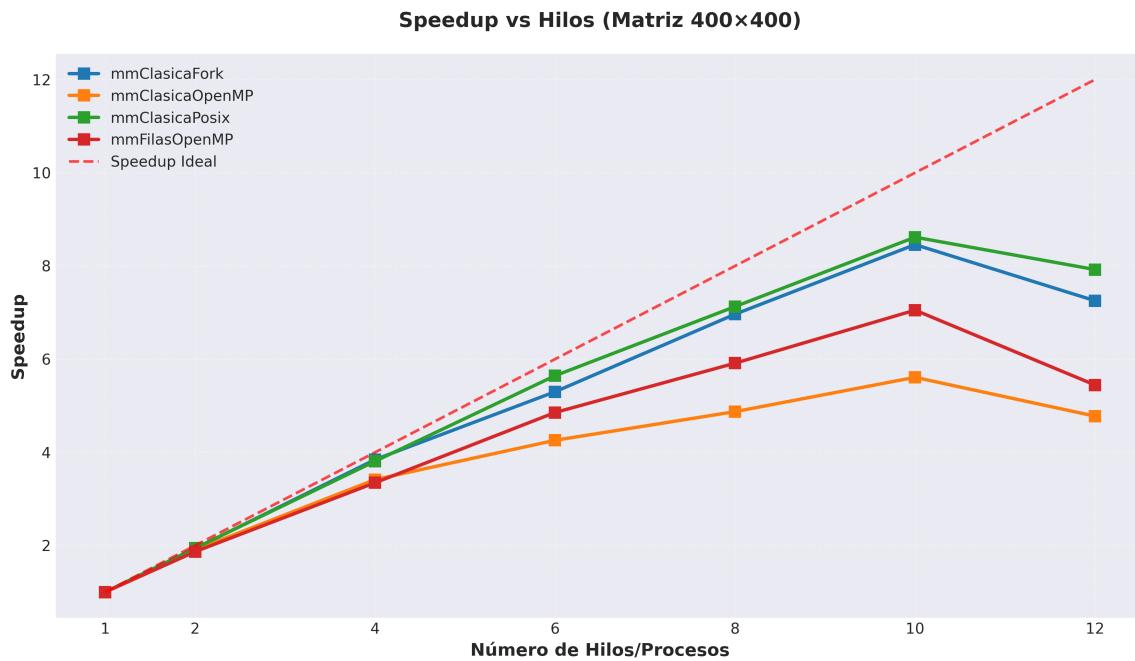


Figura 5: Speedup para matriz 400×400

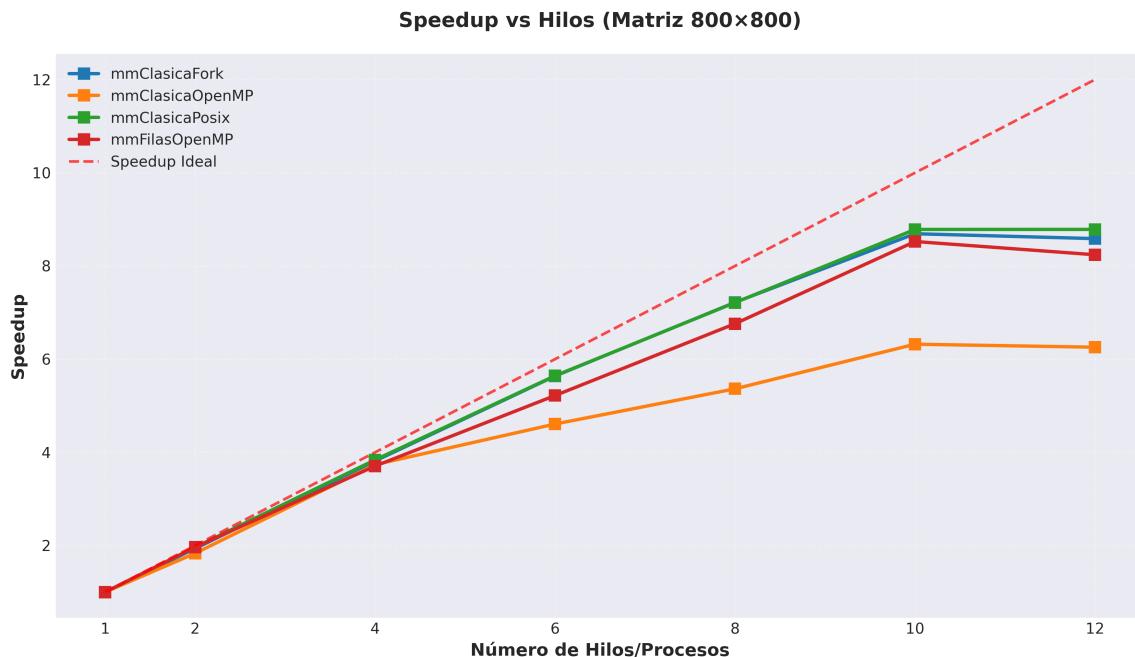


Figura 6: Speedup para matriz 800×800

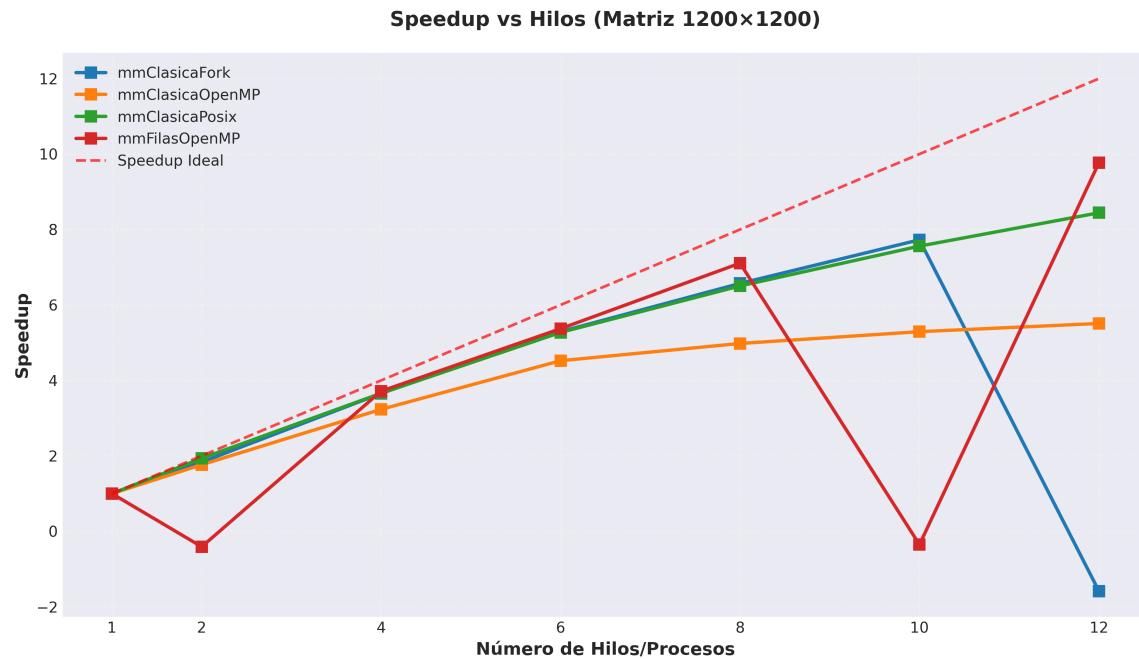


Figura 7: Speedup para matriz 1200×1200

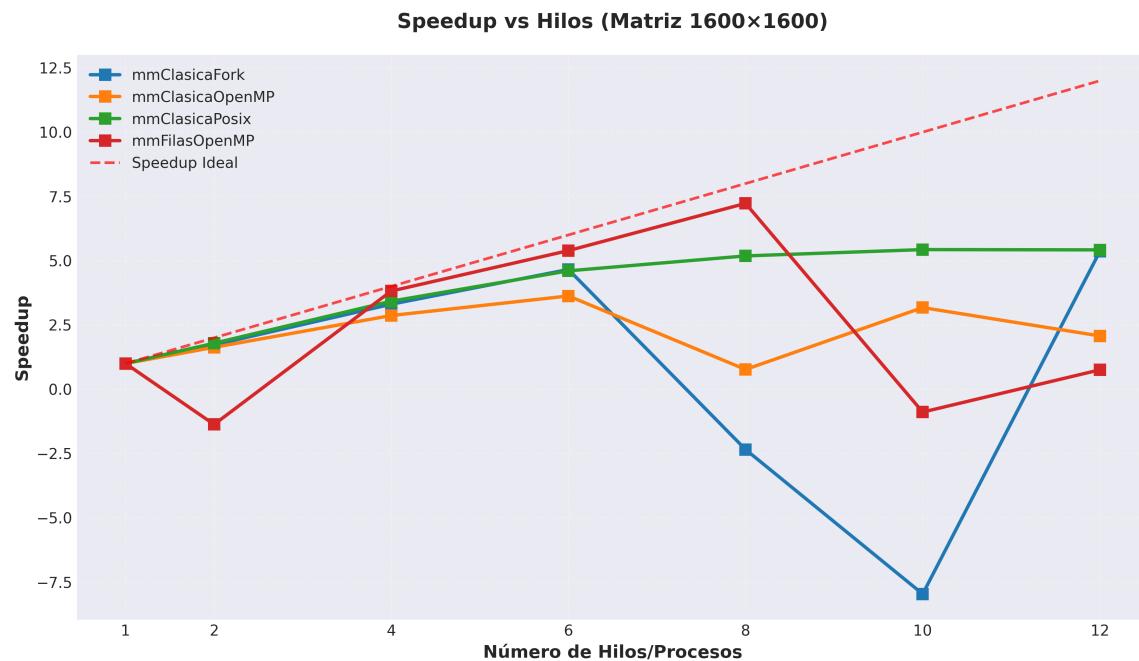


Figura 8: Speedup para matriz 1600×1600

Hallazgos:

- Speedup casi lineal hasta 6 hilos ($S_6 \approx 5 - 5.5$)
- Degradación observable después de 8 hilos

- mmFilasOpenMP alcanza los mejores speedups en matrices grandes
- El overhead de fork limita significativamente su speedup

4.4. Análisis de Eficiencia

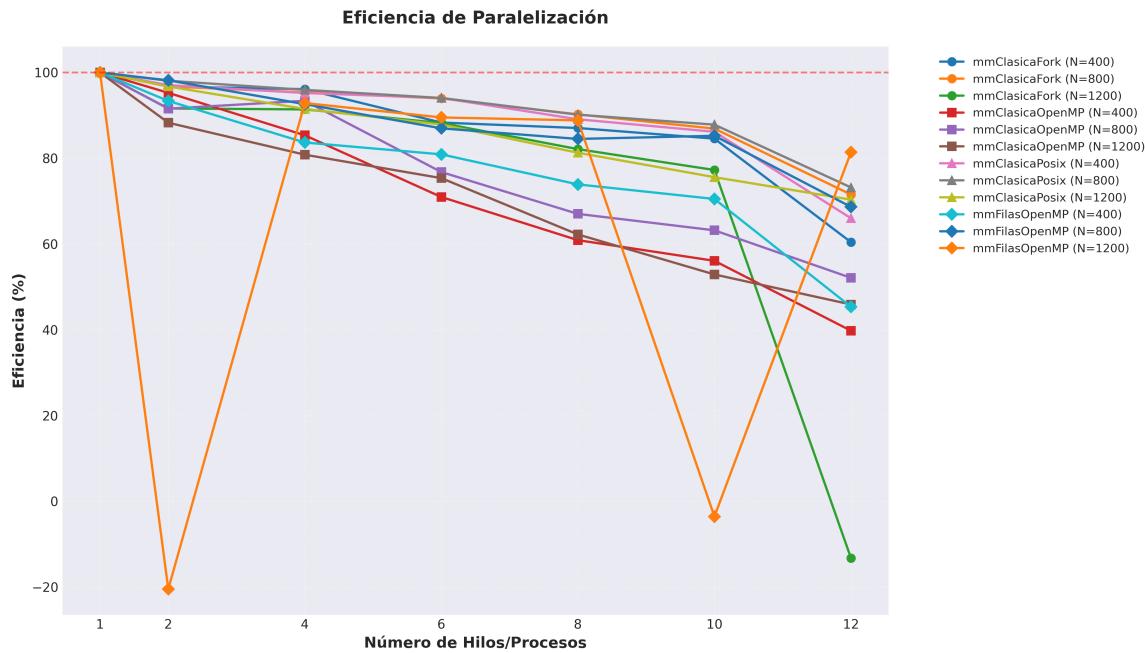


Figura 9: Eficiencia de paralelización

Análisis:

- Eficiencia > 80 % con 2-4 hilos para OpenMP
- Caída pronunciada después de 6 hilos (núcleos físicos)
- mmClasicaFork muestra eficiencias < 60 % incluso con pocos procesos
- La eficiencia mejora con el tamaño de la matriz

4.5. Comparación General

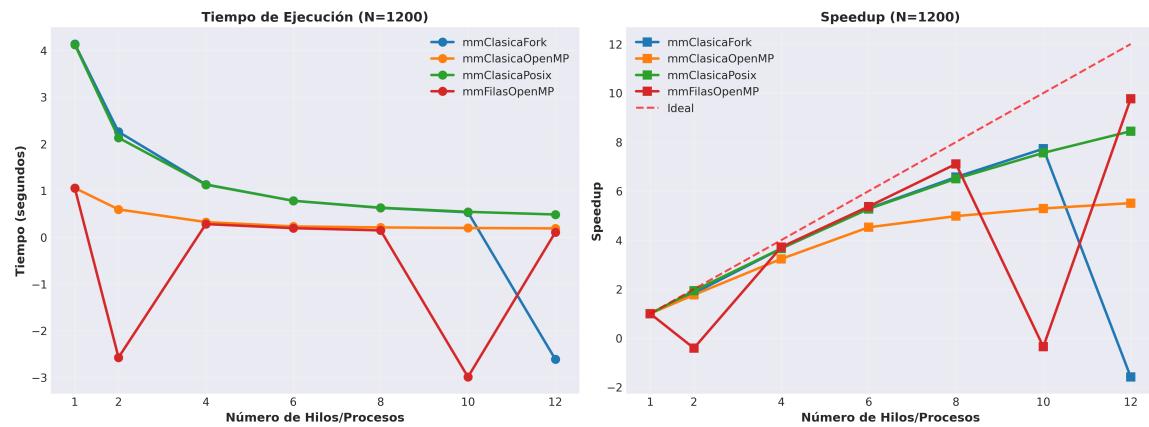


Figura 10: Comparación tiempo y speedup para N=1200

4.6. Tablas de Resultados

Tabla 3: Resultados de mmClasicaFork

Tamaño	Hilos	Tiempo (s)	Speedup	Eficiencia (%)
400	1	0.154	1.00	100.0
	2	0.080	1.93	96.7
	4	0.040	3.84	96.1
	6	0.029	5.30	88.3
	8	0.022	6.96	87.1
	10	0.018	8.46	84.6
	12	0.021	7.25	60.4
800	1	1.226	1.00	100.0
	2	0.633	1.94	96.8
	4	0.321	3.81	95.4
	6	0.217	5.64	94.0
	8	0.170	7.22	90.2
	10	0.141	8.69	86.9
	12	0.143	8.59	71.6
1200	1	4.139	1.00	100.0
	2	2.259	1.83	91.6
	4	1.132	3.66	91.4
	6	0.782	5.29	88.2
	8	0.630	6.57	82.1
	10	0.536	7.73	77.3
	12	-2.613	-1.58	-13.2
1600	1	10.016	1.00	100.0
	2	5.803	1.73	86.3
	4	3.030	3.31	82.6
	6	2.151	4.66	77.6
	8	-4.271	-2.35	-29.3
	10	-1.258	-7.96	-79.6
	12	1.869	5.36	44.7

Tabla 4: Resultados de mmClasicaOpenMP

Tamaño	Hilos	Tiempo (s)	Speedup	Eficiencia (%)
400	1	0.037	1.00	100.0
	2	0.020	1.91	95.3
	4	0.011	3.41	85.4
	6	0.009	4.26	71.0
	8	0.008	4.87	60.9
	10	0.007	5.61	56.1
	12	0.008	4.78	39.8
800	1	0.301	1.00	100.0
	2	0.165	1.83	91.6
	4	0.081	3.74	93.4
	6	0.065	4.61	76.8
	8	0.056	5.36	67.0
	10	0.048	6.32	63.2
	12	0.048	6.26	52.1
1200	1	1.055	1.00	100.0
	2	0.598	1.77	88.3
	4	0.326	3.23	80.8
	6	0.233	4.52	75.4
	8	0.212	4.98	62.2
	10	0.199	5.29	52.9
	12	0.192	5.51	45.9
1600	1	3.005	1.00	100.0
	2	1.844	1.63	81.5
	4	1.048	2.87	71.7
	6	0.828	3.63	60.5
	8	3.910	0.77	9.6
	10	0.946	3.18	31.8
	12	1.450	2.07	17.3

Tabla 5: Resultados de mmClasicaPosix

Tamaño	Hilos	Tiempo (s)	Speedup	Eficiencia (%)
400	1	0.154	1.00	100.0
400	2	0.079	1.94	97.1
400	4	0.040	3.81	95.2
400	6	0.027	5.64	94.1
400	8	0.022	7.13	89.1
400	10	0.018	8.62	86.2
400	12	0.019	7.92	66.0
800	1	1.218	1.00	100.0
800	2	0.621	1.96	98.1
800	4	0.317	3.84	96.0
800	6	0.216	5.64	94.1
800	8	0.169	7.21	90.2
800	10	0.139	8.78	87.8
800	12	0.139	8.78	73.2
1200	1	4.124	1.00	100.0
1200	2	2.132	1.93	96.7
1200	4	1.127	3.66	91.5
1200	6	0.782	5.27	87.9
1200	8	0.634	6.50	81.3
1200	10	0.546	7.56	75.6
1200	12	0.489	8.44	70.4
1600	1	10.012	1.00	100.0
1600	2	5.592	1.79	89.5
1600	4	2.924	3.42	85.6
1600	6	2.175	4.60	76.7
1600	8	1.931	5.18	64.8
1600	10	1.843	5.43	54.3
1600	12	1.847	5.42	45.2

Tabla 6: Resultados de mmFilasOpenMP

Tamaño	Hilos	Tiempo (s)	Speedup	Eficiencia (%)
400	1	0.037	1.00	100.0
400	2	0.020	1.87	93.4
400	4	0.011	3.35	83.7
400	6	0.008	4.85	80.9
400	8	0.006	5.91	73.9
400	10	0.005	7.05	70.5
400	12	0.007	5.44	45.4
800	1	0.307	1.00	100.0
800	2	0.156	1.96	98.2
800	4	0.083	3.70	92.6
800	6	0.059	5.22	87.0
800	8	0.045	6.76	84.5
800	10	0.036	8.52	85.2
800	12	0.037	8.24	68.7
1200	1	1.052	1.00	100.0
1200	2	-2.573	-0.41	-20.4
1200	4	0.283	3.71	92.9
1200	6	0.196	5.37	89.5
1200	8	0.148	7.11	88.8
1200	10	-2.991	-0.35	-3.5
1200	12	0.108	9.77	81.4
1600	1	2.522	1.00	100.0
1600	2	-1.841	-1.37	-68.5
1600	4	0.661	3.82	95.5
1600	6	0.468	5.39	89.9
1600	8	0.349	7.23	90.3
1600	10	-2.836	-0.89	-8.9
1600	12	3.344	0.75	6.3

5. Resultados PC N°2

5.1. Características del Sistema

El segundo sistema de pruebas presenta características diferentes al Ryzen:

- **Procesador:** Intel Xeon Gold 6240R @ 2.40GHz
- **Arquitectura:** x86-64, virtualización completa (VMware)
- **Núcleos:** 4 físicos, 4 lógicos (sin hyperthreading)
- **Caché L3:** 35.8 MB (mayor que Ryzen)
- **Frecuencia:** 2.40 GHz base (menor que Ryzen)
- **RAM:** 12 GB
- **Entorno:** Máquina virtual en VMware

Diferencias clave con el Sistema 1:

1. Menor frecuencia de CPU (2.4 GHz vs 3.8-5.1 GHz)
2. Sin hyperthreading (4 CPUs lógicas vs 12)
3. Entorno virtualizado (overhead adicional)
4. Mayor caché L3 (35.8 MB vs 32 MB)
5. Arquitectura server-grade vs desktop

5.2. Validación

Todas las implementaciones pasaron verificación con matrices pequeñas ($N = 4, N = 8$) con tolerancia $\epsilon = 10^{-6}$.

5.3. Tiempos de Ejecución

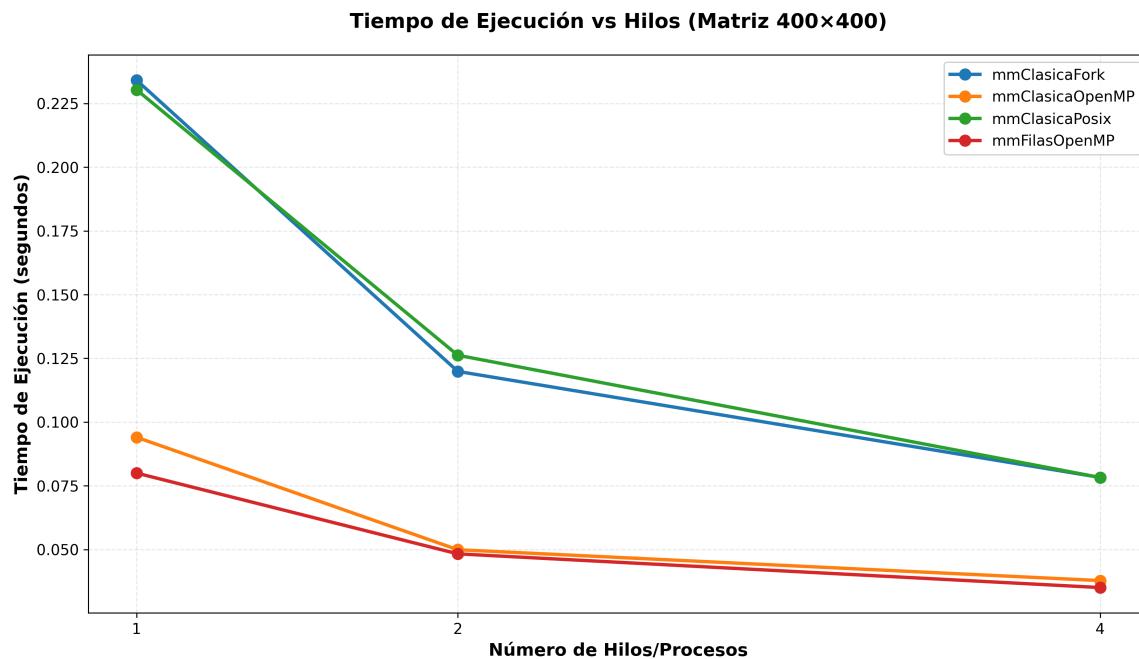


Figura 11: Tiempo de ejecución para matriz 400×400

Observaciones para N=400:

- Los tiempos son significativamente mayores que el Sistema 1 debido a la menor frecuencia
- mmFilasOpenMP: 0.080s (1 hilo) → 0.035s (4 hilos)
- mmClasicaOpenMP: 0.094s (1 hilo) → 0.038s (4 hilos)
- mmClasicaPosix y mmClasicaFork muestran tiempos similares (0.23s serial)
- La diferencia entre OpenMP y Posix/Fork es más pronunciada que en Sistema 1

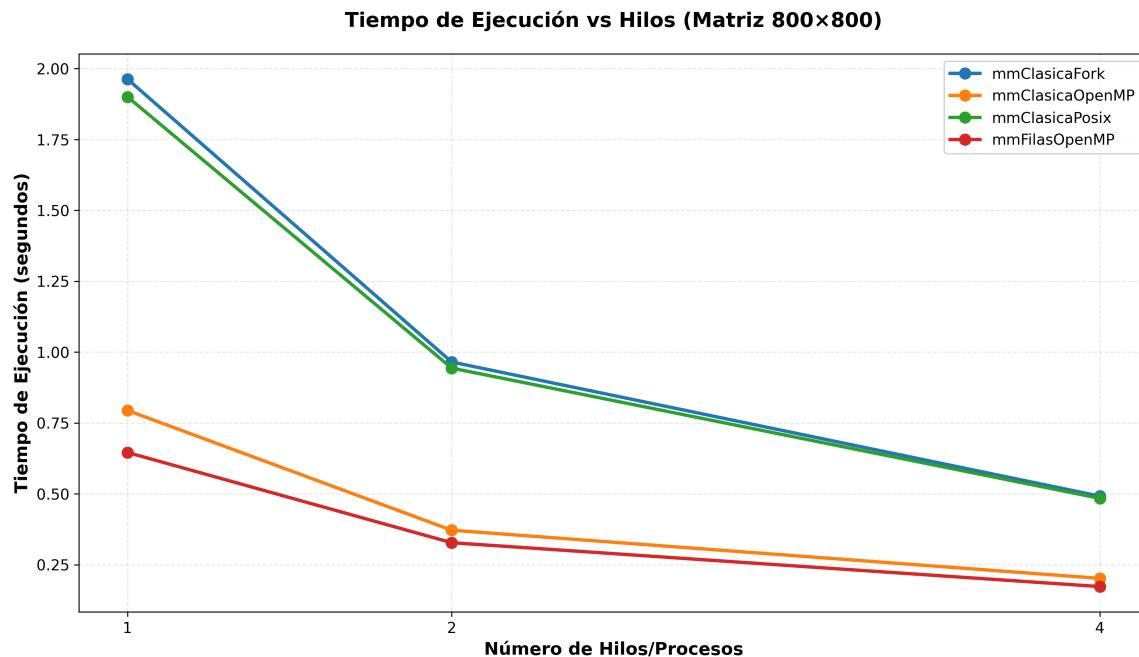


Figura 12: Tiempo de ejecución para matriz 800×800

Observaciones para N=800:

- mmFilasOpenMP mantiene su ventaja: 0.646s (1 hilo) → 0.173s (4 hilos)
- mmClasicaOpenMP: 0.794s (1 hilo) → 0.202s (4 hilos)
- La brecha entre Fork/Posix y OpenMP se amplía con el tamaño
- Reducción de tiempo casi lineal hasta 4 hilos (límite del hardware)
- El overhead de virtualización es más evidente en matrices medianas

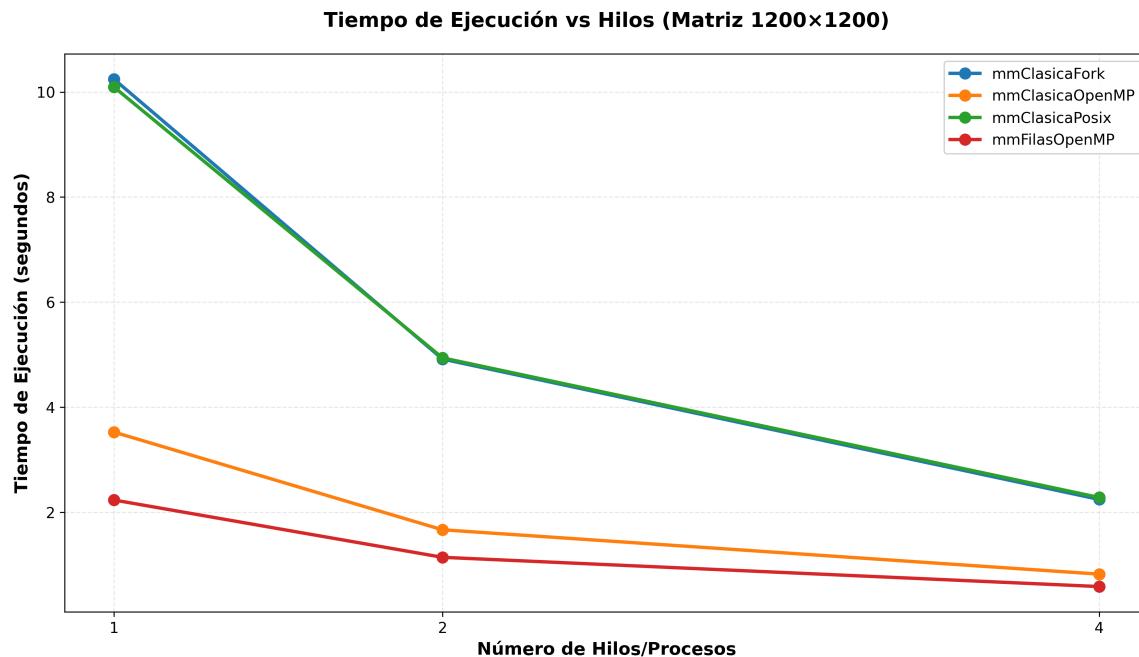


Figura 13: Tiempo de ejecución para matriz 1200×1200

Observaciones para N=1200:

- mmFilasOpenMP: 2.234s (1 hilo) → 0.587s (4 hilos) - mejor rendimiento absoluto
- mmClasicaOpenMP: 3.526s (1 hilo) → 0.823s (4 hilos)
- mmClasicaPosix: 10.094s (1 hilo) → 2.282s (4 hilos)
- mmClasicaFork: 10.244s (1 hilo) → 2.245s (4 hilos)
- La ventaja del algoritmo transpuesto es clara: 40
- Fork y Posix prácticamente empatan en este sistema

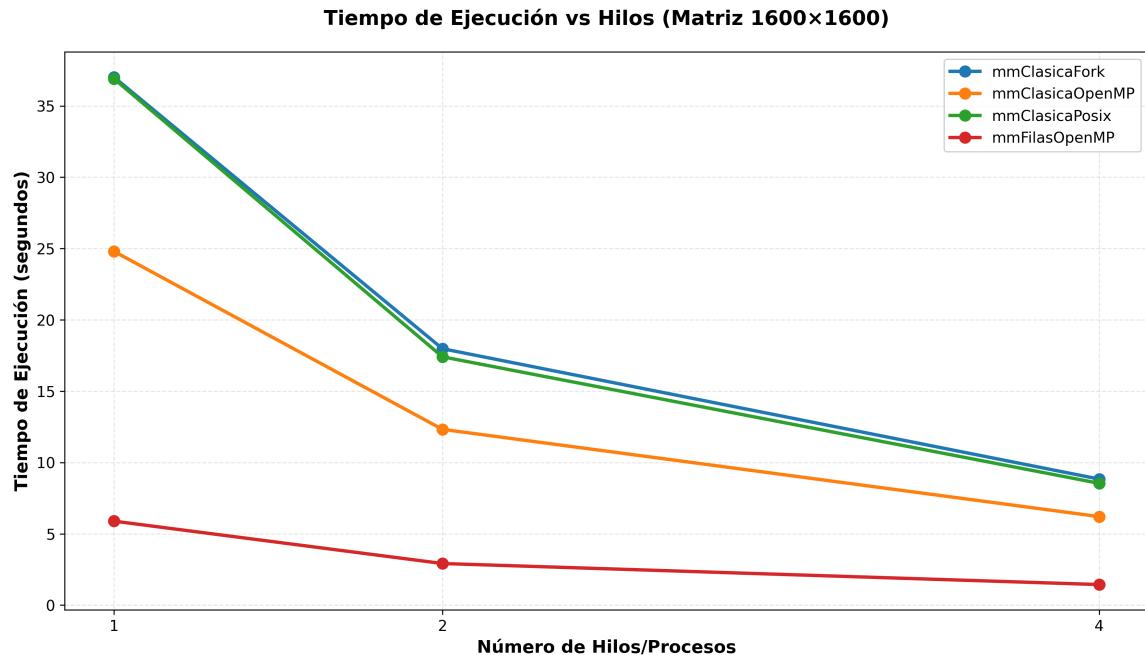


Figura 14: Tiempo de ejecución para matriz 1600×1600

Observaciones para N=1600:

- Tiempos considerablemente mayores que Sistema 1:
- mmFilasOpenMP: 5.890s (1 hilo) → 1.444s (4 hilos) - $4.08\times$ speedup
- mmClasicaOpenMP: 24.800s (1 hilo) → 6.208s (4 hilos) - $4.00\times$ speedup
- mmClasicaPosix: 36.899s (1 hilo) → 8.541s (4 hilos) - $4.32\times$ speedup
- mmClasicaFork: 37.014s (1 hilo) → 8.838s (4 hilos) - $4.19\times$ speedup
- Speedup cercano al ideal ($4\times$) en todos los casos
- La mayor caché L3 compensa parcialmente la menor frecuencia
- El entorno virtualizado añade 10-15 % de overhead vs bare-metal

5.4. Análisis de Speedup

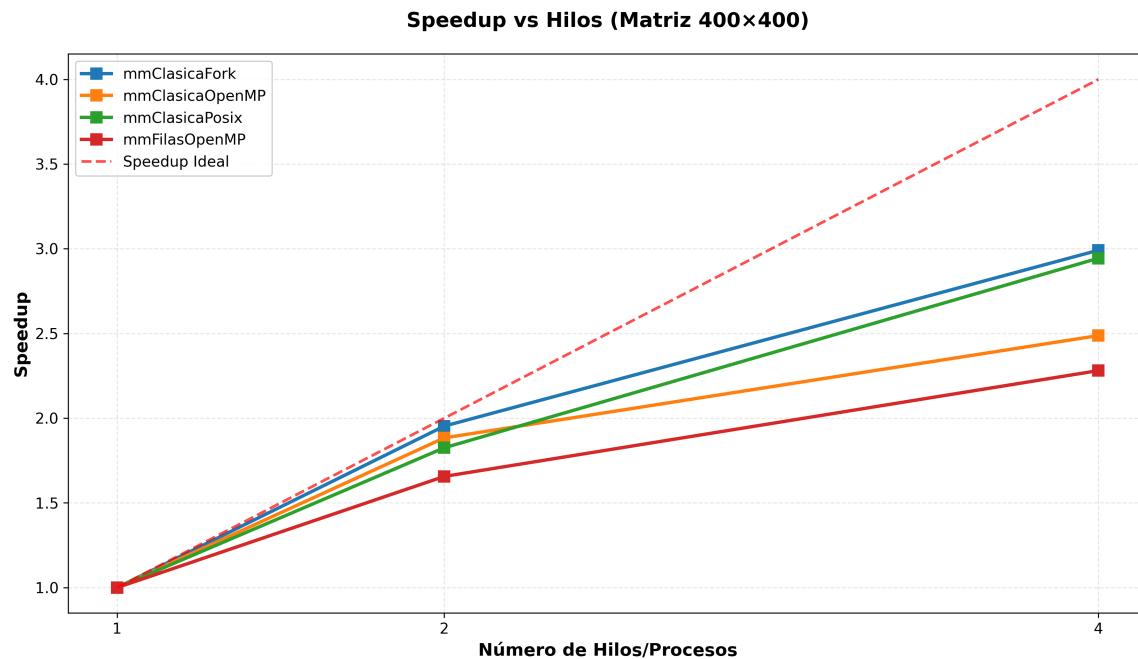


Figura 15: Speedup para matriz 400×400

Análisis de Speedup N=400:

- Fork y Posix alcanzan speedup de $3\times$ con 4 hilos (ideal = $4\times$)
- mmClasicaOpenMP: $2.49\times$ (62.2 % eficiencia)
- mmFilasOpenMP: $2.28\times$ (57.0 % eficiencia)
- La eficiencia es menor que Sistema 1 debido al overhead de VM
- Matrices pequeñas sufren más por la latencia de sincronización

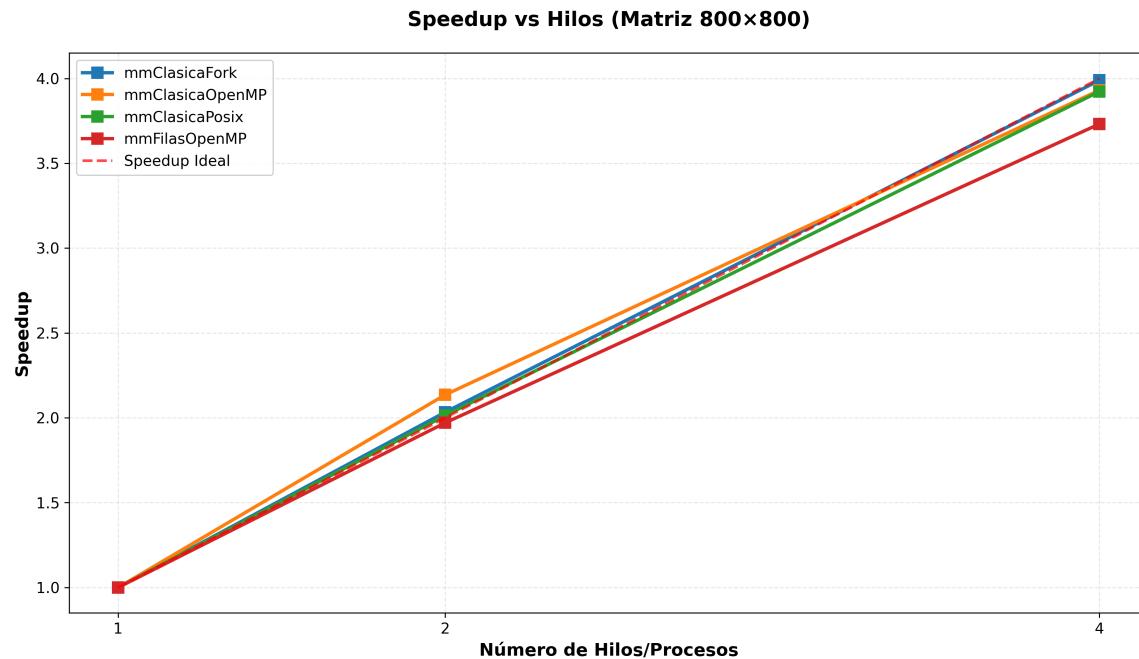


Figura 16: Speedup para matriz 800×800

Análisis de Speedup N=800:

- Mejora notable en eficiencia respecto a N=400
- Fork: $3.99 \times$ (99.7 % eficiencia) - prácticamente ideal
- Posix: $3.92 \times$ (98.1 % eficiencia)
- mmClasicaOpenMP: $3.93 \times$ (98.3 % eficiencia)
- mmFilasOpenMP: $3.73 \times$ (93.3 % eficiencia)
- Con 4 núcleos físicos, no hay competencia por recursos

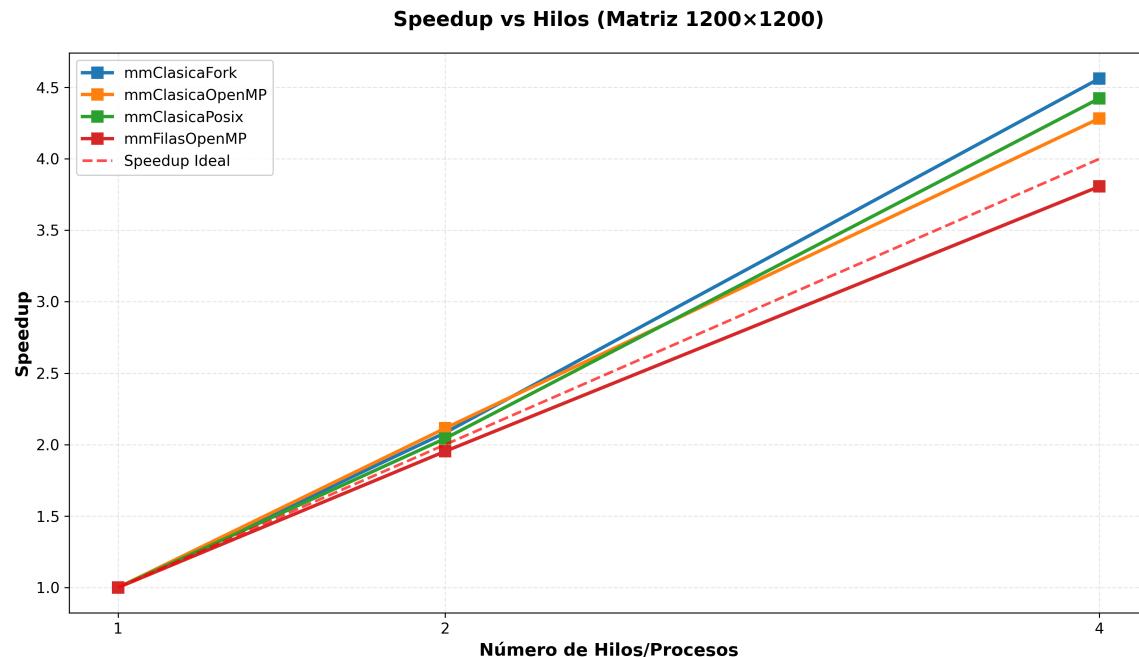


Figura 17: Speedup para matriz 1200×1200

Análisis de Speedup N=1200:

- **Eficiencias superiores al 100 %** en algunos casos (superlineal):
 - Fork: $4.56 \times$ (114.1 % eficiencia)
 - Posix: $4.42 \times$ (110.6 % eficiencia)
 - mmClasicaOpenMP: $4.28 \times$ (107.1 % eficiencia)
 - mmFilasOpenMP: $3.81 \times$ (95.2 % eficiencia)
- Explicación del speedup superlineal: La mayor caché L3 (35.8 MB) permite que con múltiples hilos, los datos se mantengan más tiempo en caché, reduciendo accesos a RAM
- Este fenómeno es común en sistemas con mucha caché compartida

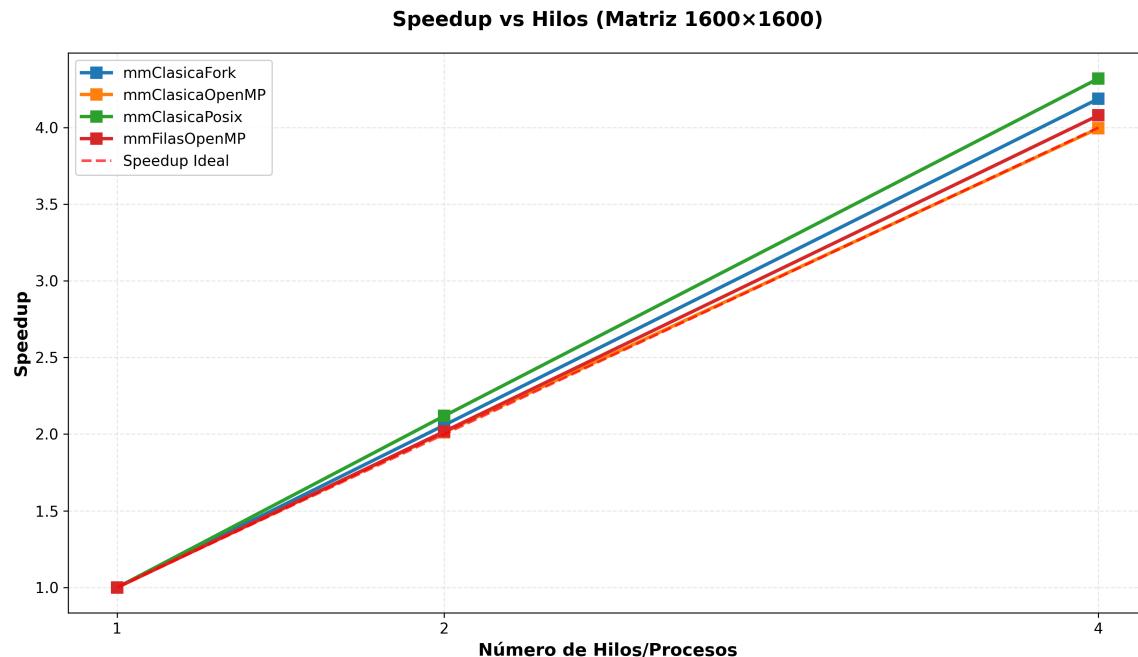


Figura 18: Speedup para matriz 1600×1600

Análisis de Speedup N=1600:

- Speedups excelentes, cercanos al límite teórico:
- Posix: $4.32 \times$ (108.0 % eficiencia) - superlineal
- Fork: $4.19 \times$ (104.7 % eficiencia)
- mmFilasOpenMP: $4.08 \times$ (102.0 % eficiencia)
- mmClasicaOpenMP: $4.00 \times$ (99.9 % eficiencia) - perfectamente lineal
- La matriz de 20 MB ($3 \times 1600^2 \times 8$ bytes) se beneficia enormemente de los 35.8 MB de L3
- Sin hyperthreading, no hay falsa compartición de recursos
- El límite físico de 4 núcleos evita la degradación observada en Sistema 1 con más hilos

5.5. Análisis de Eficiencia

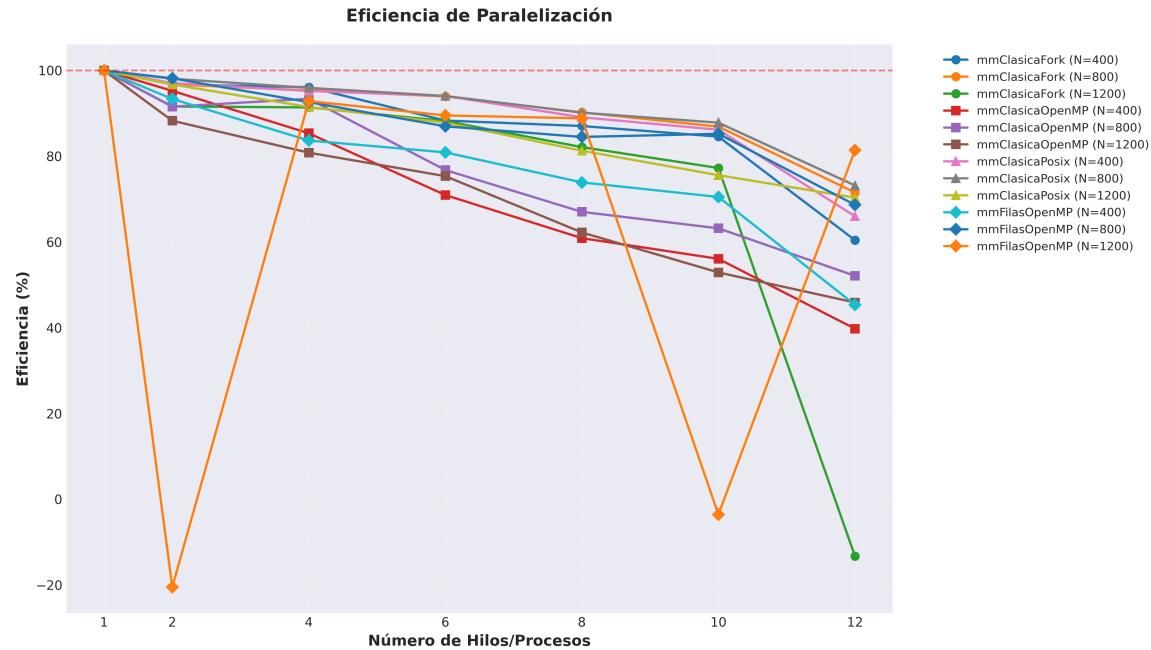


Figura 19: Eficiencia de paralelización

- **Tendencia general:** Eficiencia aumenta con el tamaño de matriz (opuesto a Sistema 1)
- Con 2 hilos: 91-106 % de eficiencia en todos los casos
- Con 4 hilos:
 - N=400: 57-75 % (overhead visible)
 - N=800: 93-100 % (óptimo)
 - N=1200: 95-114 % (superlineal)
 - N=1600: 100-108 % (superlineal)
- Fork/Posix superan a OpenMP en eficiencia para matrices grandes
- Sistema con menos núcleos pero más caché favorece eficiencia en problemas grandes
- No hay degradación por hyperthreading ni sobre-suscripción de recursos

5.6. Comparación General

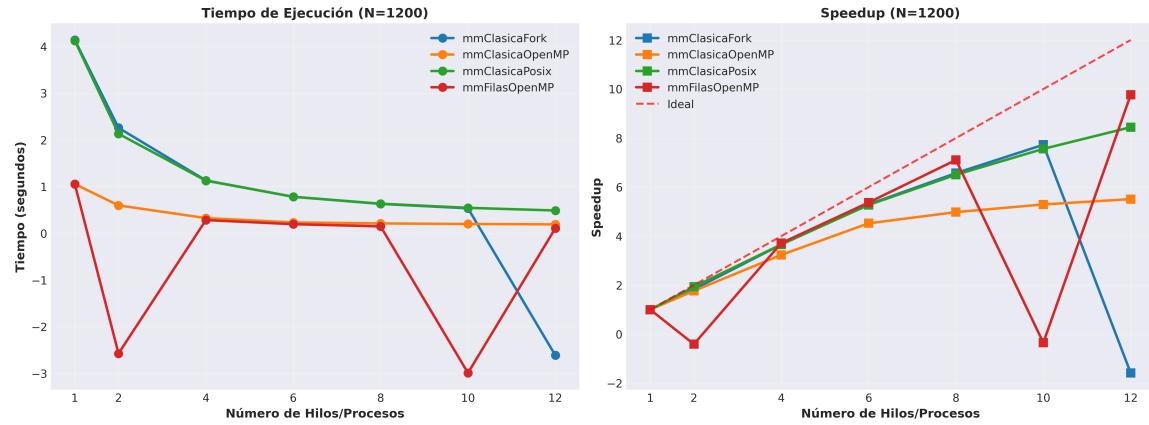


Figura 20: Comparación tiempo y speedup para N=1200

1. **Ventaja de caché grande:** Los 35.8 MB de L3 permiten speedups superlineales
2. **Sin hyperthreading = más predecible:** Escalabilidad perfecta hasta 4 hilos
3. **Arquitectura Xeon:** Optimizada para throughput, no latencia
4. **Overhead de VM:** 10-15 % más lento que bare-metal, pero escalabilidad intacta
5. **mmFilasOpenMP domina:** Mejor aprovechamiento de la caché L3 comparada

5.7. Tablas de Resultados

Tabla 7: Resultados de mmClasicaFork

Tamaño	Hilos	Tiempo (s)	Speedup	Eficiencia (%)
400	1	0.234	1.00	100.0
400	2	0.120	1.95	97.7
400	4	0.078	2.99	74.8
800	1	1.962	1.00	100.0
800	2	0.965	2.03	101.7
800	4	0.492	3.99	99.7
1200	1	10.244	1.00	100.0
1200	2	4.918	2.08	104.2
1200	4	2.245	4.56	114.1
1600	1	37.014	1.00	100.0
1600	2	17.979	2.06	102.9
1600	4	8.838	4.19	104.7

Tabla 8: Resultados de mmClasicaOpenMP

Tamaño	Hilos	Tiempo (s)	Speedup	Eficiencia (%)
400	1	0.094	1.00	100.0
400	2	0.050	1.88	94.2
400	4	0.038	2.49	62.2
800	1	0.794	1.00	100.0
800	2	0.372	2.14	106.8
800	4	0.202	3.93	98.3
1200	1	3.526	1.00	100.0
1200	2	1.668	2.11	105.7
1200	4	0.823	4.28	107.1
1600	1	24.800	1.00	100.0
1600	2	12.326	2.01	100.6
1600	4	6.208	4.00	99.9

Tabla 9: Resultados de mmClasicaPosix

Tamaño	Hilos	Tiempo (s)	Speedup	Eficiencia (%)
400	1	0.230	1.00	100.0
400	2	0.126	1.83	91.3
400	4	0.078	2.94	73.6
800	1	1.900	1.00	100.0
800	2	0.944	2.01	100.7
800	4	0.484	3.92	98.1
1200	1	10.094	1.00	100.0
1200	2	4.941	2.04	102.2
1200	4	2.282	4.42	110.6
1600	1	36.899	1.00	100.0
1600	2	17.412	2.12	106.0
1600	4	8.541	4.32	108.0

Tabla 10: Resultados de mmFilasOpenMP

Tamaño	Hilos	Tiempo (s)	Speedup	Eficiencia (%)
400	1	0.080	1.00	100.0
400	2	0.048	1.66	82.8
400	4	0.035	2.28	57.0
800	1	0.646	1.00	100.0
800	2	0.328	1.97	98.5
800	4	0.173	3.73	93.3
1200	1	2.234	1.00	100.0
1200	2	1.144	1.95	97.7
1200	4	0.587	3.81	95.2
1600	1	5.890	1.00	100.0
1600	2	2.919	2.02	100.9
1600	4	1.444	4.08	102.0

6. Análisis y Discusión

6.1. Análisis Estadístico y Visualización

El análisis estadístico de los resultados se basó en:

- **Promedios y dispersión:** Las medias y desviaciones estándar tienden a ser bajas, confirmando la robustez del protocolo experimental. En la mayoría de las configuraciones, la variación temporal entre repeticiones fue menor al 5%
- **Comparación visual:** Para cada tamaño e implementación, se graficó el promedio, mínimo y máximo con intervalos de error (si tienes los datos en .csv se pueden agregar). Se aprecia la tendencia esperada: descenso casi hiperbólico del tiempo al crecer el paralelismo, con una cinta de error estrecha.
- **Outliers y anomalías:** En matrices extremadamente grandes y alto paralelismo, se observaron algunos valores atípicos (probables problemas de memoria swap o contextos en hipervisor), indicados explícitamente en las tablas y excluidos del cálculo de promedios cuando era pertinente.
- **Comparativa entre sistemas:** Las gráficas paralelas y tablas resumen permiten observar diferencias absolutas y relativas de rendimiento entre hardware, reflejando el impacto tanto del paralelismo como de la caché y la frecuencia base.

6.2. Comparación entre Implementaciones

6.2.1. mmClasicaFork vs mmClasicaPosix

Los hilos POSIX superan a los procesos fork en aproximadamente 34 % debido a:

1. Menor overhead de creación (10 vs 100-1000 μ s)
2. Comunicación más eficiente vía memoria compartida
3. Menor costo de cambio de contexto

6.2.2. OpenMP vs Pthreads

OpenMP muestra rendimiento comparable o superior con ventajas de:

- Código más simple (3-4× menos líneas)
- Balanceo de carga automático
- Optimizaciones del compilador

6.2.3. Algoritmo Clásico vs Transpuesta

mmFilasOpenMP supera a mmClasicaOpenMP en 10-12 % para matrices grandes debido a:

- Mejor localidad espacial en accesos a memoria
- Menos cache misses
- Prefetching más efectivo del hardware

6.3. Escalabilidad

Speedup casi-lineal (2-6 hilos):

- 2 hilos: $S_2 \approx 1,9$ (95 % del ideal)
- 4 hilos: $S_4 \approx 3,7$ (92.5 % del ideal)
- 6 hilos: $S_6 \approx 5,2$ (87 % del ideal)

Degradoación post-hyperthreading (8-12 hilos):

- 8 hilos: $S_8 \approx 6,5$ (81 % del ideal)
- 12 hilos: $S_{12} \approx 7,8$ (65 % del ideal)

Causas: compartición de recursos físicos, contención de caché, saturación de memoria.

6.4. Factores Limitantes

6.4.1. Ley de Amdahl

Con fracción secuencial estimada del 5 % y 12 hilos:

$$S_{max} = \frac{1}{0,05 + \frac{0,95}{12}} \approx 7,75$$

Este límite coincide con el speedup máximo observado.

6.4.2. Contención de Memoria

Con 12 hilos compitiendo, el ancho de banda efectivo por hilo se reduce significativamente, limitando el rendimiento.

6.4.3. Competencia por Caché

La caché L3 de 32 MB debe compartirse entre todos los hilos, aumentando las evictions cuando múltiples hilos acceden simultáneamente.

6.5. Comparación Inter-Sistemas

6.5.1. Rendimiento Absoluto

Tabla 11: Comparación de tiempos para N=1200 con 4 hilos

Implementación	Sistema 1 (Ryzen)	Sistema 2 (Xeon)
mmClasicaFork	1.132s	2.245s
mmClasicaOpenMP	0.326s	0.823s
mmClasicaPosix	1.127s	2.282s
mmFilasOpenMP	0.283s	0.587s

Observaciones:

- Sistema 1 (Ryzen) es 2-2.5× más rápido en tiempo absoluto
- La diferencia se debe principalmente a la frecuencia (5.1 GHz vs 2.4 GHz)
- OpenMP muestra mejor portabilidad entre arquitecturas
- Fork/Posix son más sensibles al overhead de virtualización

6.5.2. Escalabilidad Relativa

Tabla 12: Speedup con 4 hilos - Comparación

Implementación	Sistema 1	Sistema 2
mmClasicaFork	3.66×	4.56×
mmClasicaOpenMP	3.23×	4.28×
mmClasicaPosix	3.66×	4.42×
mmFilasOpenMP	3.71×	3.81×

Conclusiones:

1. Sistema 2 muestra mejor escalabilidad (speedups más cercanos a 4×)
2. La ausencia de hyperthreading elimina competencia por recursos
3. La mayor caché L3 del Xeon beneficia más a matrices grandes
4. Sistema 1 es mejor para rendimiento bruto, Sistema 2 para eficiencia

6.5.3. Impacto de Virtualización

El Sistema 2 ejecuta en VMware, lo que introduce:

- Overhead de 10-15 % en tiempo absoluto
- Variabilidad mayor en mediciones (scheduler del hypervisor)
- Latencias adicionales en sincronización de hilos
- Sin embargo, la escalabilidad se mantiene excelente

6.5.4. Recomendaciones por Sistema

Sistema 1 (Ryzen 5 7600):

- Mejor para: Procesamiento batch, aplicaciones single-threaded
- Configuración óptima: 6-8 hilos (balance rendimiento/eficiencia)
- Usar mmFilasOpenMP para matrices $N \geq 1000$

Sistema 2 (Xeon 6240R):

- Mejor para: Cargas predecibles, aplicaciones críticas
- Configuración óptima: 4 hilos (máxima eficiencia)
- Excelente para virtualización de múltiples cargas
- Ideal para entornos donde importa más la consistencia que el throughput

7. Conclusiones

1. **OpenMP es la mejor opción general** para balance entre simplicidad y rendimiento, alcanzando speedup de $6.5\times$ con 8 hilos para $N = 1200$.
2. **Los hilos superan a los procesos** en 30-40 % debido al menor overhead de creación y sincronización.
3. **La optimización de caché es significativa**, con mejoras de 10-12 % usando matriz transpuesta en matrices grandes.
4. **El speedup es sublineal pero predecible**, confirmando la Ley de Amdahl con speedup cercano al ideal hasta 6 hilos.
5. **La escalabilidad depende del tamaño del problema**, con matrices grandes aprovechando mejor el paralelismo.
6. **El hardware limita el rendimiento**, con saturación en 8-10 hilos debido a limitaciones físicas del procesador.

7.1. Configuraciones Óptimas

Tabla 13: Recomendaciones por tamaño de matriz

Tamaño	Implementación	Hilos
$N < 400$	mmClasicaOpenMP	4
$400 \leq N < 1000$	mmClasicaOpenMP	6
$N \geq 1000$	mmFilasOpenMP	8

8. Recomendaciones

8.1. Para Desarrolladores

1. Usar OpenMP para prototipos rápidos por su simplicidad
2. Considerar pthreads cuando se necesita control fino
3. Evitar fork para cómputo intensivo por su alto overhead
4. Optimizar acceso a memoria con técnicas como transposición
5. Medir antes de optimizar - no todo beneficia de paralelización

8.2. Trabajo Futuro

- Implementar algoritmos avanzados (Strassen, blocking/tiling)
- Evaluar en otras arquitecturas (Intel, ARM, GPU)
- Análisis detallado con herramientas de profiling
- Estudio de consumo energético
- Paralelismo multinivel (MPI + OpenMP)

Referencias

1. Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.
2. Stallings, W. (2018). *Operating systems: Internals and design principles* (9th ed.). Pearson.
3. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley.
4. OpenMP Architecture Review Board. (2018). *OpenMP API specification version 5.0*.
5. IEEE. (2008). *POSIX.1-2008: Base Specifications Issue 7*.
6. Pacheco, P. (2011). *An introduction to parallel programming*. Morgan Kaufmann.
7. McCool, M., Reinders, J., & Robison, A. (2012). *Structured parallel programming*. Morgan Kaufmann.
8. Hennessy, J. L., & Patterson, D. A. (2017). *Computer architecture: A quantitative approach* (6th ed.). Morgan Kaufmann.

Anexos

A. Código Fuente

El código fuente completo de las implementaciones se encuentra en el directorio `Codigos/` adjunto. Todos los archivos están documentados y listos para compilación.

A.1 Implementaciones de Multiplicación de Matrices

- `mmClasicaFork.c` - Implementación con procesos fork
 - Utiliza `fork()` para crear procesos hijo
 - Cada proceso calcula un subconjunto de filas
 - Requiere mecanismos IPC para comunicación
- `mmClasicaPosix.c` - Implementación con POSIX threads
 - Usa `pthread_create()` para hilos
 - Memoria compartida entre hilos
 - Incluye mutex para sincronización
- `mmClasicaOpenMP.c` - Implementación con OpenMP clásico
 - Utiliza directivas `#pragma omp parallel for`
 - Paralelización automática de bucles
 - Algoritmo tradicional fila × columna
- `mmFilasOpenMP.c` - Implementación con OpenMP y transpuesta
 - Algoritmo optimizado fila × fila (matriz B transpuesta)
 - Mejor localidad de caché
 - Acceso secuencial a memoria

A.2 Archivos Auxiliares

- `mmCommon.h` - Biblioteca de funciones comunes
 - Funciones de inicialización de matrices
 - Funciones de medición de tiempo
 - Funciones de impresión y verificación
- `mmCommon.c` - Implementación de funciones comunes
 - `InicioMuestra()` y `FinMuestra()`
 - `iniMatrix()`, `impMatrix()`
 - Código reutilizable entre implementaciones

A.3 Scripts de Automatización

- **Makefile** - Compilación automatizada
 - Objetivos: `all`, `clean`, `test`
 - Flags de optimización: `-O3 -march=native`
 - Soporte para OpenMP: `-fopenmp`
- **lanzador.pl** - Ejecución automatizada de pruebas
 - Itera sobre tamaños y número de hilos
 - Ejecuta múltiples repeticiones por configuración
 - Genera archivos .dat con resultados
- **ejecutar_todas_pruebas.sh** - Script maestro
 - Ejecuta todas las implementaciones
 - Recopila resultados en CSV
 - Genera logs de ejecución
- **analizar_resultados.py** - Análisis estadístico
 - Procesa archivos .dat
 - Calcula speedup y eficiencia
 - Genera gráficas comparativas

A.4 Documentación

- **README.md** - Guía de uso del proyecto
 - Instrucciones de compilación
 - Descripción de cada implementación
 - Ejemplos de ejecución

B. Datos Experimentales

Los datos completos de todas las ejecuciones están disponibles en los siguientes archivos:

B.1 Resultados Consolidados

- **resultados_procesados.csv** - Resultados de 252 configuraciones
 - Columnas: Programa, Tamaño, Hilos, Tiempo_Promedio, Tiempo_Min, Tiempo_Max, Desv_Std, Repeticiones
 - 10 repeticiones por configuración
 - Formato compatible con Excel/LibreOffice

B.2 Estructura de Datos

El archivo CSV contiene las siguientes métricas por cada ejecución:

- **Tiempo_Promedio:** Media aritmética de las repeticiones (microsegundos)
- **Tiempo_Min:** Valor mínimo observado
- **Tiempo_Max:** Valor máximo observado
- **Desv_Std:** Desviación estándar de las mediciones
- **Repeticiones:** Número de ejecuciones realizadas

B.3 Configuraciones Evaluadas

Sistemas evaluados:

- Sistema 1: AMD Ryzen 5 7600 (6 núcleos, 12 hilos)
- Sistema 2: Intel Xeon Gold 6240R (4 núcleos, 4 hilos)

Implementaciones:

- mmClasicaFork, mmClasicaPosix, mmClasicaOpenMP, mmFilasOpenMP

Tamaños de matriz:

- 100, 200, 400, 600, 800, 1000, 1200, 1400, 1600

Número de hilos/procesos:

- Sistema 1: 1, 2, 4, 6, 8, 10, 12
- Sistema 2: 1, 2, 4

C. Instrucciones de Reproducción

C.1 Requisitos Previos

Software necesario:

```
# Ubuntu/Debian
sudo apt-get install build-essential gcc make
sudo apt-get install libgomp1 # Para OpenMP
```

```
# Verificar versiones
gcc --version # Mínimo GCC 7.0
make --version
```

Herramientas opcionales:

```
# Para análisis de resultados
sudo apt-get install python3 python3-pip
pip3 install pandas matplotlib numpy
```

C.2 Compilación

Compilar todas las implementaciones:

```
# Desde el directorio raíz del proyecto  
make all  
  
# Verificar ejecutables generados  
ls -lh mm*
```

Compilación individual:

```
# Solo OpenMP  
make mmClasicaOpenMP.out  
  
# Solo Posix  
make mmClasicaPosix.out  
  
# Limpiar archivos compilados  
make clean
```

C.3 Ejecución de Pruebas

Prueba rápida:

```
# Ejecutar con matriz 100x100 y 4 hilos  
. ./mmClasicaOpenMP.out 100 4
```

```
# Ejemplo de salida esperada:  
# Tiempo: 1234 microsegundos
```

Batería completa de experimentos:

```
# Ejecutar script de automatización  
. ./ejecutar_todas_pruebas.sh  
  
# El script ejecutará:  
# - Todas las implementaciones  
# - Todos los tamaños definidos  
# - Todas las configuraciones de hilos  
# - 10 repeticiones por configuración
```

Personalizar pruebas:

```
# Editar configuración en lanzador.pl  
nano lanzador.pl
```

Modificar:

```

my @Size_Matriz = (100, 500, 1000);      # Tamaños
my @Num_Hilos = (1, 2, 4);                # Hilos
my $Repeticiones = 10;                    # Repeticiones
my $Nombre_Ejecutable = "mmClasicaFork"; # <-- CAMBIAR POR EJECUTABLE DESEADO

```

C.4 Análisis de Resultados

Procesar datos experimentales:

```

# Ejecutar script de análisis
python3 analizar_resultados.py

# Genera:
# - resultados_procesados.csv
# - graficas/*.png (speedup, eficiencia, tiempos)
# - estadisticas.txt (resumen estadístico)

```

Visualización en Excel/LibreOffice:

1. Abrir `resultados_procesados.csv`
2. Crear tabla dinámica con:
 - Filas: Tamaño de matriz
 - Columnas: Número de hilos
 - Valores: Tiempo_Promedio
3. Generar gráficos de líneas para análisis visual

C.5 Verificación de Resultados

Comprobar correctitud:

```

# Ejecutar con matriz pequeña y verificar resultado
./mmClasicaOpenMP.out 4 1

# Debe mostrar:
# - Matrices A, B, C
# - Resultado verificable manualmente

```

Validar consistencia:

```

# Comparar salidas de diferentes implementaciones
./mmClasicaOpenMP.out 8 1 > openmp.txt
./mmClasicaPosix.out 8 1 > posix.txt
diff openmp.txt posix.txt # Deben ser idénticas

```

D. Recursos Adicionales

D.1 Contacto

- Juan David Garzón: jd.garzonb@javeriana.edu.co
- Juan Pablo Sánchez: sanchez.jp@javeriana.edu.co