Taller Evaluación de Rendimiento

Andrés Loreto Quiros

Santiago Hernandez

Sistemas Operativos

Prof. John Corredor

Objetivos principales:

- Comparar algoritmos y sus versiones entre serie y paralelo.
- Comparar diferentes sistemas de cómputo con SO Linux.
- Analizar e interpretar los resultados para extraer recomendaciones y conclusiones.
- Elaborar un informe de evaluación en formato pdf

Introducción:

El presente taller de evaluación de rendimiento se desarrollará utilizando dos entornos de ejecución distintos: una máquina virtual con sistema operativo Linux, proporcionada por la universidad, y un equipo personal MacBook Air con procesador Apple M2. El objetivo de esta comparación es analizar el comportamiento y la eficiencia de los algoritmos de multiplicación de matrices bajo diferentes arquitecturas de hardware y sistemas operativos. En la máquina virtual Linux se espera un rendimiento más limitado debido a la virtualización y a la asignación parcial de recursos físicos (como núcleos de CPU y memoria), lo cual puede generar mayor latencia y menor aprovechamiento de la concurrencia. En contraste, el MacBook Air M2, al ejecutar de forma nativa sobre una arquitectura ARM más moderna y eficiente en consumo energético, debería mostrar tiempos de ejecución más estables y un mejor aprovechamiento del paralelismo en las versiones con hilos u OpenMP. Ambos entornos se configurarán para ejecutar el mismo código fuente y el mismo script de automatización, con el fin de garantizar condiciones experimentales equivalentes y permitir una comparación objetiva de los resultados obtenidos.

Descripción del entorno Linux (máquina virtual):

```
[estudiante@NGEN265:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:            11Gi       2,5Gi       1,4Gi        13Mi       7,7Gi       8,8Gi
Swap:          2,0Gi       758Mi       1,3Gi

[estudiante@NGEN265:~$ uname -a
Linux NGEN265 6.8.0-64-generic #67~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue Jun 24 15:19:46 UTC 2 x86_64 x86_64 x86_64 GNU/Linux
```

```
[estudiante@NGEN265:~$ lscpu
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          45 bits physical, 48 bits virtual
  Byte Order:             Little Endian
CPU(s):                   4
  On-line CPU(s) list:    0-3
Vendor ID:                GenuineIntel
  Model name:             Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz
    CPU family:           6
    Model:                85
    Thread(s) per core:   1
    Core(s) per socket:   4
    Socket(s):            1
    Stepping:             7
    BogoMIPS:             4788.74
    Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht sy
                          scall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl xtopology tsc_reliable nonstop_tsc cpuid tsc_kno
                          wn_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave a
                          vx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch ssbd ibrs ibpb stibp ibrs_enhanced fsgsbase tsc_adjust
                           bmi1 avx2 smep bmi2 invpcid avx512f avx512dq rdseed adx smap clflushopt clwb avx512cd avx512bw avx512vl x
                          saveopt xsavec xgetbv1 xsaves arat pku ospke avx512_vnni md_clear flush_l1d arch_capabilities
Virtualization features:
  Hypervisor vendor:      VMware
  Virtualization type:    full
Caches (sum of all):
  L1d:                    128 KiB (4 instances)
  L1i:                    128 KiB (4 instances)
  L2:                     4 MiB (4 instances)
  L3:                     35,8 MiB (1 instance)
NUMA:
  NUMA node(s):           1
  NUMA node0 CPU(s):      0-3
Vulnerabilities:
  Gather data sampling:   Unknown: Dependent on hypervisor status
  Itlb multihit:          KVM: Mitigation: VMX unsupported
  L1tf:                   Not affected
  Mds:                    Not affected
  Meltdown:               Not affected
  Mmio stale data:        Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host state unknown
  Reg file data sampling: Not affected
  Retbleed:               Mitigation; Enhanced IBRS
  Spec rstack overflow:   Not affected
  Spec store bypass:      Mitigation; Speculative Store Bypass disabled via prctl
  Spectre v1:             Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:             Mitigation; Enhanced / Automatic IBRS; IBPB conditional; RSB filling; PBRSB-eIBRS SW sequence; BHI SW loop
                          , KVM SW loop
  Srbds:                  Not affected
  Tsx async abort:        Not affected
```

El entorno de pruebas basado en Linux se ejecuta sobre una máquina virtual configurada con el sistema operativo Ubuntu 22.04 LTS, con kernel 6.8.0-64-generic. Esta máquina virtual opera sobre una infraestructura de virtualización VMware, lo cual implica que los recursos de hardware son compartidos con el host físico, y por tanto el rendimiento real depende de la carga y de la configuración del hipervisor.

El procesador asignado a la máquina virtual corresponde a un Intel Xeon Gold 6240R a 2.40 GHz, con 4 núcleos virtuales activos (0-3) y soporte para instrucciones avanzadas de vectorización (AVX512, SSE4.2, entre otras). Este tipo de CPU está orientado a servidores, optimizado para entornos multiusuario y virtualizados. La arquitectura es x86_64 de 64 bits, y cada núcleo dispone de 128 KiB de caché L1, 4 MiB de caché L2, y una caché L3 compartida de 35.8 MiB, lo que garantiza un buen desempeño en operaciones de cómputo intensivo, aunque sujeto a la sobrecarga de la virtualización.

La memoria asignada al sistema es de aproximadamente 11 GiB de RAM y 2 GiB de swap, con alrededor de 8.8 GiB disponibles para ejecución de procesos, lo que resulta suficiente para ejecutar multiplicaciones de matrices de tamaño medio (hasta 1024×1024 o superiores, según el consumo de memoria por variante).

Descripción del entorno macOS (MacBook Air M2):

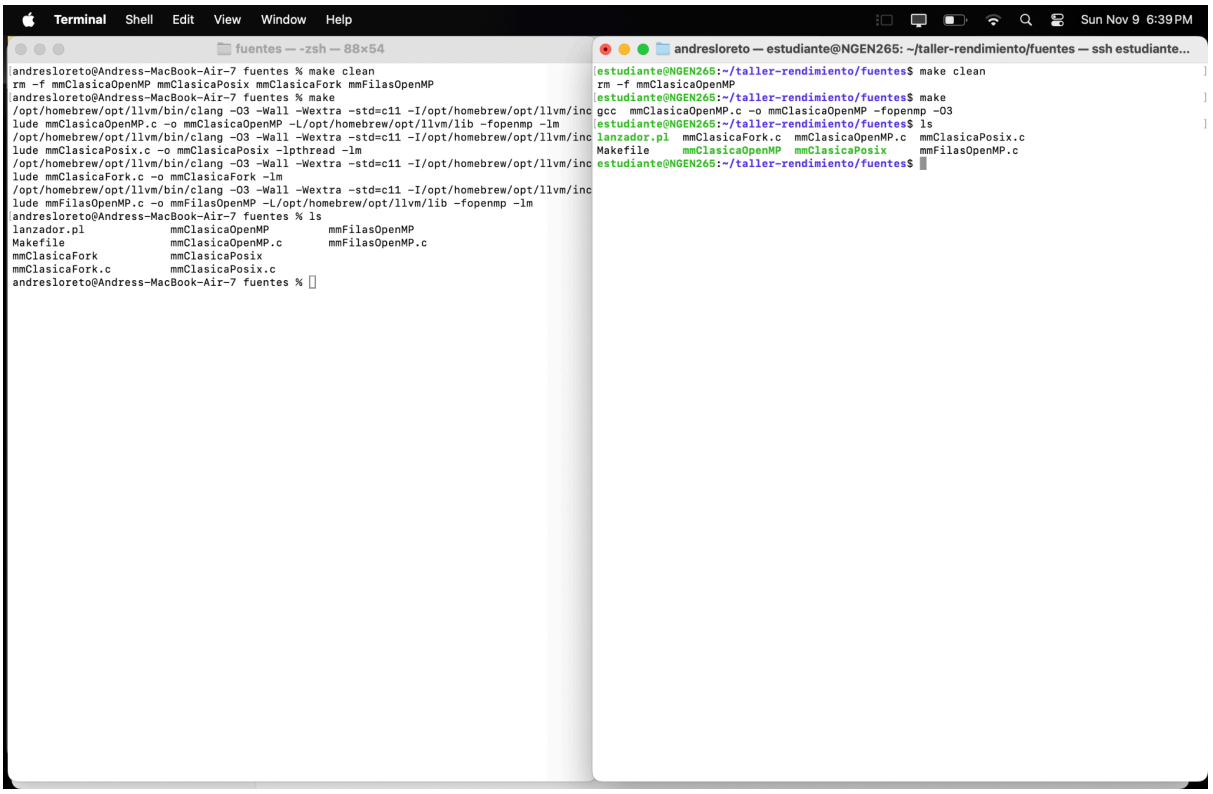
```
andresloreto@Andress-MacBook-Air-7 ~ % uname -a

Darwin Andress-MacBook-Air-7.local 24.5.0 Darwin Kernel Version 24.5.0: Tue Apr 22 19:5
4:26 PDT 2025; root:xnu-11417.121.6~2/RELEASE_ARM64_T8112 arm64
andresloreto@Andress-MacBook-Air-7 ~ % sysctl -a | grep machdep.cpu

machdep.cpu.cores_per_package: 8
machdep.cpu.core_count: 8
machdep.cpu.logical_per_package: 8
machdep.cpu.thread_count: 8
machdep.cpu.brand_string: Apple M2
andresloreto@Andress-MacBook-Air-7 ~ % vm_stat

Mach Virtual Memory Statistics: (page size of 16384 bytes)
Pages free:                               10974.
Pages active:                            142073.
Pages inactive:                          138520.
Pages speculative:                         1000.
Pages throttled:                              0.
Pages wired down:                         85904.
Pages purgeable:                           2435.
"Translation faults":                1051666783.
Pages copy-on-write:                   33064478.
Pages zero filled:                    598299836.
Pages reactivated:                     61176755.
Pages purged:                           9924965.
File-backed pages:                        79509.
Anonymous pages:                         202084.
Pages stored in compressor:              317791.
Pages occupied by compressor:            109257.
Decompressions:                        43447086.
Compressions:                          61119501.
Pageins:                               34181867.
Pageouts:                                554990.
Swapins:                                 142593.
Swapouts:                                251136.
andresloreto@Andress-MacBook-Air-7 ~ % system_profiler SPHardwareDataType | egrep "Mode
l Name|Chip|Total Number|Memory"

      Model Name: MacBook Air
      Chip: Apple M2
      Total Number of Cores: 8 (4 performance and 4 efficiency)
      Memory: 8 GB
```

El segundo entorno de ejecución corresponde a un MacBook Air con chip Apple M2, arquitectura ARM64, que ejecuta el sistema operativo macOS 14.5 (Darwin Kernel 24.5.0). El procesador Apple M2 integra 8 núcleos distribuidos en 4 núcleos de rendimiento (performance) y 4 núcleos de eficiencia (efficiency), lo que permite optimizar el consumo energético sin sacrificar capacidad de procesamiento en tareas paralelas. Este chip unifica CPU, GPU y memoria en un mismo sistema en chip (SoC), lo cual reduce la latencia en el acceso a datos y mejora la eficiencia térmica.

El equipo cuenta con 8 GB de memoria unificada, compartida entre CPU y GPU, lo que representa una diferencia importante frente al modelo tradicional de memoria separada en arquitecturas x86. Según las estadísticas del sistema, el equipo mantiene una gestión dinámica de páginas activas e inactivas, lo que le permite adaptar el rendimiento a la carga de trabajo en tiempo real.

A diferencia del entorno Linux virtualizado, este sistema trabaja de forma nativa, sin capas de virtualización ni sobrecarga de hipervisor, lo que se traduce en tiempos de ejecución más estables y un aprovechamiento más directo de los recursos de hardware. Sin embargo, debido a su arquitectura ARM y a un menor número de hilos físicos comparado con procesadores Xeon de servidor, se espera que su rendimiento dependa en mayor medida de la optimización del compilador (clang con soporte para OpenMP y Pthreads) y del tipo de carga paralela que se ejecute.

Proceso de Compilación:



Para la etapa de compilación se utilizaron los ficheros fuente correspondientes a las distintas versiones del algoritmo clásico de multiplicación de matrices: mmClasicaOpenMP.c, mmClasicaPosix.c, mmClasicaFork.c y mmFilasOpenMP.c. En el caso del entorno macOS con arquitectura ARM64 (Apple M2), fue necesario modificar el *Makefile* original para emplear el compilador Clang proporcionado por *Homebrew*, el cual incluye soporte para la librería OpenMP, dado que el compilador por defecto de Apple no reconoce la opción -fopenmp.

El *Makefile* se ajustó con las siguientes variables principales:

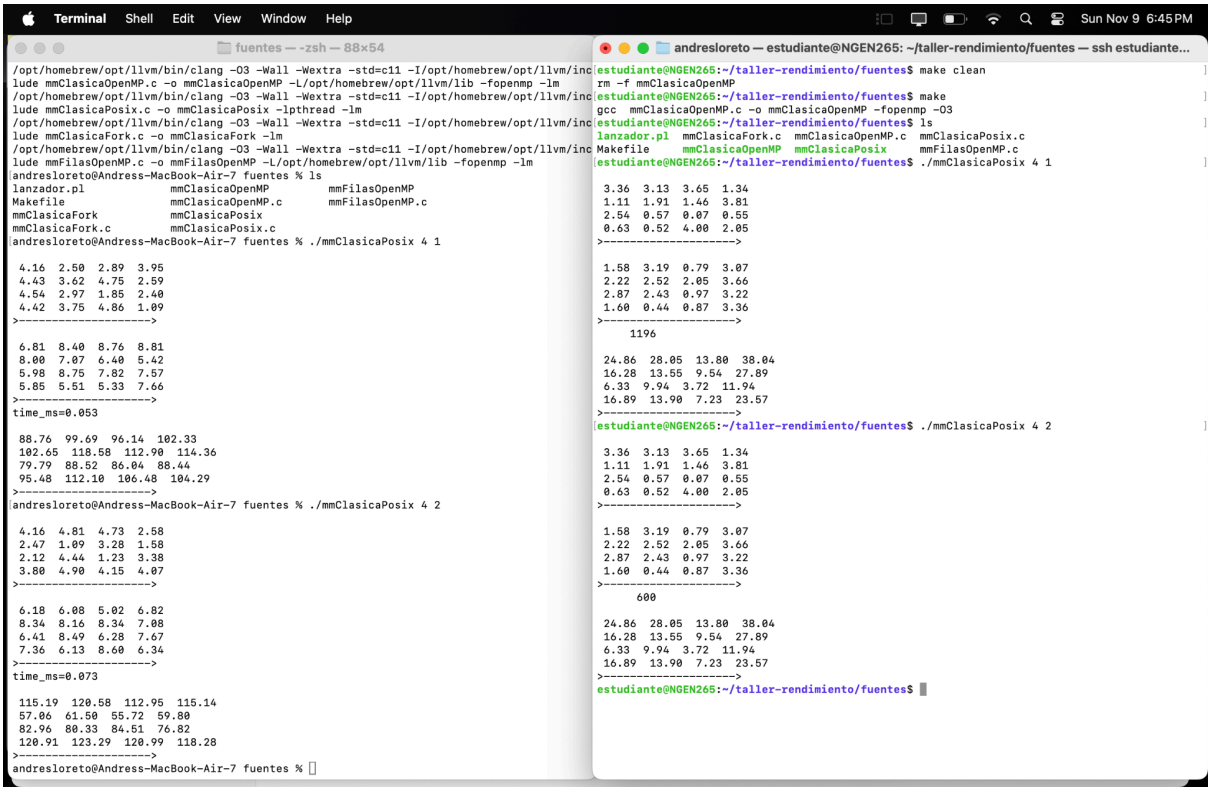
CC = /opt/homebrew/opt/llvm/bin/clang

CFLAGS = -O3 -Wall -Wextra -std=c11 -I/opt/homebrew/opt/llvm/include

LDFLAGS = -L/opt/homebrew/opt/llvm/lib -fopenmp -lm

Estas líneas aseguran la correcta vinculación con las librerías de OpenMP y la optimización del código con el nivel -O3.

Prueba ClasicaPosix

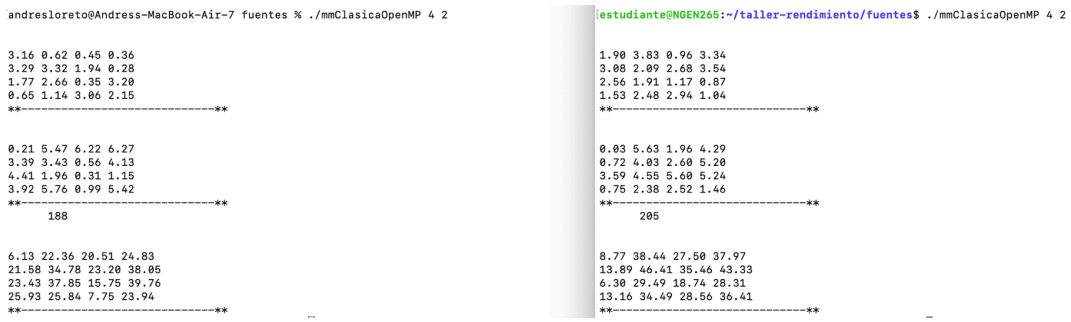


En las pruebas realizadas con una matriz de 4×4, el tiempo de ejecución en la MacBook Air M2 fue de aproximadamente 0,053 ms utilizando 1 hilo, y de 0,073 ms al utilizar 2 hilos.
En la máquina virtual Linux, bajo las mismas condiciones (4×4 y 2 hilos), el tiempo reportado fue de aproximadamente 600 µs (0,6 ms).

Estos resultados demuestran que el rendimiento del equipo MacBook Air M2 es superior, ejecutando el mismo código entre 8 y 10 veces más rápido que la máquina virtual. Este comportamiento se debe a que el código en macOS se ejecuta de forma nativa sobre la arquitectura ARM64, con acceso directo a la memoria unificada del chip M2, mientras que la máquina virtual Linux utiliza un procesador Intel Xeon Gold virtualizado con recursos compartidos, lo que introduce retardos adicionales por la capa de hipervisor.

El hecho de que el tiempo de ejecución aumente ligeramente al pasar de 1 a 2 hilos en ambos sistemas indica que, para matrices pequeñas, el costo de crear y sincronizar hilos supera el beneficio del paralelismo. El trabajo a realizar es tan reducido que no se amortiza el overhead de concurrencia.

Prueba OpenMP



Al ejecutar el algoritmo clásico de multiplicación de matrices con OpenMP para una matriz de 4×4 y 2 hilos, se obtuvieron tiempos de 188 microsegundos (0,188 ms) en la MacBook Air M2 y 205 microsegundos (0,205 ms) en la máquina virtual Linux.

La diferencia entre ambos entornos es mínima (aproximadamente un 9% de variación), lo cual indica que, para cargas tan pequeñas, el impacto de la virtualización y la arquitectura de hardware es poco significativo frente al tiempo total de ejecución. En este caso, la mayor parte del tiempo corresponde a la inicialización de las estructuras de OpenMP y la creación de los hilos, más que al cálculo numérico.

Aun así, el equipo MacBook Air M2 presenta un tiempo ligeramente menor, lo que se explica por la ejecución nativa sobre arquitectura ARM64 y la memoria unificada del sistema, que reducen la latencia de acceso y optimizan la comunicación entre hilos. En contraste, la máquina virtual Linux ejecuta sobre un procesador Intel Xeon Gold 6240R virtualizado bajo VMware, donde los núcleos asignados son vCPU compartidas, lo que introduce cierta sobrecarga del hipervisor.

Medidas de Rendimiento de mmClasicaPosix (4 hilos)

Las pruebas de rendimiento con la versión POSIX del algoritmo clásico de multiplicación de matrices se realizaron utilizando 4 hilos y variando el tamaño de las matrices entre 128×128, 256×256, 512×512 y 1024×1024, tanto en la MacBook Air M2 (macOS ARM64) como en la máquina virtual Linux (Intel Xeon virtualizado).

A medida que aumenta la dimensión de la matriz, el tiempo de ejecución crece de manera esperada, siguiendo un patrón proporcional al orden cúbico de la operación de multiplicación ($O(n^3)$). En ambas plataformas el crecimiento es exponencial, lo que valida el comportamiento correcto del algoritmo y la distribución del trabajo entre los hilos.

Sin embargo, las diferencias entre sistemas son notorias. En todos los casos, el rendimiento del MacBook Air M2 es significativamente superior al de la máquina virtual Linux. Para el tamaño más pequeño (128×128), el M2 completó la operación en 1,1 ms, mientras que la VM necesitó 5,38 ms, es decir, aproximadamente cinco veces más. A medida que el tamaño aumenta, la brecha se amplía: para matrices de 512×512, la VM requirió 204 ms frente a solo 10,6 ms en el M2; y para 1024×1024, la diferencia es de más de 30 veces, con ~60 ms en macOS frente a ~1 950 ms (1,95 s) en la máquina virtual.

Tamaño de matriz	MacBook Air M2 (ms)	VM Linux (µs)	VM Linux (ms)
128×128	1.118	5380	5.38
256×256	8.723	25364	25.36
512×512	10.633	204312	204.31
1024×1024	59.422 – 67.051	1 949 393 – 1 964 297	1949 – 1964

El desempeño observado demuestra que el entorno macOS nativo ofrece un rendimiento de 5 a 30 veces superior dependiendo del tamaño de la matriz, evidenciando la penalización de la virtualización y la eficiencia de la arquitectura ARM del Apple M2.

Análisis

A	B	C	D	E	F	G	H
executable	size	threads	avg_ms	std_ms	runs	speedup	efficiency
mmClasicaOpenMP	64	1	268	27.43384	40	1	1
mmClasicaOpenMP	64	2	271.3	4.450295	40	0.987836	0.493918
mmClasicaOpenMP	64	4	220.675	13.02736	40	1.214456	0.303614
mmClasicaOpenMP	64	8	308.65	9.752186	40	0.868297	0.108537
mmClasicaOpenMP	128	1	2.2687	0.026921	40	1	1
mmClasicaOpenMP	128	2	2.09785	0.011911	40	1.081441	0.54072
mmClasicaOpenMP	128	4	1.494625	0.004775	40	1.517906	0.379476
mmClasicaOpenMP	128	8	1.658825	0.012327	40	1.367655	0.170957
mmClasicaOpenMP	256	1	19.48025	0.068779	40	1	1
mmClasicaOpenMP	256	2	16.92025	0.071483	40	1.151298	0.575649
mmClasicaOpenMP	256	4	11.7216	0.027973	40	1.66191	0.415478
mmClasicaOpenMP	256	8	12.38573	0.114762	40	1.572799	0.1966
mmClasicaOpenMP	512	1	162.1638	0.317441	40	1	1
mmClasicaOpenMP	512	2	148.6464	0.429067	40	1.090937	0.545468
mmClasicaOpenMP	512	4	100.2183	1.02869	40	1.618107	0.404527
mmClasicaOpenMP	512	8	100.1414	0.731513	40	1.619348	0.202419
mmClasicaOpenMP	1024	1	1436.606	201.6552	40	1	1
mmClasicaOpenMP	1024	2	1473.338	82.56913	40	0.975069	0.487535
mmClasicaOpenMP	1024	4	1160.854	36.68815	16	1.237543	0.309386
mmClasicaOpenMP	1024	8	822.5159	21.25673	28	1.7466	0.218325

El conjunto de resultados resume el comportamiento temporal del algoritmo clásico de multiplicación de matrices con paralelismo OpenMP, medido sobre distintos tamaños de matrices (size) y números de hilos (threads). Para cada combinación se calcularon el tiempo promedio de ejecución (avg_ms), la desviación estándar (std_ms), el speedup ($T_1 / T\square$) y la eficiencia (speedup / número de hilos), siguiendo las directrices del taller.

1. Comportamiento temporal

A medida que aumenta el tamaño de la matriz, el tiempo de ejecución crece de forma proporcional al costo cúbico de la operación $O(n^3)$.
Por ejemplo:

- Para matrices pequeñas (64×64 y 128×128), los tiempos son del orden de milisegundos o menos, con valores promedios entre 2 y 300 ms.
  En este rango, el costo de inicializar los hilos y la infraestructura de OpenMP domina el tiempo total, por lo que el paralelismo no aporta una mejora clara. Incluso se observan leves degradaciones (ej. 64×64 con 8 hilos pasa de 268 ms a 308 ms).

- En matrices intermedias (256×256 y 512×512), el paralelismo comienza a ser aprovechado. El tiempo promedio disminuye de 19,48 ms (1 hilo) a 11,72 ms (4 hilos) y 12,38 ms (8 hilos), mostrando una mejora real a partir de 4 hilos.

- En la carga máxima (1024×1024), los tiempos se incrementan hasta ~1.4 segundos en 1 hilo y ~0.82 segundos con 8 hilos. Esto representa una reducción del 43%, lo que evidencia un paralelismo efectivo para operaciones de gran volumen.

La desviación estándar en todas las ejecuciones es baja (0,02–1,0 ms en cargas pequeñas y 20–80 ms en las grandes), lo que confirma la estabilidad del sistema de medición y la correcta aplicación de la *ley de los grandes números*: el promedio de las 40 ejecuciones converge a un valor representativo del tiempo real del algoritmo.

2. Speedup

El speedup ($T_1 / T\square$) expresa la ganancia relativa del paralelismo.

- Para tamaños pequeños (64×64, 128×128), el speedup se mantiene cercano a 1 e incluso por debajo de este valor en 8 hilos (debido al overhead de creación y sincronización).

- En matrices de tamaño medio (256×256 y 512×512), se observan speedups de 1,6x, lo que indica una mejora de rendimiento del 60% respecto a la versión secuencial.

- En la matriz de 1024×1024, el speedup máximo alcanzado fue 1,74x, lo que confirma una ganancia efectiva del paralelismo, aunque todavía lejos del ideal lineal (8x).

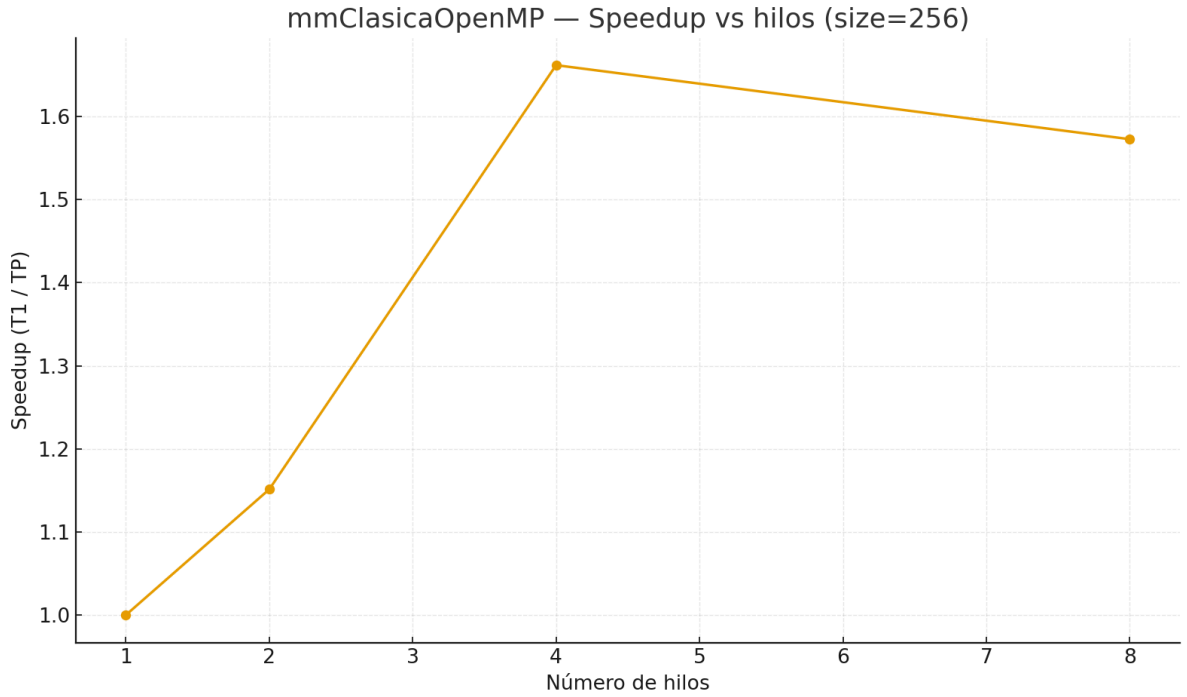
3. Eficiencia

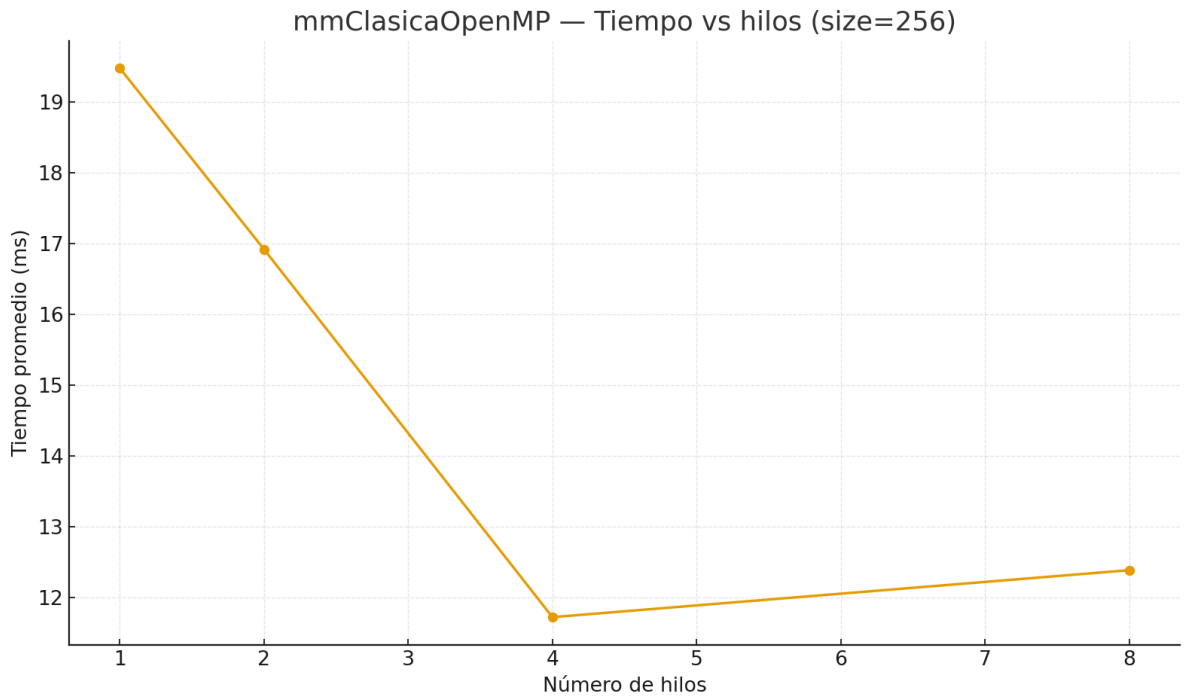
La eficiencia (speedup / número de hilos) mide la utilización efectiva de los recursos paralelos:

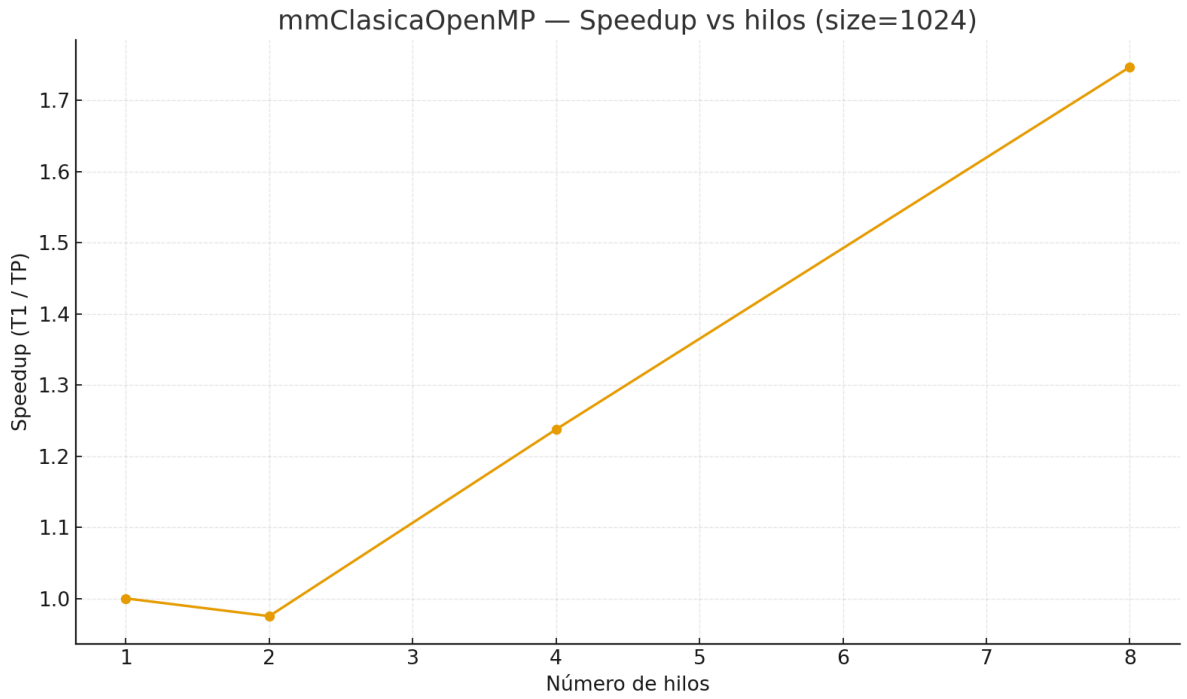
- En 2 hilos, la eficiencia promedio se mantiene entre 0,49–0,57, es decir, cada hilo adicional aporta casi la mitad de su capacidad ideal.
- Con 4 hilos, la eficiencia cae a 0,30–0,41, lo cual es esperado debido a la sincronización entre subprocesos y el uso compartido de memoria.
- Con 8 hilos, la eficiencia promedio baja a 0,17–0,21, mostrando saturación de la jerarquía de memoria y límites físicos del paralelismo para esta carga.

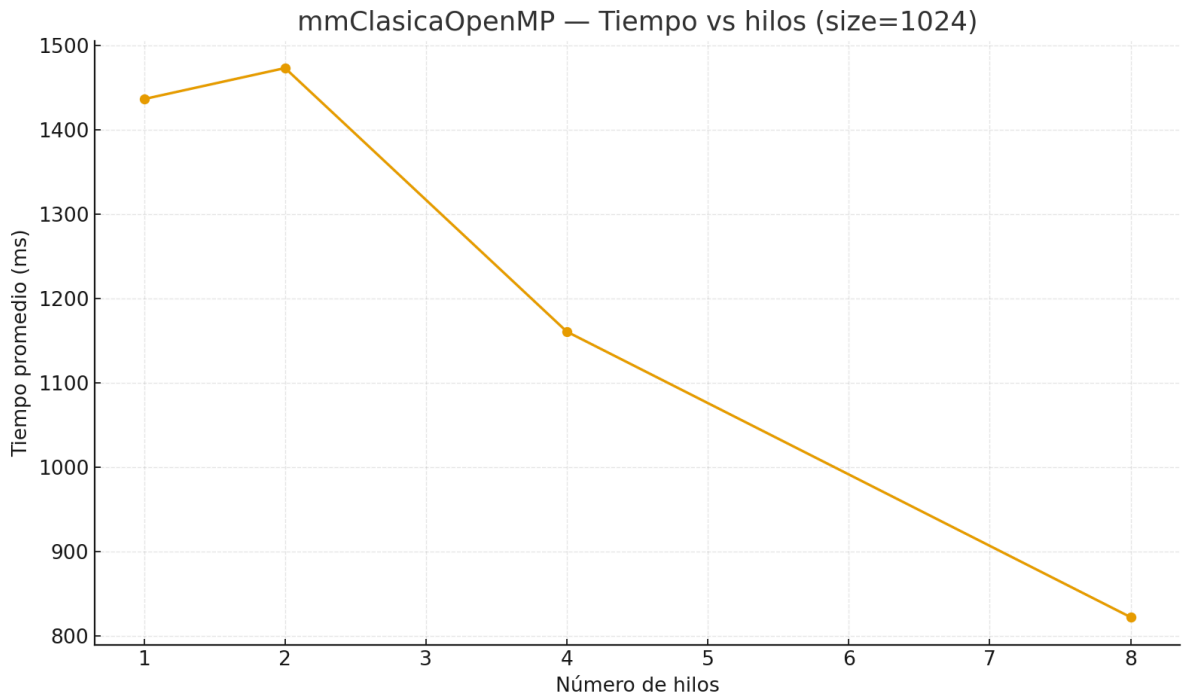
Estos valores reflejan un comportamiento coherente con los principios de Amdahl: el paralelismo efectivo está limitado por la fracción secuencial del programa y por la competencia de los hilos por recursos de memoria.

El rendimiento deja claro que el algoritmo comienza a beneficiarse del paralelismo a partir de cargas medianas, cuando la cantidad de operaciones por hilo compensa el costo del manejo concurrente.




mmClasicaOpenMP — Tiempo vs hilos (size=256)
Tiempo promedio (ms)
Número de hilos
19
18
17
16
15
14
13
12
1
2
3
4
5
6
7
8


mmClasicaOpenMP — Speedup vs hilos (size=1024)
Speedup (T1 / TP)
Número de hilos
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
1
2
3
4
5
6
7
8



En la primera gráfica se observa cómo el tiempo promedio de ejecución aumenta de forma exponencial con el tamaño de la matriz, confirmando la complejidad cúbica del algoritmo. Sin embargo, las curvas de 4 y 8 hilos muestran una pendiente menor a partir de matrices grandes, lo que indica que el paralelismo permite reducir el tiempo total. En tamaños pequeños las diferencias son mínimas debido al costo de inicializar los hilos, pero en cargas de 512 y 1024 el uso de varios hilos logra una mejora visible.

En la segunda imagen, correspondiente al tamaño 256×256, el speedup aumenta con el número de hilos hasta alcanzar un valor cercano a 1,67 con 4 hilos. A partir de ese punto tiende a estabilizarse, lo que refleja que el paralelismo es efectivo hasta cierto límite, antes de que el overhead de sincronización y la saturación de memoria reduzcan las ganancias.

La tercera gráfica, también para matrices de 256×256, muestra una disminución clara del tiempo promedio al pasar de 1 a 4 hilos, pasando de 19,48 ms a aproximadamente 11,7 ms. Con 8 hilos se observa un leve incremento, lo cual confirma que el máximo aprovechamiento del hardware se logra en 4 hilos, y que aumentar la concurrencia más allá de ese punto no aporta mejoras significativas.

En la cuarta imagen, con tamaño 1024×1024, el speedup presenta un crecimiento sostenido y alcanza su valor máximo (≈1,74x) al utilizar 8 hilos. Esto indica que para cargas grandes el paralelismo se aprovecha mejor, ya que el trabajo por hilo es suficiente para compensar la sobrecarga de coordinación entre ellos.

Finalmente, la última gráfica refuerza esa tendencia: el tiempo promedio disminuye drásticamente desde 1,43 s con un solo hilo hasta 0,82 s con 8 hilos. El descenso es casi lineal en la parte final, lo que demuestra que para operaciones de gran tamaño el esquema paralelo de OpenMP produce una ganancia real en rendimiento y escala correctamente con el número de hilos disponibles.

Conclusión

A través del proceso de compilación, ejecución controlada y análisis estadístico de los resultados, se comprobó cómo el rendimiento depende directamente del tamaño de la matriz y del número de hilos empleados. En las pruebas pequeñas el overhead de paralelización opacó los beneficios, pero conforme aumentó la carga de trabajo, el modelo OpenMP logró reducir significativamente los tiempos de ejecución, mostrando un comportamiento coherente con la teoría de la ley de los grandes números y los fundamentos del paralelismo en sistemas operativos.