Facultad de Ingeniería
Departamento de Ingeniería de Sistemas

Taller de Evaluación de Rendimiento

Noviembre 2025

Integrantes:

Danna Rojas Bernal
Giovanny Andrés Durán Rentería
Christian Becerra Enciso
María Fernanda Velandia Gracia

Asignatura: Sistemas Operativos
Docente: John Corredor Franco

Bogotá D.C., Colombia
15 de noviembre de 2025

Índice

1. Introducción

Lo desarrollado en este trabajo fue el algoritmo clásico para la multiplicación de matrices, a través del cual se pudieron evaluar diferentes métodos de paralelismo, como el uso de procesos con `fork`, otro con hilos POSIX y dos con directivas `OpenMP` (una simple y la otra mejor optimizada por las filas).

En general, lo que se hizo fue ver el contraste entre los cambios de tiempo sobre las distintas ejecuciones al modificar el tamaño de las matrices y la cantidad de hilos, así como observar cómo los diferentes métodos utilizan los recursos del sistema, teniendo en cuenta lo visto en las clases.

2. Marco teórico

2.1. Paralelismo y multiplicación de matrices

El paralelismo consiste en dividir una tarea en varias partes que se ejecutan al mismo tiempo para aprovechar mejor los recursos del procesador [1]. En este taller se tomó de base el algoritmo clásico de multiplicación de matrices, porque es un caso ideal para evaluar este tipo de procesamiento. Esto, puesto que el cálculo de cada elemento de la matriz resultado puede hacerse de forma independiente, permitiendo así repartir el trabajo sin afectar los resultados.

Entonces, lo que se hace es que cada elemento de la matriz final se obtiene multiplicando una fila de la primera matriz por una columna de la segunda, así que los cálculos no interfieren entre sí. Lo anterior es ideal, pues ayuda a que diferentes procesos o hilos trabajen al mismo tiempo en distintas partes del problema, y con esto fue posible aplicar diferentes formas de paralelismo y analizar el rendimiento teniendo en cuenta el tiempo de ejecución de cada una de esas formas.

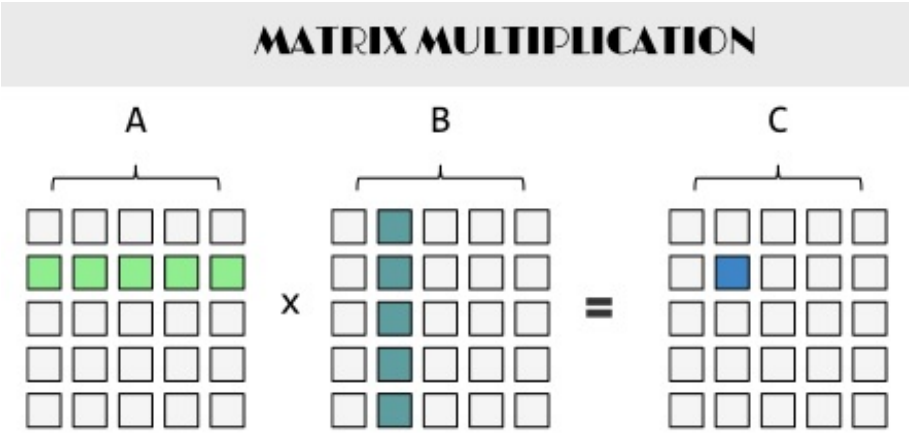

Figura 1: Esquema general de la multiplicación de matrices.

2.2. Procesos y hilos

En la primera carpeta se usó el método de procesos creados con la llamada `fork()`, en donde cada proceso tiene su propio espacio en la memoria y ejecuta su parte independiente

del resto; pero, claramente, esto genera un mayor consumo de recursos porque cada proceso es una copia del original [2].

Después se agregó otra carpeta con hilos POSIX (`pthread`), que, a diferencia de los procesos, comparten el mismo espacio de memoria y ocupan menos de la misma, lo que hace más rápida la comunicación entre ellos. Sin embargo, esto también implica tener más cuidado con las variables compartidas, ya que varios hilos pueden acceder a ellas al mismo tiempo y generar errores si no se controlan correctamente [3].

2.3. OpenMP y paralelización automática

Ahora, la siguiente carpeta del programa se desarrolló con OpenMP, que es una herramienta que facilita la programación paralela en memoria compartida. Gracias a este modelo, no es necesario crear los hilos de forma manual, sino que se agregan directivas o también llamadas "pragmas", que, cuando se compila, el compilador interpreta estas últimas para dividir el trabajo entre los núcleos del procesador (si solo se tiene un núcleo, no habrá diferencias al usar OMP o no usarlo).

Entonces, se trabajaron dos carpetas: en la primera versión con OpenMP se aplicó la directiva `#pragma omp parallel for` sobre el ciclo principal del programa, permitiendo que cada iteración se ejecute en paralelo. Luego se probó con la otra carpeta, que tiene una versión optimizada, en la que se ajustó la forma en que se repartían las filas entre los hilos para lograr reducir los tiempos de espera [4].

2.4. Medición del rendimiento

Para evaluar el desempeño de cada versión, se midió principalmente el tiempo de ejecución y se hicieron pruebas cambiando el tamaño de las matrices y la cantidad de hilos utilizados, con el fin de observar cómo variaban los resultados y qué tan eficiente era cada método de paralelismo.

Entonces, más allá de determinar cuál versión era más rápida, la intención fue entender cómo cada tipo de paralelismo afecta el uso del procesador, la memoria y el balance de trabajo entre hilos o procesos.

3. Diseño del experimento

El análisis constará de 6 partes, el objetivo será hacer una comparación de 5 exhaustivas máquinas que contienen distintos componentes de hardware. Para evaluar su rendimiento, se usarán 4 técnicas distintas, la implementación de las funciones fork(), OpenMP(), Posix() y FilasOpenMP. Se esperan distintos tiempos de ejecución que reflejaran la optimización del procesador además de la eficiencia en cada ejecución.

En este taller de rendimiento se dará suma importancia a los tiempos de ejecución y, de igual manera, que tan eficiente fue su ejecución con respecto a dos parámetros distintos que serán la cantidad de hilos y el tamaño de las matrices. Se evaluaran desde casos con índices pequeños, hasta valores grandes que reflejen un incremento relevante en los tiempos de espera por el usuario.

De igual manera, para hacer verídicos nuestros datos nos basamos en teorías probabilísticas como la ley de los grandes números. Por lo tanto, cada combinación de n hilos y tamaño de matriz se repetirá 30 veces para asegurar una dispersión mínima entre los datos y conseguir valores ponderados a su resultado real.

En primera instancia, a partir de nuestra batería de experimentación, generamos un resumen tipo CSV por cada máquina utilizada que nos muestra: el método utilizado, el número de hilos usados, el tamaño de la matriz y un tiempo ponderado del caso.

Por consiguiente, tomaremos cada caso y lo analizaremos en el programa estadístico R, que nos permitirá abstraer los datos para tener una fiel conclusión sobre la relación que existe entre componentes de la máquina, función utilizada y la influencia que tiene los hilos en los tiempos de ejecución y de eficiencia.

Como siguiente paso, una vez realizado el análisis de ejecución de cada máquina, procederemos a compararlas entre sí para observar en qué medida difieren sus tiempos de ejecución según sus componentes. Esto nos permitirá comprender mejor cómo funciona cada dispositivo y cómo aprovechar sus características bajo determinadas condiciones.

Antes de abordar los resultados, presentaremos algunos datos técnicos de cada máquina para identificar sus componentes y, posteriormente, fundamentar las conclusiones.

```
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  12
  On-line CPU(s) list:   0-11
Vendor ID:               GenuineIntel
  Model name:            12th Gen Intel(R) Core(TM) i5-12450H
    CPU family:          6
    Model:               154
    Thread(s) per core:  2
    Core(s) per socket:  6
    Socket(s):           1
    Stepping:            3
    BogoMIPS:            4992.01
    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 sys
                         call nx pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopology tsc_reliable nonstop_tsc cpuid tsc_known_fre
                         q pni pclmulqdq vmx ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx
                         f16c rdrand hypervisor lahf_lm abm 3dnowprefetch ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow ept vpid ept
                         _ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt clwb sha_ni xsaveopt xs
                         avec xgetbv1 xsaves avx_vnni vnmi umip waitpkg gfni vaes vpclmulqdq rdpid movdiri movdir64b fsrm md_clear s
                         erialize flush_l1d arch_capabilities
Virtualization features:
  Virtualization:        VT-x
  Hypervisor vendor:     Microsoft
  Virtualization type:   full
Caches (sum of all):
  L1d:                   288 KiB (6 instances)
  L1i:                   192 KiB (6 instances)
  L2:                    7.5 MiB (6 instances)
  L3:                    12 MiB (1 instance)
```

Figura 2: Componentes máquina 1

La Figura 4 consta de la máquina número 1, la cual pertenece a nuestro compañero

de Grupo Christian. Definiremos cada máquina con un número para su posterior comparación. La máquina analizada cuenta con un procesador Intel Core i5-12450H de 12ª generación con arquitectura x86-64, que dispone de 6 núcleos físicos y 12 hilos lógicos. En cuanto a la jerarquía de memoria caché, posee 288 KiB de L1d, 192 KiB de L1i, 7.5 MiB de L2 distribuidos en 6 instancias (una por núcleo), y 12 MiB de L3 compartida, elementos fundamentales para el rendimiento en operaciones con matrices de distintos tamaños.

```
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          39 bits physical, 48 bits virtual
  Byte Order:             Little Endian
CPU(s):                   8
  On-line CPU(s) list:    0-7
Vendor ID:                GenuineIntel
  Model name:             11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz
    CPU family:           6
    Model:                140
    Thread(s) per core:   2
    Core(s) per socket:   4
    Socket(s):            1
    Stepping:             1
    BogoMIPS:             6220.80
    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 syscall nx pdpe1gb
                           rdtscp lm constant_tsc arch_perfmon rep_good nopl xtopology tsc_reliable nonstop_tsc cpuid tsc_known_freq pni pclmulqdq v
                          mx ssse3 fma cx16 pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf
                          _lm abm 3dnowprefetch ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi
                          2 erms invpcid avx512f avx512dq rdseed adx smap avx512ifma clflushopt clwb avx512cd sha_ni avx512bw avx512vl xsaveopt xsav
                          ec xgetbv1 xsaves vnmi avx512vbmi umip avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx512_bitalg avx512_vpopcntdq rdpid
                          movdiri movdir64b fsrm avx512_vp2intersect md_clear flush_l1d arch_capabilities
Virtualization features:
  Virtualization:         VT-x
  Hypervisor vendor:      Microsoft
  Virtualization type:    full
Caches (sum of all):
  L1d:                    192 KiB (4 instances)
  L1i:                    128 KiB (4 instances)
  L2:                     5 MiB (4 instances)
  L3:                     8 MiB (1 instance)
NUMA:
  NUMA node(s):           1
  NUMA node0 CPU(s):      0-7
Vulnerabilities:
  Gather data sampling:   Unknown: Dependent on hypervisor status
  Itlb multihit:          Not affected
  L1tf:                   Not affected
  Mds:                    Not affected
  Meltdown:               Not affected
  Mmio stale data:        Not affected
  Reg file data sampling: Not affected
```

Figura 3: Componentes máquina 2

La figura 5 corresponde a la máquina número 2, perteneciente a nuestra compañera Danna, la máquina analizada cuenta con un procesador Intel Core i5-11300H de 11ª generación con arquitectura x86-64, que dispone de 4 núcleos físicos y 8 hilos lógicos. En cuanto a la jerarquía de memoria caché, posee 192 KiB de L1d, 128 KiB de L1i, 5 MiB de L2 distribuidos en 4 instancias (una por núcleo), y 8 MiB de L3 compartida.

```
PS C:\Users\mafev> Get-CimInstance Win32_Processor | Format-List Name, NumberOfCores, NumberOfLogicalProcessors, L2CacheSize, L3CacheSize


Name                      : AMD Ryzen 7 7735HS with Radeon Graphics
NumberOfCores             : 8
NumberOfLogicalProcessors : 16
L2CacheSize               : 4096
L3CacheSize               : 16384
```

Figura 4: Componentes máquina 3

La figura 6 muestra la máquina número 3 que corresponde a la compañera Maria Fernanda, la máquina analizada cuenta con un procesador AMD Ryzen 7 7735HS, que dispone de 8 núcleos físicos y 16 hilos lógicos, proporcionando la mayor capacidad entre todas las máquinas evaluadas. En cuanto a la jerarquía de memoria caché, posee 4096 KiB (4 MiB) de L2 y 16384 KiB (16 MiB) de L3 compartida. Esta máquina se distingue por ser la única con procesador AMD y por tener el mayor número de núcleos e hilos disponibles, lo que debería traducirse en un mejor rendimiento en operaciones. A diferencia de las otras máquinas, esta máquina podría estar ejecutándose directamente sobre hardware

físico, lo que eliminaría el overhead de virtualización y potencialmente ofrecería tiempos de ejecución más competitivos.

```
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 syscall nx pdpe1gb
                           rdtscp lm constant_tsc arch_perfmon nopl xtopology tsc_reliable nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 fma
                          cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefe
                          tch ssbd ibrs ibpb stibp ibrs_enhanced fsgsbase tsc_adjust bmi1 avx2 smep bmi2 invpcid avx512f avx512dq rdseed adx smap cl
                          flushopt clwb avx512cd avx512bw avx512vl xsaveopt 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
```

Figura 5: Componentes máquina 4

La figura 7 sera la máquina número 4 del compañero Giovanny, la máquina analizada cuenta con un procesador Intel Xeon Gold 6240R de arquitectura x86-64, que dispone de 4 núcleos físicos y 4 hilos (sin hyperthreading habilitado), operando a una frecuencia de 2.4GHz. En cuanto a la jerarquía de memoria caché, posee 128 KiB de L1d, 128 KiB de L1i, 4 MiB de L2 distribuidos en 4 instancias (una por núcleo), y 35.8 MiB de L3 compartida, siendo esta última considerablemente superior a las máquinas anteriores. A pesar de contar con menos núcleos que las máquinas anteriores, su arquitectura Xeon de servidor y su gran caché L3 pueden compensar esta diferencia en ciertos escenarios de carga de trabajo.

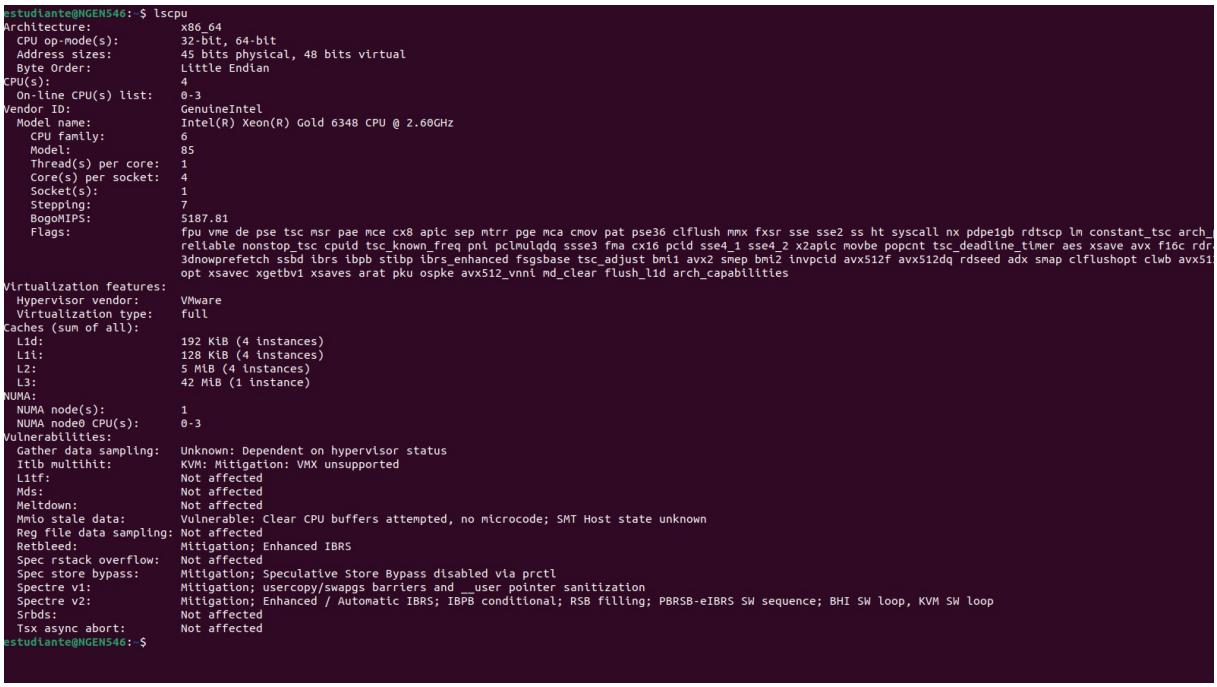

Figura 6: Componentes máquina 5

La Figura 8 correponde a la máquina número 5, proporcionada por la universidad durante el curso, la máquina analizada cuenta con un procesador Intel Xeon Gold 6348 de arquitectura x86-64, que dispone de 4 núcleos físicos y 4 hilos (sin hyperthreading habilitado), operando a una frecuencia de 2.6GHz. En cuanto a la jerarquía de memoria caché, posee 192 KiB de L1d, 128 KiB de L1i, 5 MiB de L2 distribuidos en 4 instancias (una por núcleo), y 42 MiB de L3 compartida, siendo esta última la más grande de todas las máquinas analizadas. A pesar de contar con menos núcleos que algunas máquinas anteriores, su arquitectura Xeon de servidor de última generación, su mayor frecuencia base (2.6GHz) y su enorme caché L3 pueden proporcionar ventajas significativas.

4. Resultados

4.1. Máquina de Christian

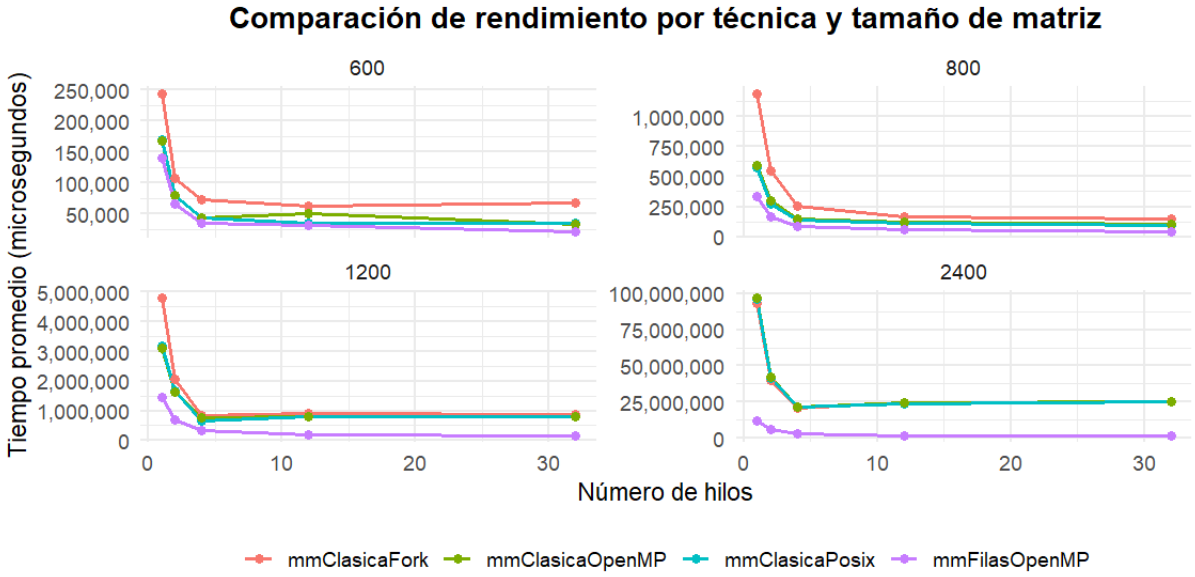

Figura 7: Comparación de rendimiento por técnica y tamaño de matriz

En la máquina 1, luego de realizar el análisis 30 veces por cada método y desde la combinación de las dos variables disponibles, hilos y matrices. Se usó como referencia 4 matrices de distintos tamaños, matrices de 600x600, 800x800, 1200x1200 y 2400x2400. Por otro lado, se definió una cierta cantidad de hilos, ejecuciones con 1, 2 ,4 ,12 y 32 hilos, usados con cada tamaño de matriz. En consecuencia, obtuvimos 4 gráficas donde cada una representa el tamaño de una matriz, en el eje y el tiempo de ejecución y en el eje x la cantidad de hilos. Cada línea representa la línea de tendencia que se obtuvo con cada función al aumentar la cantidad de hilos. Con matrices pequeñas (600x600 y 800x800), todas las técnicas presentan tiempos similares y se observa una disminución del tiempo a medida que se incrementa el número de hilos. Para matrices medianas (1200x1200), mmClasicaFork muestra tiempos iniciales más altos, pero todas las demás se estabilizan con 4 o más hilos. En el caso extremo (2400x2400), mmFilasOpenMP demuestra el mejor rendimiento general, mientras que mmClasicaOpenMP y mmClasicaPosix presentan tiempos significativamente más altos, especialmente mmClasicaOpenMP que mantiene tiempos elevados incluso con 32 hilos. En todos los casos, el mayor beneficio de la paralelización se observa entre 1 y 4 hilos, estabilizándose posteriormente.

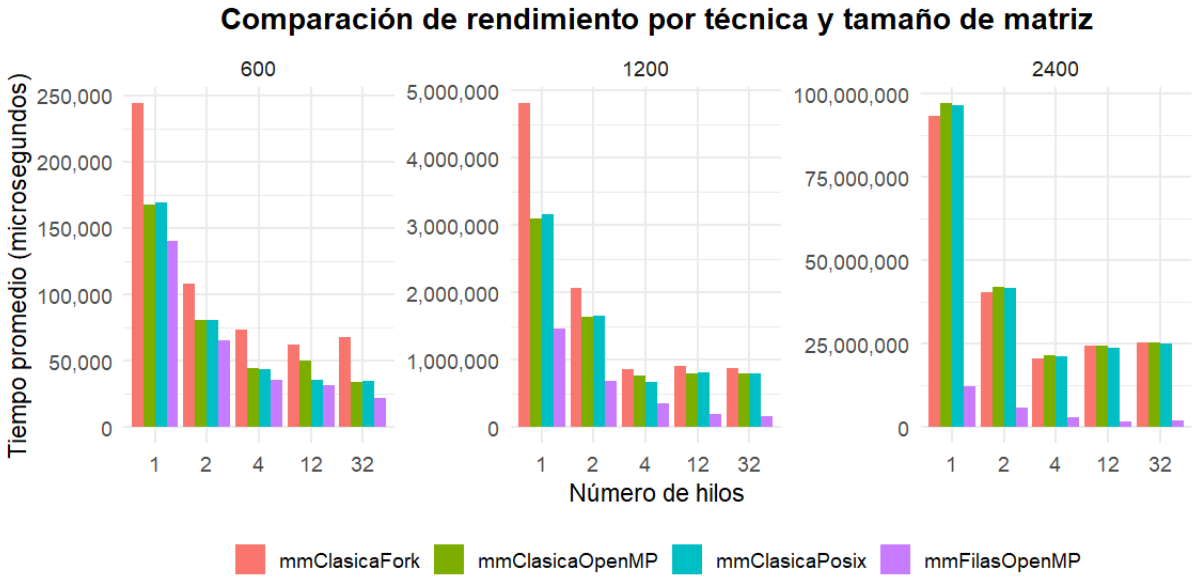

Figura 8: Comparación de rendimiento por técnica y tamaño de matriz

En la siguiente gráfica de la máquina 1 se obtuvieron los mismos resultados que la Figura 7, sin embargo, se presenta ahora un gráfico de barras, para evidenciar a partir de tres tamaños, pequeño, mediano y grande, respectivamente, el rendimiento de cada método, en la gráfica de barras se observa que para matrices pequeñas (600x600), mmClasicaFork presenta los tiempos más altos con 1 hilo, pero todas las técnicas se estabilizan con 4 o más hilos. Para matrices medianas (1200x1200), mmClasicaPosix muestra un rendimiento ligeramente inferior con 1 hilo, mientras que las demás técnicas mantienen tiempos similares al aumentar los hilos. En el caso de matrices grandes (2400x2400), todas las técnicas excepto mmFilasOpenMP presentan tiempos parecidos desde 1 hilo en adelante, sugiriendo que el cuello de botella deja puede ser por otro factor como al acceso de la memoria. El mayor impacto en la reducción de tiempo se observa al pasar de 1 a 2 hilos en todos los tamaños de matriz.

4.2. Máquina de Danna

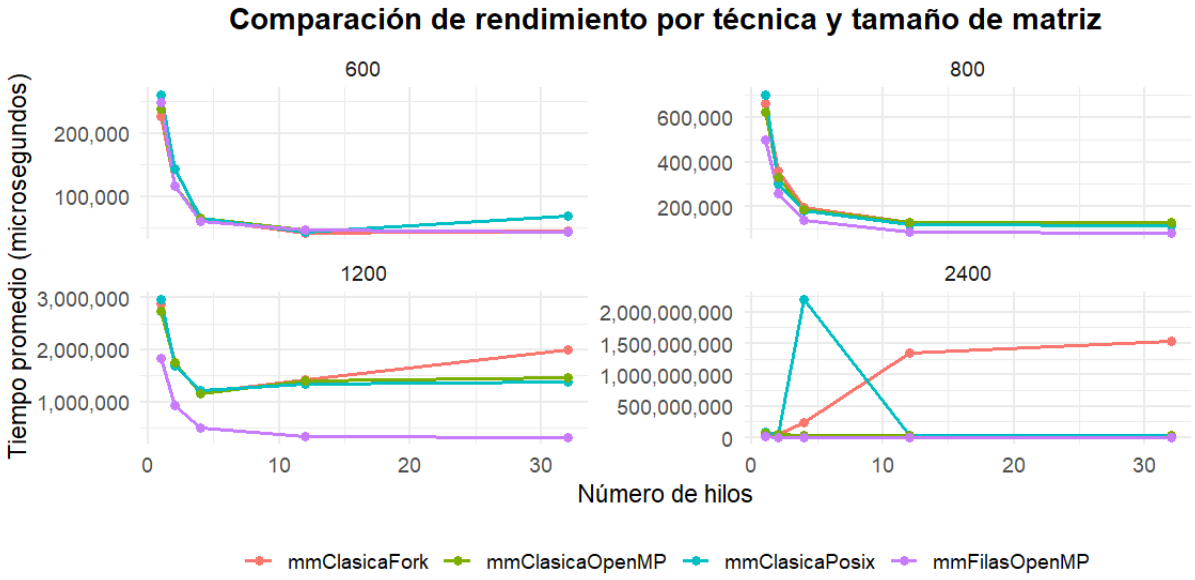

Figura 9: Comparación de rendimiento por técnica y tamaño de matriz

En la gráfica se observa que para matrices pequeñas (600x600 y 800x800) todas las técnicas convergen rápidamente a tiempos similares al aumentar los hilos. Para matrices medianas (1200x1200), mmFilasOpenMP demuestra el mejor rendimiento, mientras que mmClasicaFork muestra una tendencia creciente inusual al aumentar los hilos. En el caso extremo (2400x2400), se presenta un comportamiento crítico en mmClasicaPosix con un pico de tiempo demasiado alto en 4 hilos, seguido de una caída drástica, mientras que mmClasicaFork mantiene tiempos altos y crecientes. mmFilasOpenMP demuestran ser la técnica más estable y eficiente en todos los escenarios.

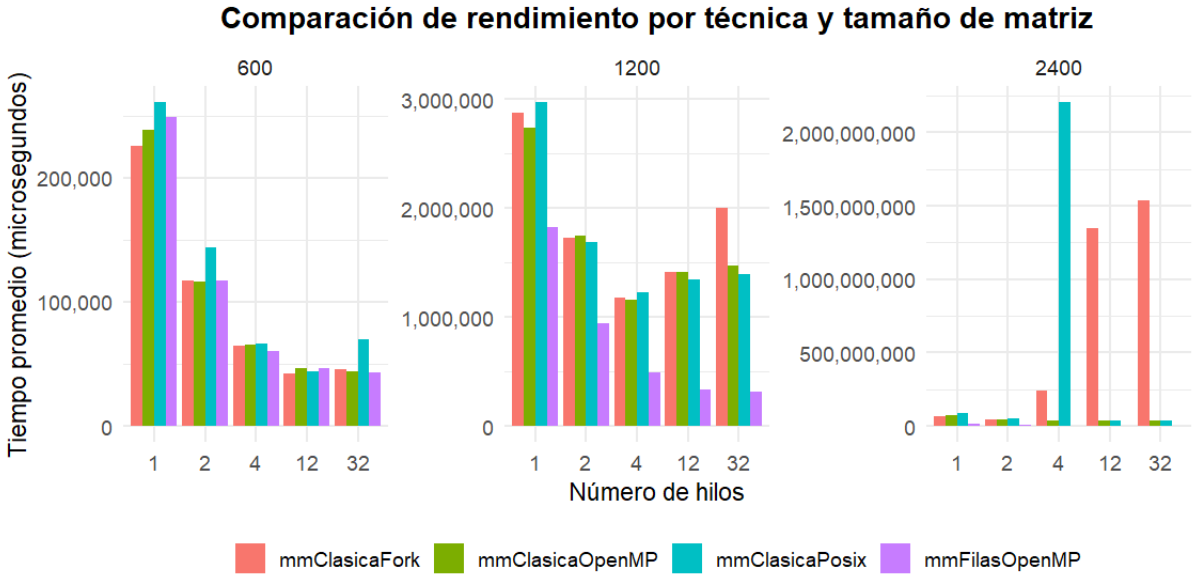

Figura 10: Comparación de rendimiento por técnica y tamaño de matriz

En la gráfica de barras se observa que para matrices pequeñas (600x600), todas las técnicas presentan tiempos iniciales similares con 1 hilo y convergen rápidamente a valores bajos con 4 o más hilos. Para matrices medianas (1200x1200), mmFilasOpenMP demuestra el mejor rendimiento, mientras que las demás técnicas mantienen tiempos similares entre sí. En el caso extremo (2400x2400), se observa un comportamiento curioso, mmClasicaPosix presenta un pico alto con 4 hilos, mmClasicaFork muestra tiempos elevados con 12 y 32 hilos, mientras que mmFilasOpenMP y mmClasicaOpenMP mantienen tiempos consistentemente bajos. Esto sugiere problemas específicos de escalabilidad en Fork y Posix para matrices muy grandes.

4.3. Máquina de Maria Fernanda

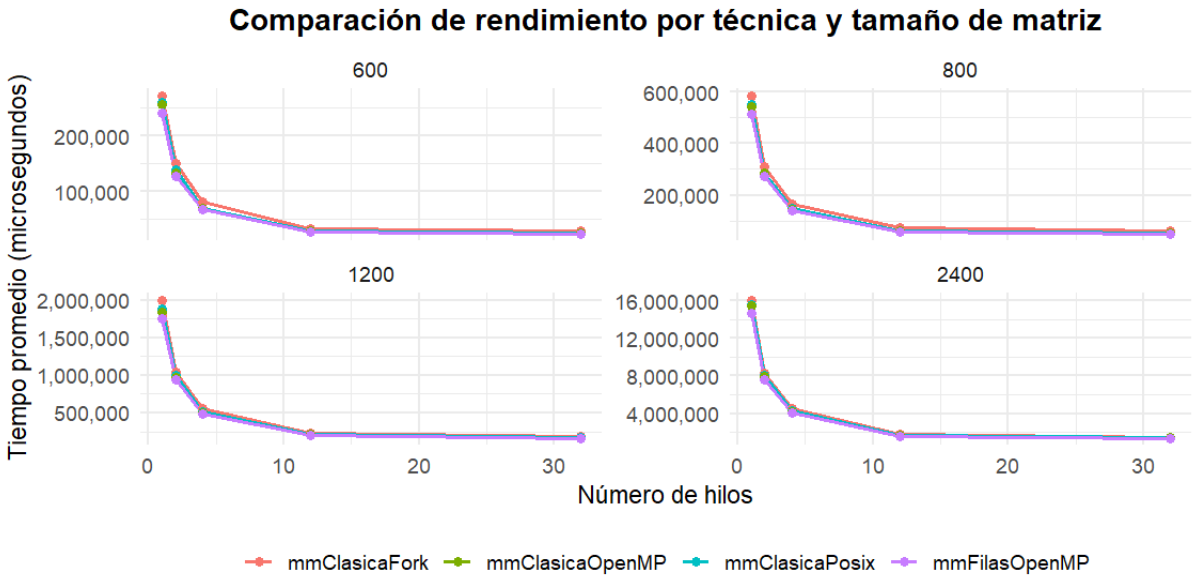

Figura 11: Comparación de rendimiento por técnica y tamaño de matriz

La gráfica muestra cómo el rendimiento mejora de forma significativa al aumentar el número de hilos en todas las técnicas, especialmente en los hilos 1, 2, 4. Donde se observa una reducción drástica del tiempo de ejecución. A partir de 12 hilos, las curvas comienzan a estabilizarse y el beneficio adicional es menor, lo que indica que el sistema alcanza un punto de saturación donde el overhead de sincronización y la competencia por recursos limitan la ganancia. Además, para todos los tamaños de matriz y técnicas (Fork, POSIX y OpenMP), los comportamientos son muy similares, lo que sugiere que el cuello de botella principal proviene de la arquitectura física de la máquina más que del modelo utilizado. En general, la gráfica confirma que la paralelización acelera la multiplicación de matrices, pero también evidencia que el rendimiento deja de escalar linealmente a medida que se incrementa el número de hilos.

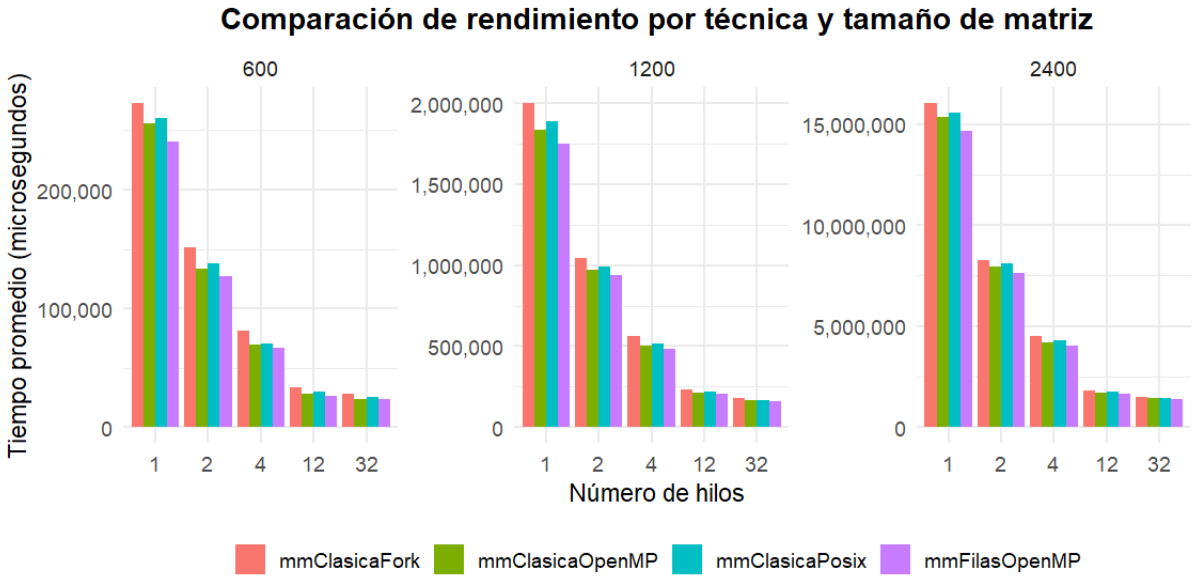

Figura 12: Comparación de rendimiento por técnica y tamaño de matriz

La gráfica muestra que todas las técnicas reducen el tiempo a medida que aumentan los hilos, con una mejora entre 1 y 4 hilos. Después de ese punto, la caída en el tiempo es menor, lo que indica que la máquina llega a su límite para sacar provecho de los hilos. Las cuatro implementaciones tienen rendimientos muy similares, así que la diferencia entre técnicas es mínima frente al efecto del número de hilos. En resumen, agregar hilos acelera la multiplicación, pero el beneficio deja de ser proporcional cuando se supera la capacidad real del hardware.

4.4. Máquina de Giovanny

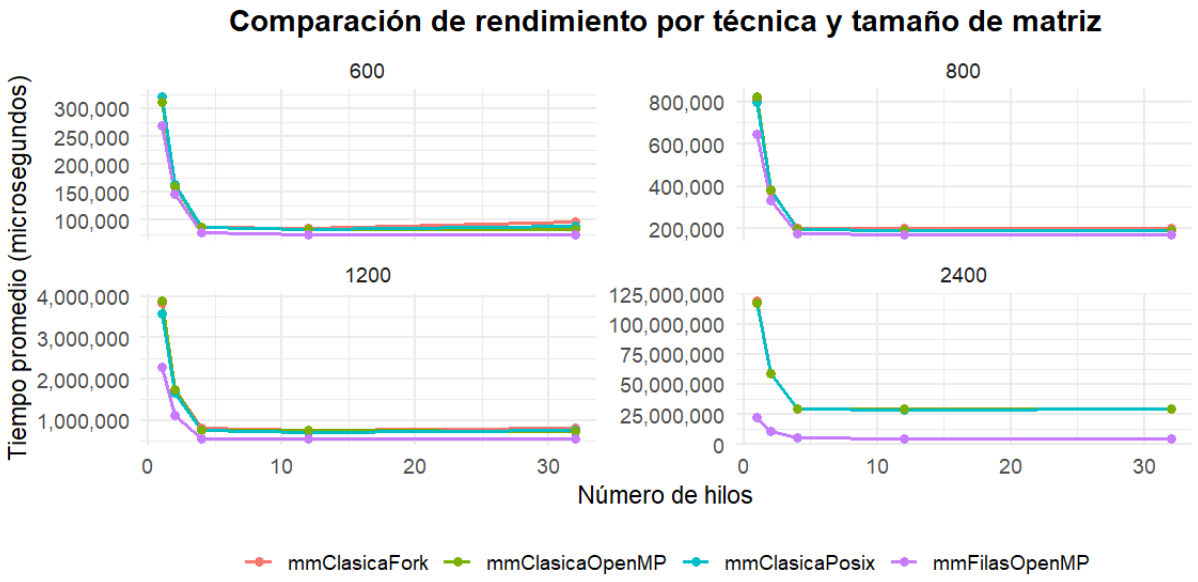

Figura 13: Comparación de rendimiento por técnica y tamaño de matriz

La gráfica muestra que el tiempo de ejecución cae de manera muy pronunciada cuando se pasa de 1 a pocos hilos, y luego se estabiliza casi horizontalmente. Esto indica que el paralelismo aporta una mejora fuerte al inicio, pero después de cierto número de hilos la ganancia adicional es mínima porque la máquina llega a su límite físico de procesamiento. Todas las técnicas presentan un comportamiento muy similar para los distintos tamaños de matriz, lo que evidencia que la reducción del tiempo depende más del hardware que de la técnica usada. En general, aumentar hilos acelera el cálculo, pero el beneficio deja de crecer cuando se supera la capacidad real de la CPU.

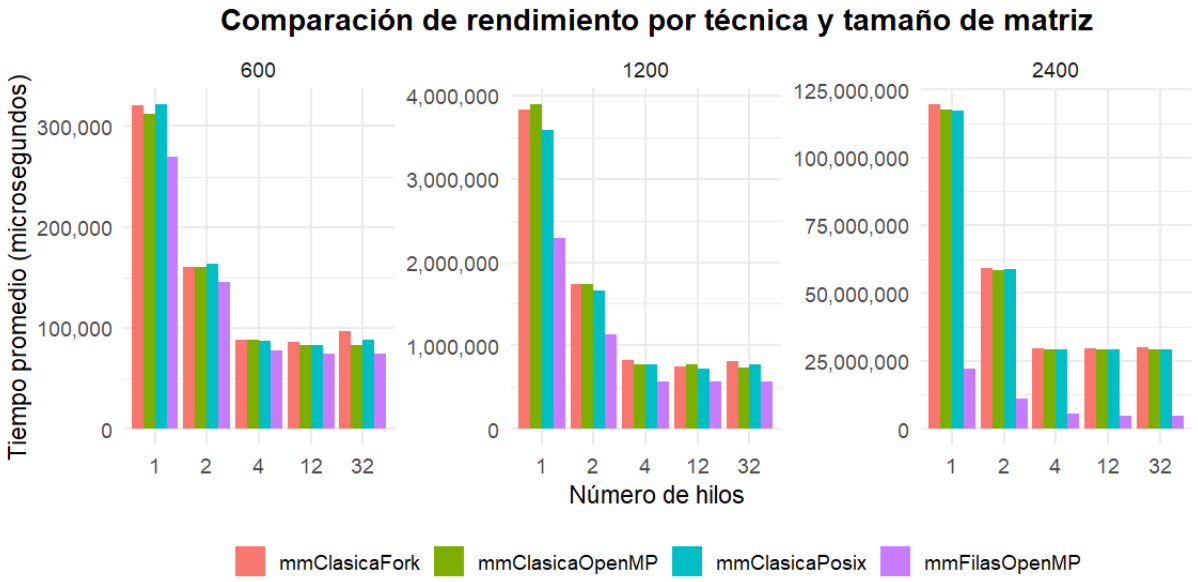

Figura 14: Comparación de rendimiento por técnica y tamaño de matriz

La gráfica muestra que todas las técnicas reducen significativamente su tiempo de ejecución al aumentar el número de hilos, evidenciando un claro beneficio de la paralelización. Sin embargo, no todas escalan de la misma manera, algunas técnicas como mmFilasOpenMP obtienen tiempos considerablemente menores, especialmente en matrices grandes, mientras que otras mantienen un rendimiento más uniforme pero menos eficiente. También se observa que, a medida que el tamaño de la matriz aumenta, las diferencias entre técnicas se vuelven más pronunciadas, indicando que la elección del método de paralelización es más crítica en problemas de mayor complejidad.

4.5. Máquina Virtual

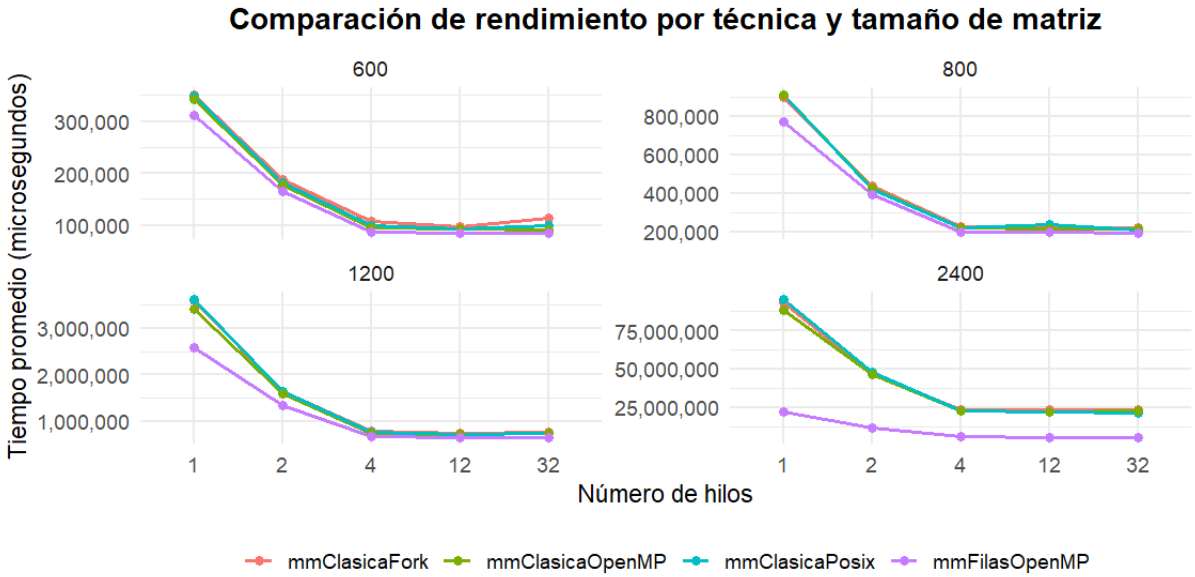

Figura 15: Comparación de rendimiento por técnica y tamaño de matriz

En el caso de la máquina 5, se evidencia una disminución transitoria en el tiempo de ejecución, donde a medida que aumenta el numero de hilos disminuye en consecuencia el tiempo. Sin embargo, a partir del uso de 4 hilos se evidencia una disminución de la ventaja de usar varios hilos, en la máquina 1 se disminuye el beneficio por este aumento en todos los casos de matrices. Por otro lado, se observa un rendimiento considerablemente mejor en términos de disminución del tiempo con mmFilasOpenMP, principalmente cuando la matriz es de tamaño 2400x2400, donde esta diferencia se hace evidente debido al gran número de iteraciones involucradas. En cambio, con matrices más pequeñas esta ventaja no alcanza a manifestarse de manera significativa. No obstante, en esta máquina los demás métodos fork(), Posix() y OpenMP() presentan diferencias mínimas entre sí.

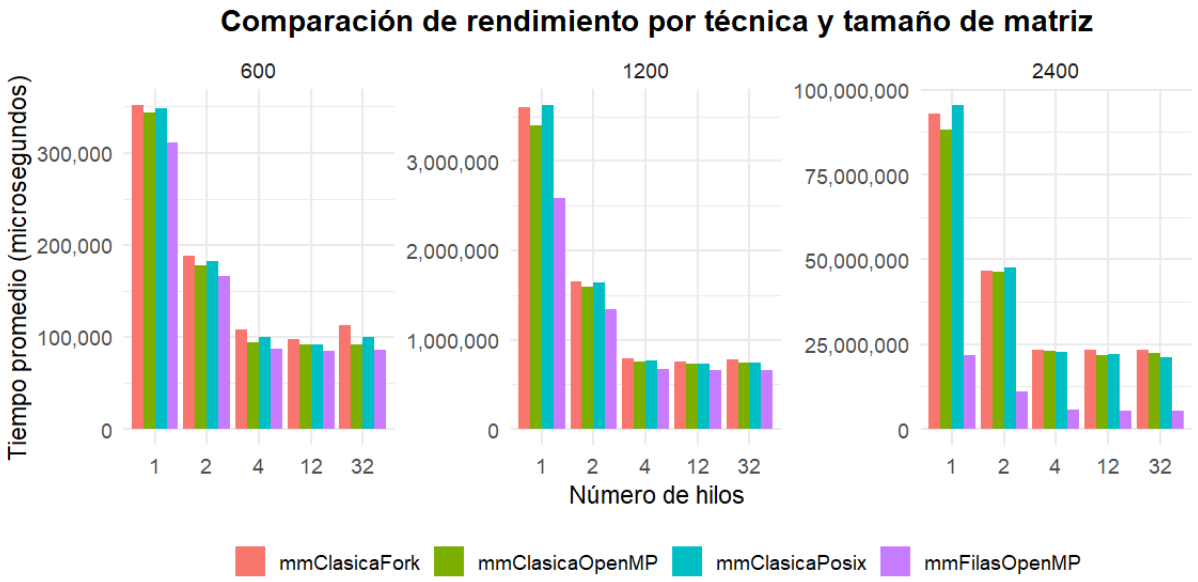

Figura 16: Comparación de rendimiento por técnica y tamaño de matriz

El rendimiento de cada método logra evidenciar nuevamente la misma tendencia hasta llegar a los 4 hilos, ya que no se ve una ventaja clara en la disminución de tiempos. Además, se observa un mejor rendimiento de la función mmFilasOpenMP, especialmente a medida que aumenta el tamaño de la matriz, donde su ventaja se vuelve más evidente. Por otro lado, las demás funciones no muestran diferencias en su ejecución.

4.6. Comparación entre máquinas

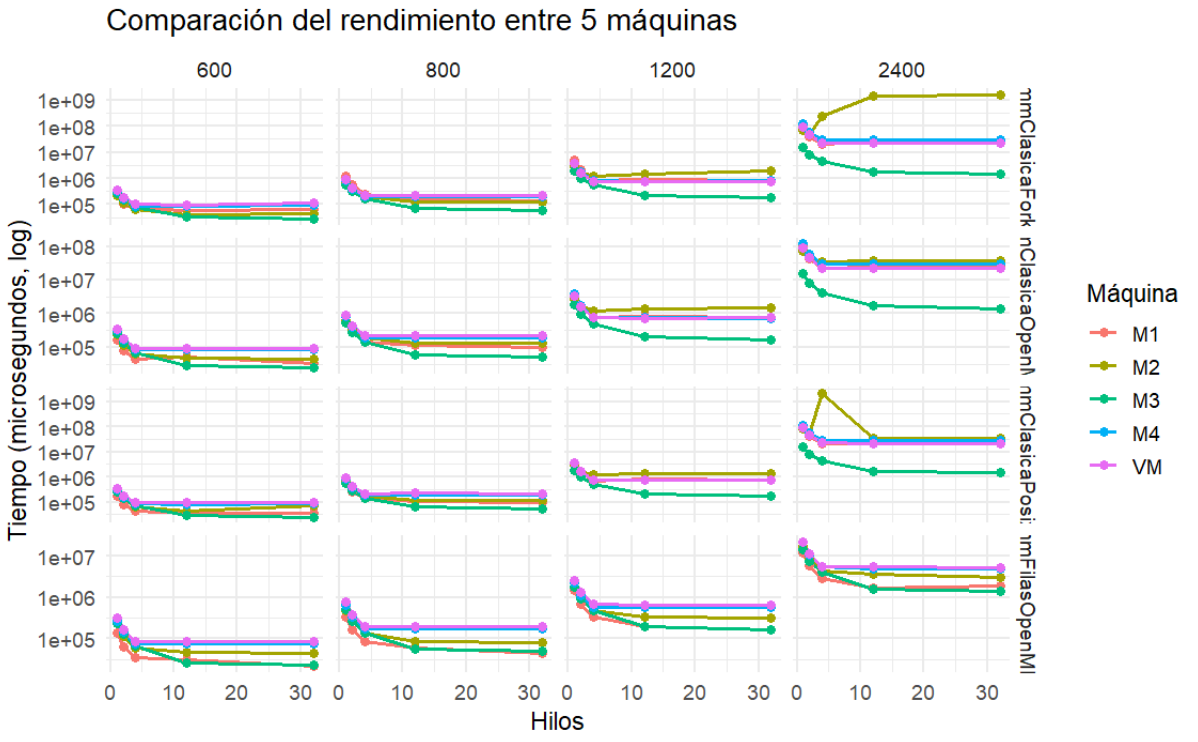

Figura 17: Comparación del rendimiento entre 5 máquinas

La gráfica compara el rendimiento de cinco máquinas ejecutando distintas técnicas de multiplicación de matrices en varios tamaños y números de hilos. En general, todas las máquinas muestran una disminución rápida del tiempo de ejecución al aumentar los hilos, lo que indica un buen aprovechamiento del paralelismo. Las diferencias entre máquinas son más evidentes en tamaños grandes, donde algunas presentan picos o tiempos notablemente mayores, sugiriendo limitaciones en su capacidad de cómputo o administración de hilos. En cambio, para tamaños pequeños o medios, las cinco máquinas mantienen comportamientos muy similares. Por otro lado, la máquina 3 demostró unos resultados más favorables en comparación a las demás máquinas con métodos como fork(), Posix() y OpenMP, donde las otras 4 fueron muy parecidas en los tiempos de ejecución. Sin embargo, algo relevante fue que con el método mmFilasOpenMP, todas tuvieron tiempos más similares en todos los tamaños llegando a tener un rendimiento parecido al de la máquina 3.

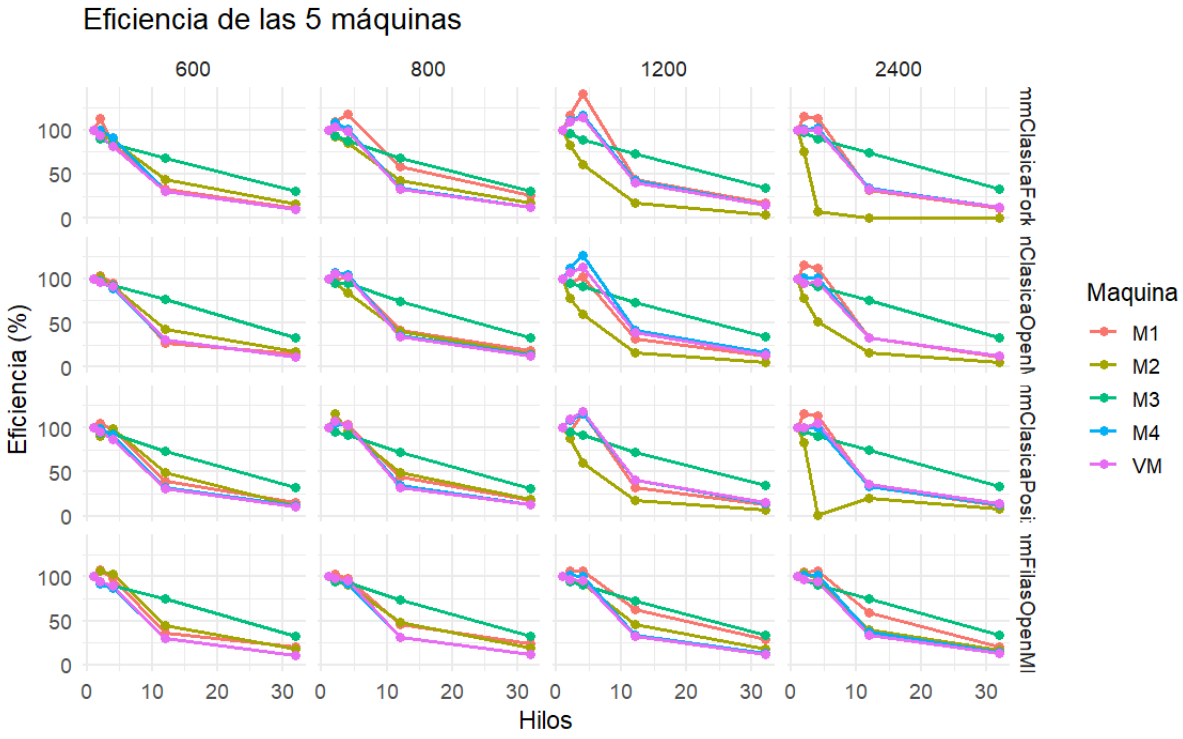

Figura 18: Comparación de eficiencia entre las 5 máquinas

En la Figura 18 se busca analizar la relación que existe entre el porcentaje de eficiencia que hay con respecto al número de hilos por cada función, para comprender como se consideró en este taller la evidencia, la eficiencia es un dato que se obtiene comparando el tiempo con 1 hilo frente al tiempo con n hilos; luego, ese valor se divide entre n para ver cuánto aporta en promedio cada hilo adicional. Es útil dado que revela el punto a partir del cual agregar más hilos deja de mejorar el rendimiento, debido a factores como sincronización, sobrecarga del sistema operativo, comunicación entre hilos o limitaciones del hardware. Así, la eficiencia nos permite identificar cuándo un algoritmo escala bien, cuándo empieza a deteriorarse y cuál es el número óptimo de hilos para cada técnica y tamaño de matriz. Dado la gráfica a medida que aumenta el número de hilos, la eficiencia general de las máquinas tiende a disminuir. Del comportamiento específico de las máquinas, se deduce que m3, y en algunos casos m1, logran mantener una mayor eficiencia (cercana al 100) incluso con un número creciente de hilos, lo que sugiere que su hardware o

configuración gestiona mejor la sobrecarga. Por el contrario, las demás gráficas muestran una caída más evidente en su eficiencia, lo que indica que son mucho más sensibles a la escalabilidad.

5. Análisis y discusión

En esta parte se revisaron los resultados obtenidos en las cinco máquinas, comparando cómo se comportó cada una bajo las cuatro técnicas usadas: *Fork*, *POSIX Threads*, *OpenMP básico* y *OpenMP por filas*. Y lo primero que se evidenció es que, aunque todas las máquinas ejecutaban exactamente el mismo código, los tiempos cambiaban bastante dependiendo del hardware, especialmente por el número de núcleos (CoREs), la frecuencia base y el tamaño de la caché.

En general, la implementación con *Fork* fue la que obtuvo los tiempos mas lentos, aunque no sea tan evidente en todos los casos. Pero esto pasa porque cada proceso creado es independiente y el sistema debe copiar memoria y manejar más contexto, lo cual genera bastante sobrecarga. Aunque si es verdad, que las diferencias se notaron más en matrices con tamaño "pequeño"(600) y "mediano"(1200), al final con el tamaño probado más grande se igualaron significativamente los tiempos.

En el caso de *POSIX Threads*, los tiempos mejoraron, solo en algunos casos y en otros se mantuvieron a la par de la implementación con Fork, porque los hilos comparten memoria y eso evita el costo de crear procesos completos. Sin embargo, su ventaja depende del balance que logre cada máquina entre los hilos, cosa que explica porque no se mejoró sustancialmente en comparación con Fork y el tamaño de matriz. En algunos casos los hilos no escalaron bien con tamaños grandes, lo que puede ser efecto de la sincronización interna y del acceso a memoria compartida.

Con *OpenMP*, en su primera versión, el desempeño fue más estable y el hecho de que el compilador distribuya las iteraciones ayudó a mantener un comportamiento más predecible y uniforme en varias máquinas. De todas formas, no siempre fue la mejor opción, especialmente cuando la máquina no tenía suficientes núcleos para aprovechar el paralelismo o cuando el tamaño de la matriz era demasiado pequeño.

La versión optimizada de *OpenMP*, la cual fue basada en distribuir las filas entre los hilos, sí mostró mejoras más evidentes con respecto al resto, porque reparte el trabajo de forma más equilibrada y en casi todas las máquinas esta fue la técnica con mejor tiempo cuando las matrices eran grandes, entonces el uso de filas completas y la distribución más ordenada ayudo a recudir los tiempos de espera y evitó que algunos hilos terminaran mucho antes que otros.

Finalmente, en la comparación entre máquinas se mostró que los núcleos y la caché L3 son determinantes. Lo anterior se debe a que fueron las máquinas con mayor capacidad de caché las que obtuvieron mejores tiempos en tamaños grandes, incluso si tenían menos hilos. Esto se vio especialmente en los equipos con procesadores Xeon, donde a pesar de no

tener tantos hilos, su caché grande ayudó a reducir los tiempos en matrices muy grandes. Al final, la técnica más rápida no fue la misma para todas las máquinas ni para todos los tamaños, pero lo que sí se mantiene es que al crecer el tamaño de la matriz, la forma en que se reparte el trabajo y la jerarquía de memoria del hardware terminan siendo más importantes que la cantidad de hilos disponible.

6. Conclusiones

A partir del trabajo realizado se pudo ver cómo las distintas formas de paralelismo afectan el rendimiento dependiendo tanto de la técnica usada como del hardware disponible además, en las pruebas se evidenció que *Fork* es la alternativa menos eficiente por el costo de crear procesos completos, mientras que *OpenMP* dio mejoras sustanciales.

Por lo dicho anteriormente, *OpenMP* resultó ser una herramienta más práctica y equilibrada, ya que permite paralelizar el código sin administrar manualmente los hilos y adicional la versión optimizada por filas fue la más estable y con mejores tiempos en la mayoría de los casos, sobre todo en matrices grandes.

También quedó claro que el rendimiento no solo depende del método, sino del hardware, cosas como, la cantidad de núcleos, la velocidad del procesador y, sobre todo, el tamaño de la caché L3 influyen directamente en los tiempos obtenidos entonces es por esto, la técnica "ganadora" puede cambiar según la máquina.

Referencias

[1] A. Silberschatz, P. Galvin y G. Gagne, *Operating Systems Concepts*, 9th ed. Wiley, 2018.

[2] M. J. Bach, *The Design of the UNIX Operating System*. Prentice-Hall, 1986.

[3] B. Nichols, D. Buttlar y J. Farrell, *Pthreads Programming*. O'Reilly Media, 1996.

[4] R. Chandra et al., *Parallel Programming in OpenMP*. Morgan Kaufmann, 2000.