Taller de Evaluación de Rendimiento

Juan José Ballesteros

Juan Diego Rojas

Nicolas Pinilla

Enlace al repositorio personal:

https://github.com/2005-jjbs/SistemasOperativos2025-03/tree/main/Taller_Rendimiento_

Pontificia Universidad Javeriana

Sistemas Operativos

Profesor: John Jairo Corredor Franco

13 de noviembre de 2025

1. Objetivos

- Evaluar comparativamente el rendimiento de tres técnicas de paralelización (POSIX Threads, OpenMP y procesos con fork()) en la multiplicación de matrices.
- Analizar la influencia del hardware y configuración del sistema en la eficiencia de algoritmos paralelos.

2. Introducción

En la actualidad al hablar de los sistemas operativos es necesario hacer mención del concepto del procesamiento paralelo, este se presenta como un recurso clave para mejorar el rendimiento de las aplicaciones, en especial aquellas que involucran una serie de operaciones para realizarse como lo es el caso de la multiplicación de matrices. El rendimiento en ultimas es una de las cosas más buscadas, sin embargo, una de las más difíciles de conseguir en este contexto. Este taller va enfocado en la evaluación del rendimiento de tres enfoques de programación paralela implementados en el lenguaje de programación C, estos son: POSIX threads, OpenMP y procesos con fork(). La demostración se realizará a través de la ejecución de distintas pruebas hechas en diferentes entornos de cómputo, preferiblemente con características diferentes, y usando distintos tamaños de matrices, esto con el fin de comparar la eficiencia de estos algoritmos en función del número de hilos utilizados y las diferentes arquitecturas de los dispositivos. El resultado de los tiempos de ejecución de estos algoritmos permitirá identificar limitaciones, ventajas y en especial, comportamientos particulares de cada uno de estos, lo cual nos llevaría a evidenciar unas posibles ventajas o desventajas de la implementación o uso del paralelismo.

A través de este taller, se busca comprender como el entorno de ejecución, la técnica de paralelización empleada y el número de hilos influyen o afectan el desempeño de la operación de multiplicación. Además, surge como objetivo la identificación de debilidades y fortalezas que presentan los diferentes sistemas de cómputo, teniendo en cuenta aspectos como la sobrecarga de creación de hilos o procesos, la escalabilidad, la contención de recursos y la eficiencia en el uso del hardware disponible. Por último, el análisis realizado en el taller podría construir un cimiento, una base, que ayude al proceso de tomar decisiones a la hora de desarrollar un programa para lograr la máxima eficiencia y rendimiento posibles.

3. Resumen

3.1 Planteamiento del Problema

A pesar de los avances en la computación paralela y la disponibilidad de múltiples núcleos en los procesadores modernos, no siempre se logra un aumento proporcional en el rendimiento al aplicar diversas técnicas de concurrencia. En la práctica, el desempeño de un programa paralelo depende de diversos factores como el número de hilos, la arquitectura del procesador, la distribución de memoria caché, la sobrecarga del sistema operativo y el tipo de técnica de paralización utilizada.

En este contexto, surge la necesidad de analizar de forma experimental cómo influyen estas variables en el rendimiento real de un algoritmo de multiplicación de matrices. Si bien las técnicas de paralelismo como POSIX threads, OpenMP o el uso de fork() prometen mejorar la eficiencia, su impacto puede variar significativamente entre distintos entornos de ejecución y configuraciones de hardware.

El problema central se plantea entonces en determinar cuál de los enfoques de programación paralela ofrecen un mayor rendimiento bajo distintas condiciones, y cómo las características de los equipos de cómputo modifican los resultados obtenidos. Además, es necesario identificar las posibles causas de degradación del desempeño, como la contención de recursos, la falta de escalabilidad o la sobrecarga del manejo de hilos.

En consecuencia, el taller busca dar respuesta a la problemática del rendimiento variable en algoritmos paralelos, mediante la comparación controlada de tres técnicas de programación paralela en C, evaluada sobre diferentes arquitecturas y con diversos tamaños de matrices, para comprender las condiciones bajo las cuales el paralelismo resulta realmente beneficioso.

3.2 Propuesta de Solución

La solución propuesta para abordar el problema de variabilidad en el rendimiento de los algoritmos paralelos consiste en diseñar y ejecutar un estudio experimental controlado, siguiendo un enfoque sistemático de evaluación de desempeño. El objetivo central es comparar el comportamiento de los algoritmos de multiplicación de matrices implementados con POSIX threads, OpenMP y procesos con fork(), evaluados en distintos entornos de cómputo y bajo distintas configuraciones de concurrencia.

Para logarlo, se plantea la siguiente estrategia metodológica:

1) Modularización y documentación del código:

Siguiendo las especificaciones del taller, el código fuente será reorganizado en:

- Implementación de funciones (.c)
- Archivos de Interfaz (.h)
- Archivo principal (main.c)

Cada función será documentada, y se verificará el correcto funcionamiento de los métodos de multiplicación antes de proceder a las mediciones, cumpliendo con los requisitos del taller.

2) Construcción de un Makefile y validación de compilación:

Se utilizará un Makefile para automatizar la compilación y garantizar:

- Compilación homogénea en todos los equipos
- Reducción de errores manuales
- Generación de ejecutables diferenciados por técnica (POSIX, OpenMP, fork, clásica).

3) Diseño del experimento de rendimiento:

El diseño experimental considerará tres variables fundamentales:

-Tamaño de la matriz:

Se seleccionarán diferentes dimensiones teniendo en cuenta: jerarquía de memoria, capacidad de cómputo de cada equipo, necesidad de observar escalabilidad.

-Número de hilos:

Para cumplir las exigencias del taller se evaluará: serie (1 hilo) y paralelo (2, 4, 8, ... según la maquina).

Esto permite comparar escalabilidad y una sobrecarga de cada enfoque, como exige el diseño de experimentos.

-Equipos con diferente hardware y SO:

Cada equipo se caracterizó previamente (procesador, arquitectura, cachés, etc). Esto permite analizar cómo influye el harware real o virtualizado en ejecución.

4) Batería de experimentos con repetición estadística:

Para cada combinación se ejecutarán 30 repeticiones, siguiendo lo estipulado en el taller sobre la necesidad de aplicar la ley de los grandes números para obtener valores promedio representativos.

Los datos se procesarán para obtener: promedio, velocidad al aumentar hilos, velocidad relativa entre algoritmos.

5) Automatización con el script lanzador.pl:

El archivo lanzador.pl servirá para:

-Iterar sobre todos los tamaños de matriz.

-Iterar sobre la lista de hilos definidos.

-Ejecutar 30 veces cada caso.

-Almacenar los resultados en archivos .dat separados por experimento.

El fin es evitar intervención humana, evitar errores manuales y asegurar reproducibilidad, tal como requiere el taller.

6) Recolección y análisis de datos

Los .dat generados se exportarán a Excel, donde se:

-calcularán promedios

-generarán gráficas comparativas por máquina

-compararán algoritmos

-analizarán comportamientos de escalabilidad

-identificarán cuellos de botella

7) Interpretación, conclusiones y recomendaciones:

Finalmente, los resultados experimentales permitirán:

-determinar cuál técnica paralela presenta mejor rendimiento

-evaluar la influencia del hardware

-detectar la existencia de sobrecarga de hilos o procesos

-comprender cuándo la paralelización deja de ser beneficiosa

-proponer recomendaciones sobre modelos de ejecución eficientes

3.3 Método de Prueba

El método de prueba usado en este taller se basa en un proceso experimental sistemático orientado a medir, comparar y analizar el rendimiento de distintos algoritmos de multiplicación de matrices bajos diferentes configuraciones de concurrencia y distintos entornos de hardware. Para garantizar la reproducibilidad y la validez estadística de los resultados se siguieron los lineamientos del diseño experimental propuesto en el taller, complementado con buenas prácticas de evaluación de desempeño.

El método se desarrolló en las siguientes etapas:

1) **Preparación y validación del entorno de pruebas**

   Antes de realizar cualquier medición se consolidó la estructura del proyecto separando los ficheros fuente en:

   -Archivo principal

   -Encabezado de funciones de multiplicación

   -Archivo de interfaz

   Se validó que cada implementación de multiplicación (serial, POSIX threads, OpenMP y fork()) funcionara correctamente en matrices pequeñas (menores a 5x5), como recomienda el taller, asegurando que los resultados matemáticos fueran correctos y consistentes entre algoritmos.

2) Automatización de ejecuciones mediante lanzador.pl

Para garantizar uniformidad y repetibilidad en las pruebas de rendimiento, se utilizó un script en Perl denominado lanzador.pl, encargado de automatizar la ejecución de los programas implementados en diferentes modelos de paralelismo.

El script realiza la ejecución sistemática de cada programa bajo distintas condiciones controladas, variando el tamaño de las matrices de entrada y el número de hilos o procesos utilizados. Este enfoque permite recolectar múltiples muestras por configuración, reduciendo la variabilidad y facilitando el cálculo de promedios representativos.

3) Ejecución controlada en múltiples plataformas

Cada algoritmo fue ejecutado en los distintos entornos de cómputo previamente descritos. Para cada equipo se procuró:

- Minimizar procesos en segundo plano

- Cerrar aplicaciones innecesarias

- Ejecutar las pruebas en condiciones comparables

- Registrar las características de hardware relevantes

Esto permitió identificar la influencia del hardware en el desempeño y reducir riesgos asociados a condiciones externas.

4) Recolección y procesamiento estadístico de datos

Cada archivo .dat contiene los tiempos de ejecución para las 30 repeticiones de una misma configuración. Estos valores fueron posteriormente exportados a hojas de cálculo, donde se procesaron según los siguientes criterios:

-Promedio de tiempo de ejecución como métrica principal de desempeño.

-Organización por algoritmo, máquina, tamaño de matriz y número de hilos.

Estas métricas permiten una comparación objetiva entre versiones en serie y paralelas del algoritmo y entre plataformas de hardware.

5) Construcción de gráficas comparativas

Posteriormente, los datos promediados se visualizaron mediante gráficas que muestran:

-evolución del tiempo de ejecución en función del número de hilos

-comparación entre algoritmos con diversas entradas en un mismo equipo

-comparación entre equipos bajo un mismo algoritmo

Estas gráficas constituyen un instrumento fundamental para el análisis, permitiendo detectar cuellos de botella, límites de paralelización y diferencias entre arquitecturas.

6) Análisis comparativo y elaboración de conclusiones

Finalmente, se realizó un análisis detallado de los resultados obtenidos para:

-evaluar el impacto del algoritmo paralelo utilizado

-examinar el comportamiento particular de cada equipo

-identificar mejoras o degradaciones del rendimiento al incrementar los hilos

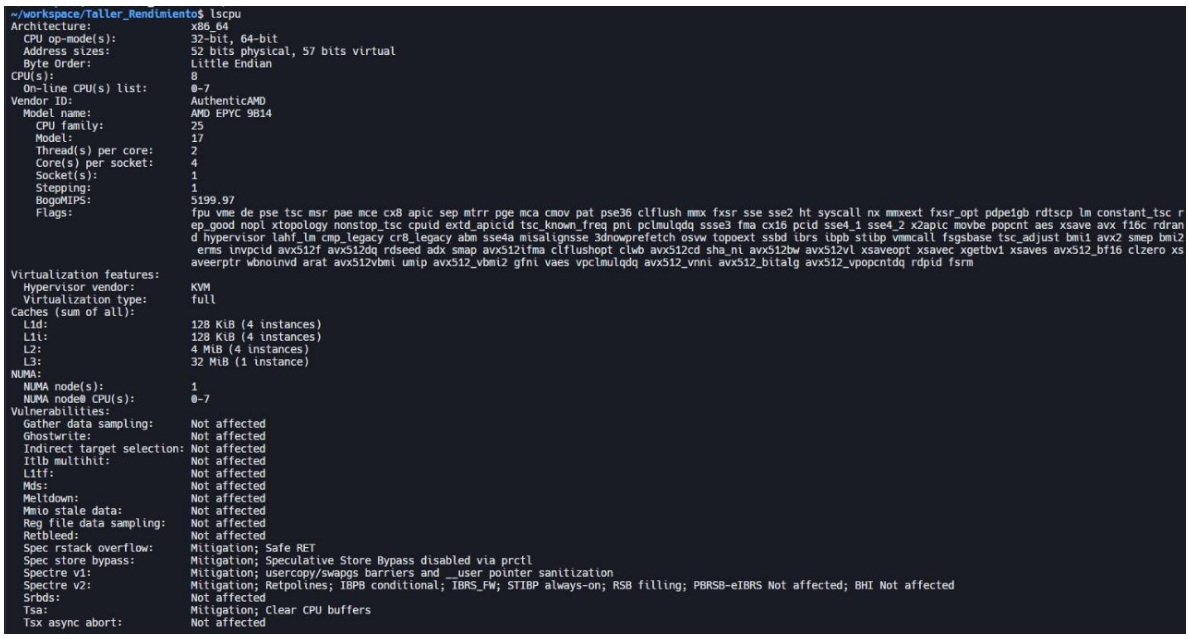
-proponer explicaciones basadas en la arquitectura y la jerarquía de memoria

Este análisis permite responder al problema planteado y fundamentar conclusiones sólidas sobre la eficiencia de cada enfoque de paralelización.

3.4 Descripción de los equipos de cómputo

A continuación, se describen las características de hardware y sistema operativo en los diferentes entornos de cómputo utilizados en el taller. Cada equipo fue identificado mediante el uso del comando **lscpu**, lo que permitió registrar información relevante para el análisis de rendimiento, como número de núcleos, hilos arquitectura, caché y tipo de virtualización. Estas características permiten interpretar adecuadamente las diferencias en tiempos de ejecución observadas entre máquinas.

Equipo 1 – Replit



-Procesador: AMD EPYC 9814 (4 núcleos / 8 hilos, frecuencia base 3.5 GHz aprox.)

-Arquitectura: x86_64

-Virtualización: KVM (Kernel-based Virtual Machine)

-Memoria caché:

-L1: 128 KiB × 4

-L2: 4 MiB × 4

-L3: 32 MiB × 1

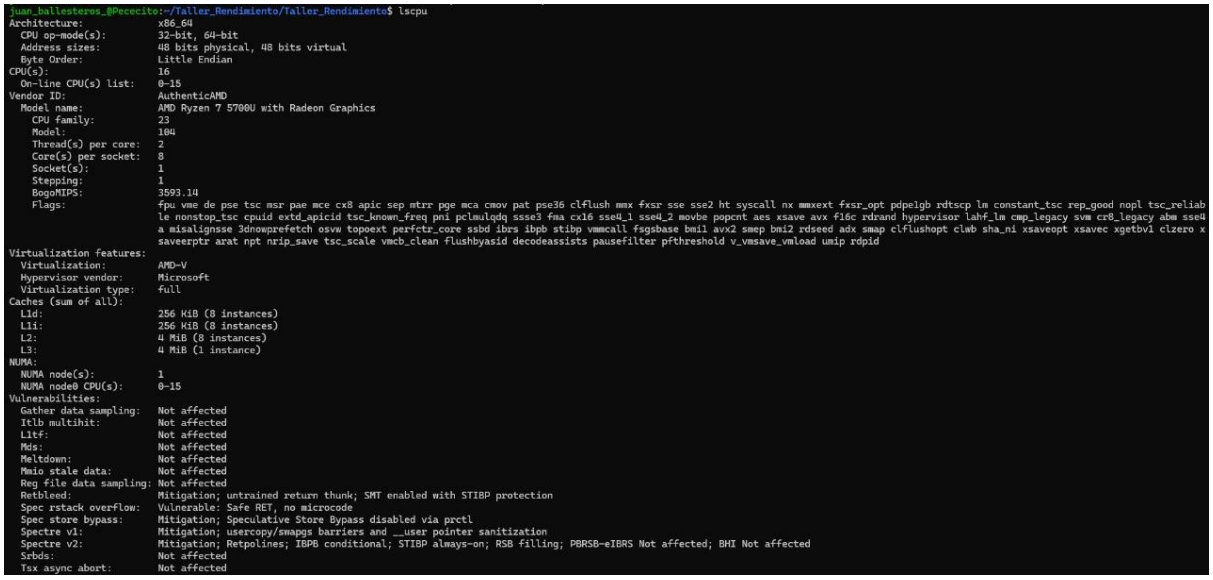
-Sistema Operativo: Linux (versión de kernel provista por Replit, entorno Ubunto/Debian)

-**Tipo de ejecución:** Máquina virtual (entorno cloud)

Este equipo virtualizado permitió ejecutar el algoritmo de multiplicación de matrices tanto en modo serie como paralelo, aprovechando la disponibilidad de múltiples hilos del procesador. La virtualización mediante KVM introduce una capa adicional entre el hardware físico y el sistema operativo, lo cual puede influir en los tiempos de ejecución debido a la gestión del hypervisor y a la distribución compartida de recursos. Sin embargo, el

procesador AMD EPYC está optimizado para cargas paralelas, por lo que el desempeño en ejecuciones con varios hilos resultó estable y representativo para análisis comparativos.

Equipo 2 - Ubuntu



-Arquitectura: x86_64

-CPU: AMD Ryzen 7 5700U

-Núcleos / Hilos: 8 núcleos físicos, 16 hilos.

-Frecuencia: 1.8 GHz – 4.3 GHz

-Memoria caché:

-L1d: 256 KiB × 8

-L1i: 256 KiB × 8

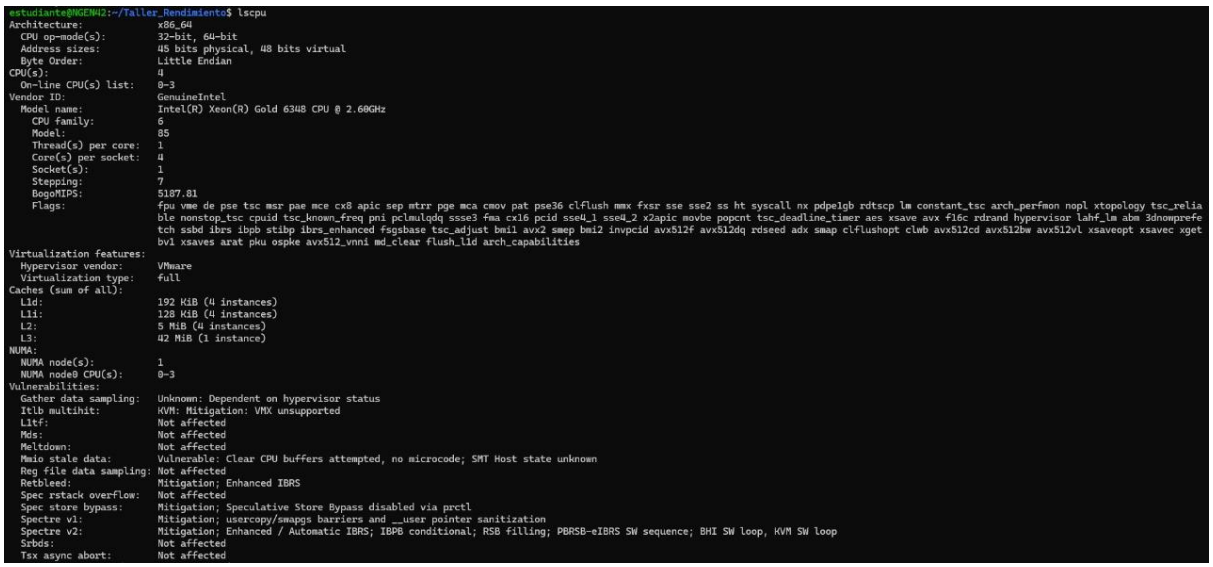
-L2: 4 MiB × 8

-L3: 8 MiB × 1

-Sistema Operativo: Ubuntu bajo WLS2.

Este equipo posee la mayor cantidad de núcleos/hilos entre las plataformas probadas, lo que lo hace ideal para evaluar escalabilidad. No obstante, al ejecutar bajo WLS2, el sistema opera bajo un entorno virtualizado que puede limitar el acceso completo al hardware físico, en especial gestión de memoria y planificación de hilos. Aun así, la CPU es eficiente para paralelización ligera y media.

Equipo 3 - Ubuntu



```
estudiante@NGENH2:~/Taller_Rendimiento$ lscpu
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          48 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 6348 CPU @ 2.60GHz
    CPU family:           6
    Model:                85
    Thread(s) per core:   1
    Core(s) per socket:   4
    Socket(s):            1
    Stepping:             7
    BogoMIPS:             5187.81
    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 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 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:                    192 KiB (4 instances)
  L1i:                    128 KiB (4 instances)
  L2:                     5 MiB (4 instances)
  L3:                     42 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
```

-Arquitectura: x86_64

-CPU: Intel Xeon Gold 6348 @ 2.6 GHz

-Núcleos / Hilos: 4 núcleos físicos, 8 hilos.

-Memoria caché:

-L1d: 192 KiB × 4

-L1i: 128 KiB x 4

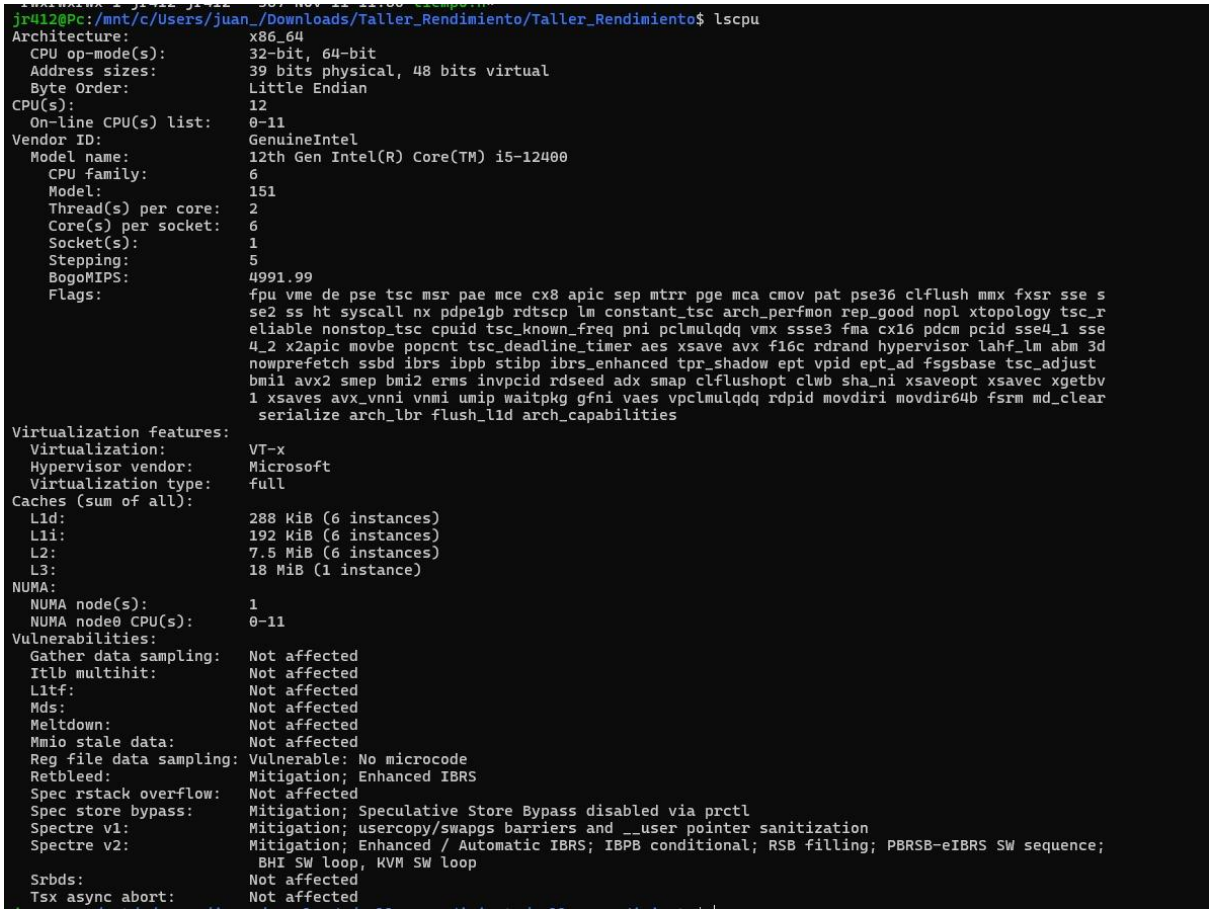
-L2: 4 MiB × 4

-L3: 15 MiB × 1

-Sistema Operativo: Ubuntu

Aunque el Xeon Gold 6348 es un procesador de alto rendimiento, la máquina virtual solo tiene asignada una fracción de los núcleos/hilos disponibles físicamente. Esto afecta directamente la escalabilidad y puede introducir tiempos más altos.

Equipo 4 - Ubuntu



-Arquitectura: x86_64

-CPU: 12th Gen Intel Core i5-12400

-Núcleos / Hilos: 6 núcleos físicos, 12 hilos.

-Memoria caché:

-L1d: 288 KiB × 6

-L1i: 192 KiB x 6

-L2: 7.5 MiB × 6

-L3: 18 MiB × 1

-Sistema Operativo: Ubuntu en WSL2

El Intel Core i5-12400 cuenta con un excelente rendimiento por núcleo y buena jerarquía de caché, lo que beneficia operaciones intensivas como la multiplicación de matrices. Sin embargo, al ejecutarse bajo WSL2, existe una capa de virtualización que puede limitar la eficiencia en hilos múltiples. Aun así, es uno de los equipos con mayor estabilidad de tiempos esperada gracias a su arquitectura moderna.

3.5 Resultados Esperados

1) Diferencias de rendimiento entre algoritmos paralelos

Cada enfoque de paralelización presenta características particulares, por lo que se esperan comportamiento diferenciados:

-POSIX threads (pthread): debería ofrecer una buena escalabilidad y tiempos relativamente bajos debido a su control fino sobre la creación y gestión de hilos.

-OpenMP: se espera un desempeño competitivo, especialmente en equipos con buen soporte de compilador y múltiples núcleos, dado su bajo nivel de sobrecarga en paralelización de bucles.

-Fork(): se prevé que presente los peores tiempos entre las versiones paralelas, ya que la creación de procesos independientes tiene un costo significativamente mayor que la creación de hilos y puede generar mayor presión sobre la memoria.

También se espera que la versión clásica secuencial actúe como referencia base para evaluar los beneficios del paralelismo.

2) Variación del rendimiento entre máquinas

Se espera observar diferencias notables entre los equipos utilizados debido a factores como: número de núcleos e hilos físicos disponibles, tamaño y organización de la memoria caché, eficiencia de scheduler del sistema operativo.

En particular:

-Los equipos con mayor número de hilos reales deberían mostrar mejores ganancia al incrementar hilos.

-Equipos con cachés más amplias podrían manejar matrices medianas con menor penalización por accesos a memoria.

3) Consolidación de resultados estadísticamente representativos

Dado que cada experimento se repite 30 veces, siguiendo la recomendación basada en la ley de los grandes números, se espera obtener valores promediados estables y confiables. Esto permitirá:

-Reducir efectos aleatorios del sistema operativo.

-Minimizar interferencias de procesos externos.

-Identificar claramente tendencias en el rendimiento.

4) Gráficas que evidencien las tendencias de escalabilidad

Al representar los promedios en gráficas, se espera obtener:

- Puntos de saturación o estancamiento para determinados tamaños de matriz.

- Diferencias claras entre algoritmos, sobre todo entre hilos (pthread/OpenMP) y procesos (fork).

- Comparaciones entre máquinas que permitan visualizar qué equipos aprovechan mejor la concurrencia.

Estas gráficas servirán como soporte visual clave para el análisis y las conclusiones.

5) Identificación de cuellos de botella y límites de paralelización

Finalmente, se espera poder determinar:

- En qué punto el uso de más hilos deja de mejorar el rendimiento.

- Como impacta la jerarquía de memoria en el uso de matrices grandes.

- Cuando la sobrecarga de paralelización supera los beneficios.

-Que combinación de algoritmo-máquina ofrece el mejor rendimiento global.

Estos hallazgos serán la base para formular conclusiones y recomendaciones sobre el uso eficiente de técnicas de paralelización en programas intensivos en cómputo.

4. Resultados

Después de realizar las pruebas, se registraron en un libro de Excel los tiempos resultantes de las ejecuciones para los tamaños de matrices y cantidades de hilos previamente definidos en cada uno de los cinco entornos de cómputo. Para cada combinación de tamaño de matriz, número de hilos y sistema operativo, se calculó el promedio del tiempo de ejecución para 30 repeticiones. Posteriormente, estos promedios se utilizaron para construir gráficas comparativas, organizados por tamaño de matriz (50x50, 100x100, 500x500, 1000x1000, 1500x1500).

Cada grupo de graficas muestra cómo varía el tiempo de ejecución en función del número de hilos utilizados, permitiendo observar la evolución del rendimiento a medida que se incrementa la cantidad de hilos. El objetivo principal de estas gráficas es comparar el comportamiento de los distintos sistemas de cómputo y detectar posibles cuellos de botella. Además, permiten identificar qué entornos aprovechan mejor los recursos y hasta qué punto la utilización de más hilos proporciona mejoras en el rendimiento. Este procedimiento se repitió para los cuatro algoritmos evaluados: Fork, filasOpenMP, clasicaOpenMP y POSIX.

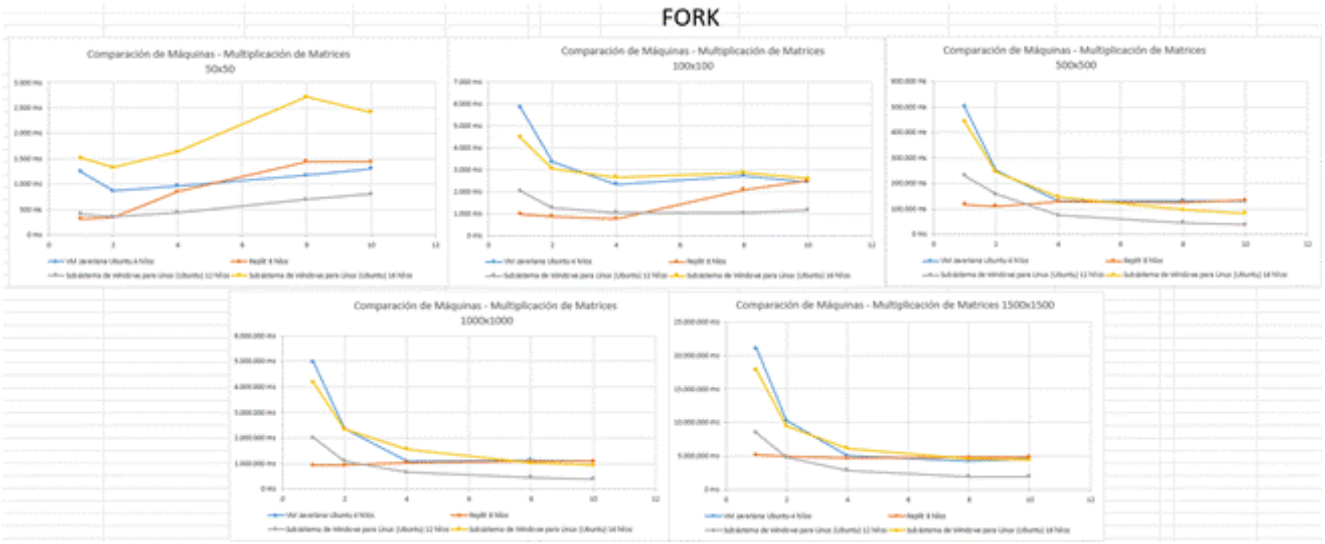


Figura 1. Grupo de graficas comparativas de máquinas para los promedios del algoritmo FORK.

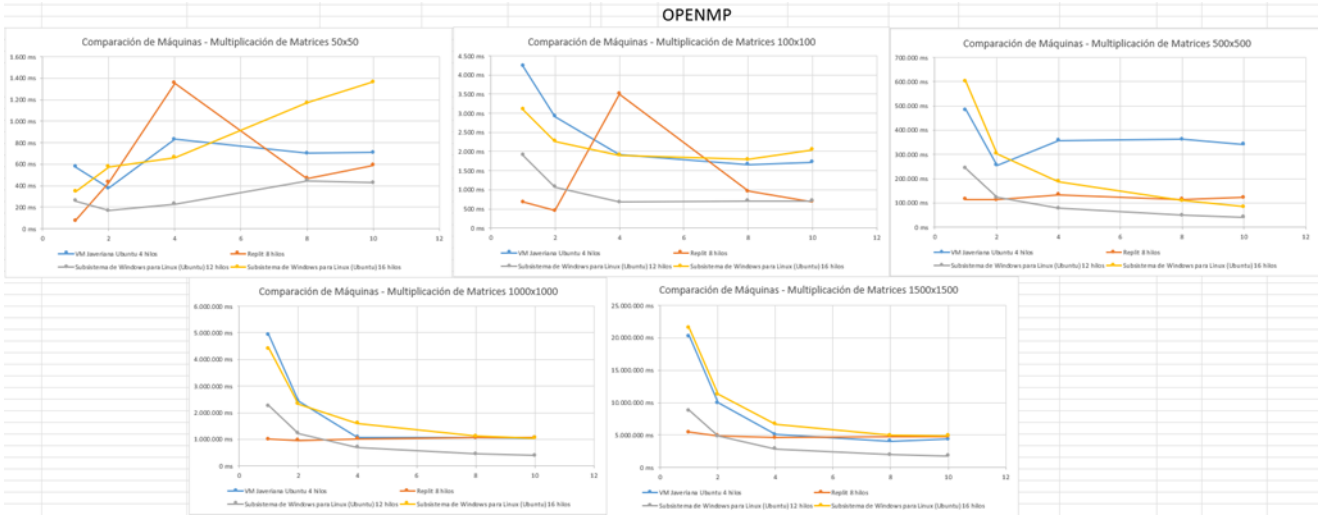


Figura 2. Grupo de graficas comparativas de máquinas para los promedios del algoritmo OpenMP.

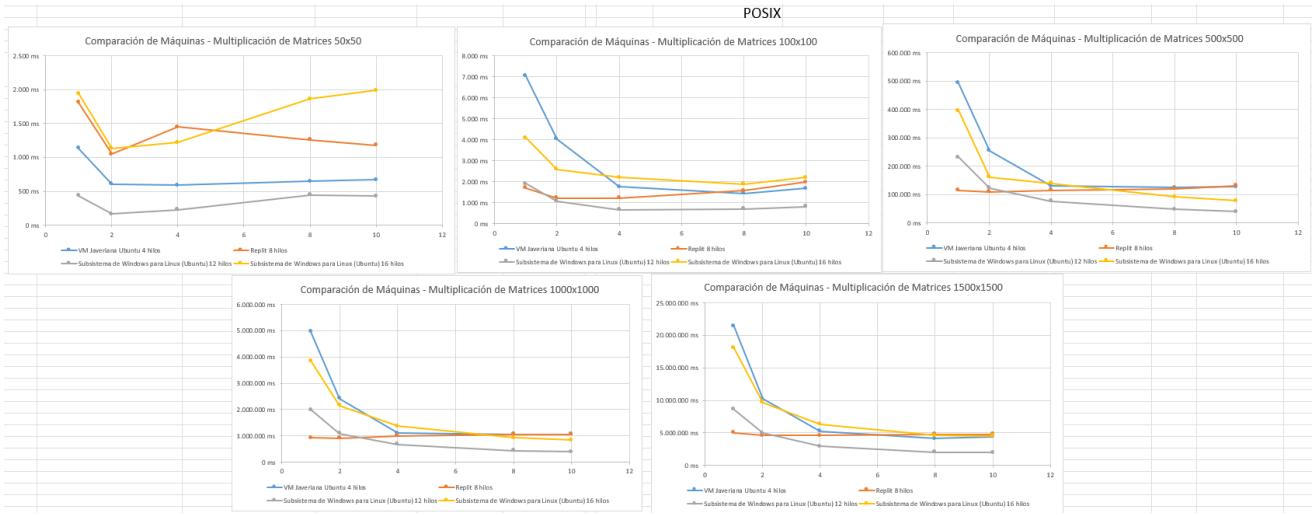


Figura 3. Grupo de graficas comparativas de máquinas para los promedios del algoritmo Posix.

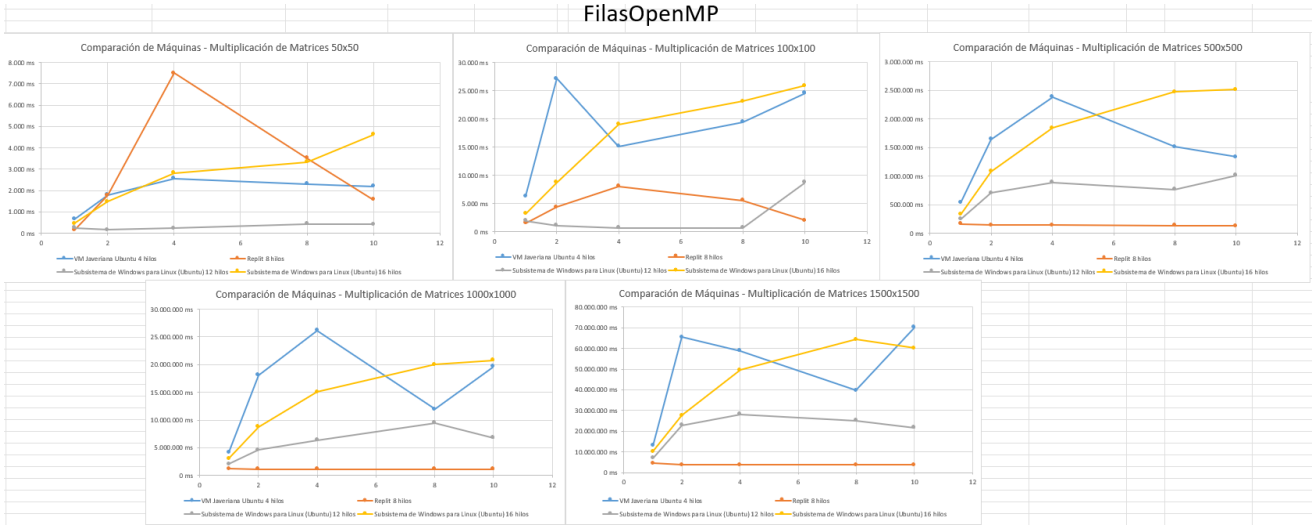


Figura 4. Grupo de graficas comparativas de máquinas para los promedios del algoritmo filasOpenMP.

5. Análisis de resultados

5.1 Análisis del rendimiento de algoritmos

A continuación, se presentan las gráficas correspondientes al rendimiento mostrado por cada dispositivo de cómputo frente a cada uno de los siguientes algoritmos: Fork, OpenMP, POSIX y FilasOpenMP, los cuales fueron utilizados en el desarrollo de este taller. Las gráficas muestran los tiempos promedio de ejecución de cada algoritmo, permitiendo ilustrar la eficiencia de cada uno en diferentes entornos de cómputo. Esta comparación facilitara la identificación del algoritmo que presenta mayores mejoras en el rendimiento al incrementar el número de hilos, además de su comportamiento frente a distintas cargas.

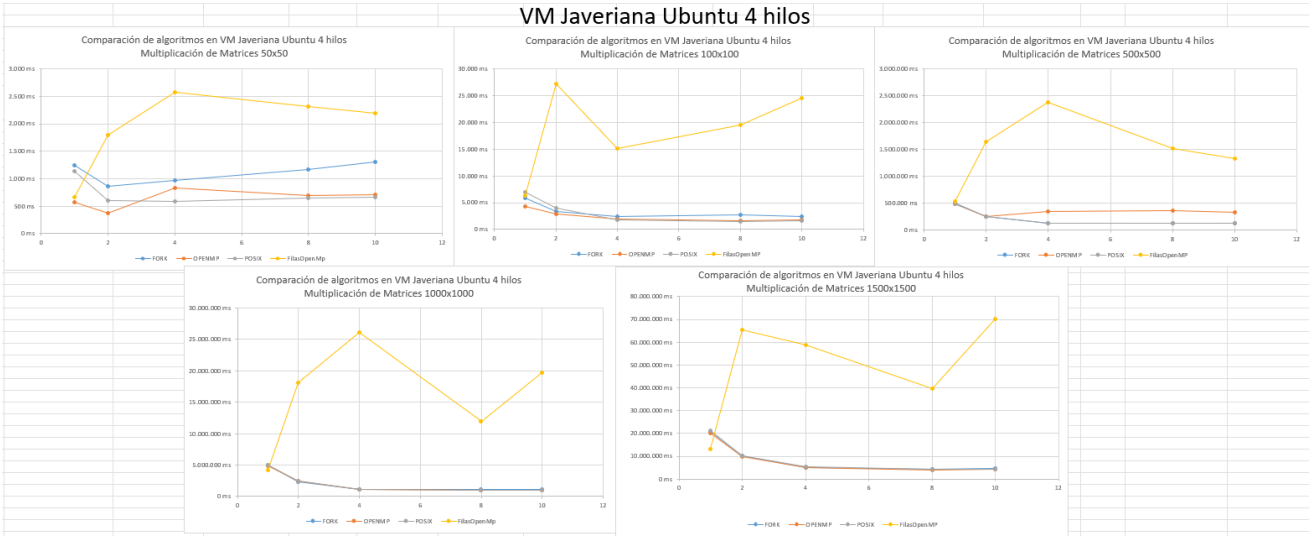


Figura 5. Grupo de gráficas comparativas de algoritmos para la VM de la Javeriana (Ubuntu 4 hilos).

Empezaremos el análisis con el entorno de cómputo VM Javeriana Ubuntu 4 hilos. Como se puede apreciar en las gráficas los algoritmos POSIX y Fork se destacan por encima de los otros para este entorno, en especial para matrices medianas y grandes, siendo el caso los casos menores para esta prueba los que mostraron peor rendimiento para estos. Además, se muestran los mejores tiempos de ejecución y que no presentan picos en su recorrido, lo cual nos indica cierta linealidad. Por otro lado, OpenMP muestra un desempeño aceptable, donde su eficiencia depende del número de hilos y del tamaño de la matriz. Por último, FilasOpenMP demuestra ser el peor algoritmo para este entorno, obteniendo los tiempos de ejecución más altos en cada tamaño de matriz con cualquier número de hilos.

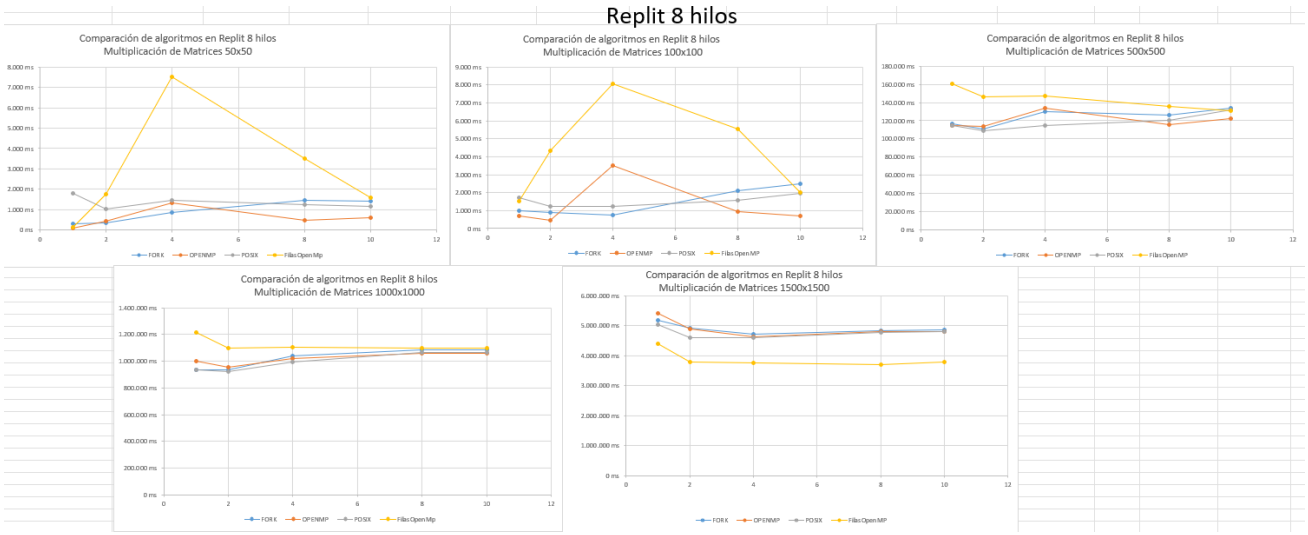


Figura 6. Grupo de gráficas comparativas de algoritmos para Replit (8hilos).

Continuamos el análisis con el entorno de cómputo Replit 8 hilos, acá se muestran tiempos de ejecución a nivel general frente al anterior entorno, siendo la matriz 50x50 la excepción. Se evidencian picos muy bruscos del algoritmo FilasOpenMP en los primeros dos tamaños de matrices, lo que se podría traducir que para esos tamaños este algoritmo no es óptimo, sin embargo, esto va cambiando a medida que el tamaño de la matriz aumenta, en especial en el último de 1500x1500, donde demuestra ser el mejor algoritmo, con los tiempos más rápidos con respecto a sus contrapartes. Además, se evidencia cierta estabilidad con los demás algoritmos con los diferentes tamaños de matriz, contando con tiempos de ejecución muy similares, mostrando cierta diferencia con un numero especifico de hilos. Por último, podemos decir que a medida que el tamaño en la matriz aumenta, los tiempos van mejorando cuando se usan más hilos, lo que demuestra que el uso del paralelismo ayuda con la eficiencia en este entorno de cómputo.

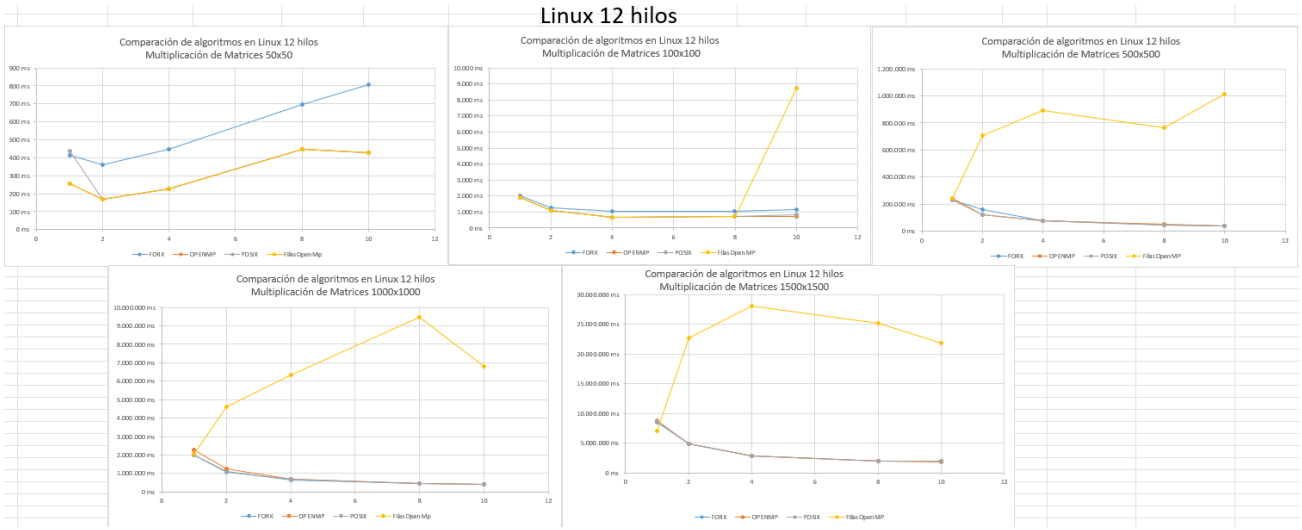


Figura 7. Grupo de gráficas comparativas de algoritmos para Linux (12hilos).

Continuamos con el entorno de cómputo de Linux 12 hilos, donde observamos una notable disminución en el rendimiento del algoritmo FilasOpenMP para tamaños de matriz mayores o iguales a 500x500, donde demuestra tener promedios mucho más altos que los demás algoritmos, sugiriendo que es el peor algoritmo para usar en esas condiciones. Por otro lado, evidenciamos que los tiempos de los otros algoritmos son muy similares, mostrando una diferencia muy pequeña, excepto en el caso peculiar de la matriz con el menor tamaño de 50x50, donde el mejor algoritmo resulta ser FilasOpenMP y el peor Fork. Por último, se muestra que, a mayores tamaños de matriz,

el paralelismo y sus ventajas se hacen mas evidentes, reduciendo el tiempo a mayor cantidad de hilos trabajando, y así lograr un mejor rendimiento.

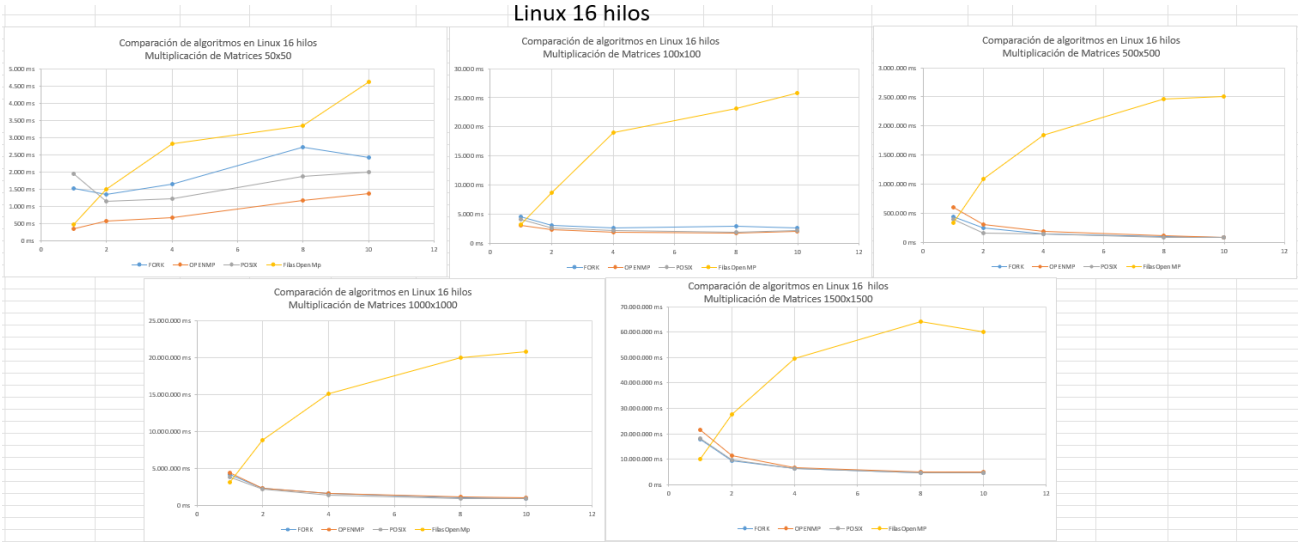


Figura 8. Grupo de gráficas comparativas de algoritmos para Linux (16hilos).

Por último, analizaremos los promedios en el entorno de cómputo Linux 16 hilos. Acá se evidencia nuevamente un pésimo rendimiento del algoritmo FilasOpenMP con respecto a los otros, esta vez siendo el peor en todos los tamaños de matriz analizados. Al igual que con el entorno Linux 12 hilos, vemos que el resto de los algoritmos presentan tiempos muy similares, con una diferencia casi imperceptible, a excepción de la matriz más pequeña, donde se evidencia que el mejor rendimiento los tuvo el algoritmo OpenMP. Nuevamente evidenciamos la considerable mejora en el rendimiento que nos otorga el paralelismo en las matrices con los mayores tamaños, ya que a medida que se aumentan los hilos, el tiempo de ejecución va disminuyendo.

5.2 Análisis del comportamiento de las maquinas

Con base a los resultados obtenidos y las gráficas de comportamiento de los entornos de cómputo, se realizó un análisis de los rendimientos teniendo en cuenta las características de cada máquina, y los análisis realizados previamente para cada algoritmo. Inicialmente se pensaba que, al aumentar el número de hilos usados al ejecutar cada algoritmo, se obtendría una disminución en el tiempo de ejecución, teniendo a su vez una mejor eficiencia en cuanto al uso de recursos. Sin embargo, como podemos apreciar en la figura 1, este no siempre es el caso. Para el entorno de cómputo Linux 16, se puede apreciar como en varias ocasiones al aumentar el número de hilos también se aumentaba el tiempo de ejecución, en especial en matrices más pequeñas, lo cual se conoce como un cuello de botella. Esto se puede deber a múltiples factores, como lo puede ser la ejecución en simultaneo de múltiples procesos ajenos al experimento o una posible limitación en sus capacidades generales, ya que podemos evidenciar que la arquitectura de este dispositivo, cuenta con un numero de hilos disponibles mayores a los de la mayor cantidad puesta en las pruebas, llegando a limitar incluso el número de hilos disponibles para la ejecución del experimento, generando así una sobrecarga o “overhead”, la cual, pese a estar siempre presente, se busca que sea mínima, sin embargo, al crear esta sobrecarga de la administración de los hilos, este aumenta, causando mayores tiempos de ejecución.

Por otro lado, para los demás entornos de cómputo Linux 12 hilos y VM Javeriana Ubuntu 4 hilos, podemos evidenciar un comportamiento esperado, es decir, a mayor cantidad de hilos, menor tiempo de ejecución, siempre y cuando se cuente con el numero necesario de hilos, sin embargo, resulta necesario realizar la observación de que a partir de ciertos puntos los tiempos de ejecución se empiezan a estabilizar, como en una línea recta, incluso llegan a aumentar. Esto nos sugiere hay un límite máximo útil de hilos a usar al momento de paralelizar. Esto lo podemos comparar con el ejemplo de pintar una pared. Digamos que se desea pintar una pared, digamos de 20

metros de ancho por 10 de altura, entonces se contrata a un trabajador para llevar a cabo esta tarea, a este trabajador le llevaría pintarla completa un total de 6 horas, si se contrata otro se espera que este tiempo se reduzca a la mitad, llevándonos a la conclusión de que entre más trabajadores se contraten menos tiempo se demora el trabajo de pintar la pared. Sin embargo, llegara un punto donde el número de trabajadores es tan alto, que se estorbarían entre ellos al hacer el trabajo, afectando el tiempo en completar el trabajo. Con este ejemplo se busca decir que, no necesariamente a mayor cantidad de hilos, menor será el tiempo de ejecución, ya que se genera un "overhead" muy alto, derivando en un cuello de botella. Esto señala la importancia de comprender la arquitectura y distribución de hardware de las máquinas para determinar su número optimo de hilos a ejecutar en paralelo para hacer un uso eficiente de recursos.

Por último, en el entorno de cómputo Replit 8 hilos, se evidencia que, al aumentar la cantidad de hilos, el tiempo de ejecución de los algoritmos disminuye. Esto indica una relación inversamente proporcional para este sistema. Esto puede deberse a alguna de las siguientes razones:

- **Limitación de recursos compartidos:** Al ser Replit un entorno de desarrollo en la nube cuenta con recursos limitados, como CPU, memoria y ancho de banda. Al momento de agregar más hilos, estos consumen estos recursos. Si hay demasiados hilos, la sobrecarga en el sistema podría generar que los hilos se peleen por estos recursos, entorpeciendo el tiempo de ejecución.
- **Overhead de la gestión de hilos:** Crear y administrar hilos tiene un costo. A medida que se agreguen más hilos, el overhead también va aumentando. Esto puede generar que la paralelización sea más dañina que beneficiosa.
- **Contencion de recursos:** Si se cuenta con varios hilos creados y estos intentan acceder a recursos compartidos, como lo son la memoria o la CPU, al mismo tiempo, pueden experimentar una contención. Esto a su vez genera que los demás hilos se queden esperando acceder a los recursos, lo cual reduce el rendimiento.
- **Gastos de sincronización:** Al momento de sincronización de los hilos, ya sea para compartirse datos entre ellos o realizar tareas de forma coherente, se puede generar un retraso o una demora en la sincronización, lo que puede llevar a una sobrecargar significativa, afectando los tiempos de ejecucion
- **Estrategias de planificación de hilos:** En un entorno con múltiples hilos, el sistema operativo debe decidir cuál de los hilos se ejecuta en qué momento. Si hay demasiados hilos, la planificación se vuelve más costosa y menos eficiente, ya que el sistema tiene que alternar entre muchos hilos. Esto puede generar un cambio de contexto más frecuente, lo que puede ralentizar el rendimiento.
- **Escalabilidad limitada de la infraestructura de replit:** La infraestructura de Replit probablemente está diseñada para manejar cargas de trabajo de usuarios individuales, pero no para ejecutar aplicaciones de alto rendimiento con múltiples hilos.

6. Conclusiones

Después de analizar los diferentes resultados obtenidos, podemos concluir que el paralelismo no siempre garantiza mejor rendimiento: Se evidenció que para matrices pequeñas (50×50, 100×100), la sobrecarga de la paralelización supera los beneficios, resultando en tiempos de ejecución mayores que la versión secuencial.

Así mismo se pueden resaltar ciertos aspectos de cada técnica de paralelización:

- POSIX Threads: Mostró mejor escalabilidad y rendimiento consistente en la mayoría de los escenarios.
- OpenMP: Excelente rendimiento en equipos con múltiples núcleos, con menor complejidad de implementación
- FilasOpenMP: Comportamiento variable según el entorno - excelente para matrices muy grandes en algunos sistemas (como Replit con 1500×1500), pero con rendimiento deficiente en otros escenarios, especialmente con matrices medianas.
- Fork(): Es el menos eficiente debido al alto costo de creación de procesos

El tamaño de matriz determina la efectividad del paralelismo: Para matrices grandes (≥500×500), todas las técnicas paralelas mostraron mejoras significativas, mientras que, para matrices pequeñas, el enfoque secuencial fue más eficiente.

En cuanto a los entornos que manejamos podemos concluir:

- **Linux 16 hilos:** Muestra que no necesariamente tener más hilos asegura un rendimiento optimo a la hora de ejecutar un programa, esto se debe a que el dispositivo pudo tener más tareas en ejecución ocasionando un gran ruido, afectando el rendimiento de ejecución de los algoritmos.
- **Linux 12 hilos:** Presentó el mejor balance entre capacidad de procesamiento y eficiencia, con rendimiento estable y dentro de los sistemas comparados fue el mejor en tiempos de ejecución.
- **Replit (8 hilos):** Este entorno tuvo un comportamiento muy similar a lo largo de todas las pruebas, este presento resultados esperados, como lo es el de a mayor número de hilos mejor es el rendimiento, exceptuando cuando se alcanza el numero optimo de hilos. Sin embargo, estos resultados fueron mejores que los esperados debido a las características con las que cuenta este entorno, ya que no son las mejores si se comparan con el resto de los entornos usados para la elaboración de este taller.
- **VM Javeriana (4 hilos):** Este entorno, al igual que el anterior, mostro resultados esperados, incluso teniendo el menor número de hilos disponibles, ya que demostró que a mayor número de hilos mejor es el rendimiento, hasta encontrar ese punto óptimo.

7. Bibliografía

- Kernighan, Brian W., and Dennis M. Ritchie. The C Programming Language. Vol. 2. Englewood Cliffs: prentice-Hall, 1988.
- Silberschatz, Abraham, Peter B. Galvin, and Greg Gagne. Operating System Concepts. Vol. 8. Wiley, 2013.
- Milenkovic, Milan. *SISTEMAS OPERATIVOS. CONCEPTOS Y DISEÑO*. McGraw-Hill