Taller de Evaluación de Rendimiento

Luz Salazar, Gabriel Jaramillo, Roberth Méndez, Guden Silva, Viviana Gómez

Enlace al repositorio personal:

https://github.com/luzasalazar/Taller-Fork-Grupo-Alpha/tree/main/Taller%20Evaluacion%20Rendimiento

Pontificia Universidad Javeriana

Sistemas Operativos

Profesor: John Jairo Corredor Franco

6 de mayo de 2025

1. Introducción

En el contexto de los sistemas operativos modernos, el procesamiento paralelo es un recurso clave para mejorar el rendimiento de las aplicaciones, especialmente aquellas que requieren una intensa serie de operaciones para realizarse como lo es en el caso de la multiplicación de matrices. El taller se enfoca en la evaluación del rendimiento de tres enfoques de programación paralela implementados en lenguaje C: OpenMP, procesos con fork(), y POSIX threads. A través de la ejecución de pruebas controladas en diferentes entornos de cómputo y usando varias matrices de distintos tamaños, se busca comparar la eficiencia de estos algoritmos en función del número de hilos utilizados y la arquitectura del sistema. El análisis de los tiempos de ejecución permite identificar ventajas, limitaciones y comportamientos particulares de cada técnica, lo cual muestra las posibles ventajas y desventajas del uso o la implementación del paralelismo.

A través de este trabajo, se busca comprender cómo el número de hilos, el entorno de ejecución y la técnica de paralelización empleada influyen en el desempeño de la operación de multiplicación de matrices. Además, se tiene como objetivo identificar las fortalezas y debilidades de cada sistema de cómputo, considerando aspectos como la sobrecarga de creación de hilos o procesos, la contención de recursos, la escalabilidad y la eficiencia en el uso del hardware disponible. El análisis del taller puede construir una base que ayude a tomar decisiones a la hora de desarrollar un programa para lograr su máxima eficiencia y rendimiento.

2. Resumen

Este taller tiene como objetivo la comparación de algoritmos en serie y paralelos con tres programas en lenguaje C que implementan la multiplicación de matrices cuadradas utilizando OpenMP, procesos (fork) y POSIX threads ejecutados en distintos entornos. Se busca evaluar cómo varía el rendimiento según el algoritmo, la cantidad de hilos y el sistema de cómputo utilizado. Los programas leen matrices generadas dinámicamente y ejecutan la operación de multiplicación en distintos sistemas de cómputo, midiendo los tiempos de ejecución. Las pruebas se realizaron 30 veces por combinación de tamaño de matriz e hilos, con matrices de tamaño 600, 800, 1000 y 1200, y con 1, 2, 4, 8, 16 y 10 hilos, y los resultados se guardan en archivos para realizar el análisis comparativo.

2.1 Planteamiento del Problema

El objetivo es analizar el impacto del paralelismo en la eficiencia de la multiplicación de matrices cuadradas. Evaluando lo siguiente:

- Cómo varía el rendimiento según el tamaño de matrices y el número de hilos utilizados.
- Cómo se comportan los distintos algoritmos (OpenMP, fork, POSIX) en diferentes sistemas de cómputo.

Entrada de los casos de prueba

- Tamaños de matriz: 600, 800, 1000 y 1200.
- Cantidad de hilos: 1, 2, 4, 8, 16, 20 (cada maquina hasta su máximo de hilos).
- Sistemas de cómputo: WSL, Replit, Ubuntu (4, 8, 20 núcleos).

Salida esperada

Tiempos de ejecución por cada repetición en un archivo correspondiente a cada caso de prueba, para cada algoritmo y sistema de cómputo.

2.2 Propuesta de Solución

Se ejecutaron tres programas en C, con las siguientes características:

- Algoritmo fork(): Crea procesos hijos para distribuir el trabajo por filas. Cada hijo escribe su resultado parcial a través de un pipe.
- Algoritmo POSIX threads: Utiliza pthread_create() para distribuir las filas de la matriz entre múltiples hilos.
- Algoritmo OpenMP: Utiliza la biblioteca OpenMP para distribuir el trabajo por filas entre hilos usando directivas paralelas y hace uso de memoria compartida para acumular resultados.

Cada programa ejecuta las siguientes etapas:

- Reserva dinámica de memoria para las matrices.
- Inicialización de matrices.
- Ejecución de la multiplicación.
- Medición del tiempo total usando gettimeofday().
- Liberación de memoria dinámica.

Para automatizar la ejecución de los programas, se implementó un archivo Perl que recorre combinaciones de 4 tamaños de matriz y 5 cantidades de hilos, ejecutando cada una 30 veces por programa. Los resultados se almacenan en archivos .dat con el nombre del programa, tamaño de matriz e hilos para facilitar la recolección y análisis de los tiempos de ejecución en los cinco sistemas de cómputo. Cabe destacar que para cada máquina solo se hará la evaluación hasta la cantidad de hilos que cada sistema tenga. Es decir, el registro de los resultados de las operaciones se hará hasta alcanzar la cantidad de hilos en los sistemas de cómputo, por lo que el único sistema del cual se registrarán todos los resultados es el sistema de la máquina virtual de Ubuntu de 20 hilos.

2.3 Método de Prueba

Para probar la funcionalidad de los programas y la automatización de su ejecución, se sigue el siguiente método de prueba:

1. Pruebas funcionales iniciales

   Antes de las mediciones, se realizaron pruebas con matrices pequeñas para:

   - Confirmar el correcto funcionamiento de la paralelización y distribución del trabajo.
   - Comprobar la correcta medición de los tiempos de ejecución y asegurar que todos los algoritmos generaran resultados válidos y comparables.

2. Revisión de uso de memoria y finalización adecuada

- Se verificó la correcta liberación de memoria dinámica en cada programa.
- Se comprobó que todos los procesos e hilos finalizaran correctamente al terminar su ejecución.

3. Automatización de casos de prueba

   Se desarrolló un archivo lanza.pl que automatiza la ejecución de los tres programas. Teniendo en cuenta todas las combinaciones posibles de:

   - Programas: mmClasicaFork, mmClasicaOpenMP, mmClasicaPosix
   - Tamaños de matriz: 600, 800, 1000, 1200
   - Cantidad de hilos: 1, 2, 4, 8, 16, 20 (cada maquina hasta su máximo de hilos)

   Se comprobó que se crearan correctamente los archivos de cada caso de prueba y se registraran los tiempos de ejecución de las 30 repeticiones.

4. Evaluación en múltiples sistemas de cómputo

   Las pruebas se repitieron en cinco entornos distintos (WSL, Replit, y tres Ubuntu con 4, 8 y 20 CPUs) para comparar el comportamiento de los algoritmos bajo diferentes arquitecturas de hardware y capacidades de procesamiento.

2.4 Descripción de los equipos de cómputo

En la siguiente tabla se muestran algunas características de hardware relevantes de los 5 sistemas de cómputo en donde se ejecutaron las pruebas.

Equipo	**WSL**	**VM Ubuntu (4 hilos)**	**Replit**	**VM Ubuntu (8 hilos)**	**VM Ubuntu (20 hilos)**
L1d, L1i	128 KiB / 128 KiB	128 KiB / 128 KiB	128 KiB / 128 KiB	256KiB/256KiB	320 KiB / 320 KiB
L2	1 MiB	4 MiB	2 MiB	8Mib	2.5 MiB
L3	6 MiB	143 MiB	32 MiB	35,8Mib	20 MiB
RAM	3.8 MiB	11 Gb	64 Gb	15Gb	64 Gb
# Hilos	8	4	8	8	20
GHz	1.6 GHz	2.40 GHz	3 GHz	2.4GHz	3.2 GHz

2.5 Resultados Esperados

Se espera observar que:

- Con 1 hilo, los tres algoritmos tengan tiempos similares de ejecución.
- Con más hilos, los algoritmos de OpenMP y POSIX deben mostrar disminución del tiempo de ejecución, especialmente en sistemas con más núcleos.

- El algoritmo de fork() sería el algoritmo con menos rendimiento debido a un cuello de botella causado por una sobrecarga en la sincronización de los procesos y la creación que implica duplicar el espacio de memoria del proceso padre, por lo tanto, no sería eficiente con grandes cantidades de hilos.
- WSL y Replit tendrán un menor rendimiento en comparación con Ubuntu, ya que en el WSL el acceso a los recursos de CPU y planificación de hilos está controlado por el sistema Windows y Replit tiene limitaciones en los recursos que se asignan a los usuarios.

3. Resultados

Luego 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 paneles de gráficas comparativas, organizados por tamaño de matriz (600x600, 800x800, 1000x1000 y 1200x1200).

Cada panel 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 tres algoritmos evaluados: Fork, OpenMP y POSIX.

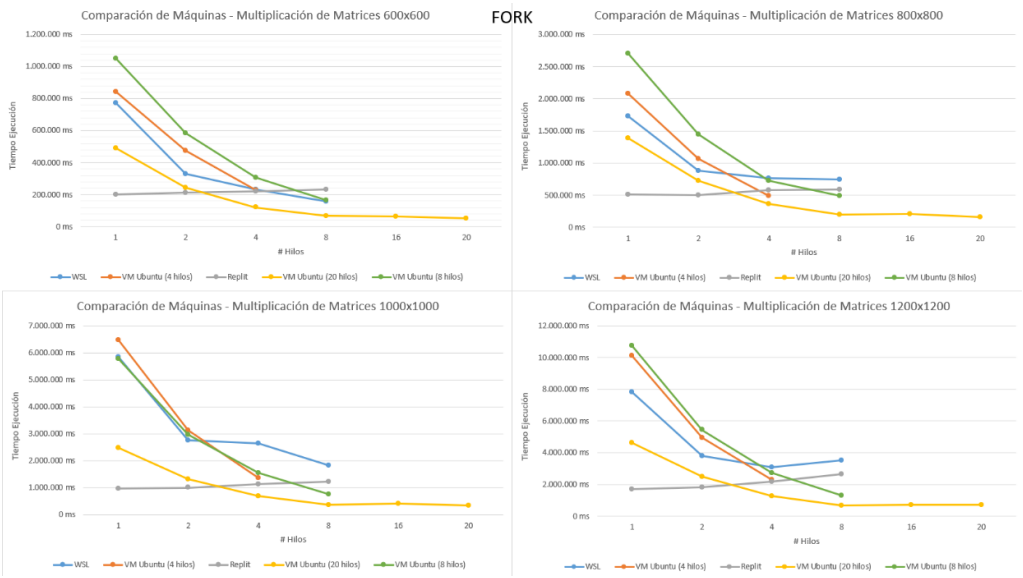

Figura 1. Panel de gráficas comparativas de máquinas para el algoritmo Fork

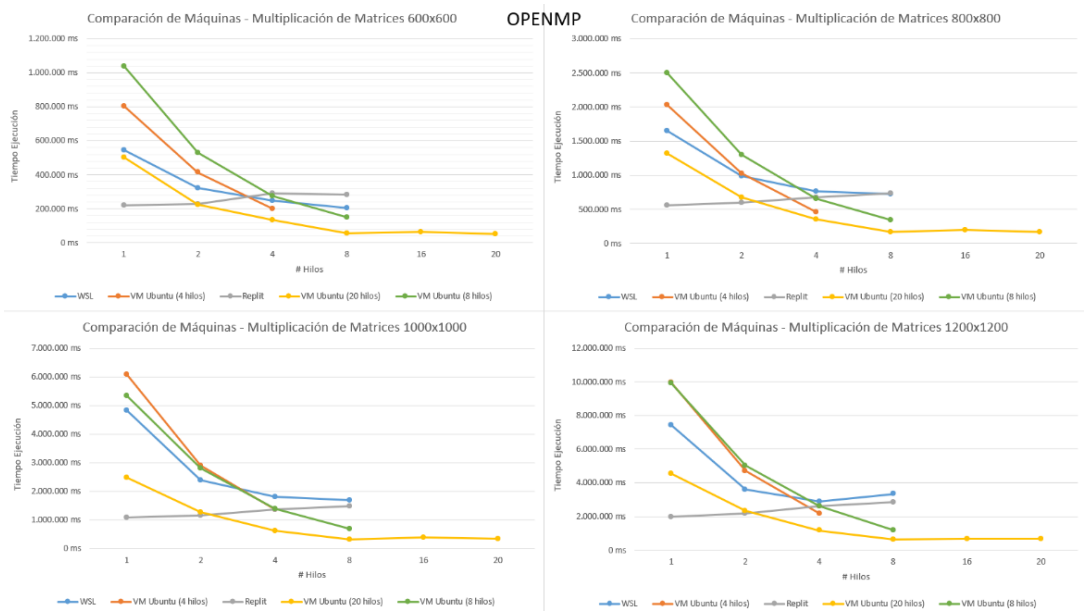

Figura 2. Panel de gráficas comparativas de máquinas para el algoritmo OpenMP

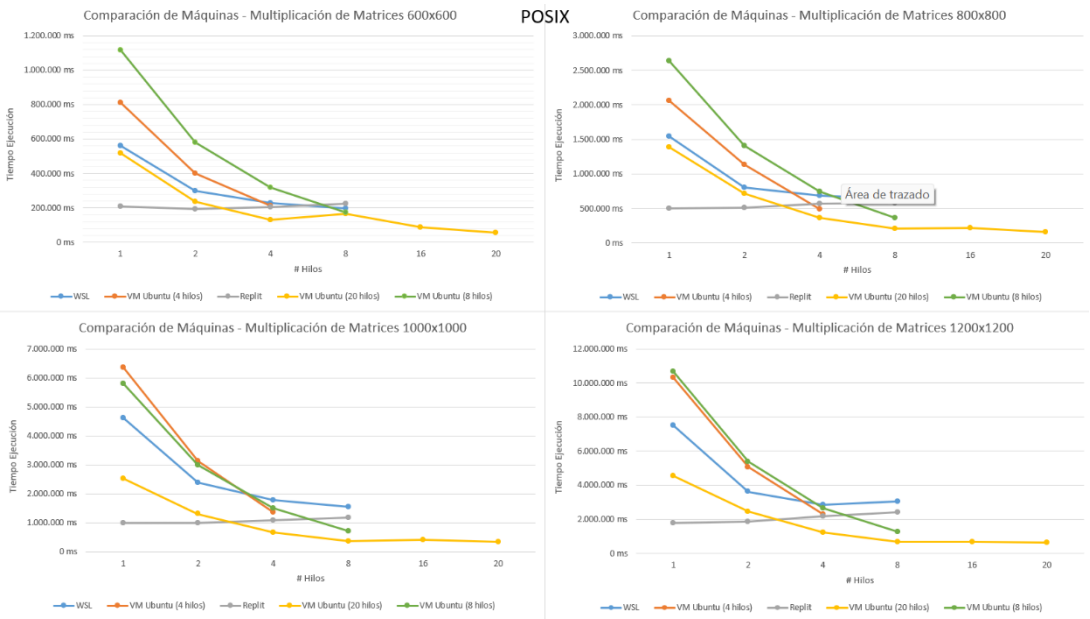

Figura 3. Panel de gráficas comparativas de máquinas para el algoritmo Posix

4. Análisis de resultados

4.1. Análisis del rendimiento de algoritmos

Al igual que las gráficas anteriores, se creo este panel que compara el rendimiento de los algoritmos Fork, OpenMP y POSIX. Las gráficas muestran los tiempos de ejecución promedio para cada algoritmo, permitiendo observar la eficiencia de cada uno en diferentes entornos de cómputo. Esta comparación facilita identificar cuál de los algoritmos logra mayores mejoras en desempeño al incrementar la cantidad de hilos y cómo se comportan

frente a distintas cargas de trabajo.

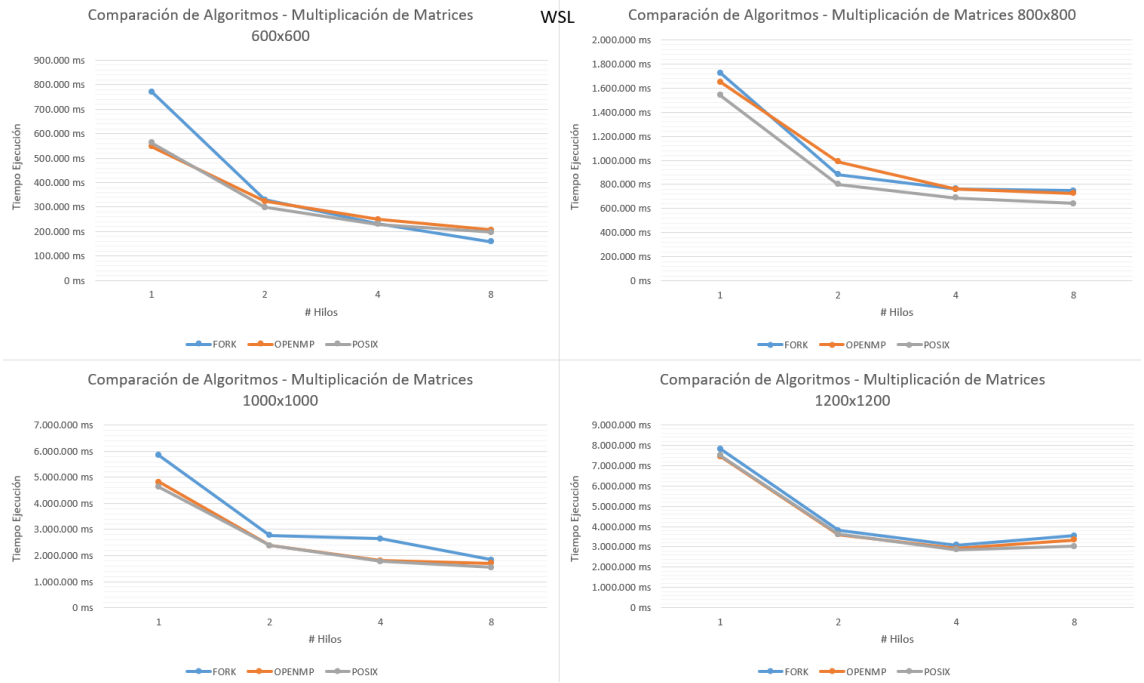

Figura 4. Panel de gráficas comparativas de algoritmos para el WSL

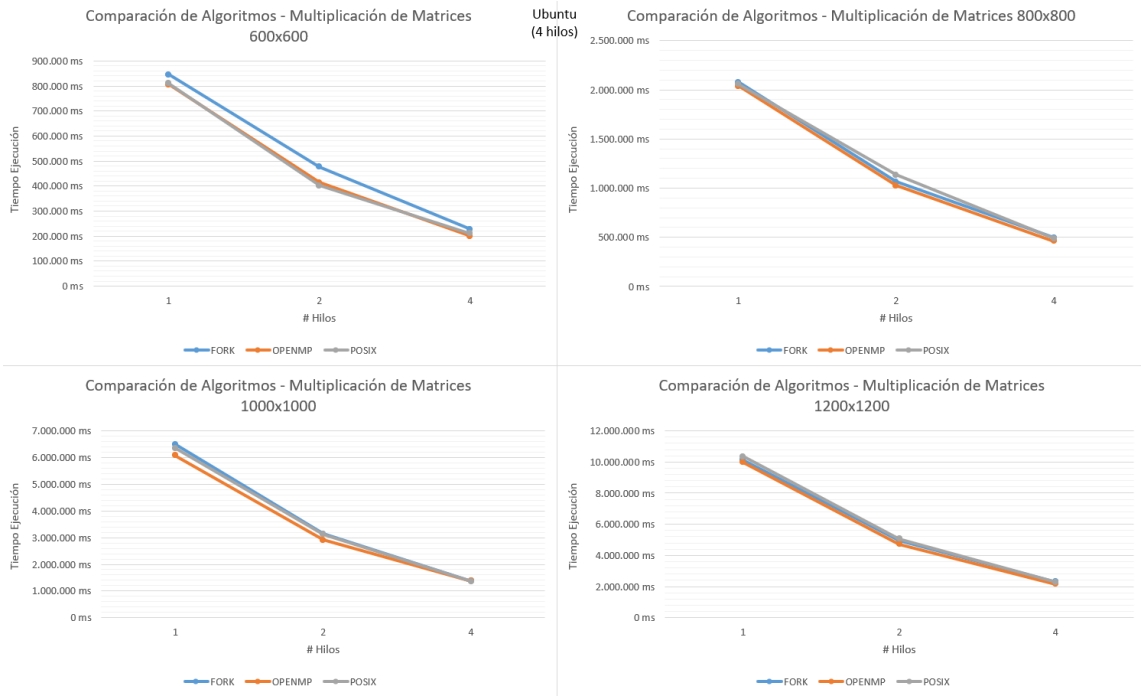

Figura 5. Panel de gráficas comparativas de algoritmos para el Ubuntu de 4 hilos

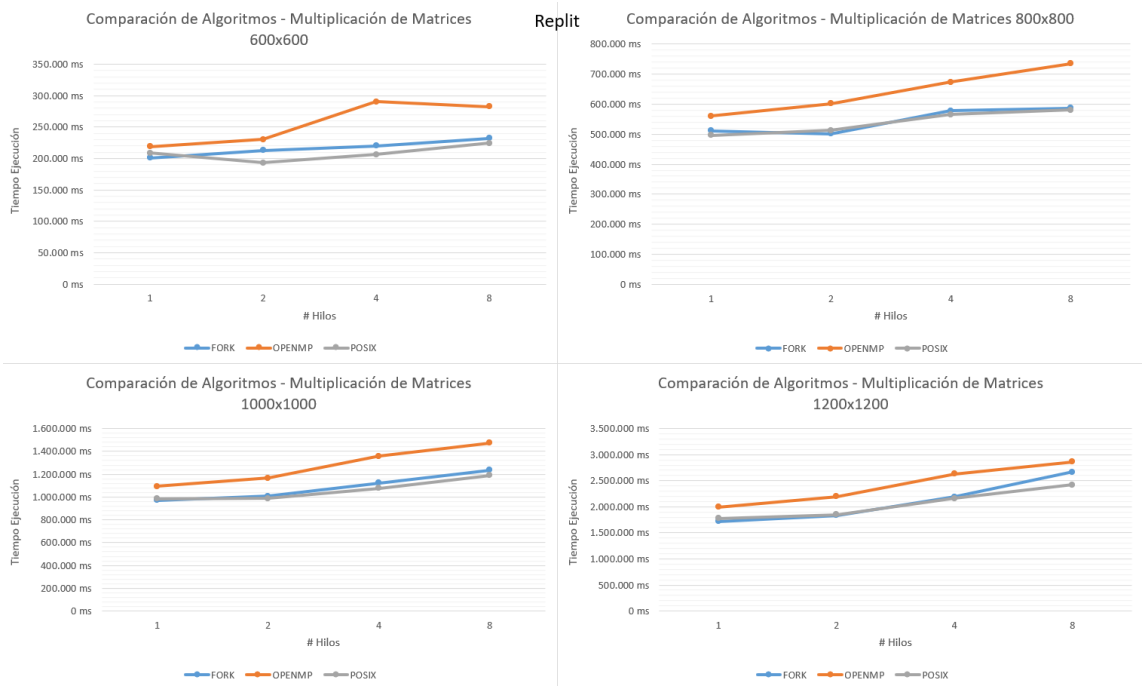

Figura 6. Panel de gráficas comparativas de algoritmos para el entorno de Replit

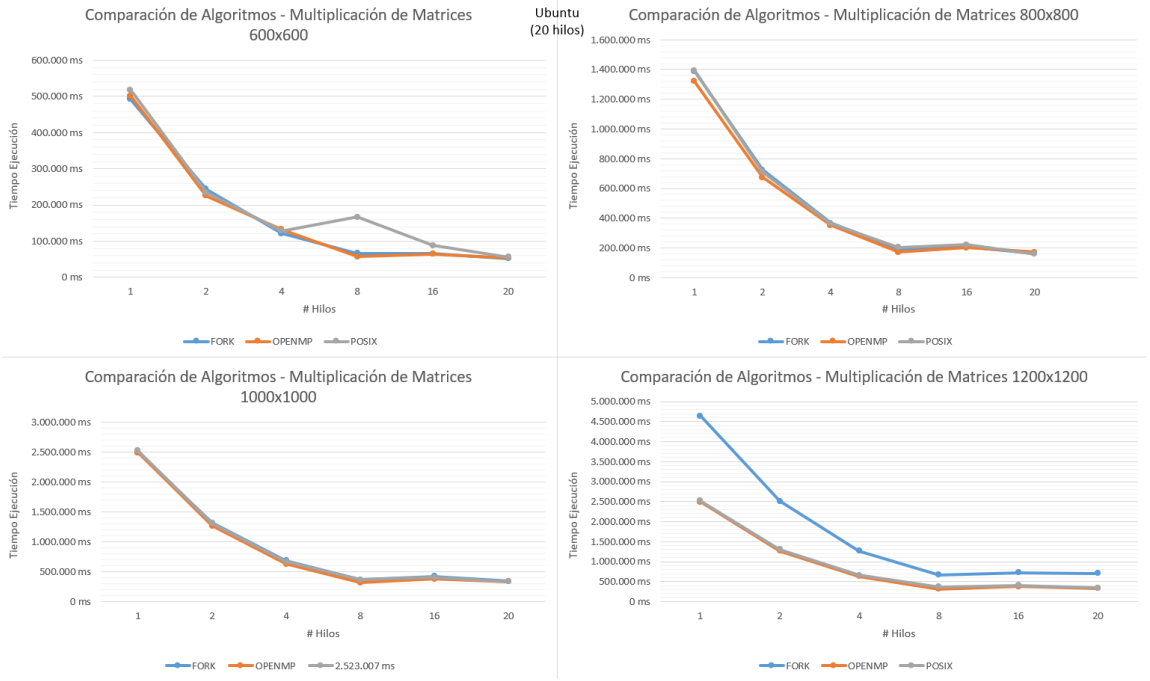

Figura 7. Panel de gráficas comparativas de algoritmos para el Ubuntu de 20 hilos

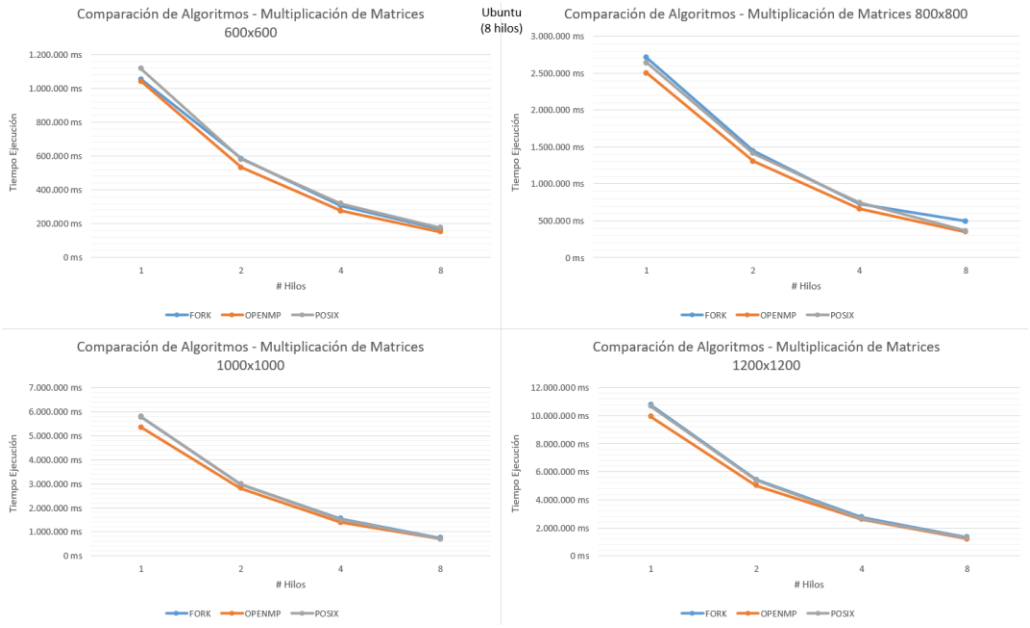

Figura 8. Panel de gráficas comparativas de algoritmos para el Ubuntu de 8 hilos

A partir de los resultados se observa que, en efecto, los algoritmos OpenMP y POSIX reducen su tiempo de ejecución a medida que se añadiendo más núcleos al procesamiento. Del mismo modo, se evidenció que el algoritmo de fork() es el de menos rendimiento a medida que van aumentando la cantidad de hilos.

Se puede apreciar que, para la gran mayoría de casos, el mejor algoritmo es el de OpenMP, pues tiene tiempos de ejecución menores a los demás, en algunos casos, sobre todo en matrices de 1200x1200 en la máquina VM Ubuntu de 8 Hilos, la diferencia de tiempos de ejecución entre OpenMP y fork() puede llegar cerca de 700,000ms.

Para la mayoría de los casos, el algoritmo de fork() es el de peor rendimiento, sin embargo, hay algunos, aquellos que se procesan en un solo núcleo, el algoritmo que utiliza POSIX es el peor, siendo así significa que el algoritmo fork() es mejor que POSIX para este caso en concreto.

Hay algunos casos en los que fork() tiene un mejor rendimiento que OpenMP, sin embargo, para estos mismo la diferencia no es mucha, por ejemplo, la matriz 1000x1000 en la máquina VM Ubuntu de 20 hilos, aun así, la diferencia es muy poca, 400ms por lo cual no debería tomarse mucho en cuenta.

De igual manera, se pudo evidenciar que el comportamiento de la máquina WSL es muy distinto a los de Ubuntu. Para esta máquina, el mejor algoritmo no es OpenMP como en las otras máquinas, sino es Posix o fork() dependiendo del caso, para la matriz de 600x600 en 8 núcleos, fork es la mejor opción, de resto, posix tiene un mejor rendimiento.

Una posible razón de este comportamiento es que WSL no maneja los hilos de la misma forma que un sistema Linux nativo como Ubuntu. Aunque WSL emula un entorno Linux, internamente sigue usando el schedule y kernel de Windows, lo cual puede afectar el rendimiento de bibliotecas como OpenMP, que dependen fuertemente de una buena gestión de hilos.

4.2. Análisis del comportamiento de las maquinas

Con base en los resultados obtenidos y las gráficas del 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. Como noción inicial, se creería que, al aumentar el número de hilos al ejecutar cada algoritmo, este tendría un menor tiempo de ejecución, siendo cada vez más eficiente en cuanto a sus recursos.

Sin embargo, se puede apreciar como en la **Figura 1**, este no siempre es el caso. Para la maquina WSL (Azul), se puede apreciar como para operaciones más complejas (matriz de 1200x1200), el desempeño con 4 hilos es peor comparado con el desempeño de 2 hilos, evidenciando un cuello de botella. Esto se puede deber a limitaciones en la capacidad de procesamiento paralelo del sistema, tal como una sobrecarga por la administración de múltiples hilos sin el hardware o arquitectura apropiado para esto. Esta sobrecarga de la coordinación de los hilos genera un "overhead" más grande del deseado. Mientras que el "overhead", es un gasto que siempre está presente, siempre se busca que este sea mínimo, sin embargo, al crear esta sobrecarga de la administración de hilos, este aumenta, causando mayores tiempos de ejecución.

Por otro lado, para las maquinas Ubuntu de 4, 8 y 20 hilos, se aprecia un comportamiento esperado al propuesto inicialmente, con la observación de que, a partir de determinado punto, los tiempos se estabilizan o incluso aumentan ligeramente. Esto sugiere que existe un límite en el cual añadir más hilos no contribuye de manera notable al rendimiento, e incluso puede generar tiempos más altos por la sobrecarga de administración de los procesos. 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.

Ahora bien, se observa que en Replit, el rendimiento de la ejecución de la multiplicación de las matrices disminuye a medida que aumenta la cantidad de hilos usados. Esto quiere decir que la relación entre el tiempo y los hilos es inversa para este sistema de cómputo. Esto puede deberse a algunas de las siguientes razones:

- **Limitación de recursos compartidos:** Replit es un entorno de desarrollo en la nube que tiene recursos limitados, como CPU, memoria y ancho de banda. Cuando se agregan más hilos, cada uno consume recursos de estos. Si hay demasiados hilos, la sobrecarga en el sistema puede hacer que los hilos se bloqueen entre sí o se peleen por recursos, lo que provoca una caída en el rendimiento.

- **Overhead de la gestión de hilos:** Crear y administrar hilos tiene un costo. A medida que se aumenta el número de hilos, hay un overhead adicional por la gestión de estos, como la creación de los hilos, la sincronización, la planificación y la conmutación de contexto entre hilos. Esto puede llegar a ser más costoso que los beneficios de la paralelización si no se gestiona adecuadamente.

- **Contención de recursos:** Si varios hilos intentan acceder a recursos compartidos (como memoria o CPU) al mismo tiempo, pueden experimentar contención. Esto puede hacer que los hilos pasen más tiempo esperando a acceder a los recursos, lo que reduce el rendimiento global. En sistemas de múltiples hilos, se requiere un buen diseño para evitar esta contención.

- **Gastos de sincronización:** Si los hilos necesitan sincronizarse entre sí para compartir datos o realizar tareas de forma coherente, lo que esto puede generar un retraso. Demasiada sincronización puede llevar a una sobrecarga significativa, ya que los hilos no pueden hacer su trabajo de forma independiente.

- **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.

5. Conclusiones

En primera instancia, se concluye de gran importancia comprender la arquitectura interna de cada sistema con el objetivo de maximizar el rendimiento y uso de los recursos de la máquina. Esto permite identificar como el cuello

de botella varía entre máquinas y como el uso de múltiples hilos no siempre garantiza una mejora lineal en el desempeño. Además, se debe considerar el "overhead" asociado a la creación y gestión de hilos, ya que un número excesivo de estos puede generar una sobrecarga que impacte negativamente el rendimiento.

Además de esto, el paralelismo demuestra ser más efectivo en sistemas de cómputo que cuentan con una mayor cantidad de hilos, en este caso fue la VM Ubuntu con 20 hilos. Esto se evidencia en la reducción significativa que tenían los tiempos de ejecución a medida que aumentaba la cantidad de hilos, esto indica una alta capacidad de distribución del trabajo entre varias unidades de procesamiento. Esta eficiencia es más evidente al trabajar con matrices de mayor tamaño (como las de 1000x1000 y 1200x1200), donde la carga computacional es más intensa y permite que el paralelismo tenga un mayor impacto en la optimización del rendimiento.

De igual manera, aunque OpenMP demuestra ser el algoritmo con mejor rendimiento en la mayoría de los escenarios, especialmente al aumentar la cantidad de núcleos, su eficiencia se ve afectada en entornos como WSL debido a las diferencias en la gestión de hilos respecto a sistemas Linux nativos. POSIX y fork(), si bien generalmente son menos eficientes, presentan ventajas puntuales dependiendo del entorno y del tamaño de la matriz, lo que resalta la importancia de considerar tanto el algoritmo como el sistema operativo al evaluar el rendimiento paralelo.