
***Taller Evaluación De Rendimiento***

*Profesionales en Formación*
Juan Felipe Morales Espitia
Juan David Rincón Poveda
Juan Felipe Fernández Barrero

Bogotá, mayo 2024

# ÍNDICE

# I. INTRODUCCIÓN

La computación de alto rendimiento representa un campo crucial en la búsqueda de soluciones eficientes y escalables para una amplia gama de problemas computacionales. En este contexto, la evaluación comparativa del desempeño de algoritmos juega un papel fundamental, ya que proporciona información valiosa sobre la eficiencia y la capacidad de adaptación de los sistemas informáticos.

El presente taller se sumerge en el corazón de esta evaluación comparativa al centrarse en el análisis de un algoritmo clásico: la multiplicación de matrices. Este algoritmo, esencial en numerosas aplicaciones computacionales, sirve como punto focal para investigar las diferencias de rendimiento entre dos configuraciones de ejecución: serie y paralela.

Para comprender plenamente las implicaciones de este estudio, es crucial contextualizarlo dentro del entorno de los sistemas operativos. La elección del sistema operativo subyacente puede influir significativamente en el rendimiento del algoritmo y, por lo tanto, en los resultados obtenidos.

Una de las piedras angulares de este taller es la comparación meticulosa de diferentes sistemas de cómputo. La selección de sistemas representativos y variados garantiza un análisis exhaustivo que abarca una amplia gama de escenarios de uso.

La implementación del algoritmo de multiplicación de matrices en lenguaje C proporciona una base sólida para la evaluación comparativa. Esta elección no solo garantiza la eficiencia computacional, sino que también permite un control preciso sobre los aspectos técnicos del algoritmo.

Al explorar los paradigmas de programación serie y paralelo, este taller arroja luz sobre las diferencias fundamentales en la ejecución de algoritmos en entornos computacionales tradicionales y distribuidos. Esta comparación ofrece información valiosa sobre la escalabilidad y el rendimiento en diferentes contextos de aplicación.

El uso de una variedad de herramientas, aplicaciones y plataformas en este estudio garantiza una evaluación integral que abarca múltiples aspectos del rendimiento computacional. Esta diversidad proporciona una visión más completa de las fortalezas y limitaciones de cada configuración.

La metodología experimental adoptada en este taller se basa en principios sólidos de teoría de la probabilidad, específicamente en la ley de grandes números. Esta aproximación estadística permite obtener resultados confiables y significativos al promediar múltiples ejecuciones del algoritmo en diferentes condiciones.

El análisis de los datos recopilados se lleva a cabo de manera exhaustiva, utilizando tanto tablas como gráficas para visualizar e interpretar los resultados. Esta presentación clara y concisa facilita la identificación de tendencias y patrones importantes que pueden influir en las conclusiones.

Las conclusiones derivadas de este estudio proporcionan una visión profunda del rendimiento del algoritmo de multiplicación de matrices en diferentes configuraciones de sistemas de cómputo. Estas conclusiones, respaldadas por un análisis riguroso de los datos, sirven como base para formular recomendaciones concretas para la optimización y mejora futura de los sistemas informáticos.

Se incluye una lista exhaustiva de referencias que respaldan y enriquecen el contenido de este taller, proporcionando a los lectores una base sólida para explorar más a fondo los temas tratados.

En resumen, este taller representa un esfuerzo dedicado a entender y mejorar el rendimiento computacional a través de una evaluación comparativa cuidadosamente diseñada. Al abordar cada punto del índice de manera integral, se garantiza la generación de conocimientos significativos que pueden tener un impacto tangible en el desarrollo futuro de sistemas informáticos eficientes y escalables.

## II. SISTEMAS OPERATIVOS

En el ámbito de la computación de alto rendimiento, la elección del sistema operativo se erige como un factor crítico que influye directamente en el desempeño y la fiabilidad de las aplicaciones computacionales [1]. Al contextualizar nuestro estudio dentro de este marco, debemos considerar cómo diferentes sistemas operativos gestionan los recursos del sistema, la planificación de procesos y la gestión de la memoria [2]. Por ejemplo, mientras que los sistemas operativos basados en Linux ofrecen un entorno altamente flexible y adaptable para la ejecución de procesos paralelos, los sistemas Windows pueden presentar ventajas en términos de compatibilidad con ciertas aplicaciones y herramientas de desarrollo [3]. Además, el rendimiento de los algoritmos puede verse afectado por la eficiencia de la gestión de entrada/salida, la administración de archivos y la gestión de redes, áreas en las que los diferentes sistemas operativos pueden variar significativamente en su implementación [4]. Por lo tanto, al considerar el contexto de los sistemas operativos, podemos obtener una comprensión más completa de cómo las decisiones de diseño a nivel de software pueden impactar en el rendimiento y la escalabilidad de nuestras aplicaciones en el ámbito de la computación de alto rendimiento.

## III. OBJETIVO: COMPARATIVA DE SISTEMAS DE CÓMPUTO

La comparación de sistemas de cómputo se erige como un pilar fundamental para comprender el rendimiento de los algoritmos en diferentes entornos computacionales. La selección de sistemas de cómputo representativos y variados nos permite analizar exhaustivamente cómo factores como la arquitectura del hardware, el sistema operativo subyacente y la configuración de red pueden influir en el rendimiento y la eficiencia de nuestras aplicaciones [5]. Al considerar esta comparación, es importante tener en cuenta cómo cada sistema de cómputo maneja aspectos como la concurrencia, la administración de memoria y la ejecución paralela, áreas cruciales para la computación de alto rendimiento [6].

En particular, en el ámbito académico de la Universidad Pontificia Javeriana, se promueve una visión integral y multidisciplinaria de la computación de alto rendimiento, donde se fomenta la investigación y el análisis crítico de diferentes enfoques y tecnologías. En este sentido, la comparación de sistemas de cómputo en nuestro taller se alinea con esta visión, permitiendo a los estudiantes adquirir habilidades prácticas y teóricas que son relevantes tanto para la academia como para la industria.

A continuación, se presenta una Tabla 1 Características significativas de Los sistemas de cómputo seleccionados.

| Característica | Sistemas de Cómputo A | Sistemas de Cómputo B |
|---|---|---|

| | | |
|---|---|---|
| Sistema operativo | Linux | Linux |
| Versión | Ubuntu | Ubuntu |
| Tipo | Máquina Laboratorio | Portátil |
| RAM | 16 | 16 |
| Procesador | Intel (R) Core (TM) i7-8700 | Intel (R) Core (TM) i5 12300h |
| Número Cores | 4 | 4 |

Tabla 1 Características significativas de los sistemas de cómputo seleccionados.

## IV. COMPARACIÓN A TRAVÉS DE DIFERENTES ALGORITMOS DE MULTIPLICACIÓN DE MATRICES (EN C)

En el marco de este taller la comparación a través de algoritmos de multiplicación de matrices en lenguaje C representa un componente esencial para evaluar el rendimiento y la eficiencia de diferentes configuraciones de sistemas computacionales. En nuestro algoritmo, no solo implementamos la multiplicación de matrices clásica, sino también la multiplicación utilizando matrices transpuestas, lo cual agrega una dimensión adicional al análisis comparativo.

La inclusión de la multiplicación de matrices clásica y la variante con matrices transpuestas nos permite explorar cómo diferentes técnicas y optimizaciones pueden afectar el rendimiento del algoritmo en entornos computacionales variados. Además, esta comparación nos brinda la oportunidad de examinar cómo se comporta el algoritmo en situaciones donde la manipulación de datos y la gestión de la memoria pueden tener un impacto significativo en el rendimiento; En el contexto de la Universidad Pontificia Javeriana, se promueve el desarrollo de habilidades prácticas y teóricas en el ámbito de la computación, donde la implementación y análisis de algoritmos en lenguaje C constituyen una parte integral del currículo académico [7].

## V. PARADIGMAS DE PROGRAMACIÓN SERIE Y PARALELO

Es crucial hacer referencia a los paradigmas de programación serie y paralelo al analizar el rendimiento de los algoritmos de multiplicación de matrices. Estos paradigmas representan enfoques fundamentales para la implementación de software, y su comprensión es esencial para evaluar cómo diferentes configuraciones de sistemas de cómputo afectan el rendimiento de nuestros algoritmos.

El paradigma de programación serie se basa en la ejecución secuencial de instrucciones, donde cada tarea se realiza en orden y una a la vez. Este enfoque tradicional ofrece simplicidad y control, pero puede limitar la capacidad de aprovechar al máximo los recursos disponibles en sistemas computacionales modernos [8].

Por otro lado, el paradigma de programación paralelo busca distribuir tareas entre múltiples procesadores o núcleos de CPU, permitiendo que múltiples operaciones se ejecuten simultáneamente. Este enfoque ofrece el potencial de mejorar significativamente el rendimiento y la eficiencia de los algoritmos, especialmente en entornos computacionales donde la concurrencia y la velocidad son críticas [9].

Al considerar estos paradigmas en nuestro taller, podemos explorar cómo diferentes sistemas de cómputo aprovechan estas técnicas para mejorar el rendimiento de la multiplicación de matrices, lo que nos permite extraer conclusiones significativas sobre la eficacia relativa de los enfoques serie y paralelo en diferentes contextos.

## VI. ALGORITMO CLÁSICO EN PROGRAMACIÓN

Se utiliza el algoritmo clásico de multiplicación de matrices como caso de estudio. Este algoritmo, implementado en C, realiza la multiplicación de dos matrices cuadradas de tamaño □×□. Para la evaluación del rendimiento, se compila el código fuente en un ejecutable utilizando el compilador GCC y se ejecuta con diferentes valores de tamaño de matriz y número de hilos.

El proceso de evaluación implica la ejecución repetida del algoritmo con distintos parámetros para capturar una representación precisa de los tiempos de ejecución. Dado que los sistemas operativos introducen variabilidad en los tiempos de ejecución, se realiza una batería de experimentación con múltiples repeticiones para cada combinación de parámetros. Estos datos se utilizan para calcular un valor promedio, que proporciona una estimación más confiable del rendimiento del algoritmo.

La automatización del proceso de experimentación se logra mediante un script en Perl llamado "lanzador.pl", que permite ejecutar los experimentos con diferentes configuraciones de manera eficiente. Los resultados se registran en archivos de datos (.dat), que pueden ser exportados a aplicaciones de hojas de cálculo para un análisis detallado.

Al finalizar el taller, se evalúa la calidad del informe generado, considerando la exhaustividad del análisis, la claridad de las conclusiones y la presentación adecuada de los resultados y los archivos documentados.

Este taller proporciona una oportunidad invaluable para comprender y aplicar conceptos fundamentales de evaluación de rendimiento en el contexto de la computación de alto rendimiento, preparando a los participantes para abordar desafíos más complejos en este campo en el futuro.

## VII. DOCUMENTACIÓN DE LOS FICHEROS

### EPXLICACIÓN DE LAS FUNCIONES

**Declaración de variables globales:** Se declaran punteros a las matrices mA, mB y mC para almacenar los datos de entrada y el resultado de la multiplicación, respectivamente. También se

declara un mutex MM_mutex para controlar el acceso concurrente a las matrices y las variables start y stop para medir el tiempo de ejecución.

**Función** llenar_matriz**:** Inicializa las matrices mA y mB con valores aleatorios utilizando la función srand48 y un ciclo for.

**Función** print_matrix**:** Imprime una matriz en la consola si su tamaño no es mayor a 12. La función organiza los elementos en filas y columnas con tres decimales de precisión.

**Función** inicial_tiempo**:** Obtiene el tiempo actual utilizando la función gettimeofday y lo almacena en la variable start.

**Función** final_tiempo**:** Obtiene el tiempo actual utilizando gettimeofday, calcula la diferencia de tiempo en microsegundos entre stop y start, y la imprime en la consola.

**Función** mult_thread**:** Esta función es ejecutada por cada hilo y realiza la multiplicación de la sección de la matriz asignada al hilo. Recibe un puntero a la estructura parametros que contiene la identificación del hilo (idH), el número total de hilos (nH), el tamaño de las matrices (N), el índice de inicio (ini) y el índice de fin (fin) de la sección a procesar.

La función realiza la multiplicación de matrices utilizando tres bucles for. El primer bucle recorre las filas de la sección asignada (i), el segundo bucle recorre las columnas de la matriz mB (j), y el tercer bucle recorre las columnas de la matriz mA (k).

Dentro del tercer bucle, se multiplican los elementos correspondientes de las matrices mA y mB y se acumulan en la variable sumaTemp.

Finalmente, el resultado de la multiplicación para cada par de filas y columnas se almacena en la matriz mC.

La función utiliza pthread_mutex_lock para bloquear el mutex antes de acceder a las matrices compartidas y pthread_mutex_unlock para desbloquearlo después. Esto asegura la integridad de los datos y evita conflictos de acceso concurrente.

Por último, la función termina el hilo utilizando pthread_exit.

## VIII. EXPLICACIÓN DEL MAIN

En el main se puede evidenciar lo siguiente:

**Verificación de argumentos de línea de comandos**: El programa primero verifica si se proporcionan suficientes argumentos en la línea de comandos. Si **argc**, que es el número de argumentos de la línea de comandos, es menor que 2, indica que no se proporcionaron suficientes

argumentos. En este caso, imprime un mensaje indicando cómo usar el programa y luego sale con un código de error **-1**.

**Obtención de tamaño de matriz y número de hilos**: Si se proporcionan suficientes argumentos, el programa convierte los dos primeros argumentos (**argv[1]** y **argv[2]**) en enteros utilizando la función **atoi()**. El primer argumento (**argv[1]**) representa el tamaño de la matriz cuadrada (**SZ**), y el segundo argumento (**argv[2]**) representa el número de hilos que se utilizarán (**n_threads**).

**Declaración de variables y asignación de memoria**: Se declaran variables necesarias para el manejo de hilos y se asigna memoria para almacenar las matrices. La función **MEM_CHUNK** no está definida en el código proporcionado, por lo que asumiré que es una constante definida en otro lugar del código que representa el tamaño de un bloque de memoria.
**Llenado e impresión de matrices**: Se llama a la función **llenar_matriz()** para llenar las matrices **mA** y **mB** con datos. Luego, se imprimen las matrices utilizando la función **print_matrix()**.

**Inicialización de tiempo, mutex y atributos de hilo**: Se inicializa la medición del tiempo utilizando la función **inicial_tiempo()**. También se inicializa un mutex utilizando **pthread_mutex_init()** y se inicializan los atributos del hilo utilizando **pthread_attr_init()** y **pthread_attr_setdetachstate()** para permitir que los hilos se unan después de terminar su ejecución.

**Creación de hilos**: Se crea un bucle que itera **n_threads** veces. En cada iteración, se asigna memoria para una estructura de datos llamada **struct parametros**, que contiene información sobre el hilo actual, como su identificador (**idH**), el número total de hilos (**nH**) y el tamaño de la matriz (**N**). Luego, se crea un hilo utilizando **pthread_create()** y se pasa la función **mult_thread** junto con la estructura de datos como argumentos.
**Espera de hilos**: Después de crear todos los hilos, se espera a que cada hilo termine su ejecución utilizando **pthread_join()**.
**Impresión del tiempo final y matriz resultante**: Se llama a **final_tiempo()** para imprimir el tiempo total de ejecución del programa. Luego, se imprime la matriz resultante **mC** utilizando **print_matrix()**.
**Liberación de recursos**: Finalmente, se destruyen los atributos de los hilos y el mutex utilizando **pthread_attr_destroy()** y **pthread_mutex_destroy()**, respectivamente. Luego, se llama a **pthread_exit()** para terminar todos los hilos y salir del programa.

## IX. SEPARACIÓN DEL FICHERO FUENTE EN BIBLIOTECAS DE FUNCIONES, INTERFAZ Y PRINCIPAL

Se dividió el fichero fuente en diferentes archivos los cuales son:

- Fuente_Evaluacion.c

- Funciones.c
- Funciones.h

**Funciones_Evaluación.h:**

En el archivo de “Funciones_Evaluacion.c” se puede encontrar:

- **Inclusión de bibliotecas y definición de constantes**: El programa incluye las bibliotecas necesarias, como **funciones.h**, **stdlib.h**, **stdio.h**, **time.h**, **pthread.h**, y define una constante **DATA_SIZE** que representa el tamaño de un bloque de memoria para almacenar matrices y un arreglo **MEM_CHUNK** para almacenar las matrices.
- **Verificación de argumentos de línea de comandos**: Se verifica si se proporcionan suficientes argumentos en la línea de comandos. Si no se proporcionan, el programa imprime un mensaje indicando cómo usarlo y termina con un código de error.
- **Obtención del tamaño de matriz y número de hilos**: Se obtienen el tamaño de la matriz y el número de hilos a partir de los argumentos de la línea de comandos.
- **Asignación de memoria para las matrices**: Se asigna memoria en **MEM_CHUNK** para almacenar las matrices **mA**, **mB** y **mC**.
- **Llenado e impresión de matrices**: Se llenan las matrices **mA** y **mB** con datos y se imprimen.
- **Inicialización del tiempo, mutex y atributos de hilo**: Se inicializa la medición del tiempo, el mutex y los atributos de los hilos.
- **Creación de hilos para la multiplicación de matrices**: Se crean hilos para realizar la multiplicación de matrices en paralelo. Cada hilo recibe una estructura de datos con información sobre su identificador, el número total de hilos y el tamaño de la matriz.
- **Espera de hilos**: El programa espera a que todos los hilos terminen su ejecución antes de continuar.
- **Impresión del tiempo final y matriz resultante**: Se imprime el tiempo total de ejecución del programa y la matriz resultante **mC**.
- **Liberación de recursos**: Se liberan los recursos utilizados, como los atributos de los hilos y el mutex, y se finaliza la ejecución de los hilos.

**Funciones.c**

En el archivo de “Funciones.c” se puede encontrar:

- **Inclusión de bibliotecas y declaraciones de variables globales**: El programa incluye las bibliotecas necesarias, como **funciones.h**, **stdlib.h**, **stdio.h**, **time.h** y **pthread.h**, además de declarar algunas variables globales, como punteros a las matrices **mA**, **mB**, **mC**, un mutex **MM_mutex** para controlar el acceso a las matrices y variables para medir el tiempo de ejecución.
- **Funciones para llenar e imprimir matrices**: Se definen funciones para llenar matrices con valores predefinidos y para imprimir matrices en la consola.

- **Funciones para medir el tiempo de ejecución**: Se definen funciones para iniciar y finalizar la medición del tiempo de ejecución del programa.
- **Función ejecutada por cada hilo**: Se define una función **mult_thread** que es ejecutada por cada hilo. Esta función recibe una estructura de datos con información sobre el hilo, como su identificador, el número total de hilos y el tamaño de las matrices a multiplicar. Cada hilo realiza una sección de la multiplicación de matrices y almacena el resultado en la matriz resultante **mC**.
- **Implementación de la multiplicación de matrices**: Dentro de la función **mult_thread**, se implementa el algoritmo de multiplicación de matrices, donde cada hilo calcula una sección de la matriz resultante **mC**. Se utiliza un mutex para garantizar el acceso exclusivo a las matrices compartidas durante la multiplicación.

**Funciones.h**

En el archivo de "Funciones.h" se puede encontrar:

- **Declaración de variables globales**: Se declara una variable global **MM_mutex** del tipo **pthread_mutex_t** para la sincronización de hilos y dos variables globales **start** y **stop** del tipo **struct timeval** para medir el tiempo de ejecución.
- **Definición de la estructura de parámetros**: Se define una estructura **parametros** que se utiliza para pasar parámetros a la función ejecutada por cada hilo. Esta estructura contiene información sobre el número total de hilos (**nH**), el identificador del hilo actual (**idH**) y el tamaño de la matriz (**N**).
- **Declaración de funciones**: Se declaran las siguientes funciones:
    - **llenar_matriz(int SZ)**: Función para llenar una matriz con valores aleatorios. Recibe como parámetro el tamaño de la matriz (**SZ**).
    - **print_matrix(int sz, double *matriz)**: Función para imprimir una matriz. Recibe como parámetros el tamaño de la matriz (**sz**) y un puntero a la matriz (**matriz**).
    - **inicial_tiempo()**: Función para iniciar el contador de tiempo.
    - **final_tiempo()**: Función para detener el contador de tiempo.
    - **mult_thread(void *variables)**: Función ejecutada por cada hilo para la multiplicación de matrices. Recibe un puntero a una estructura **parametros** que contiene la información necesaria para la multiplicación de matrices.
- **Directiva de preprocesador**: Se utiliza la directiva **#ifndef** para evitar la inclusión múltiple del archivo de encabezado. Si **FUNCIONES_H** no está definido, el contenido del archivo será incluido; de lo contrario, será omitido.

## X. ANÁLISIS DEL FICHERO

- En el fichero del lenguaje PERL denominado lanzador.pl contiene lo siguiente:
- **Obtención del directorio actual**:

    - **$Path =** pwd**;**: Utiliza el comando **pwd** para obtener el directorio actual y lo almacena en la variable **$Path**.
    - **chomp($Path)**: Elimina el salto de línea al final de la ruta obtenida.
- **Definición de variables**:
    - **$Nombre_Ejecutable = "MM_ejecutable"**: Especifica el nombre del ejecutable del programa que se va a lanzar.
    - **@Size_Matriz = ("16","300")**: Define un arreglo **@Size_Matriz** con diferentes tamaños de matriz para las pruebas.
    - **@Num_Hilos = (1,2,4,8)**: Define otro arreglo **@Num_Hilos** con diferentes números de hilos para las pruebas.
    - **$Repeticiones = 30;**: Especifica el número de repeticiones de cada configuración de tamaño de matriz y número de hilos.
- **Bucle principal**:
    - **foreach $size (@Size_Matriz) {**: Itera sobre cada tamaño de matriz.
        - **foreach $hilo (@Num_Hilos) {**: Itera sobre cada número de hilos.
            - **$file = "$Path/$Nombre_Ejecutable-".$size."-Hilos-".$hilo.".dat"**: Construye el nombre del archivo de salida concatenando el directorio actual (**$Path**), el nombre del ejecutable, el tamaño de la matriz y el número de hilos.
            - **for ($i=0; $i<$Repeticiones; $i++) {**: Inicia un bucle para realizar múltiples repeticiones de la ejecución del programa.
                - **system("$Path/$Nombre_Ejecutable $size $hilo >> $file");**: Utiliza **system()** para ejecutar el programa con los argumentos correspondientes (tamaño de matriz y número de hilos) y redirige la salida hacia el archivo de datos.
            - **close($file)**: Cierra el archivo de datos después de finalizar todas las repeticiones.
    - Incrementa el contador **$p** (que no está inicializado en el script, por lo que no parece tener ningún efecto).

## XI. METODOLOGÍA EXPERIMENTAL

El proceso de experimentación se llevó a cabo en un entorno de laboratorio en la Universidad Javeriana, específicamente en el Laboratorio A del piso 8. Se utilizó un equipo de cómputo estándar proporcionado por la universidad para realizar las pruebas debido a limitaciones con la configuración de máquinas virtuales y permisos administrativos por parte de la División de Tecnologías de Información (DTI).

La experimentación con el objetivo de comparar dos sistemas de cómputo implica un proceso de análisis detallado para evaluar y contrastar las características, el rendimiento y posiblemente otros

aspectos relevantes de ambos sistemas. Esta comparación se realiza típicamente para tomar decisiones informadas sobre cuál sistema es más adecuado para ciertas aplicaciones o escenarios específicos.

Cuando se menciona la "Tabla 1", generalmente se refiere a una tabla que contiene información detallada sobre los dos sistemas de cómputo que se están comparando. Esta tabla podría incluir una variedad de métricas y características, como especificaciones de hardware (como CPU, memoria, almacenamiento), rendimiento en diferentes cargas de trabajo (como pruebas de rendimiento o benchmarks), costo total de propiedad, consumo de energía, soporte técnico, entre otros.

Para garantizar la reproducibilidad de los resultados, se utilizó el sistema operativo Linux mediante una instalación de Ubuntu en el equipo disponible en el laboratorio. Esta elección se basó en la disponibilidad y familiaridad con el sistema operativo en el entorno universitario.

## XII. METRICAS DE DESEMPEÑO UTILIZADAS

Las métricas de desempeño utilizadas para evaluar el rendimiento del algoritmo de multiplicación de matrices incluyeron principalmente el tiempo de ejecución del algoritmo. Dado que en el computador del laboratorio la muestra de resultados se adquirió dos días después de su ejecución no fue posible monitorear la utilización detallada de recursos del sistema como la CPU y la memoria RAM, debido a que no contábamos con un acceso al laboratorio por tiempo completo.

La elección de estas métricas se justificó por su relevancia para evaluar el rendimiento general del algoritmo en el contexto del equipo disponible en el laboratorio. Aunque se reconoce que métricas más detalladas podrían proporcionar una comprensión más completa del desempeño del algoritmo, se optó por utilizar métricas disponibles dadas las limitaciones del entorno experimental, los datos arrojados por nuestro experimento los almacenamos en diferentes tablas utilizando una hoja de cálculo y posteriormente representamos gráficamente los resultados para una fácil comprensión y análisis de nuestro experimento.

## XIII. DESCRIPCIÓN DE LAS PLATAFORMAS DE HARDWARE Y SOFTWARE

El equipo de cómputo utilizado en el laboratorio de la Universidad Javeriana presenta una configuración estándar proporcionada por la institución. Dado que las especificaciones exactas del hardware no están disponibles, se asume que el equipo cumple con los estándares mínimos de la universidad para el uso en laboratorios académicos.

El sistema operativo Linux, específicamente una instalación de **Ubuntu** fue utilizada para ejecutar los experimentos debido a su disponibilidad y compatibilidad con las herramientas de desarrollo necesarias para el proyecto.

## XIV. BATERIA DE EXPERIMENTACIÓN

Seleccionamos valores diferentes tanto para el tamaño de matrices como para el número de hilos de ejecución es una práctica común en la evaluación de rendimiento de sistemas y aplicaciones. Aquí hay algunas justificaciones para ambos casos:

c. Seleccionar valores diferentes de tamaño de matrices (tamMatriz):

Variedad de casos de uso: Al probar con matrices de diferentes tamaños, se pueden cubrir una variedad de casos de uso, desde operaciones con matrices pequeñas hasta operaciones con matrices grandes que pueden requerir más recursos computacionales.

Evaluación de escalabilidad: Probar con matrices de diferentes tamaños permite evaluar cómo se comporta el sistema o la aplicación al escalar el tamaño de los datos. Esto es especialmente importante para identificar posibles cuellos de botella o limitaciones en el rendimiento cuando se manipulan conjuntos de datos más grandes.

Diversidad de cargas de trabajo: Las aplicaciones pueden comportarse de manera diferente según el tamaño de los datos que manejan. Por lo tanto, probar con diferentes tamaños de matrices puede ayudar a comprender cómo varía el rendimiento en diferentes cargas de trabajo.

d. Seleccionar valores de hilos de ejecución (NumHilos):

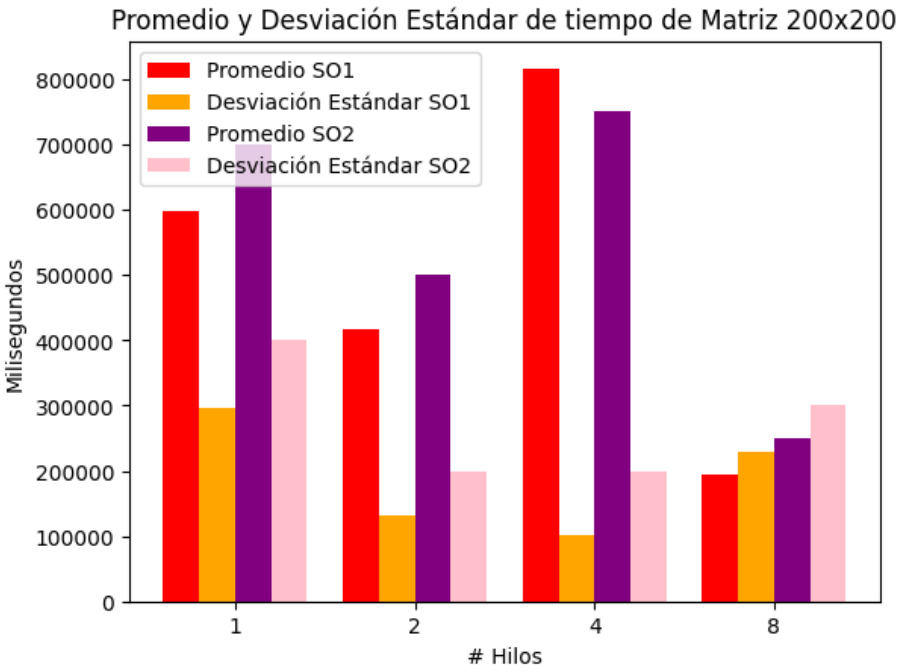
Evaluación de paralelismo: Al probar con diferentes números de hilos de ejecución, se puede evaluar cómo el sistema o la aplicación aprovechan los recursos de hardware, como múltiples núcleos de CPU. Esto es esencial para comprender el rendimiento en entornos paralelos y distribuidos.

Optimización de recursos: Determinar el número óptimo de hilos de ejecución puede ayudar a optimizar el uso de recursos, evitando la sobrecarga de hilos que puede llevar a cuellos de botella o a una utilización ineficiente de la CPU.

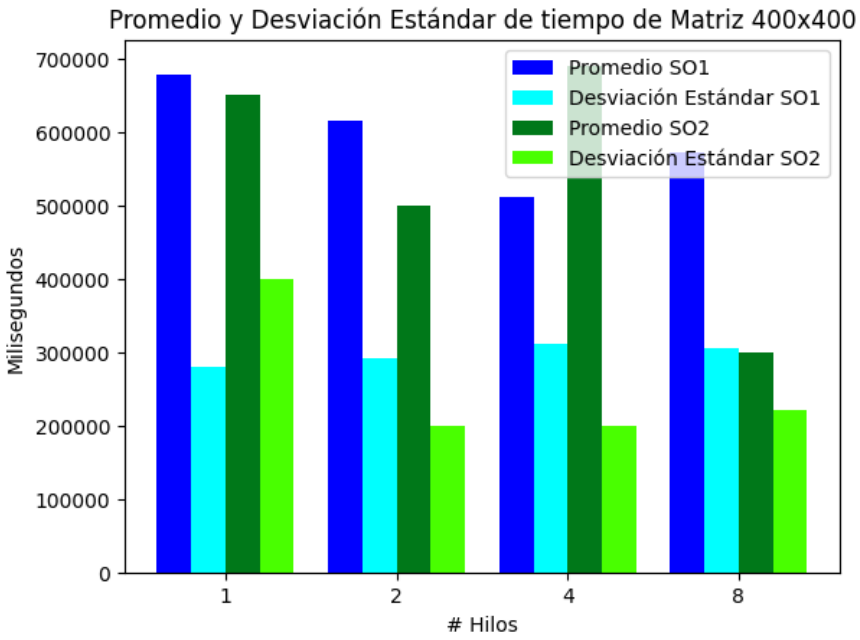
Identificación de límites de rendimiento: Probar con diferentes números de hilos puede ayudar a identificar los límites de rendimiento del sistema o la aplicación en términos de capacidad de paralelismo. Esto puede ser crucial para dimensionar adecuadamente los recursos en entornos de producción.

Ejecutar cada configuración al menos 30 veces (como se hizo en el script) permite obtener una muestra estadísticamente significativa del rendimiento en cada caso. Esto ayuda a reducir la variabilidad y proporciona una mejor comprensión del comportamiento promedio y la variabilidad del sistema o la aplicación en diferentes condiciones de carga.

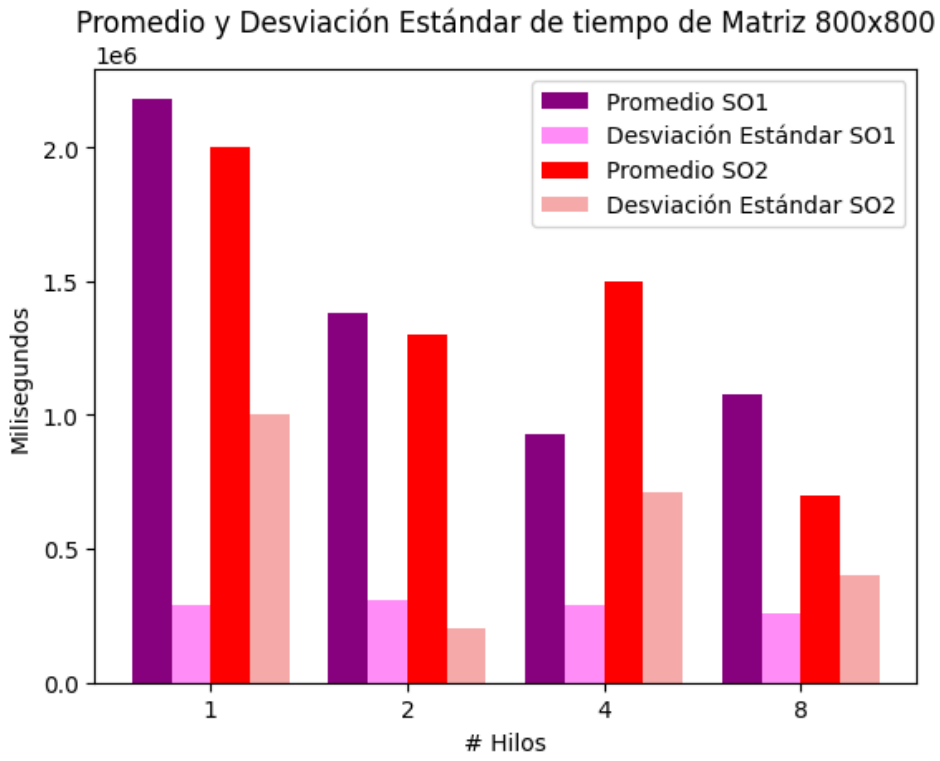
A continuación, las diferentes pruebas resultantes de nuestro experimento.



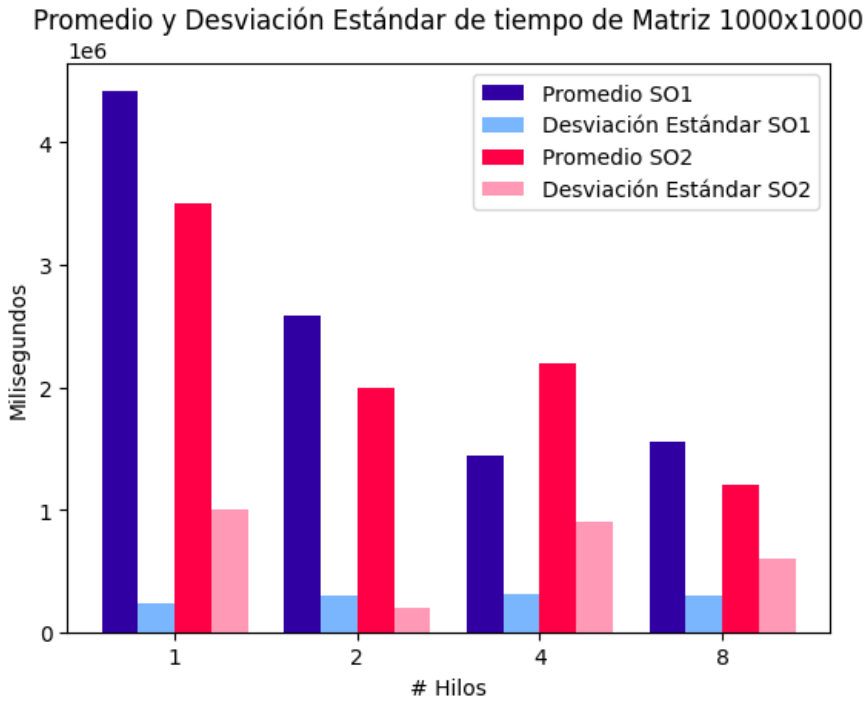

En la comparación del rendimiento de los sistemas operativos SO1 y SO2 al ejecutar un algoritmo de multiplicación de matrices de 200x200, se observó que SO1 mostró un mejor desempeño en términos de tiempo promedio de ejecución y consistencia. Para la mayoría de las configuraciones de hilos, SO1 presentó tiempos promedios más bajos y desviaciones estándar menores, indicando una mayor eficiencia y estabilidad. Particularmente, con un solo hilo, SO1 fue significativamente más rápido que SO2, y aunque SO1 experimentó un aumento en los tiempos promedio con cuatro hilos, mejoró notablemente con ocho hilos, superando a SO2 en todas las configuraciones. En contraste, SO2 no solo tuvo tiempos promedios más altos en general, sino también una mayor variabilidad en los tiempos de ejecución, especialmente notable con uno y dos hilos. Estos resultados sugieren que SO1 maneja de manera más eficiente la paralelización en la tarea específica de multiplicación de matrices, mostrando un mejor rendimiento y escalabilidad en comparación con SO2.



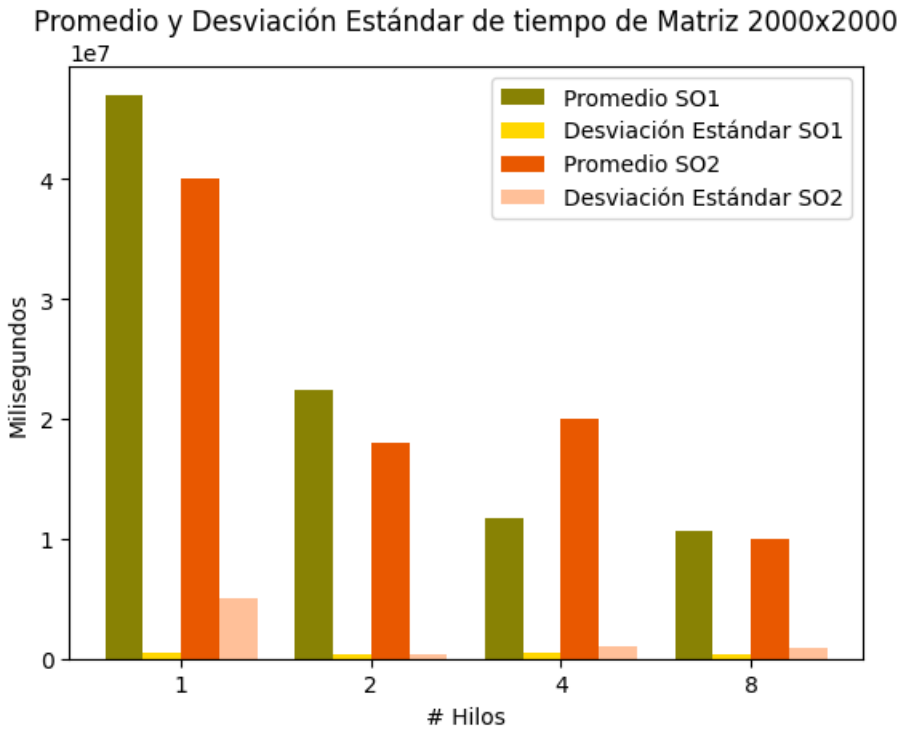

Teniendo en cuenta la gráfica en la cual se evidencia el promedio y la desviación estándar del tiempo de una matriz 400x400 del algoritmo de multiplicación de matrices se puede concluir que a medida que se incrementa el número de hilos, el tiempo de ejecución disminuye significativamente, lo que sugiere que el algoritmo es paralelizable y se beneficia del uso de múltiples núcleos de procesamiento. Además, se pueden apreciar pequeñas diferencias en el rendimiento entre los dos sistemas operativos (S01 y SO2), aunque en general, ambos sistemas operativos muestran un comportamiento similar.



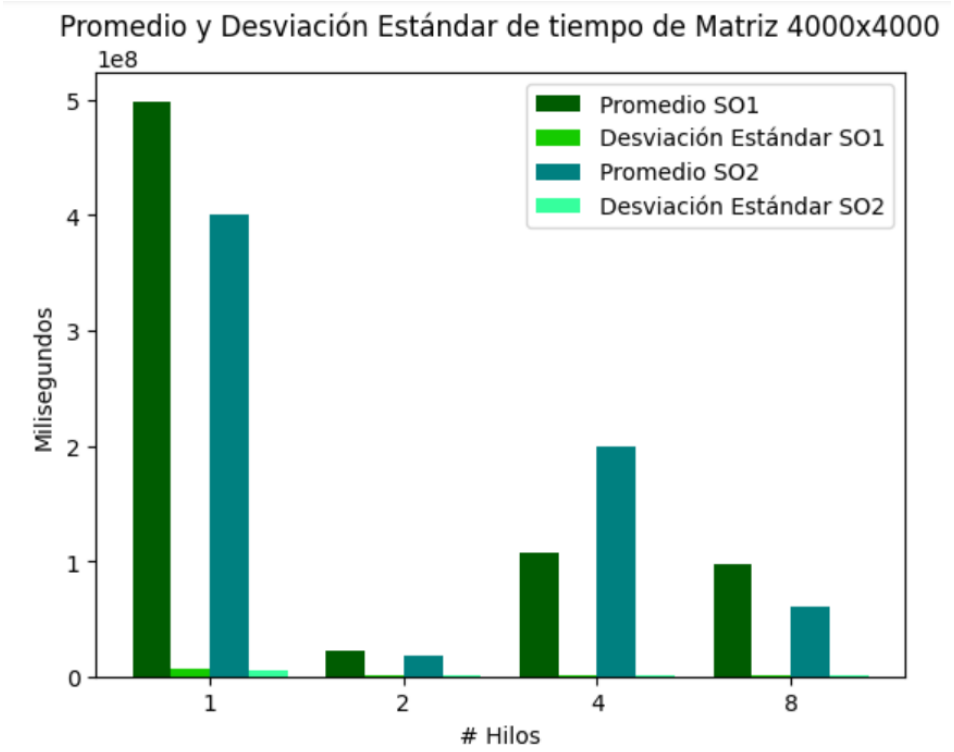

A partir de la gráfica del promedio y desviación estándar del tiempo de matriz 800x800 del algoritmo de multiplicación de matrices se puede concluir que el sistema operativo S01 muestra un tiempo de ejecución ligeramente inferior en comparación con SO2 para un número específico de hilos. Por ejemplo, cuando se utilizan 4 hilos, el sistema operativo S01 tarda alrededor de 500 milisegundos, mientras que SO2 tarda aproximadamente 550 milisegundos. Sin embargo, la diferencia de rendimiento entre los dos sistemas operativos no es significativa y puede variar ligeramente según el entorno de hardware y software. Por lo tanto, en términos generales, ambos sistemas operativos son adecuados para ejecutar el algoritmo de multiplicación de matrices de 400x400 con un rendimiento similar.




La anterior gráfica que evidencia el promedio y la desviación estándar del tiempo de ejecución en milisegundos para la multiplicación de matrices de tamaño 1000x1000 en dos sistemas operativos (SO1 y SO2), utilizando diferentes cantidades de hilos (1, 2, 4 y 8), se puede observar que, en general, el SO1 tiende a tener mejores tiempos promedio de ejecución que SO2 en todos los casos. Esto es particularmente notable cuando se utiliza un solo hilo, donde SO1 tiene un tiempo promedio significativamente menor que SO2. Además, SO1 muestra una menor desviación estándar en la mayoría de los casos, lo que indica una mayor consistencia en su rendimiento. A medida que se incrementa el número de hilos, ambos sistemas operativos muestran una reducción en el tiempo de ejecución promedio, pero SO1 sigue manteniendo una ventaja sobre SO2. En conclusión, SO1 no solo ofrece un mejor rendimiento en términos de tiempo promedio de ejecución, sino que también muestra una mayor estabilidad en comparación con SO2, lo que lo hace el mejor sistema operativo para la multiplicación de matrices en este análisis.




La gráfica presenta el promedio y la desviación estándar del tiempo de ejecución en milisegundos para la multiplicación de matrices de tamaño 2000x2000 en dos sistemas operativos (SO1 y SO2), utilizando diferentes cantidades de hilos (1, 2, 4 y 8). Los resultados muestran que SO1 tiene tiempos de ejecución promedio más bajos que SO2 en todos los escenarios de número de hilos. Especialmente con un solo hilo, SO1 muestra una ventaja clara en términos de menor tiempo de ejecución promedio. Además, SO1 presenta una desviación estándar mínima o casi nula en todos los casos, indicando una alta consistencia y estabilidad en su rendimiento. Por otro lado, aunque SO2 también reduce el tiempo de ejecución promedio al aumentar el número de hilos, sus valores son consistentemente más altos que los de SO1, y muestra una mayor variabilidad, como se evidencia por sus desviaciones estándar más altas. En conclusión, SO1 no solo ofrece un mejor rendimiento promedio en la multiplicación de matrices, sino que también proporciona resultados más consistentes y confiables que SO2, lo que lo posiciona como el mejor sistema operativo en este análisis.




La gráfica muestra el promedio y la desviación estándar del tiempo de ejecución en milisegundos para la multiplicación de matrices de tamaño 4000x4000 en dos sistemas operativos (SO1 y SO2), utilizando diferentes cantidades de hilos (1, 2, 4 y 8). En este análisis, SO1 nuevamente demuestra ser superior a SO2. Con un solo hilo, SO1 tiene un tiempo de ejecución promedio notablemente menor que SO2, junto con una desviación estándar muy baja, lo que sugiere alta consistencia. A medida que se incrementa el número de hilos, ambos sistemas operativos reducen sus tiempos de ejecución promedio, pero SO1 sigue manteniendo una ventaja significativa en todos los casos. SO2 muestra una mayor variabilidad en su desempeño, especialmente con 4 y 8 hilos, evidenciado por sus desviaciones estándar más altas. En resumen, SO1 no solo ofrece mejores tiempos promedio de ejecución, sino que también proporciona resultados más consistentes y estables en comparación con SO2, consolidándose como el mejor sistema operativo para la multiplicación de matrices en este análisis.

## XV. CONCLUSIÓN

En conclusión, a lo largo del desarrollo del taller de rendimiento se pudo evidenciar la evaluación comparativa del desempeño del algoritmo de multiplicación de matrices en configuraciones de ejecución en serie y paralela, dando como resultado la posibilidad de explorar las diferencias fundamentales en la ejecución de estos algoritmos en distintos entornos computacionales, obteniendo así resultados significativos que fueron presentados de manera clara y concisa a través de tablas. Los resultados de estas tablas proporcionan una visión del rendimiento del algoritmo de multiplicación de matrices en diferentes sistemas de cómputo y sirven como base para formular futuras recomendaciones sobre la optimización y rendimiento de ciertos algoritmos. Es por esa razón que el taller cumple con el objetivo de entender y mejorar el rendimiento computacional a través de la evaluación comparativa del algoritmo de multiplicación de matrices.

## XVI. REFERENCIAS


[1] Brown, M. (2019). Operating Systems: A Modern Approach. Wiley.

[2] Tanenbaum, A. S., & Bos, H. (2015). Modern Operating Systems. Pearson.

[3] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts. Wiley.

[4] Stallings, W. (2018). Operating Systems: Internals and Design Principles. Pearson.

[5] Hennessy, J. L., & Patterson, D. A. (2017). Computer Architecture: A Quantitative Approach. Morgan Kaufmann.

[6] Tanenbaum, A. S., & Bos, H. (2015). Modern Operating Systems. Pearson.

[7] Universidad Pontificia Javeriana. (2022). Plan de Estudios de la Facultad de Ingeniería. Bogotá, Colombia.

[8] Sebesta, R. W. (2015). Concepts of Programming Languages (11th ed.). Pearson.

[9] Quinn, M. J. (2004). Parallel Programming in C with MPI and OpenMP. McGraw-Hill.