RENDIMIENTO DE SISTEMAS DE COMPUTO

María José Cárdenas Machaca mariacardenasm@javeriana.edu.co

Juan Pablo Hernández Jp-hernandez@javeriana.edu.co

Juan Diego Palacios Toledo Ju.palacios@javeriana.edu.co

Andrés Leonardo Manrique Hernández andresl-manrique@javeriana.edu.co

1. RESUMEN

La presente investigación se centra en evaluar y terminar el mejor sistema de computo entre una muestra de tres computadores. Para ello, se utilizará un benchmark donde se ejecutarán los algoritmos de multiplicación de matrices clásico y transpuesto, variando el número de hilos utilizados en cada maquinas. De esta forma, se obtendrá el tiempo medio de ejecución de cada algoritmo y, se visualizará los datos mediante el uso de Pandas. Es así que, a partir de dicha visualización, se concluye que la maquina de Lenovo es la más estable y recomendada para el usuario debido a su rendimiento consistente.

2. INTRODUCCIÓN

El avance tecnológico ha revolucionado el ámbito de la computación, impulsando mejoras continuas en hardware y software. Este progreso ha generado una diversidad de computadoras en el mercado, cada una con características y capacidades distintas. Para los consumidores, elegir la máquina adecuada para sus necesidades puede ser una tarea complicada debido a las variaciones en precio, modelo y componentes.

En este contexto, la presente investigación tiene como objetivo evaluar el rendimiento de cinco sistemas de cómputo diferentes al ejecutar algoritmos de multiplicación de matrices, al haber dos sistemas con un número tan insignificante de hilos se consideró para las pruebas dejarlos por aparte, tanto en serie como en paralelo. El estudio se centra en la medición del tiempo de ejecución para diferentes tamaños de matrices y números de hilos, utilizando para ello herramientas de benchmarking y automatización de pruebas.

La investigación se plantea responder a la pregunta: ¿Cuál de los tres sistemas de cómputo ofrece el mejor rendimiento al ejecutar algoritmos de multiplicación de matrices? La respuesta a esta pregunta permitirá proporcionar recomendaciones informadas para seleccionar una computadora que optimice el rendimiento en tareas específicas de cómputo intensivo. Esta evaluación no solo ayudará a identificar la máquina más eficiente, sino también a entender cómo diferentes configuraciones de hardware y software influyen en el desempeño de algoritmos paralelos y secuenciales.

3. PROBLEMA

En el contexto actual, el entorno tecnológico ha evolucionado en gran medida, dando lugar a impresionantes avances tanto de hardware como en software. Es así que, este progreso refleja una amplia variedad de computadoras disponibles en el mercado, que difieren en aspecto como precio, modelo y componentes internos y externos. Esta diversidad puede resultar abrumadora para un consumidor promedio, quien enfrenta la difícil tarea en seleccionar entre estas múltiples maquinas, además de seleccionar una de ellas que se adapte adecuadamente a los labores y necesidades específicas del usuario.

Es así que, mediante este ensayo, se propone una investigación de campo con el fin de determinar la mejor máquina de tres muestras abordadas, las cuales se destacan por usar el sistema operativo Windows, pero con diferentes modelos y recursos, aplicando así, el algoritmo de multiplicación de matrices clásico y transpuesto.

Con la anterior premisa, se aborda la siguiente pregunta de investigación: ¿Cuál de las tres máquinas propuestas se puede recomendar a un usuario, considerando que proporciona el mejor rendimiento al ejecutar tanto el algoritmo clásico como transpuesto de multiplicación de matrices?

4. EVALUACIÓN DE RENDIMEINTO

Lo que realmente queremos evaluar es la capacidad de cómputo de cada máquina al momento de ejecutar cierto algoritmo de categoría serie o paralelo. Por eso usamos una técnica llamada Benchmarking para comparar el rendimiento de ciertas máquinas con condiciones justas (lograr la justicia en estos casos no suele suceder por los factores que interfieren) en un cierto criterio (Bailey 1985), como el procesamiento en hilos de un programa. En ese sentido, usaremos una aplicación benchmark para medir la eficiencia por cantidad de hilos. Es de suponer que el tiempo t de ejecución del programa, a medida que se incrementen los hilos, disminuirá de tal forma que el tiempo de ejecución para n hilos sería t/n.

También, nos apoyaremos en *la ley de los grandes números* para llegar a un resultado más acertado y cercano a la realidad. Por lo tanto, según esta ley, haremos comparativas de 30 ejecuciones por cada número de hilos. La idea es variar este último factor para conseguir resultados que nos sustenten la premisa de que, a mayor número de hilos, menor tiempo de ejecución.

5. EJECUCIÓN

5.1. Motivación

Dado que lo que queremos es comparar los tiempos de ejecución de un programa de acuerdo al número de hilos con que este se ejecute, necesitamos una aplicación de juguete

(benchmark) que funcione en paralelo, para que así los hilos puedan realizar tareas de forma independiente. En ese sentido, de acuerdo con (Bailey 1985), uno de los 7 tests intensivos que permiten probar sistemas paralelos de grano grueso es el famoso algoritmo de la multiplicación de matrices (MxM). Esto se debe a que, dada la forma en la que se multiplican matrices, es fácil dividir la tarea en subprocesos sin que uno dependa de otro, esto es, se sigue la filosofía de un algoritmo de *divide y vencerás*.

5.2. Sistemas de computo

Inicialmente, para el desarrollo de la investigación se tenía dispuesto utilizar 5 máquinas de estudio, las descritas en las FIG 1 y, adicionalmente, una máquina de Windows con 4 hilos y una de Linux con 6 hilos.

Sin embargo, con el fin de hacer una comparación más justa, se optó por seleccionar las tres máquinas que compartieran el mismo sistema operativo, siendo en este caso Windows, además de rectificar que el número de hilos de cada máquina no fueran tan distantes entre sí.

(1)

Modelo	Sistema operativo	Procesador	Proce. lógicos
HP Pavilion Gaming Laptop 15-dk0xxx	Microsoft Windows 11	Intel Core i7-9750H	12
Vivobook_ASUS Laptop K6500ZC	Microsoft Windows 11	12th Gen Intel Core i7-12700H	20
Lenovo IdeaPad 1 14ALC7 Laptop	Microsoft Windows 11	AMD Ryzen 7 5700U with Radeon Graphics	16

FIG 1. Detalle de los sistemas de cómputo a evaluar

Frente a lo anterior, y destacando la cantidad de procesadores lógicos, los cuales son de gran importancia para la ejecución de este algoritmo, se puede dar a suponer que la máquina más optima es la ASUS al tener 20 procesadores a su disposición, lo que debería brindarle mayor rapidez en la ejecución de tareas.

6. EVALUACIÓN DEL PARADIGMA DE PROGRAMACIÓN

6.1. Descripción algoritmo de multiplicación de matrices clásico y transpuesto

Para el desarrollo de esta investigación se utilizará el algoritmo de multiplicación de matrices clásico y transpuesto. Por ello, es importante comprender la aplicación de cada una para entender su implementación y diferencias en su versión en código.

En el caso del algoritmo clásico, se ejemplifica con la FIG 2. El procedimiento de este tipo de algoritmo consiste en tomar la primera fila de la matriz A, donde cada valor de la fila multiplicará a cada valor respectivamente de la primera columna de la matriz B, por lo que, el producto de estas será el resultado del primer valor de la matriz resultado.

(2)

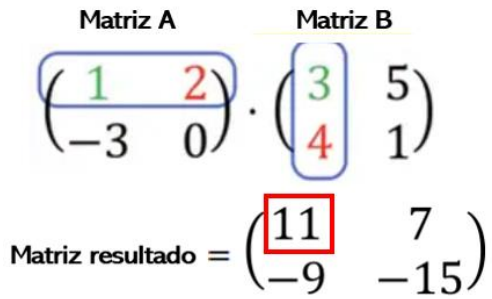

FIG 2. Algoritmo multiplicación de matrices clásico

De esa forma, la primera fila de la matriz A, debe repetir el proceso con la siguiente columna de la matriz B, encontrando así, el resultado de la siguiente fila de la matriz resultado. Así, se repite el desarrollo para las filas restantes de la matriz A.

Por otro lado, se tiene la FIG 3 la cual refleja el comportamiento del algoritmo transpuesto. En este ejemplo, se tiene una matriz A y una matriz B, donde la matriz A será transpuesta antes de realizar la multiplicación, para ello, las filas pasan a ser columnas y las columnas pasaran a ser filas, una vez acomodada la matriz, se procede con la multiplicación clásica que se conoce.

(3)

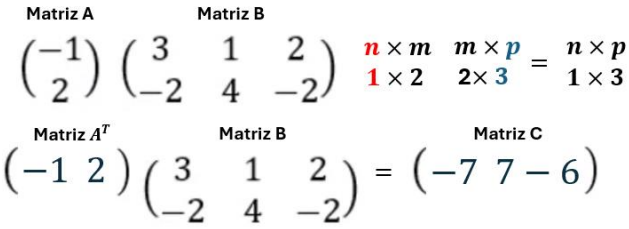

FIG 3. Algoritmo multiplicación de matrices transpuesta

Asimismo, es importante destacar que conociendo el tamaño de las matrices A y B, se puede determinar el tamaño de la matriz B o resultante.

6.2. Programación secuencial y paralela

La programación secuencial es aquella donde una acción o instrucción sigue a otra en secuencia. Por lo que la salida de una tarea es la entrada o indicador de inicio para que se ejecute la siguiente, así sucesivamente hasta finalizar el proceso. Este enfoque es fácil de comprender, pero puede llegar a ser ineficiente al momento de su ejecución, especialmente si hay tareas pueden realizarse simultáneamente.

Asimismo, la programación paralela, consisten en el uso de múltiples procesadores para ejecutar varias tareas simultáneamente, lo que conlleva a una mayor eficiencia en términos de tiempo a comparación con la programación en serie.

Con ello, es importante mencionar la funcionalidad de

los hilos, los cuales consisten en unidades de ejecución dentro de un proceso, por lo que comparten el mismo espacio de memoria y recursos que el proceso principal. Ante esto, tanto el código de algoritmo clásico como transpuesto utilizan múltiples hilos para dividir la tarea de multiplicar matrices en partes más pequeñas. Sin embargo, debido a que se utiliza la función pthread_join, el programa tiene que esperar a que todos los hilos finalicen antes de continuar por lo que se puede considerar una aplicación secuencial para ambos algoritmos.

7. METODOLOGÍA DE EXPERIMENTACIÓN

7.1. Leyes

En el contexto de la evaluación del rendimiento de algoritmos, es fundamental aplicar la teoría de probabilidad, particularmente la Ley de los Grandes Números e sus variantes. Esta ley establece que, conforme aumenta el número de ensayos o experimentos, el promedio de los resultados obtenidos tiende a aproximarse al valor esperado. En otras palabras, al realizar una serie de ejecuciones repetidas de un algoritmo, podemos obtener una estimación más precisa de su tiempo de ejecución promedio y, por ende, una mejor evaluación de rendimiento.

La ley de los Grandes Números se descompone en dos formas principales: la Ley Débil de los Grandes Números y la Ley Fuerte de los Grandes Números. Ambas indican que, con un número suficiente de observaciones, el promedio de los resultados converge al valor esperado, aunque la ley fuerte hace esta afirmación de manera más estricta. Para los propósitos de esta investigación, utilizaremos este principio para asegurar que nuestras mediciones de tiempo de ejecución sean representativas y confiables.

7.2. Lenguaje de programación

Para esta investigación se seleccionó el lenguaje C debido a su eficiencia y control sobre el hardware, características esenciales para la evaluación precisa de algoritmos de alto rendimiento. Además, se empleará el lenguaje PERL para automatizar la ejecución de múltiples pruebas, lo cual facilita la recolección de datos necesaria para la aplicación de la Ley de los Grandes Números.

7.3. Automatización

La automatización del proceso de experimentación se llevará a cabo mediante un script en PERL, denominado ***lanzador.pl*** . Este script se encargará de ejecutar el algoritmo de multiplicación de matrices, tanto en su versión clásica como transpuesta, con diferentes configuraciones de tamaño de matriz (tamMattriz) y número de hilos (NumHilos)

7.4. Pasos

- **Preparación de los sistemas de cómputo:**

  Seleccionar al menos dos sistemas de cómputo diferentes para realizar las pruebas.

  Instalar un entorno con el sistema operativo Linux, ya sea de forma nativa o en una máquina virtual.

- **Descompresión y documentación:**

  Descomprimir la carpeta enviada que contiene los ficheros fuente.

  Documentar cada función dentro de los ficheros fuente, incluyendo la función ***main***

- **Separación del fichero fuente:**

Dividir el fichero fuente (.c) en una biblioteca de funciones, interfaz y principal.

- **Análisis y documentación del script PERL:**

  Analizar y documentar el fichero en PERL ***lanzador.pl***, que servirá para automatizar la ejecución de los experimentos**.**

- **Compilación y ejecución:**

  Compilar los ficheros fuente utilizando el siguiente comando: *gcc *.c -o MM_ejecutable*

  Ejecutar el programa con diferentes valores de *tamMatriz* y *NumHilos* para verificar su correcto funcionamiento: *./MM_ejecutable tamMatriz NumHilos*

- **Batería de experimentación:**

  Seleccionar diferentes tamaños de matrices (justificando la elección de valores).

  Seleccionar diferentes números de hilos de ejecución (justificando la elección de valores).

  Ejecutar el programa al menos 30 veces para cada combinación de *tamMatriz* y *NumHilos*.

  Elaborar una tabla que represente la batería de experimentación para el análisis de rendimiento, tanto en serie (1 hilo) como en paralelo (2, 4, 8,… hilos). Hasta el número máximo de hilos que tenga disponible su procesador.

- **Automatización de la experimentación:**

  Modificar el fichero lanzador.pl para automatizar la captura de todas las ejecuciones.

  Ejecutar el script PERL para llevar a cabo los experimentos: *./lanzador.pl*

  Exportar los resultados a una hoja de cálculo para su análisis, obteniendo el promedio de los tiempos de ejecución para cada combinación de parámetros.

- **Evaluación y análisis de resultados:**

  Analizar los datos recopilados para determinar el rendimiento promedio de los algoritmos de multiplicación de matrices.

  Comparar los resultados obtenidos en diferentes sistemas de cómputo y bajo diferentes configuraciones de hilos.

8. EVALUACIÓN EXHAUSTIVA

8.1. Herramientas

Para esta evaluación, se utilizaron una combinación de herramientas y bibliotecas que facilitan tanto la recolección de datos como su análisis.

- Python: Lenguaje de programación utilizado para escribir los scripts de procesamiento y análisis.
- Pandas: Biblioteca de Python para la manipulación y análisis de datos. Esencial para

organizar y calcular estadísticas a partir de datos.

- Matplotlib: Biblioteca de Python para la visualización de datos. Utilizada para crear graficas que ayuden a interpretar los resultados.
- Expresiones regulares (`re`): Utilizado para extraer información específica de los nombres de los archivos de datos.
- Glob: Utilizada para encontrar todos los archivos de datos que coinciden con un patrón especifico en un directorio.

8.2. Análisis y resultados

Se siguieron unos pasos para lograr analizar los datos después de la ejecución y la obtención de los archivos con extensión (.dat).

- *Procesamiento de Datos:* Se utiliza el script de Python para leer estos archivos de datos, extraer la información relevante y calcular estadísticas descriptivas (media, varianza y desviación estándar).
- *Visualización de datos:* Se crean las graficas que muestran las medias de los tiempos de ejecución para diferentes configuraciones, lo que facilita la comprensión visual del rendimiento.

Script Detallado

- Importación de Módulos: Importamos las bibliotecas necesarias para el procesamiento visualización de datos.

```
import pandas as pd
import glob
import os
import matplotlib.pyplot as plt
import re
```

Función `process_files`:

- *Recolección de Archivos:* Utilizamos `glob` para listar todos los archivos `.dat` en el directorio especificado.
- *Extracción de información:* Utilizamos expresiones regulares para extraer el tamaño de la matriz y el número de hilos del nombre de cada archivo.
- *Lectura y conversión de Datos:* Leemos los tiempos de ejecución desde los archivos y los convertimos de microsegundos a segundos.
- *Creación de DataFrame:* Almacenamos los datos en un Dataframe de pandas.
- *Cálculo de Estadísticas:* Calculamos la media de los tiempos de ejecución para cada combinación de tamaño de matriz y numero de hilos.

```
def process_files(dir_path,
output_csv):
    files =
glob.glob(os.path.join(dir_path,
"*.dat"))
    all_data = []

    for file in files:
        # Extraer detalles del archivo
a partir del nombre del archivo
        match =
re.search(r'MM_ejecutable-(\d+)-Hilos-
(\d+).dat', os.path.basename(file))
        if match:
            matrix_size =
int(match.group(1))
            thread_count =
int(match.group(2))
            data = []

            with open(file, 'r') as f:
                for line in f:
                    if
line.strip().startswith(':->'):
                        time_us =
int(re.search(r'\d+', line).group())

data.append(time_us / 1000000.0)  #
Convert from µs to seconds

            if data:
                df = pd.DataFrame({
                    'Matrix Size':
[matrix_size] * len(data),
                    'Thread Count':
[thread_count] * len(data),
                    'Time (s)': data
                })
                all_data.append(df)

    if not all_data:
        print("No valid data found.")
        return

    combined_data = pd.concat(all_data,
ignore_index=True)
    combined_data.to_csv(output_csv,
index=False)
    print(f"Data saved to
{output_csv}")
combined_data = pd.concat(all_data,
ignore_index=True)
combined_data.to_csv(output_csv,
index=False)
print(f"Data saved to {output_csv}")
```

Visualización:

- *Configuración de la gráfica:* Configuramos la gráfica con títulos y etiquetas.
- *Trazado de datos:* Trazamos las medias de los tiempos de ejecución en la grafica
- *Guardado y Mostrado:* Guardamos la grafica en un archivo PNG y la mostramos en pantalla.

```
# Calculate and plot the mean execution
time for each matrix size and thread
```

```
count combination
mean_data =
combined_data.groupby(['Matrix Size',
'Thread Count'])['Time
(s)'].mean().reset_index()

plt.figure(figsize=(12, 8))
for (matrix_size, group) in
mean_data.groupby('Matrix Size'):
    plt.plot(group['Thread Count'],
group['Time (s)'], marker='o',
linestyle='-', label=f'Matrix Size
{matrix_size}')

plt.title('Tiempo por Matriz y Cantidad
de Hilos')
plt.xlabel('Contador de Hilos')
plt.ylabel('Tiempo (s)')
plt.legend(title='Matrix Size')
plt.grid(True)
plt.savefig(os.path.join(dir_path,
'average_execution_times.png'))
plt.show()
```

Resultados

- *Algoritmo de multiplicación Clásica:* A continuación, se presentan el comportamiento de la ejecución del algoritmo de multiplicación clásico hasta el tamaño 3200 para cada una de las tres máquinas.

(4)

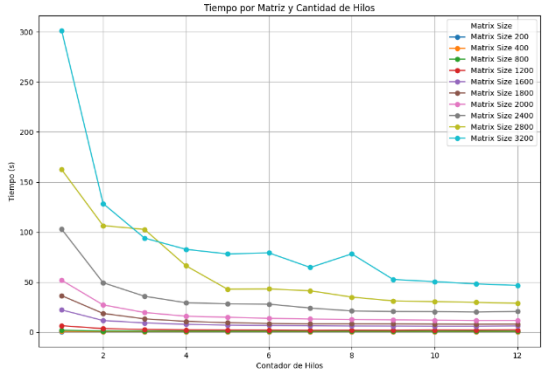

FIG 4. Comportamiento algoritmo clásico máquina 1 - HP Pavilion Gaming Laptop 15-dk0xxx

(5)

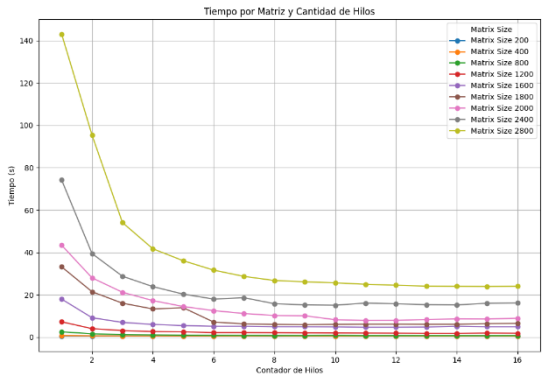

FIG 5. Comportamiento algoritmo clásico máquina 2 - Lenovo IdeaPad 1 14ALC7 Laptop

(6)

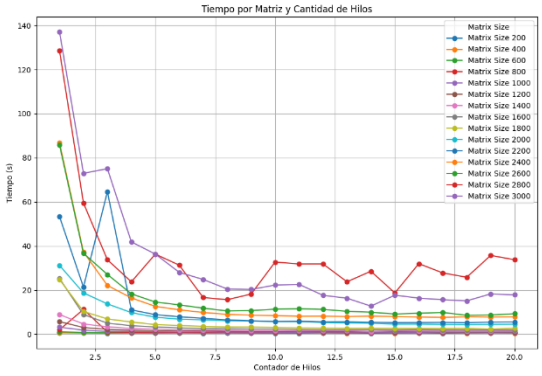

FIG 6. Comportamiento algoritmo clásico máquina 3 - Vivobook_ASUSLaptop K6500ZC

- *Comparativa de sistemas Clásico:* Tomando los resultados obtenidos al ejecutar el algoritmo clásico de tamaño 3200 en un hilo de ejecución, se hacer la comparación de las tres máquinas, observando que la maquina 1 es la que más intermitencia presenta en su ejecución.

(7)

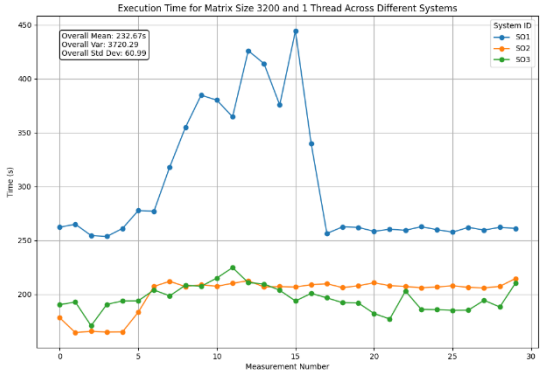

FIG 7. Comparativo algoritmo clásico de tamaño 3200 en un hilo de ejecución

- *Algoritmo de multiplicación Transpuesta:* A continuación, se presentan el comportamiento de la ejecución del algoritmo de multiplicación transpuesta hasta el tamaño 3200 para cada una de las tres máquinas.

(8)

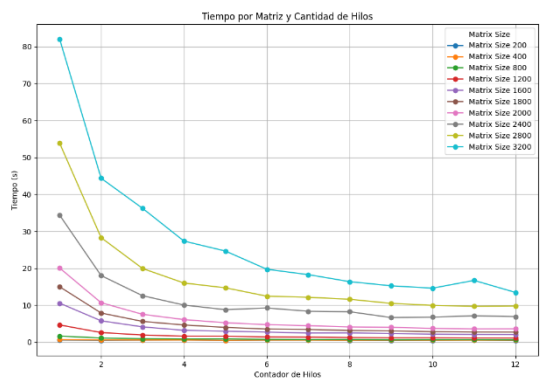

FIG 8. Comportamiento algoritmo transpuesta máquina 1 - HP Pavilion Gaming Laptop 15-dk0xxx

(9)

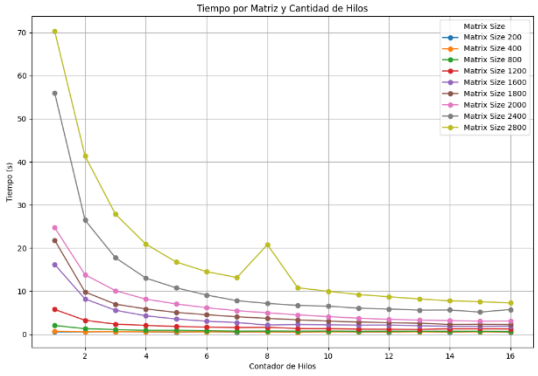

FIG 9. Comportamiento algoritmo transpuesta máquina 2 - Lenovo IdeaPad 1 14ALC7 Laptop

(10)

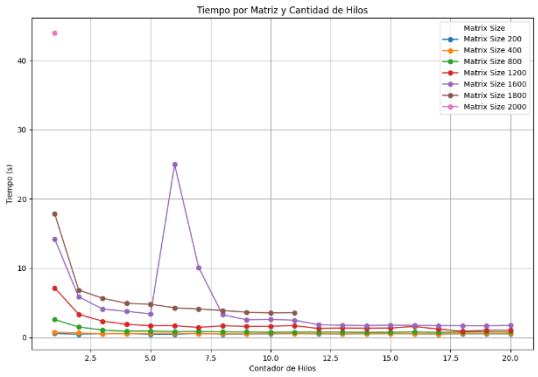

FIG 10. Comportamiento algoritmo transpuesta máquina 3 - Vivobook_ASUSLaptop K6500ZC

- *Comparativa de sistemas Transpuesta:* Tomando los resultados obtenidos al ejecutar el algoritmo transpuesta de tamaño 3200 en un hilo de ejecución, se hacer la comparación de las tres máquinas, observando que la maquina 3 es la que más intermitencia presenta en su ejecución.

(11)

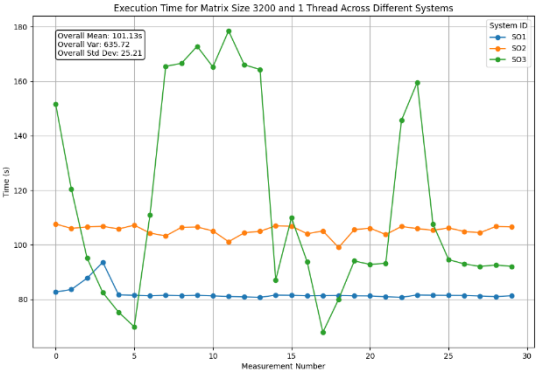

FIG 11. Comparativo algoritmo transpuesta de tamaño 3200 en un hilo de ejecución

9. CONCLUSIONES Y RECOMENDACIONES

A partir del desarrollo investigativo, análisis y visualización de gráficas se destaca la importancia de un enfoque estadístico, el uso eficiente del lenguaje C y la automatización con scripts en PERL. La paralelización de los algoritmos mostró mejoras significativas en el rendimiento, pero estas son dependientes del hardware y la configuración del sistema.

Asimismo, como se puede observar en la FIG 7 y FIG 11 revelan que las maquina 1 y 3, respectivamente, presentaban picos en su rendimiento. Dicho comportamiento puede deberse a dos motivos principales: El primero, se ejecutaban nuevas tareas donde el sistema operativo les daba mayor prioridad de ejecución, relegando al algoritmo de multiplicación de matrices a un segundo plano, ocasionando que el tiempo de ejecución se extendiera, y, como segunda opción, las maquinas constantemente ejecutan programas predeterminados que requieren ciertos recursos, interfiriendo con la ejecución del algoritmo.

Por otro lado, es importe hacer mención del tiempo de ejecución, pues se pudo concluir que este factor depende en gran medida del tamaño de la matriz y del número de hilos, ya que, el tiempo de ejecución aumenta con el tamaño de la matriz, pero disminuye considerablemente al incrementar el número de hilos de ejecución.

De igual forma, es importante hacer ciertas recomendaciones, si bien el desarrollo de esta prueba se ejecuto en maquinas personas, es necesario utilizar maquinas dedicadas, es decir, maquinas libres de oras tareas o programas predeterminados, que únicamente sean utilizadas par ejecutar tareas relacionadas al benchmark de prueba, ya que esto, ayudará a obtener resultados más precisos y adaptados a las necesidades del usuario. De igual forma, al tener una maquina dedicada a una tarea es posible asegurar la optimización del sistema operativo y configurarlo de tal manera que se minimicen las interferencias y se le prioridad al algoritmo de multiplicación de matrices.

Finalmente, en base a las FIG 7 y FIG 11 se considera que la maquina Lenovo IdeaPad 1 14ALC7 Laptop con un procesador AMD Ryzen 7 5700U with Radeon Graphics, es la mejor máquina, y por ende, la recomendada para el usuario, ya que esta, se mantuvo más estable en la ejecución tanto del algoritmo clásico como en el transpuesto, evitando picos de gran escala e impredecibles, como sucedió con las maquinas 1 y 3. Si bien está maquina no tuvo el mejor tiempo de ejecución en el algoritmo transpuesto, es posible notar como aprovecho los recursos disponibles de forma óptima.

10. REFERENCIAS

[1] Bailey, D. H., & Barton, J. T. (1985). *The NAS kernel benchmark program* (No. NAS 1.15: 86711)

[2] McKinney, W. (2011). pandas: a foundational Python library for data analysis and statistics. *Python for high performance and scientific computing*, *14*(9), 1-9.

[3] McKinney, W., & Team, P. D. (2015). Pandas-Powerful python data analysis toolkit. *Pandas—Powerful Python Data Analysis Toolkit*, *1625*.

[4] McKinney, W. (2012). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc.".

[5] Tosi, S. (2009). *Matplotlib for Python developers*. Packt Publishing Ltd.

[6] Yim, A., Chung, C., & Yu, A. (2018). *Matplotlib for Python Developers: Effective techniques for data visualization with Python*. Packt Publishing Ltd.

[7] Edan (2009). *Algoritmos y estructuras de programación*.