Reporte técnico: Proyecto final de Sistemas Operativos y Laboratorio

1. Información del Proyecto

- **Título del Proyecto:** Comparación de Rendimiento entre Ubuntu y Raspberry Pi OS en Aplicaciones IoT
- **Curso/Materia:** Sistemas Operativos
- **Integrantes:** Sofia Vanegas Cordoba sofia.vanegasc@udea.edu.co, Samuel David Montoya montoyacanos@gmail.com
- **Fecha de Entrega:** 16/07/2025

2. Introducción

2.1. Objetivo del Proyecto

El objetivo principal del proyecto es analizar y comparar el rendimiento de la ejecución secuencial y paralela de un algoritmo de multiplicación de matrices, utilizando métricas como uso de CPU, memoria y tiempo de ejecución. A través de la implementación en diferentes entornos (por ejemplo, en Ubuntu y Raspberry Pi OS), se busca demostrar cómo la paralelización puede mejorar el desempeño computacional y qué impacto tiene en sistemas operativos con recursos limitados. Este análisis permite entender mejor el comportamiento del procesamiento paralelo en plataformas de bajo consumo energético.

2.2. Motivación y Justificación

La multiplicación de matrices es una operación fundamental en muchos campos como la inteligencia artificial, procesamiento de imágenes, simulaciones físicas y modelado numérico. Sin embargo, esta operación puede ser muy costosa computacionalmente, especialmente cuando se realizan con grandes volúmenes de datos. Evaluar su implementación en modo secuencial y paralelo, en plataformas limitadas como una Raspberry Pi, permite estudiar los beneficios reales del paralelismo y las limitaciones prácticas del hardware. Además, el análisis de rendimiento es un tema esencial dentro del estudio de sistemas operativos y computación concurrente, lo que justifica su elección académica y técnica.

2.3. Alcance del Proyecto

Este proyecto se enfoca en la implementación de un algoritmo de multiplicación de matrices en dos versiones: una secuencial y otra paralela, utilizando pipes. Se medirán y compararán métricas como tiempo de ejecución, uso de CPU y memoria, en dos sistemas operativos: Ubuntu y Raspberry Pi OS.

Quedan fuera del alcance del proyecto aspectos como la visualización de resultados gráficos, el uso de GPUs, optimizaciones matemáticas avanzadas (como Strassen o algoritmos con SIMD), o el análisis en arquitecturas multinodo.

3. Marco Teórico / Conceptos Fundamentales

Este proyecto se enmarca en el estudio del rendimiento computacional en operaciones intensivas como la multiplicación de matrices, bajo distintos paradigmas de ejecución (secuencial y paralela). A continuación se presentan los conceptos clave que sustentan el trabajo:

- **Multiplicación de matrices**: Es una operación aritmética básica que consiste en combinar dos matrices para generar una nueva. Es intensiva en cómputo, especialmente cuando las matrices son grandes, y su complejidad es **$O(n^3)$** para la implementación clásica.

- **Ejecución secuencial vs. paralela**: En la versión secuencial, las operaciones se realizan una tras otra utilizando un solo hilo de ejecución. En la paralela, se divide el trabajo entre múltiples **hilos**, **procesos** o **goroutines** (en Go), lo que permite aprovechar múltiples núcleos del procesador para realizar cálculos simultáneamente.

- Paralelismo en sistemas operativos: Los sistemas modernos soportan múltiples hilos y procesos, lo que permite implementar técnicas de paralelismo a nivel de software. El rendimiento puede variar significativamente dependiendo de cómo el sistema operativo administra el planificador de procesos, la memoria, y los recursos del sistema.

- **Métricas de rendimiento**: Las principales métricas analizadas incluyen:

    - **Tiempo de ejecución** total del algoritmo.

    - **Uso de CPU**, medido en porcentaje.

    - **Consumo de memoria RAM** durante la operación.

    - **Speedup**, definido como el cociente entre el tiempo secuencial y el tiempo paralelo.

- **Plataforma Raspberry Pi**: Este hardware representa un entorno de recursos limitados ideal para evaluar cómo se comportan algoritmos computacionalmente intensivos en sistemas embebidos.

4. Diseño e Implementación

El proyecto implementa el algoritmo de **multiplicación de matrices** en dos modalidades:

- **Secuencial**: Un solo proceso realiza toda la multiplicación usando los bucles clásicos `i-j-k`.

- **Paralela**: El proceso principal crea múltiples **procesos hijos** mediante `fork()`, y distribuye el cálculo de la matriz resultado. Cada hijo realiza una porción del trabajo y comunica los resultados al padre mediante **pipes anónimos**.

Adicionalmente, tras la ejecución, el programa compila las métricas recolectadas (tiempo, uso de CPU, uso de memoria, tamaño de matriz, número de procesos, etc.) y las envía a **Firebase Realtime Database**. Esto permite:

- **Centralizar resultados** de distintas pruebas (por ejemplo, usando varias Raspberry Pi o corriendo en distintos sistemas operativos).

- **Visualizar remotamente** los resultados en tiempo real o almacenarlos para análisis posterior.

El diagrama de arquitectura podría mostrar tres partes:

1. Proceso padre e hijos (pipe IPC)

2. Recopilación de métricas de ejecución

3. Envío de datos estructurados a Firebase

4.2. Tecnologías y Herramientas

- **Lenguaje**: C++17

- **Modelo de paralelismo**: Procesos (`fork`) + comunicación mediante **pipes**

- **Sistemas operativos**:

    - Ubuntu Server

    - Raspberry Pi OS Lite

- **Compilador**: `g++`

- **Librerías y utilidades**:
    - `unistd.h`, `sys/wait.h`, `sys/resource.h` para procesos e IPC
    - `chrono` para medición de tiempo
    - `fstream`, `vector`, `iostream` para manejo de estructuras
    - `libcurl` para enviar datos a **Firebase** mediante HTTP
- **Herramientas de monitoreo**:
    - `htop`/`top` (interactivo)
    - `getrusage()` (uso de CPU y memoria)
    - `time` (medición externa)
- **Firebase Realtime Database**:
    - Se envían métricas de cada ejecución en formato JSON a una base de datos centralizada.
    - Permite almacenar: tamaño de matriz, número de procesos, tiempo total, uso de CPU/RAM, speedup.
- **Scripts auxiliares**: Bash para automatizar corridas y generar carga de trabajo variada

4.3. Detalles de Implementación

La implementación del sistema se organizó en módulos bien definidos para facilitar el mantenimiento y la portabilidad entre sistemas operativos. Los aspectos clave son:

- **Lectura de sensores**: Se utilizaron bibliotecas específicas para acceder a los sensores conectados a los pines GPIO de la Raspberry Pi. Por ejemplo, se configuró un pin de entrada para el sensor de sonido y un protocolo de comunicación para el sensor de temperatura y humedad (como el protocolo de un DHT11).
- **Monitoreo del sistema**: Se accedió a archivos del sistema como `/proc/stat` y `/proc/meminfo` para obtener métricas del uso de CPU y RAM desde C++, sin depender de herramientas externas. También se consideró el uso de comandos como `top` o `htop` para validación.
- **Envío a Firebase**: Se construyó una solicitud HTTP tipo POST usando una biblioteca como `libcurl`, donde se enviaban los datos recolectados en formato JSON al endpoint

de Firebase Realtime Database.

- **Control de flujo**: Se utilizó un bucle principal que implementa un retardo (por ejemplo, con `std::this_thread::sleep_for`) para realizar la lectura en intervalos constantes (por ejemplo, cada 10 segundos).

- **Configuración**: Los parámetros como intervalos de lectura, rutas de acceso a sensores o URL del endpoint de Firebase fueron definidos en archivos de configuración o constantes dentro del código fuente (estos se pueden referenciar en los anexos del informe).

Este enfoque modular permitió reutilizar la misma aplicación en ambas Raspberry Pi, variando únicamente el sistema operativo para comparar su rendimiento bajo condiciones idénticas.

5. Pruebas y Evaluación

5.1. Metodología de Pruebas

Para validar el correcto funcionamiento y medir el rendimiento del proyecto, se diseñó y ejecutó un conjunto de **casos de prueba sistemáticos**. El objetivo de estas pruebas fue comparar la ejecución del algoritmo de multiplicación de matrices en sus versiones **secuencial y paralela** (utilizando procesos e IPC con pipes), evaluando métricas como **tiempo total de ejecución**, **uso de CPU**, **uso de memoria** y **speedup**.

Enfoque de pruebas

Se utilizó un enfoque de **pruebas experimentales controladas**, variando dos parámetros clave:

- El **tamaño de las matrices**, para simular distintas cargas de trabajo.

- La **cantidad de procesos**, para evaluar la escalabilidad del paralelismo basado en `fork()`.

Las pruebas fueron diseñadas para cubrir tanto casos de carga baja como alta, así como situaciones de **sobrecarga de procesos**, donde la cantidad de procesos es mayor al número de filas disponibles.

Casos de prueba definidos

Se probaron tres tamaños de matrices cuadradas:

- 400×400

- 800×800
- 1600×1600

Para cada tamaño, se ejecutó el algoritmo paralelo utilizando 1, 100 y 500 procesos. Esto generó un total de **9 combinaciones únicas**, además de su respectiva versión secuencial como referencia.

Cada combinación se ejecutó **tres veces** para reducir el efecto de variaciones en el sistema operativo, usando la media como valor representativo.

Métricas recolectadas

Durante cada ejecución, el sistema capturó automáticamente las siguientes métricas:

- Tiempo de ejecución secuencial (`timeSeqMs`)
- Tiempo de ejecución paralelo (`timeParMs`)
- Speedup (`timeSeqMs / timeParMs`)
- Uso de CPU en modo usuario y sistema (`cpuUserMs`, `cpuSysMs`)
- Memoria máxima utilizada (`maxRssKB`)
- Número de procesos usados (`numProcesses`)
- Timestamp de la ejecución

Estas métricas se empaquetaron en un objeto JSON y se enviaron a **Firebase Realtime Database**, permitiendo la consulta centralizada de resultados y facilitando su visualización y análisis posterior.

5.2. Casos de Prueba y Resultados

Ubuntu

ID Caso de prueba	Descripción del caso de prueba	Resultado obtenido
CP-001	400X400 1 proceso	timestamp : 1752426048 matrixRows : 400 matrixCols : 400 numProcesses : 1 timeSeqMs : 2399

		timeParMs : 2359 speedup : 1.0169563374311148 cpuUserMs : 9 cpuSysMs : 7 maxRssKB : 12160
CP-002	400X400 100 procesos	timestamp : 1752427474 matrixRows : 400 matrixCols : 400 numProcesses : 100 timeSeqMs : 2362 timeParMs : 801 speedup : 2.948813982521848 cpuUserMs : 18 cpuSysMs : 225 maxRssKB : 11648
CP-003	400X400 500 procesos	timestamp : 1752428019 matrixRows : 400 matrixCols : 400 numProcesses : 500 timeSeqMs : 2415 timeParMs : 3376 speedup : 0.7153436018957346 cpuUserMs : 19 cpuSysMs : 2466 maxRssKB : 11648
CP-004	800X800 1 procesos	timestamp : 1752426297 matrixRows : 800 matrixCols : 800 numProcesses : 1 timeSeqMs : 25090 timeParMs : 24624 speedup : 1.0189246263807668 cpuUserMs : 25 cpuSysMs : 22 maxRssKB : 22912
CP-005	800X800 100 procesos	timestamp : 1752427545 matrixRows : 800 matrixCols : 800 numProcesses : 100 timeSeqMs : 26641 timeParMs : 15879 speedup : 1.677750488065999 cpuUserMs : 69 cpuSysMs : 583 maxRssKB : 20480
CP-006	800X800 500 procesos	timestamp : 1752428075 matrixRows : 800 matrixCols : 800

		numProcesses : 500 timeSeqMs : 24354 timeParMs : 12450 speedup : 1.956144578313253 cpuUserMs : 179 cpuSysMs : 2236 maxRssKB : 20480
CP-007	1600X1600 1 proceso	timestamp : 1752426873 matrixRows : 1600 matrixCols : 1600 numProcesses : 1 timeSeqMs : 248749 timeParMs : 249142 speedup : 0.9984225863162374 cpuUserMs : 97 cpuSysMs : 56 maxRssKB : 65664
CP-008	1600X1600 100 procesos	timestamp : 1752427994 matrixRows : 1600 matrixCols : 1600 numProcesses : 100 timeSeqMs : 249050 timeParMs : 170821 speedup : 1.4579589160583302 cpuUserMs : 174 cpuSysMs : 628 maxRssKB : 55808
CP-009	1600X1600 500 procesos	RB : Samuel timestamp : 1752428507 matrixRows : 1600 matrixCols : 1600 numProcesses : 500 timeSeqMs : 248458 timeParMs : 157965 speedup : 1.5728674073370683 cpuUserMs : 446 cpuSysMs : 3312 maxRssKB : 55808

RaspberryPiOs

ID Caso de prueba	**Descripción del caso de prueba**	**Resultado obtenido**
CP-001	400X400 1 proceso	timestamp : 1752426063 matrixRows : 400 matrixCols : 400

		numProcesses : 1 timeSeqMs : 2354 timeParMs : 2376 speedup : 0.9907407407407407 cpuUserMs : 7 cpuSysMs : 0 maxRssKB : 12072
CP-002	400X400 100 procesos	timestamp : 1752427473 matrixRows : 400 matrixCols : 400 numProcesses : 100 timeSeqMs : 2342 timeParMs : 700 speedup : 3.3457142857142856 cpuUserMs : 25 cpuSysMs : 72 maxRssKB : 11444
CP-003	400X400 500 procesos	timestamp : 1752428024 matrixRows : 400 matrixCols : 400 numProcesses : 500 timeSeqMs : 2328 timeParMs : 1019 speedup : 2.2845927379784103 cpuUserMs : 0 cpuSysMs : 447 maxRssKB : 11448
CP-004	800X800 1 procesos	timestamp : 1752426337 matrixRows : 800 matrixCols : 800 numProcesses : 1 timeSeqMs : 24695 timeParMs : 25463 speedup : 0.9698385893256882 cpuUserMs : 20 cpuSysMs : 8 maxRssKB : 22840
CP-005	800X800 100 procesos	timestamp : 1752427531 matrixRows : 800 matrixCols : 800 numProcesses : 100 timeSeqMs : 24676 timeParMs : 6802 speedup : 3.6277565421934725 cpuUserMs : 87 cpuSysMs : 80 maxRssKB : 20280
CP-006	800X800 500 procesos	timestamp : 1752428074 matrixRows : 800 matrixCols : 800

		numProcesses : 500 timeSeqMs : 24185 timeParMs : 7161 speedup : 3.377321603128055 cpuUserMs : 160 cpuSysMs : 575 maxRssKB : 20280
CP-007	1600X1600 1 proceso	timestamp : 1752426881 matrixRows : 1600 matrixCols : 1600 numProcesses : 1 timeSeqMs : 244646 timeParMs : 256507 speedup : 0.9537595465230968 cpuUserMs : 88 cpuSysMs : 20 maxRssKB : 65592
CP-008	1600X1600 100 procesos	timestamp : 1752427891 matrixRows : 1600 matrixCols : 1600 numProcesses : 100 timeSeqMs : 245598 timeParMs : 73120 speedup : 3.358834792122538 cpuUserMs : 173 cpuSysMs : 156 maxRssKB : 55608
CP-009	1600X1600 500 procesos	timestamp : 1752428426 matrixRows : 1600 matrixCols : 1600 numProcesses : 500 timeSeqMs : 244765 timeParMs : 75116 speedup : 3.2584935300069224 cpuUserMs : 899 cpuSysMs : 313 maxRssKB : 55608

5.3. Análisis de resultados

Escalabilidad

La escalabilidad se refiere a cómo el rendimiento de la aplicación cambia a medida que se aumenta el número de procesos.

Ubuntu:

- **Speedup:** Para la matriz de 400×400, el speedup aumenta significativamente hasta los 100 procesos, alcanzando casi 3x, pero luego cae drásticamente a 500 procesos. Para las matrices de 800×800 y 1600×1600, el speedup aumenta de manera más gradual con

el número de procesos, sin la caída pronunciada que se observa en la matriz más pequeña al aumentar a 500 procesos. Esto sugiere que para matrices más grandes, Ubuntu puede aprovechar mejor el paralelismo a medida que aumenta el número de procesos, aunque con retornos decrecientes a medida que se añaden más procesos (de 100 a 500).
- **Eficiencia:** La eficiencia disminuye consistentemente a medida que aumenta el número de procesos para todos los tamaños de matriz. Esto es esperado, ya que la sobrecarga de comunicación y gestión de procesos tiende a aumentar, superando los beneficios del paralelismo extremo. A 500 procesos, la eficiencia es casi nula, lo que indica que añadir más procesos a partir de cierto punto no es beneficioso y puede incluso ser perjudicial para el rendimiento.

RaspberryPiOS:

- **Speedup:** Similar a Ubuntu, el speedup en RaspberryPiOS muestra un aumento para todos los tamaños de matriz hasta los 100 procesos, siendo el tamaño de 800×800 el que presenta el mayor speedup. Sin embargo, al igual que en Ubuntu, el speedup también disminuye significativamente al pasar de 100 a 500 procesos para el tamaño de 400×400. Para 800×800 y 1600×1600, el speedup se mantiene más estable o incluso disminuye ligeramente entre 100 y 500 procesos, pero no tan drásticamente como en el caso de 400×400.
- **Eficiencia:** La eficiencia en RaspberryPiOS también muestra una disminución similar a la de Ubuntu a medida que aumenta el número de procesos. Los beneficios de añadir más procesos disminuyen rápidamente después de 100 procesos.

Comparación: Ambos sistemas muestran un patrón similar de aumento de speedup hasta cierto punto (aproximadamente 100 procesos) seguido de una disminución o estancamiento. La eficiencia en ambos cae a medida que se añaden más procesos. Curiosamente, para la matriz 400×400 con 500 procesos, el speedup de Ubuntu cae por debajo de 1.0 (significando que es más lento que la versión secuencial), mientras que RaspberryPiOS aún mantiene un speedup por encima de 2.0 en esa misma configuración. Esto sugiere una sobrecarga de procesos mucho mayor o una gestión de recursos menos eficiente en Ubuntu para un gran número de procesos en problemas pequeños.

2. Tamaño del Problema

Se analiza si el beneficio del paralelismo mejora con matrices más grandes.

Ubuntu:

- **Tiempo Secuencial vs. Paralelo:** El gráfico de barras muestra que tanto el tiempo secuencial como el paralelo aumentan significativamente con el tamaño de la matriz. El tiempo paralelo es consistentemente menor que el tiempo secuencial para todos los tamaños de matriz, lo que indica el beneficio del paralelismo. La reducción de tiempo es más pronunciada para el tamaño de matriz más grande (1600×1600).
- **Log-Log Plot:** El gráfico log-log del tiempo vs. tamaño de la matriz (N) muestra una relación que se acerca a O(N3) para el tiempo secuencial, lo cual es consistente con la complejidad de la multiplicación de matrices estándar. El tiempo paralelo también sigue

una tendencia similar, pero a una escala de tiempo significativamente menor, lo que confirma que el paralelismo es más beneficioso para problemas de mayor tamaño.

RaspberryPiOS :

- **Tiempo Secuencial vs. Paralelo:** Como era de esperar, tanto el tiempo secuencial como el paralelo aumentan significativamente con el tamaño de la matriz. El tiempo paralelo es consistentemente menor que el tiempo secuencial, lo que indica el beneficio del paralelismo. La reducción de tiempo es más pronunciada para los tamaños de matriz más grandes (1600×1600).
- **Log-Log Plot:** El gráfico log-log del tiempo vs. tamaño de la matriz (N) muestra una relación que se acerca a O(N3) para el tiempo secuencial. El tiempo paralelo también sigue una tendencia similar, pero a una escala de tiempo significativamente menor, lo que confirma que el paralelismo es más beneficioso para problemas de mayor tamaño.

Conclusión: Ambos sistemas confirman que el paralelismo es más ventajoso para problemas de mayor tamaño, ya que la proporción del tiempo paralelo respecto al secuencial es menor para matrices más grandes. Esto se debe a que la sobrecarga del paralelismo se amortiza mejor en cargas de trabajo computacionales más grandes.

3. Sobrecarga de IPC (Comunicación entre Procesos)

Este análisis examina el consumo de CPU en modo sistema debido a la comunicación entre procesos.

Ubuntu:

- **CPU System Usage Ratio:** Para Ubuntu, el ratio de uso de CPU en modo sistema (cpuSysMs) aumenta drásticamente con el número de procesos. Para 100 y 500 procesos, el porcentaje de CPU del sistema es muy alto, superando el 80% e incluso llegando al 100% para la matriz de 400×400 con 500 procesos. Esto indica que una gran parte del tiempo de CPU se dedica a la gestión de procesos y la comunicación, lo que sugiere una alta sobrecarga de IPC.
- **CPU System vs CPU User Time:** El gráfico muestra claramente que el tiempo de CPU en modo sistema (cpuSysMs) domina el tiempo de CPU en modo usuario (cpuUserMs) a medida que aumenta el número de procesos, especialmente a 500 procesos. El tiempo de CPU del sistema es significativamente mayor que el tiempo de CPU de usuario, lo que implica que el sistema está gastando una cantidad considerable de recursos en la gestión de los procesos paralelos.

RaspberryPiOS:

- **CPU System Usage Ratio:** En RaspberryPiOS, el patrón es similar al de Ubuntu, con un aumento en el ratio de uso de CPU en modo sistema a medida que aumenta el número de procesos. Sin embargo, los valores son generalmente un poco más bajos que en Ubuntu para los tamaños de matriz más grandes.
- **CPU System vs CPU User Time:** Al igual que en Ubuntu, el tiempo de CPU en modo sistema supera al tiempo de CPU en modo usuario a medida que se incrementan los procesos. Sin embargo, la diferencia parece ser menos extrema en RaspberryPiOS en

comparación con Ubuntu para un número muy alto de procesos (500), donde el tiempo de CPU de usuario tiene una contribución más notable.

Comparación: Ambos sistemas operativos sufren de una sobrecarga de IPC significativa a medida que el número de procesos aumenta, evidenciada por el incremento en el uso de CPU en modo sistema. Ubuntu parece ser más susceptible a esta sobrecarga, especialmente para problemas pequeños con muchos procesos, donde el tiempo de CPU en modo sistema consume casi todos los recursos, lo que explica la caída del speedup por debajo de 1.0 en el caso de la matriz de 400×400 con 500 procesos. RaspberryPiOS también muestra un aumento en la sobrecarga, pero el tiempo de CPU de usuario sigue siendo una parte más sustancial del tiempo total en comparación con Ubuntu en escenarios de alta concurrencia.

4. Uso de Memoria

Este análisis se centra en cómo la RAM pico utilizada crece con el número de procesos o el tamaño de las matrices.

Ubuntu :

- **Peak RAM Usage:** El uso de RAM pico (maxRssKB) es principalmente afectado por el tamaño de la matriz y no tanto por el número de procesos. Para cada tamaño de matriz, el uso de RAM es relativamente constante a través de los diferentes números de procesos. Las matrices más grandes (ej. 1600×1600) consumen significativamente más RAM que las pequeñas (400×400). Esto es de esperar, ya que el tamaño de la matriz afecta directamente la cantidad de datos que deben almacenarse en memoria.

RaspberryPiOS :

- **Peak RAM Usage:** El comportamiento de uso de RAM en RaspberryPiOS es muy similar al de Ubuntu. El uso de RAM pico está directamente relacionado con el tamaño de la matriz, aumentando significativamente con matrices más grandes. Para un tamaño de matriz dado, el uso de RAM se mantiene relativamente estable independientemente del número de procesos.

Comparación: Ambos sistemas muestran que el uso de RAM está dominado por el tamaño de la matriz. El número de procesos tiene un impacto mínimo en el uso de RAM pico, lo que sugiere que la aplicación no duplica significativamente las estructuras de datos grandes por cada proceso, sino que quizás comparte memoria o distribuye los datos de manera eficiente.

5.4. Problemas Encontrados y Soluciones

Durante el desarrollo del proyecto se presentaron algunos inconvenientes técnicos que requirieron ajustes en la planificación y configuración del sistema. A continuación, se describen los problemas más relevantes y cómo fueron solucionados:

Problema 1: Sincronización de hora en las Raspberry Pi

Al intentar utilizar el gestor de paquetes `apt` en las Raspberry Pi, se detectaron errores relacionados con la falta de sincronización horaria del sistema, lo cual impedía la descarga correcta de algunos paquetes. Esto se debía a que la fecha y hora del sistema no coincidían con los certificados de los repositorios seguros.

Solución:
Se configuró manualmente el servicio Network Time Protocol (NTP) editando el archivo de configuración correspondiente. Se activaron las líneas comentadas que habilitan el acceso a los servidores de tiempo adecuados para la región geográfica. Posteriormente, se reinició el servicio de sincronización con el comando adecuado (`sudo systemctl restart systemd-timesyncd.service`) y se verificó que la hora estuviera correctamente actualizada con `timedatectl`.

Problema 2: Fallo en la conexión de sensores

Aunque originalmente se consideró la integración de sensores (temperatura, sonido, luz) para realizar lecturas periódicas, se identificó que la mayoría de los sensores disponibles estaban diseñados para plataformas como Arduino, y las librerías asociadas no estaban plenamente adaptadas para trabajar en C + + nativamente. Nos planteamos acondicionar las librerías para que funcionaran en este entorno pero era un trabajo demasiado extenso.

Solución:
Dado que el componente de lectura de sensores tenía como objetivo principal generar carga computacional realista y enviar datos de rendimiento, se reemplazó por una alternativa más estable y controlable: un proceso de multiplicación de matrices, diseñado para ejecutarse en versiones secuencial y paralela. Esta nueva actividad representa de forma efectiva el uso intensivo de CPU en modo usuario, además de activar el uso de recursos en modo sistema (cuando se utiliza IPC con pipes).
Además, se mantuvo el flujo original de transmitir resultados hacia Firebase, asegurando que la arquitectura general del sistema —lectura/procesamiento + envío de datos— se conservara, alineada con los objetivos iniciales del proyecto.

Este cambio no solo resolvió el inconveniente técnico, sino que permitió reforzar el enfoque analítico del proyecto, centrado en el rendimiento del sistema operativo bajo distintas condiciones de ejecución.

6. Conclusiones

El proyecto ha logrado exitosamente sus objetivos iniciales, que se centraban en analizar y comparar el rendimiento de un algoritmo de multiplicación de matrices en ejecución secuencial y paralela, evaluando métricas clave como el uso de CPU, memoria y tiempo de ejecución en distintos entornos operativos (Ubuntu y Raspberry Pi OS). Se demostró cómo la paralelización puede mejorar el desempeño computacional, especialmente para cargas de trabajo más grandes, y se identificaron los impactos del paralelismo en sistemas con recursos limitados.

Durante la realización de este proyecto, se obtuvieron aprendizajes significativos en relación con conceptos fundamentales de los sistemas operativos:

- **Impacto de la Complejidad Algorítmica y el Paralelismo:** Se validó que la multiplicación de matrices presenta una complejidad computacional de O(N3), observando cómo el tiempo de ejecución (tanto secuencial como paralelo) escala con el tamaño de la matriz. La implementación paralela demostró una reducción significativa en el tiempo de ejecución en comparación con la versión secuencial, destacando los beneficios inherentes del paralelismo para problemas computacionalmente intensivos.
- **Gestión de Procesos y Comunicación Inter-Procesos (IPC):** El uso de `fork()` para la creación de procesos y `pipes` para la comunicación entre ellos fue una parte central del diseño. El análisis reveló que, aunque el paralelismo mejora el rendimiento inicialmente, un número excesivo de procesos (ej., 500 procesos) introduce una sobrecarga considerable en la comunicación y gestión de recursos. Esto se manifestó en un incremento drástico del uso de CPU en modo sistema (`cpuSysMs`), lo que indica que una parte significativa del tiempo de CPU se dedicó a la coordinación y sobrecarga del sistema operativo, en lugar de a la computación útil.
- **Influencia del Sistema Operativo en el Rendimiento:** Se observaron diferencias clave en el comportamiento entre Ubuntu y Raspberry Pi OS. Ubuntu mostró una mayor sensibilidad a la sobrecarga de IPC en escenarios de alta concurrencia y matrices pequeñas (ej., 400×400 con 500 procesos), donde el speedup incluso cayó por debajo de 1.0, indicando que la versión paralela era más lenta que la secuencial. Por el contrario, Raspberry Pi OS gestionó esta sobrecarga de manera relativamente más eficiente en el mismo escenario, manteniendo un speedup positivo. Esto subraya cómo el planificador de procesos y los mecanismos de gestión de recursos de cada sistema operativo pueden afectar críticamente el rendimiento de aplicaciones paralelas.
- **Amortización de la Sobrecarga:** Los resultados confirmaron que la sobrecarga del paralelismo se amortiza mejor en problemas de mayor tamaño. A medida que el tamaño de la matriz aumenta, los beneficios del speedup se mantienen por un mayor número de procesos, ya que el tiempo de cómputo domina sobre el tiempo de comunicación.
- **Uso de Recursos Físicos:** El análisis del uso de memoria (maxRssKB) demostró que el consumo de RAM está predominantemente determinado por el tamaño de la matriz y es relativamente independiente del número de procesos. Esto sugiere una gestión eficiente de la memoria, evitando duplicaciones excesivas de datos entre procesos.

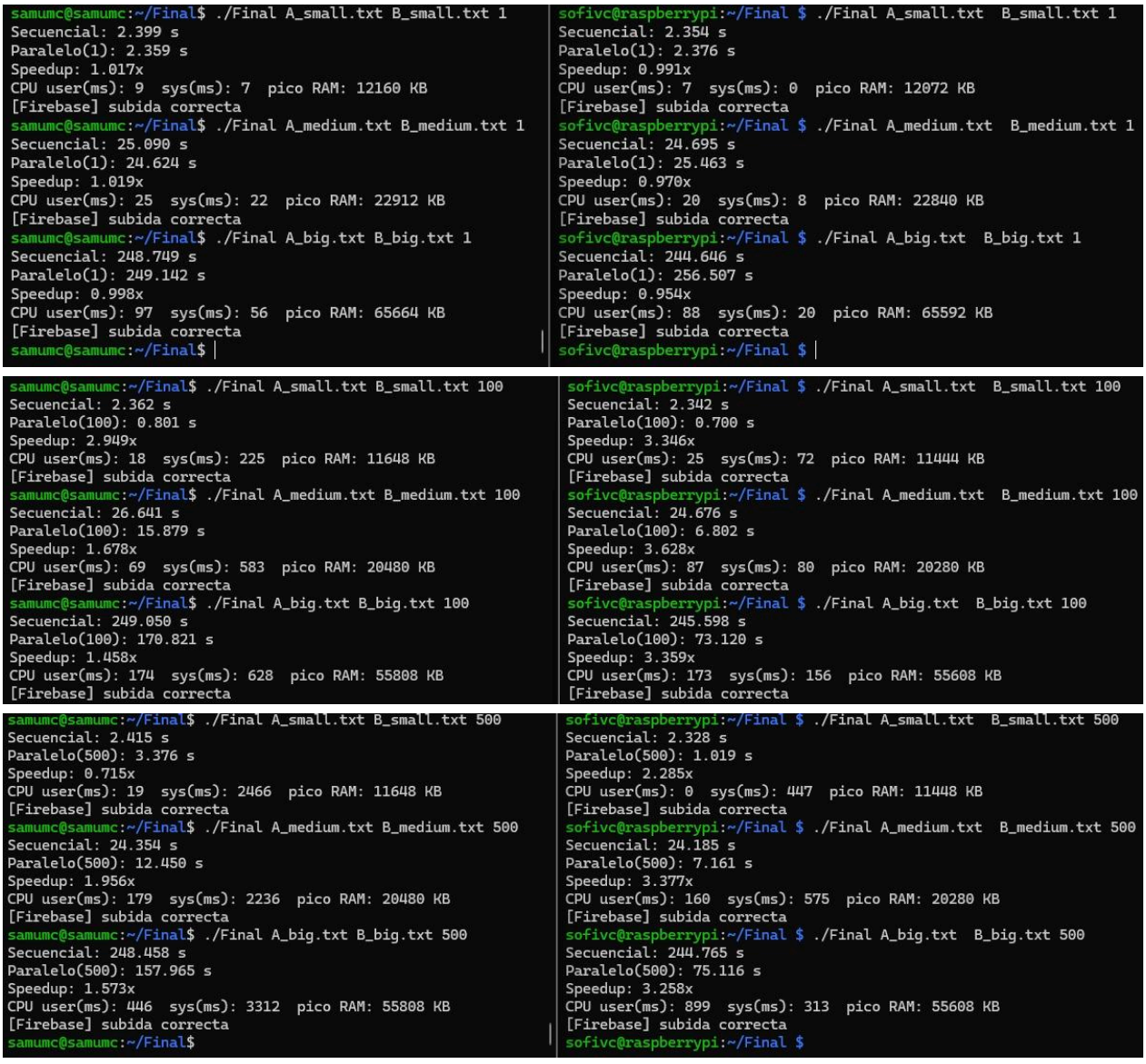
El éxito general del proyecto radica en su capacidad para no solo implementar y ejecutar un algoritmo paralelo de multiplicación de matrices, sino también en la recopilación sistemática y el análisis profundo de sus métricas de rendimiento. La integración con Firebase para la centralización de datos fue una estrategia efectiva para la evaluación a gran escala. A pesar de las limitaciones de hardware en el caso de la Raspberry Pi y las sobrecargas inherentes al paralelismo basado en procesos, el proyecto logró ofrecer una comprensión clara de los compromisos entre la paralelización, la eficiencia y el uso de recursos en distintos entornos de sistemas operativos.

7. Referencias

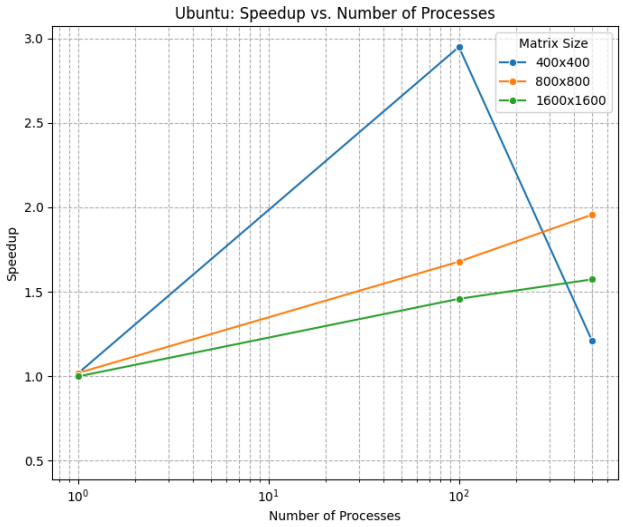
Laboratorio 3 - Sistemas Operativos

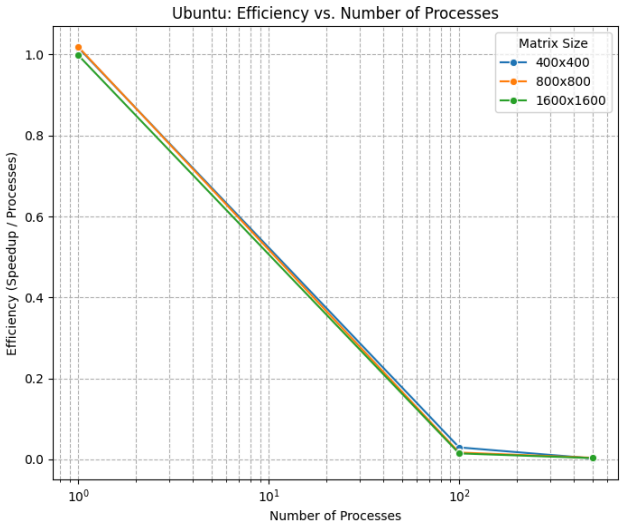
9. Anexos

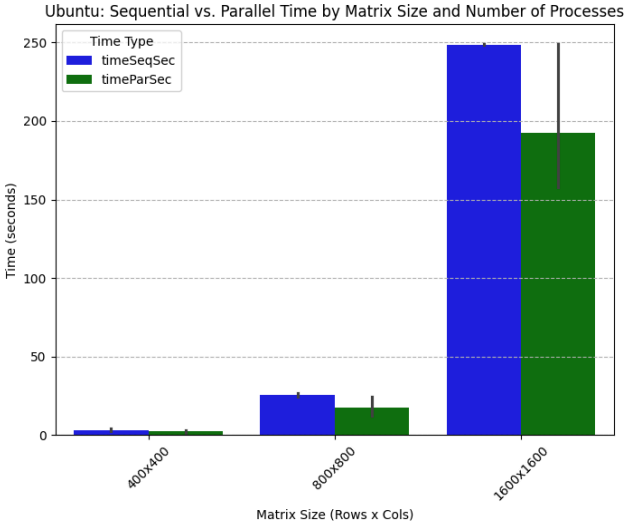
Ejecucion:

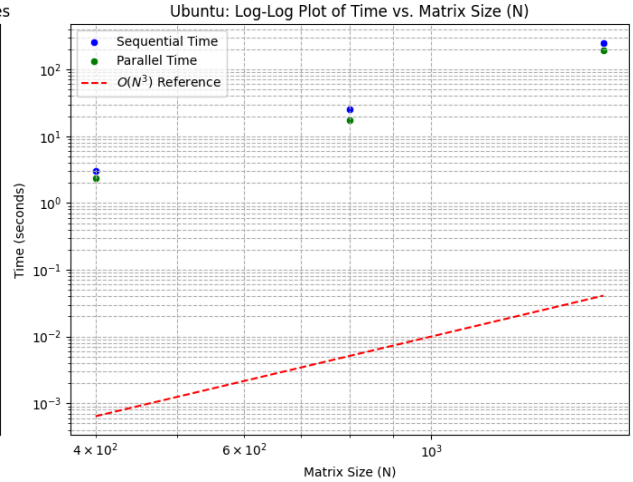


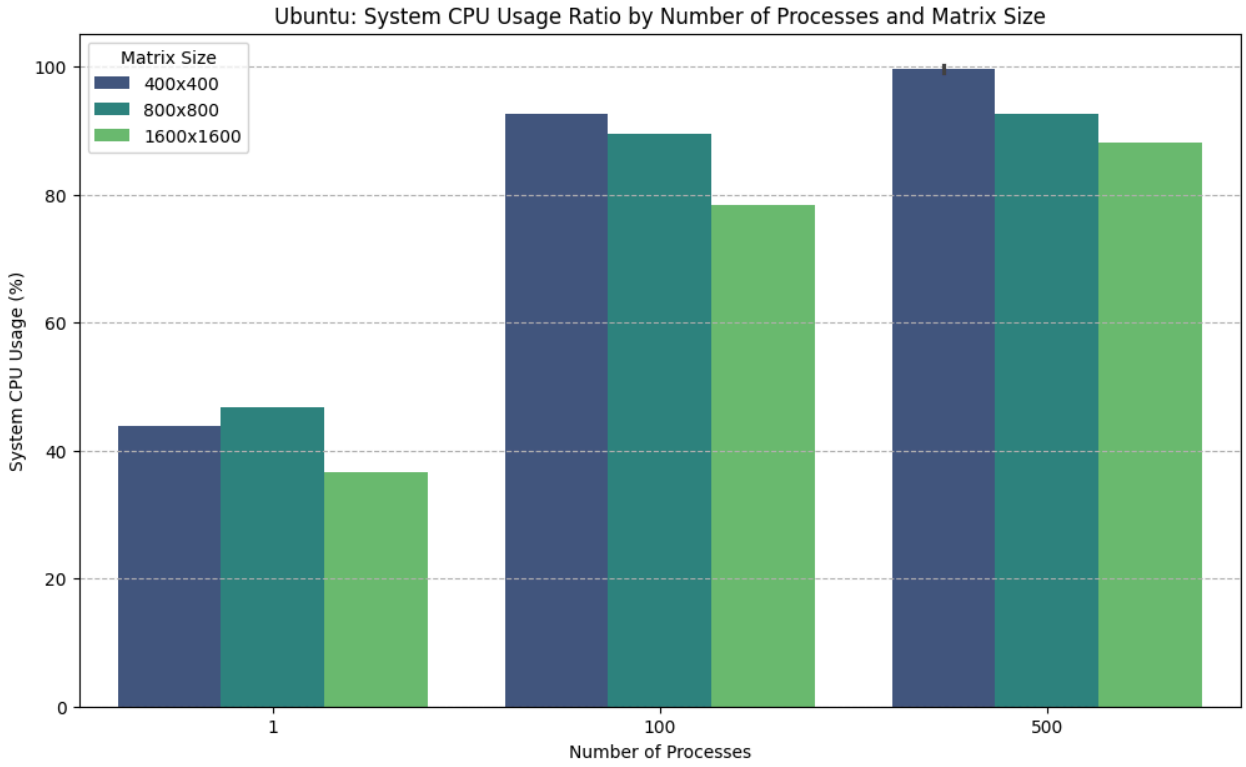
Graficas ubuntu:

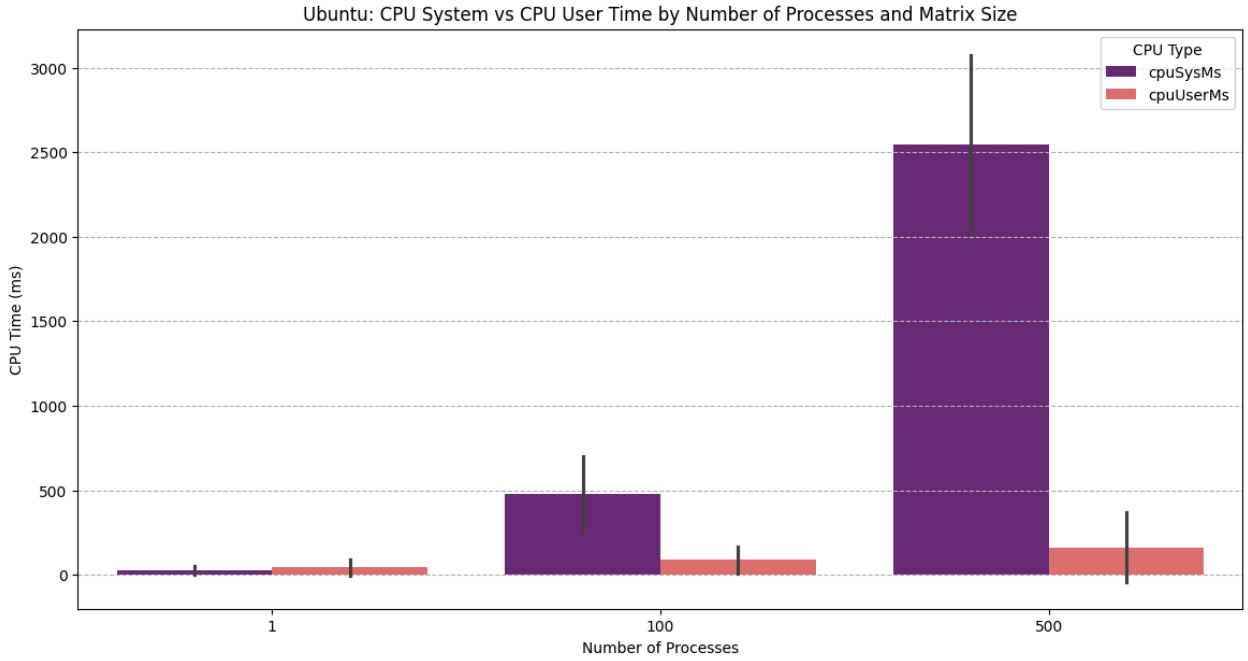


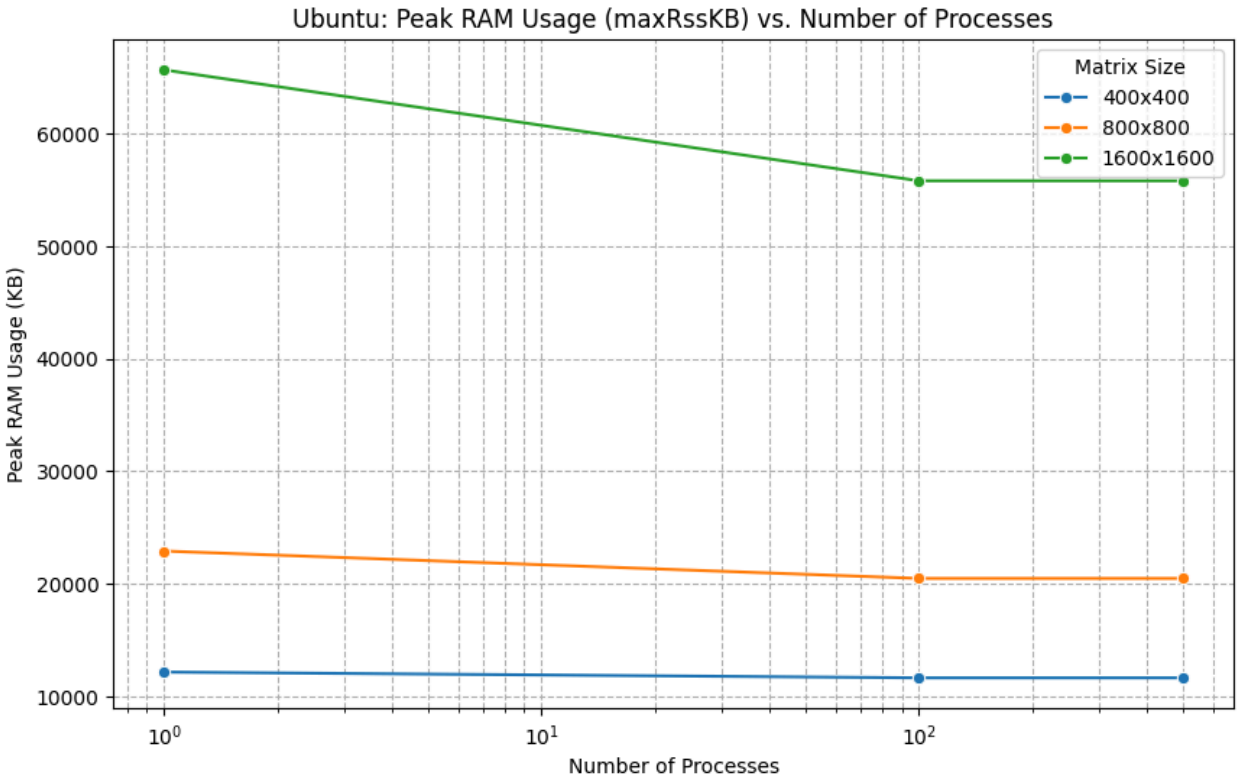




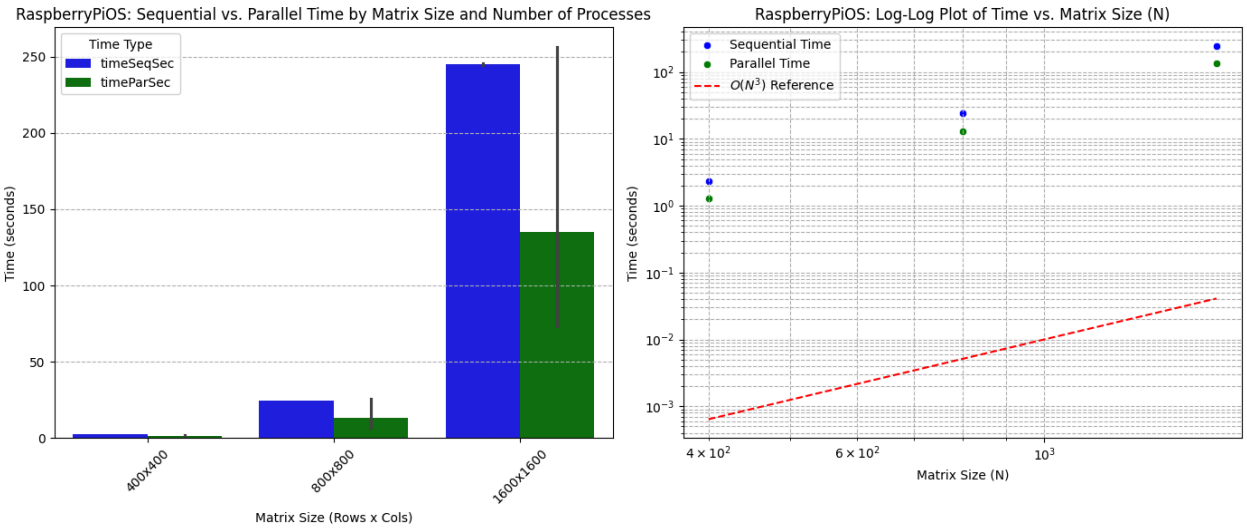


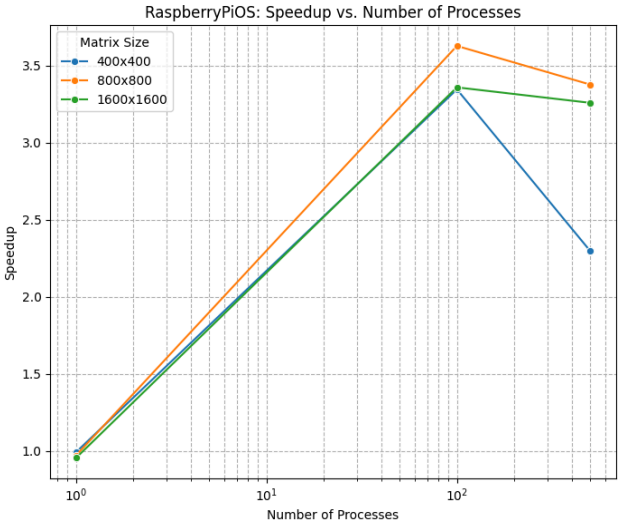

Ubuntu: System CPU Usage Ratio by Number of Processes and Matrix Size
Matrix Size
400x400
800x800
1600x1600
System CPU Usage (%)
0
20
40
60
80
100
1
100
500
Number of Processes

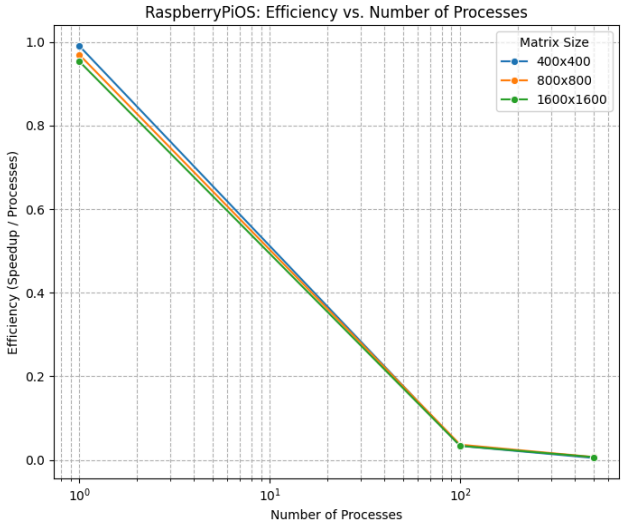

Ubuntu: CPU System vs CPU User Time by Number of Processes and Matrix Size
CPU Type
cpuSysMs
cpuUserMs
CPU Time (ms)
0
500
1000
1500
2000
2500
3000
1
100
500
Number of Processes

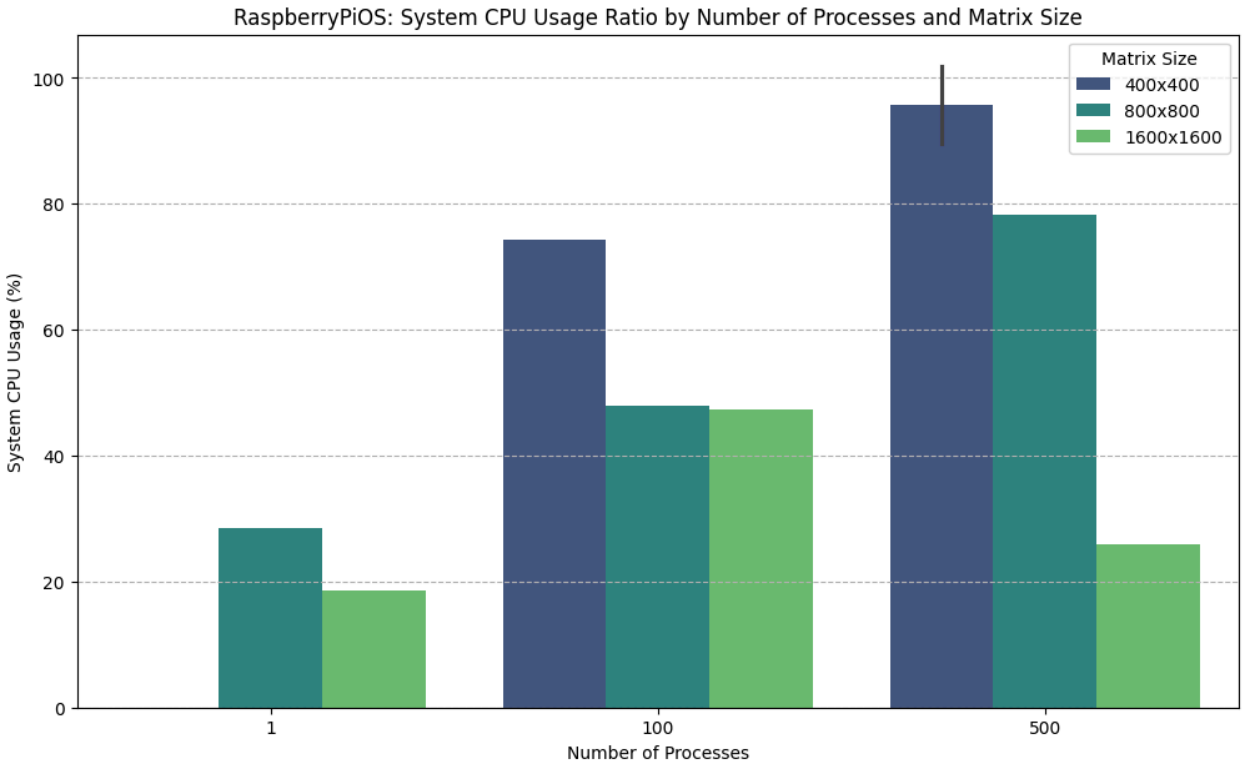


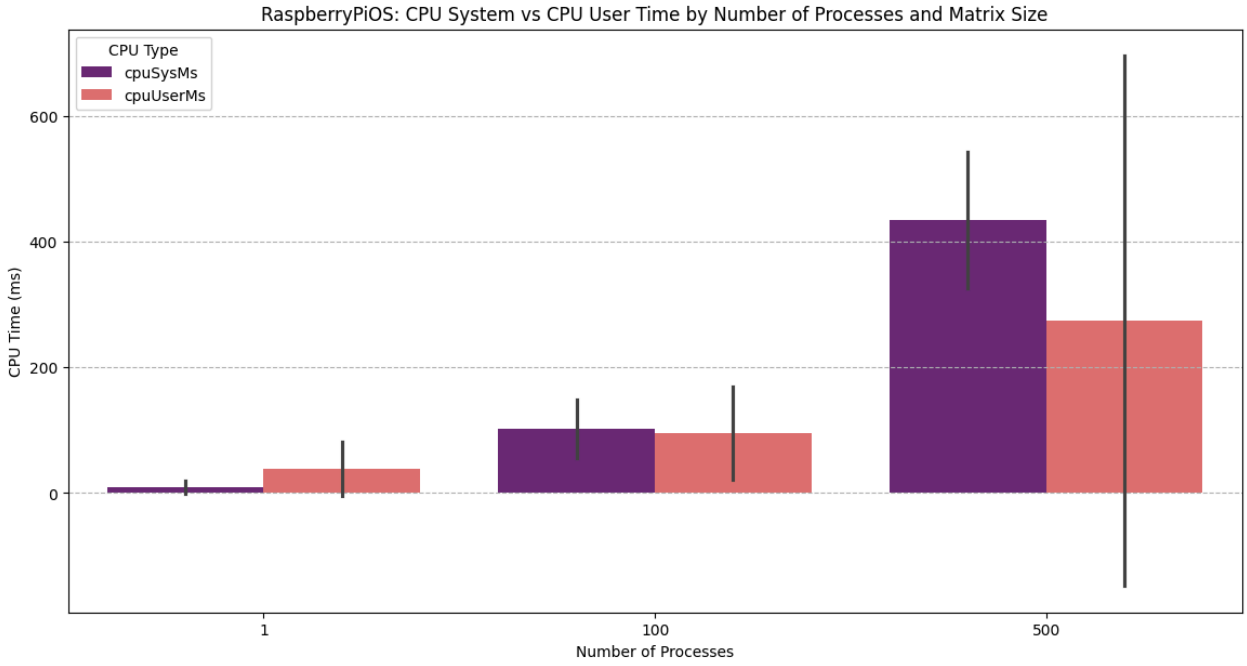
Graficas raspberryPiOS:

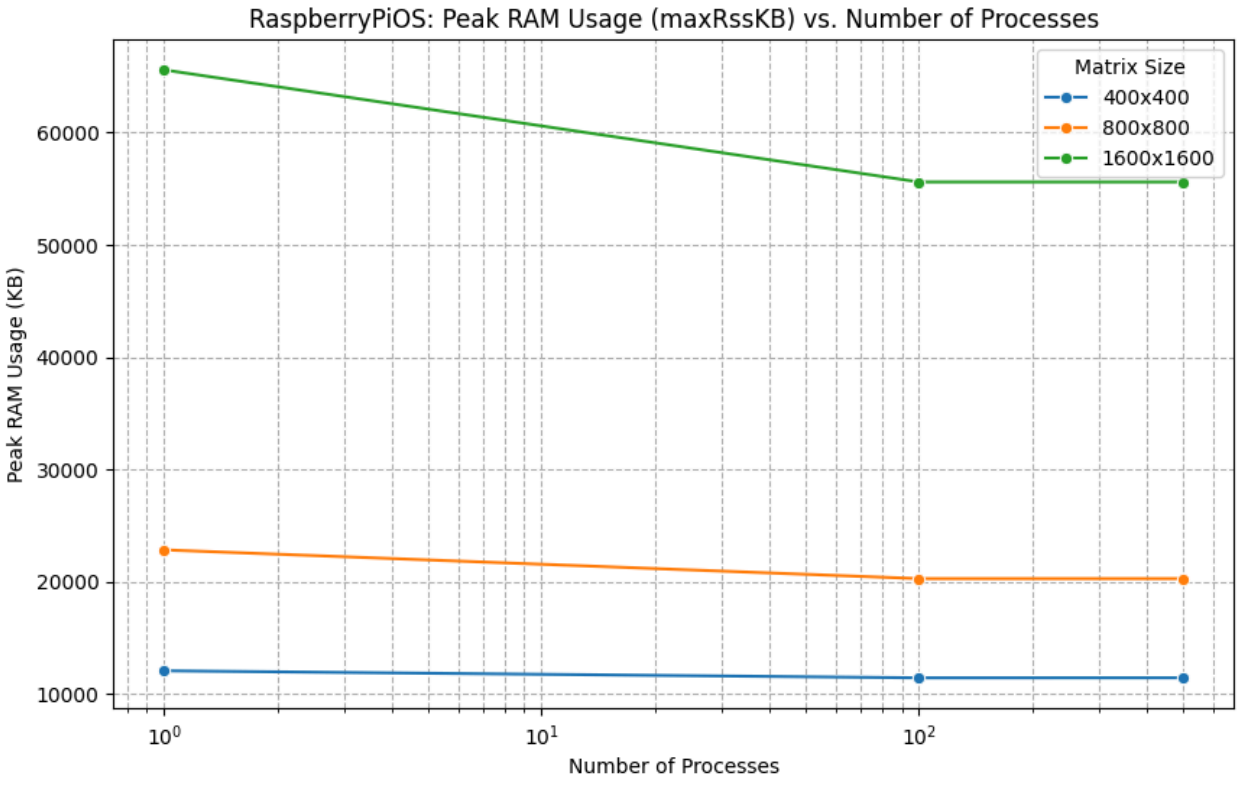



RaspberryPiOS: Speedup vs. Number of Processes
Matrix Size
400x400
800x800
1600x1600
Speedup
Number of Processes


RaspberryPiOS: Efficiency vs. Number of Processes
Matrix Size
400x400
800x800
1600x1600
Efficiency (Speedup / Processes)
Number of Processes


RaspberryPiOS: System CPU Usage Ratio by Number of Processes and Matrix Size
Matrix Size
400x400
800x800
1600x1600
System CPU Usage (%)
Number of Processes


RaspberryPiOS: CPU System vs CPU User Time by Number of Processes and Matrix Size
CPU Type
cpuSysMs
cpuUserMs
CPU Time (ms)
600
400
200
0
1
100
500
Number of Processes


RaspberryPiOS: Peak RAM Usage (maxRssKB) vs. Number of Processes
Matrix Size
400x400
800x800
1600x1600
Peak RAM Usage (KB)
60000
50000
40000
30000
20000
10000
10^0
10^1
10^2
Number of Processes