

Inversión de matrices de gran tamaño

Sistemas Operativos
2025



UNIVERSIDAD
DE ANTIOQUIA



Integrantes

**Cristian Daniel
Muñoz Botero**

**Jonathan Andrés
Granda Orrego**

**Brandon Duque
Garcia**

**Jhon Alexander
Botero Gómez**

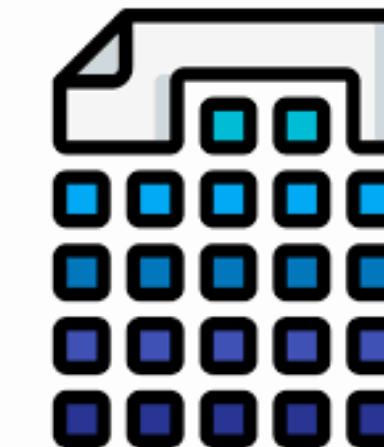


Introducción

“La inversión de matrices es fundamental en IA y simulaciones”

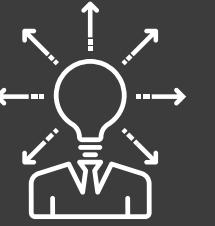
Algunos problemas o uso de estas

- En muchas aplicaciones científicas e industriales se requiere invertir matrices de gran tamaño
- Sin embargo, esta operación es muy costosa computacionalmente en especial a gran escala.
- Esto genera la necesidad de buscar algoritmos más eficientes y analizar su desempeño en diferentes arquitecturas (CPU vs GPU)



Objetivo

Se va a analizar y comparar diferentes aspectos al hacer el proyecto con el cual se pretende sacar diferentes conclusiones



Comparación entre diferentes métodos

C vs Python



Análisis del rendimiento del lenguaje

C vs Python

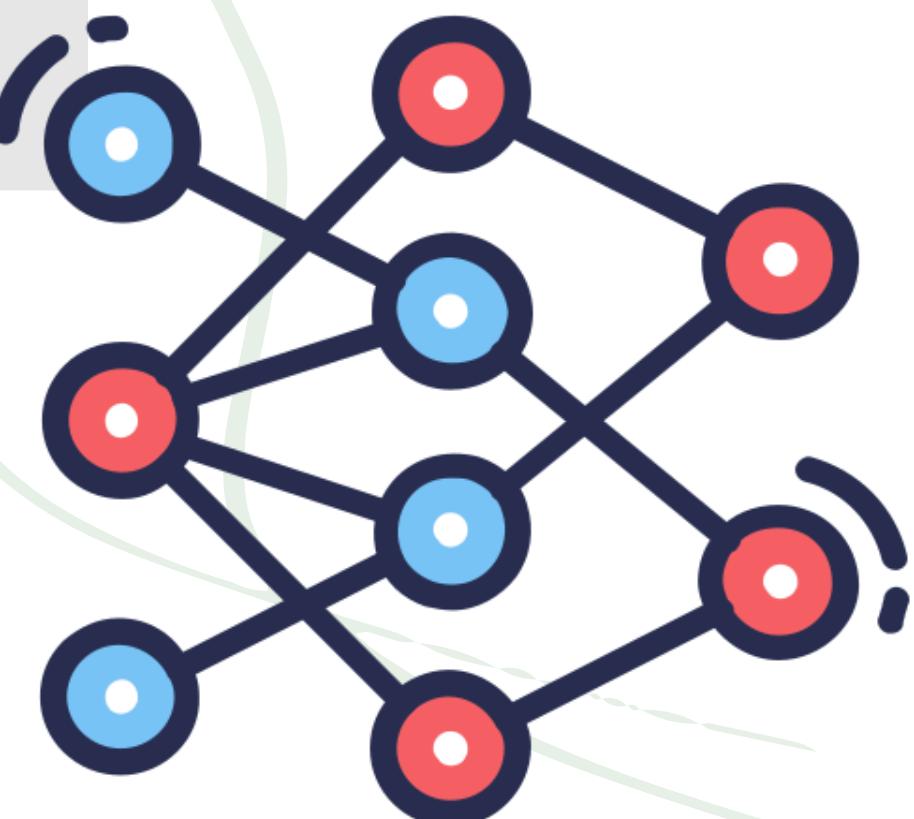


Evaluar desempeño

CPU VS GPU

Motivación y Justificación

- Relaciona conceptos clave de sistemas operativos con aplicaciones reales.
- Requiere gestión eficiente de memoria y recursos del sistema
- Permite aplicar paralelismo y gestión de hilos, optimizando cálculos intensivos
- Conecta un problema matemático complejo con la computación a bajo nivel
- Es relevante en la era del procesamiento masivo de datos y computación paralela



Alcance

- Implementación y análisis de algoritmos de inversión de matrices.
- Ejecución en versiones secuenciales y paralelas (con hilos).
- Comparación de resultados con y sin paralelismo en CPU.
- Uso de CUDA en Python para explorar el potencial GPU.

01

Descomposición de Cholesky

02

Reducción Gauss-Jordan

03

Descomposición LU



NVIDIA®

Marco Teórico

Inversión de matrices

Obtener la matriz inversa que, al multiplicarse por la original, da la identidad; esencial en álgebra lineal, IA y simulaciones.

Lenguaje C

Manejo directo de memoria y hilos; ideal para algoritmos numéricos de alto rendimiento con paralelismo (OpenMP, MPI).

Lenguaje Python

Facilita el prototipado rápido y permite comparar rendimiento frente a C usando librerías de alto nivel.

CUDA

Plataforma de NVIDIA para computación paralela en GPU; miles de núcleos para cálculos intensivos.

CuPy

Biblioteca de Python similar a NumPy, pero ejecuta en GPU con CUDA, acelerando operaciones matriciales.

PyTorch

Framework flexible para operaciones numéricas y machine learning, con aceleración en GPU.

Paralelismo e hilos

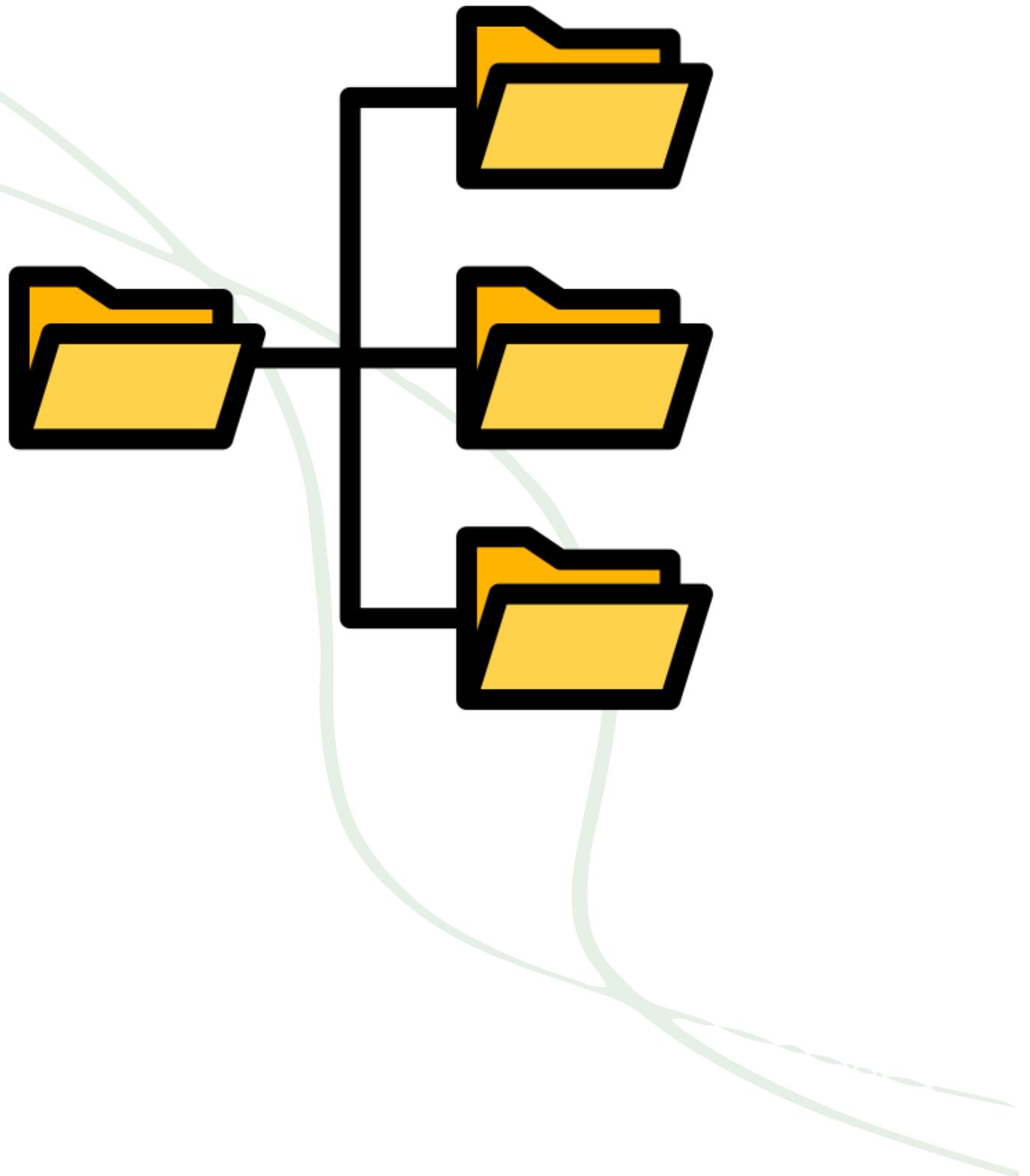
Distribuyen cálculos en tareas simultáneas, mejorando tiempos de ejecución.

Gestión de memoria

Clave con matrices grandes; CUDA y librerías que controlan memoria global y compartida eficientemente.

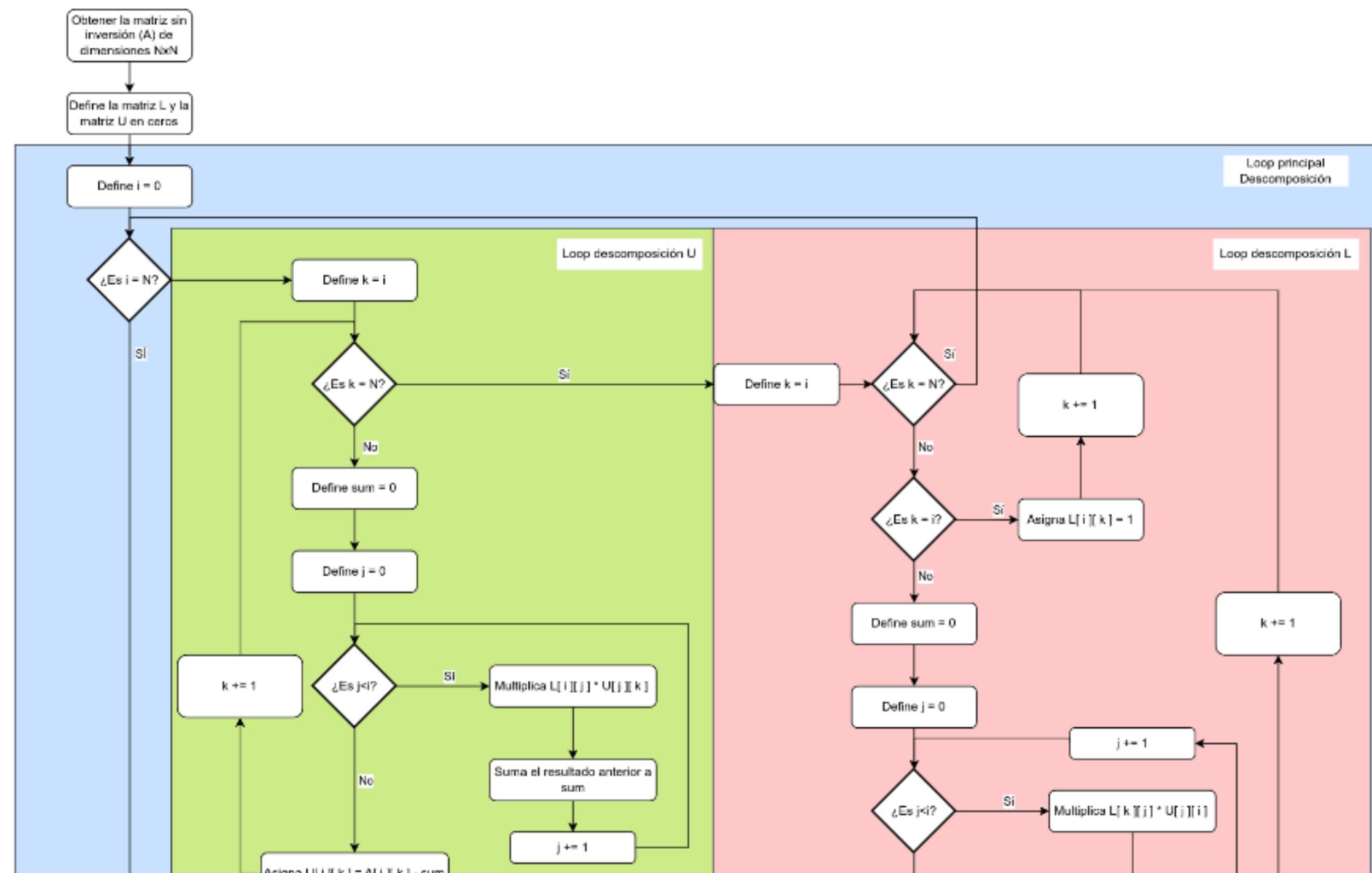
Diseño de la solución

```
README.md  
cholesky  
    cholesky.c  
    cholesky.py  
    cholesky_paralelo.c  
    cholesky_paralelo.py  
cuda_implementaciones  
    inversion_manual_impl.ipynb  
    matrix_inversión_cuda.ipynb  
gaussiana  
    gaussiana.c  
    gaussiana.py  
    parallel_gaussiana.c  
    parallel_gaussiana.py  
lu  
    lu_matrix_inversion.c  
    lu_matrix_inversion.py  
    parallel_lu_matrix_inversion.c  
    parallel_lu_matrix_inversion.py  
resultados  
    resultados_cholesky.ipynb  
    resultados_gaussiano.ipynb  
    resultados_lu.ipynb
```



Diseño de la solución

El diseño de cada algoritmo base (LU, gaussiana y Cholesky) se realizó en un diagrama de flujo donde se hizo el esqueleto de los métodos de inversión posteriormente implementado en código.



[Link de la carpeta de diagramas de flujo](#)

Implementación

- Lectura de matrices desde archivos .csv (invertibles y definidas positivas o invertibles, simétricas y positivas).
- **12 algoritmos codificados:**
 - LU, Gaussiana, Cholesky en C y Python, con versiones secuenciales y paralelas.
- **Paralelismo aplicado:**
 - LU y Cholesky paralelización por columnas (hilos en C con Pthreads/OpenMP, Python con threading).
- **Comparativa secuencial vs paralela:**
 - LU: Función paralela divide cálculo en columnas, procesadas por hilos.
 - Gaussiana: El paralelismo mejora tiempos solo en matrices muy grandes debido a la sobrecarga de Numba.
 - Cholesky: Usa OpenMP para parallelizar el ciclo for de inversión por columnas y reglas adicionales (matrices simétricas).
- Codificación basada en diagramas de flujo, con mayor fidelidad en C por su bajo nivel.
- Estructura modular facilita pruebas y comparación de resultados.
- Acercamiento a procesamiento con GPUs con la herramienta CUDA

Pruebas y Evaluación

- **Enfoque de pruebas**

- Pruebas funcionales (matemática verificando $A \cdot A^{-1}$).
- Pruebas de rendimiento (tiempos de ejecución y escalabilidad).

- **Matrices probadas**

- Generar matrices en archivos .csv para LU y Gaussiana
 - Generar matrices simétricas positivas en archivos .csv para Cholesky
- Tamaños de matrices evaluados 10x10, 100x100, 200x200, 300x300, 400x400, hasta 500x500.

- **Algoritmos probados:** LU, Gaussiana, Cholesky (en versiones C, Python, CuPy, PyTorch).

- **Medición de desempeño**

- Medición de tiempo de cada algoritmo en sus respectivos lenguajes
- Resultados graficados para comparar rendimiento según tamaño y tecnología

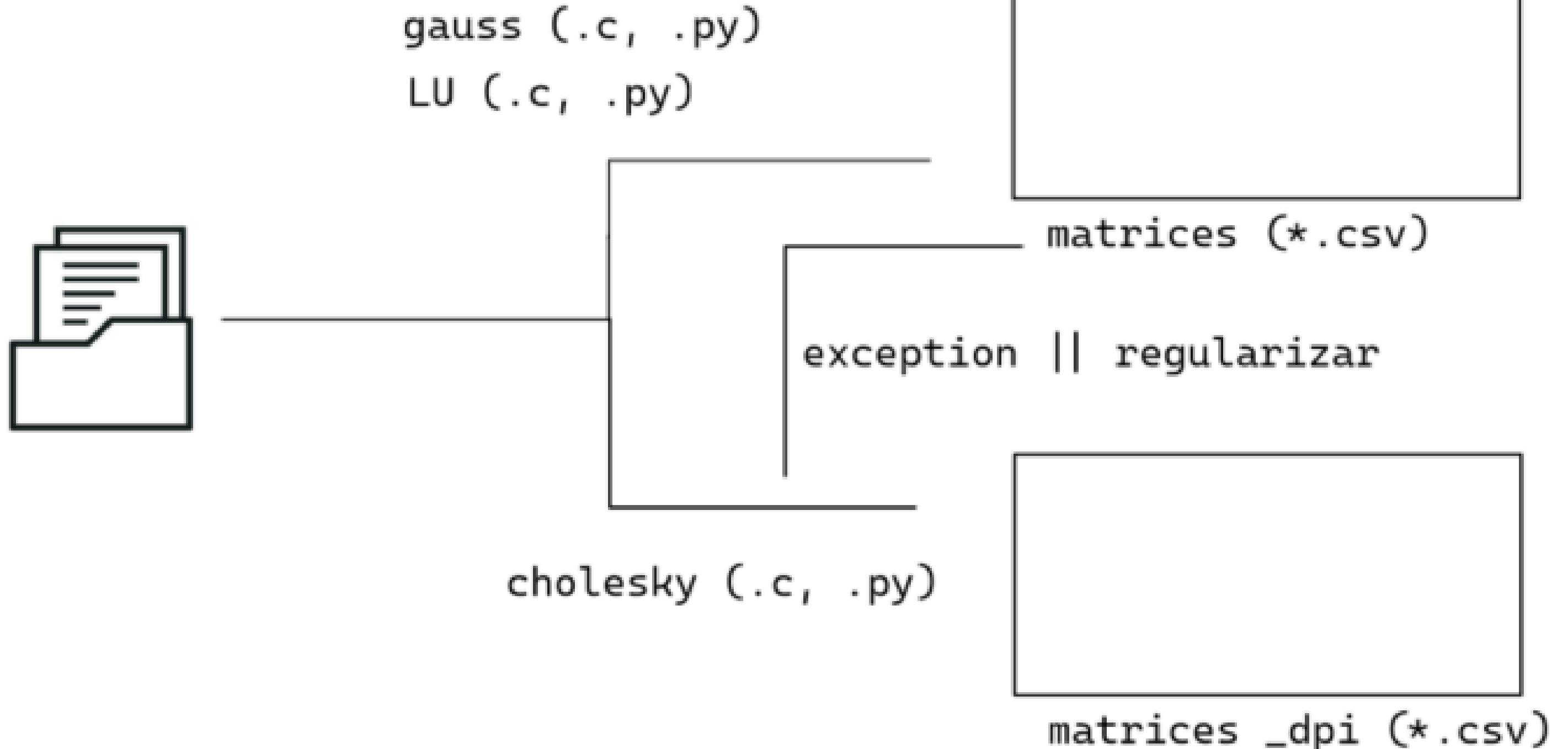
- **Observaciones clave**

- Paralelismo mejora tiempos al repartir cálculos en hilos.
- En matrices pequeñas, crear múltiples hilos puede aumentar el tiempo por sobrecargo.

- **Casos de prueba:**

- Ejecuciones secuenciales y paralelas de cada algoritmo en C y Python.

Pruebas y Evaluación





Resultados y Comparación

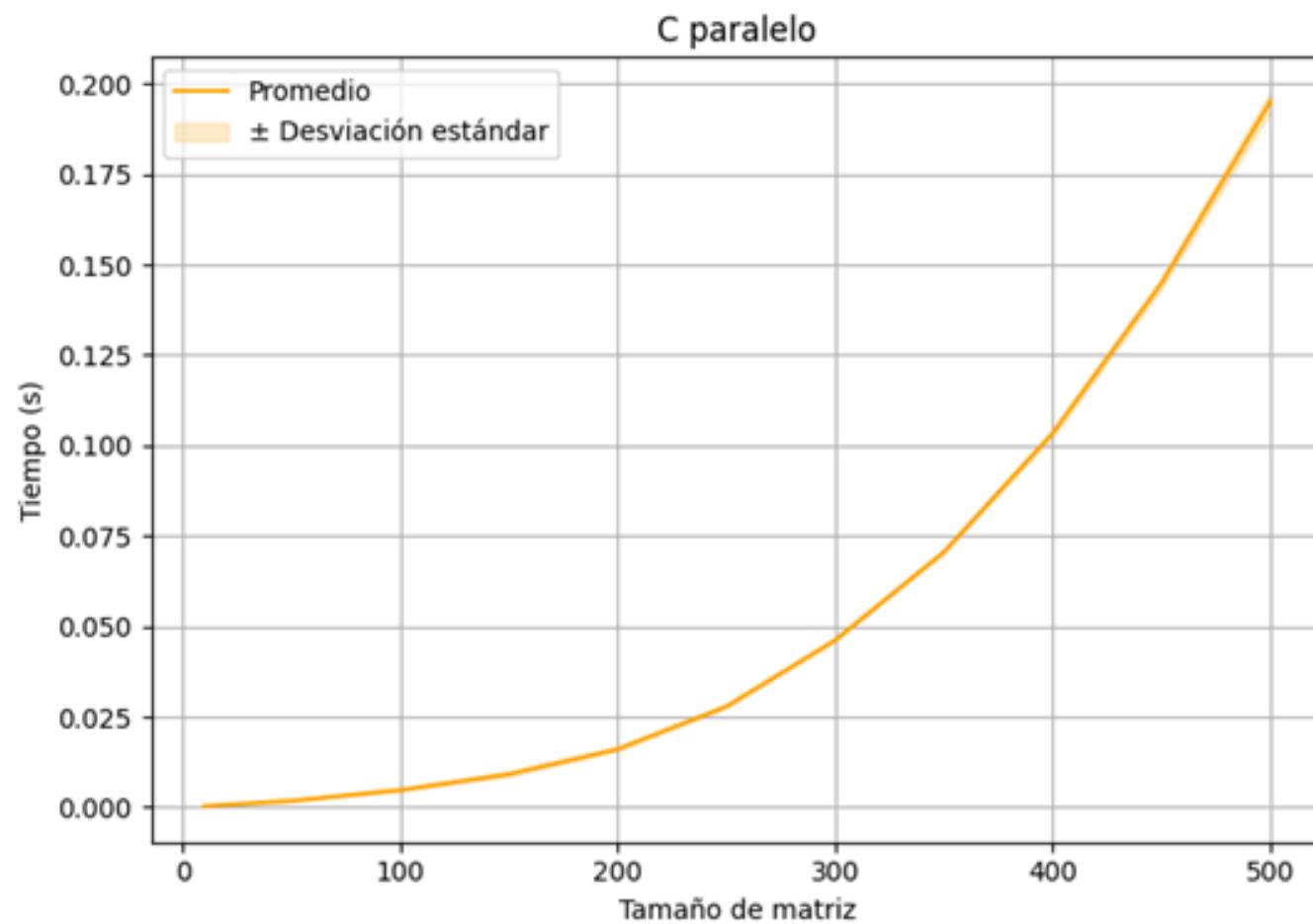
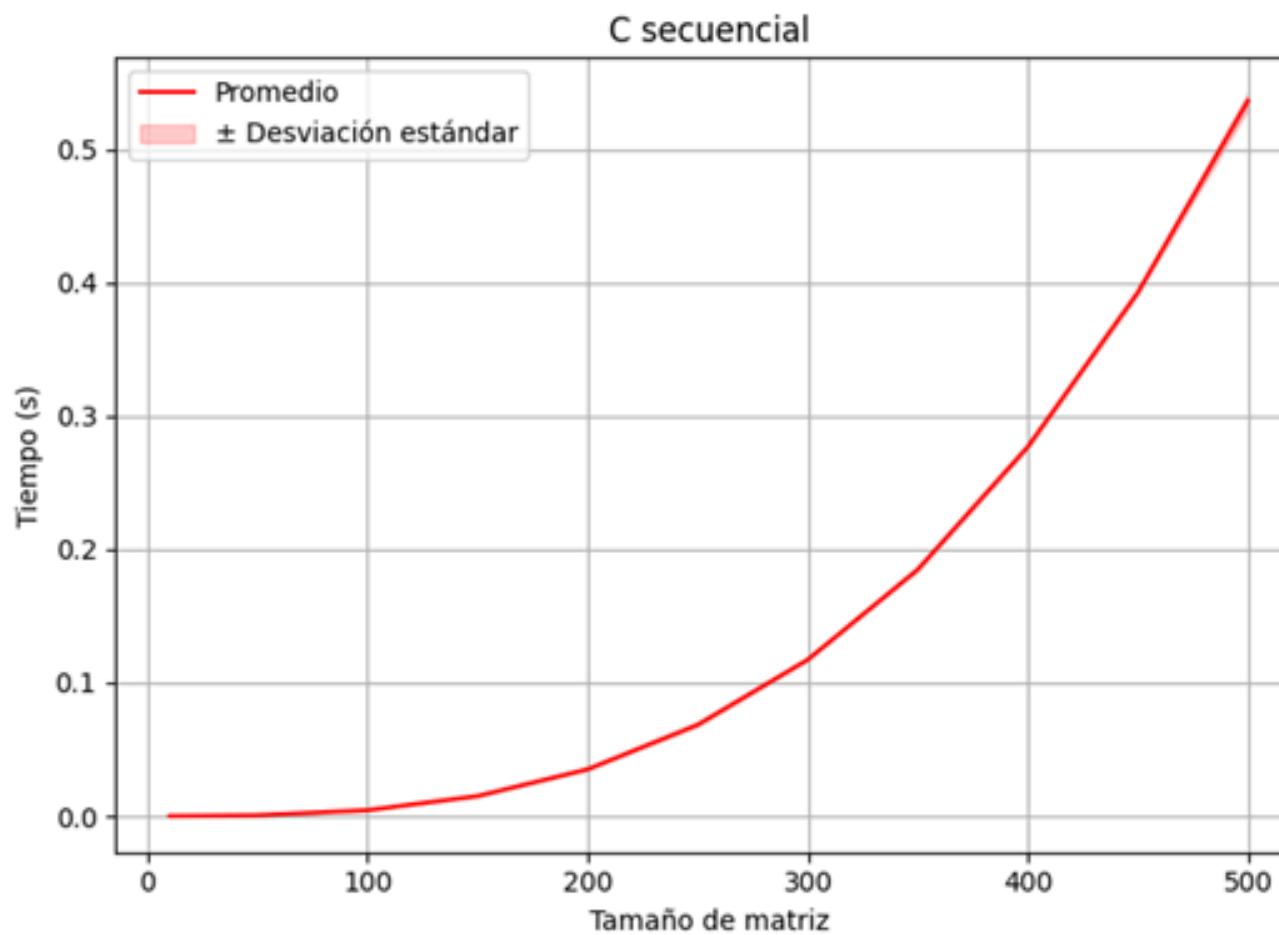
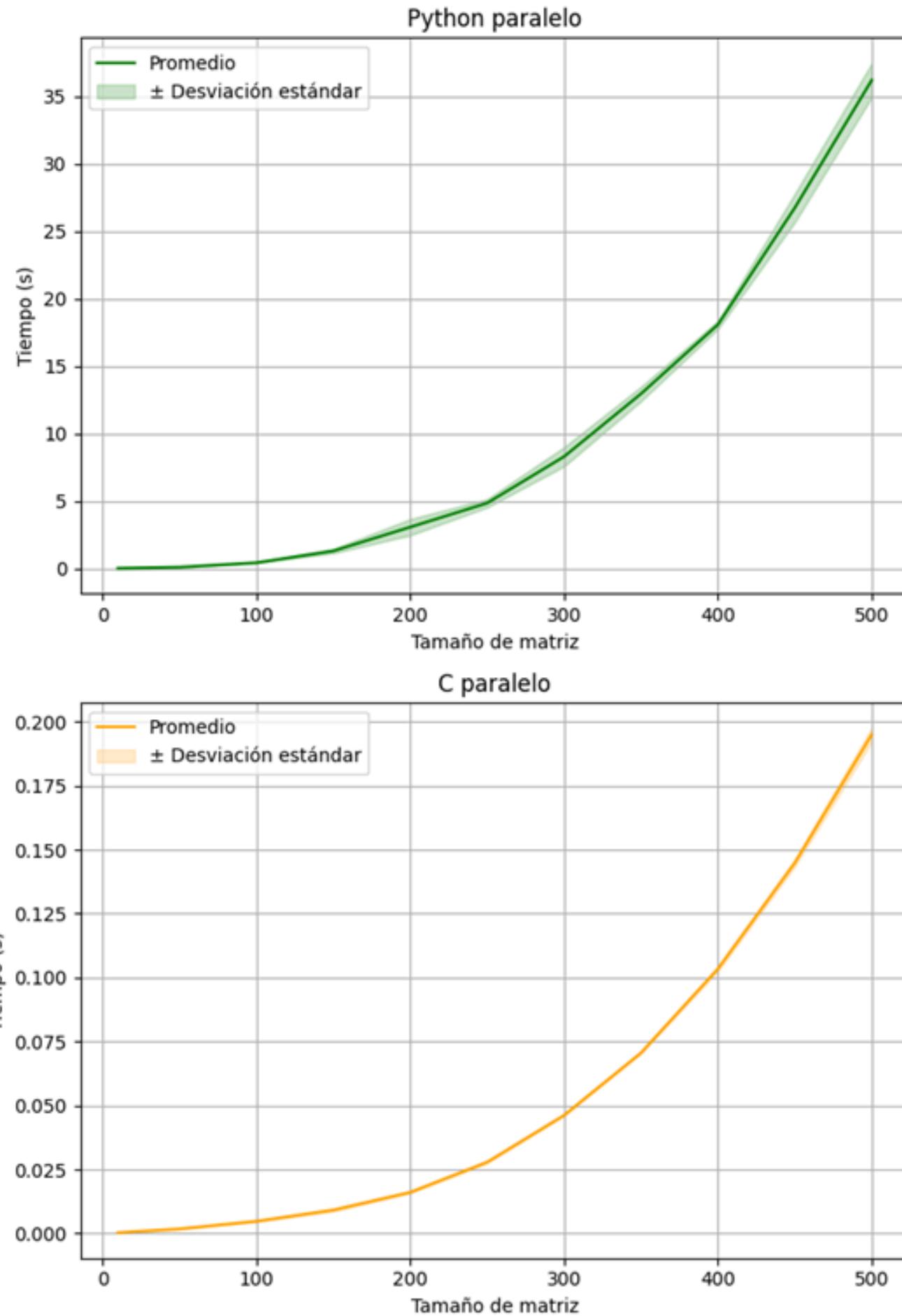
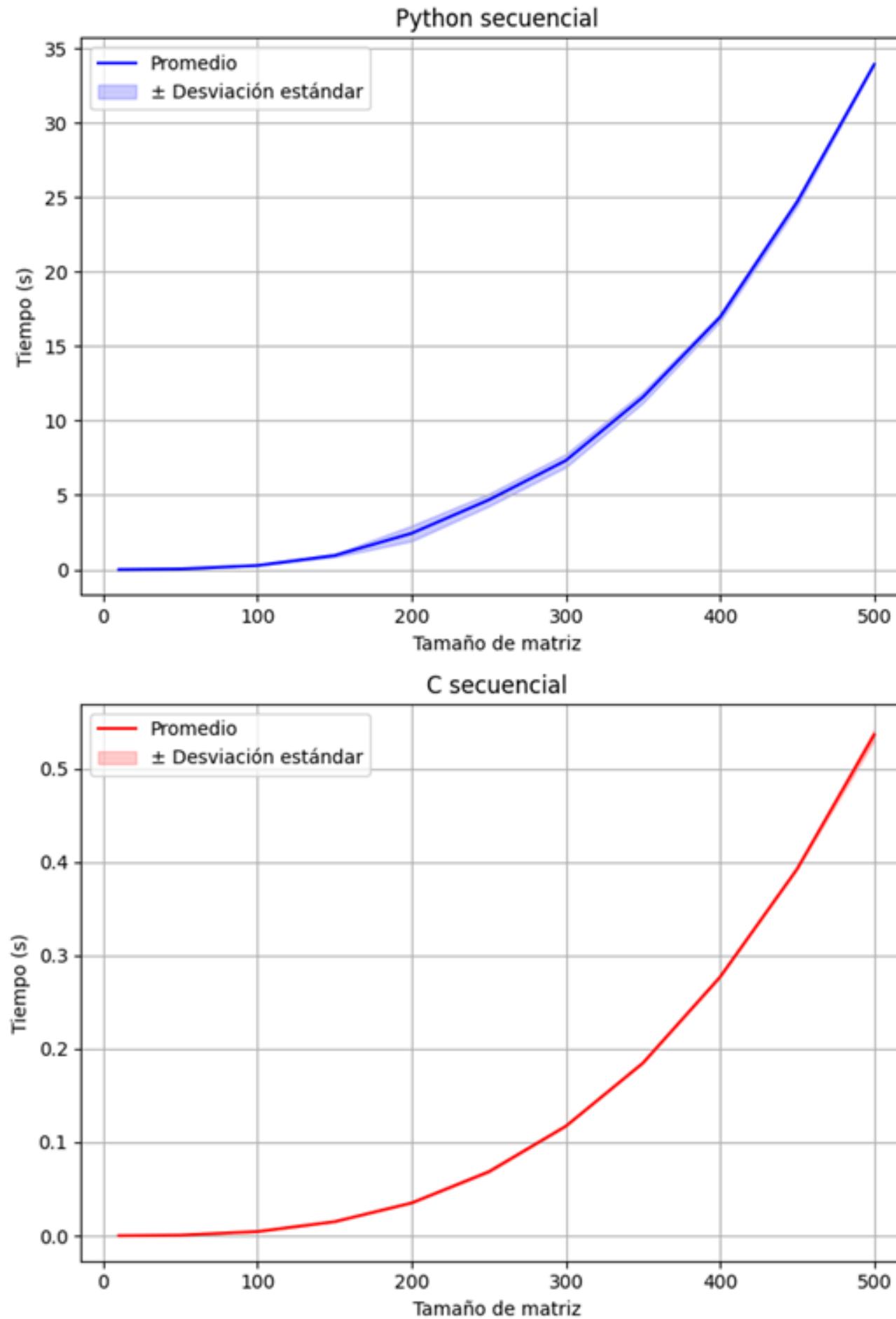
Resultados

Algoritmo	C Secuencial	C Paralelo	Python Secuencial	Python Paralelo
Eliminación Gaussiana	$0.098 \pm 0.023\text{s}$	$0.290\text{s} \pm 0.029\text{s}$	$0.833\text{s} \pm 0.027\text{s}$	$7.005\text{s} \pm 0.125\text{s}$
Descomposición LU	$0.536\text{s} \pm 0.006\text{s}$	$0.195\text{s} \pm 0.003\text{s}$	$33.91\text{s} \pm 0.08\text{s}$	$36.21\text{s} \pm 1.26\text{s}$
Descomposición Cholesky	$62.062\text{s} \pm 0.627$	$0.107\text{s} \pm 0.786\text{s}$	$62.062\text{s} \pm 0.627\text{s}$	$62.141\text{s} \pm 0.293\text{s}$

Algoritmo	Cupy	Pytorch
Eliminación Gaussiana	12.840	10.613
Descomposición LU	97.844	52.987
Descomposición Cholesky	50.892	16.975

Descomposición LU

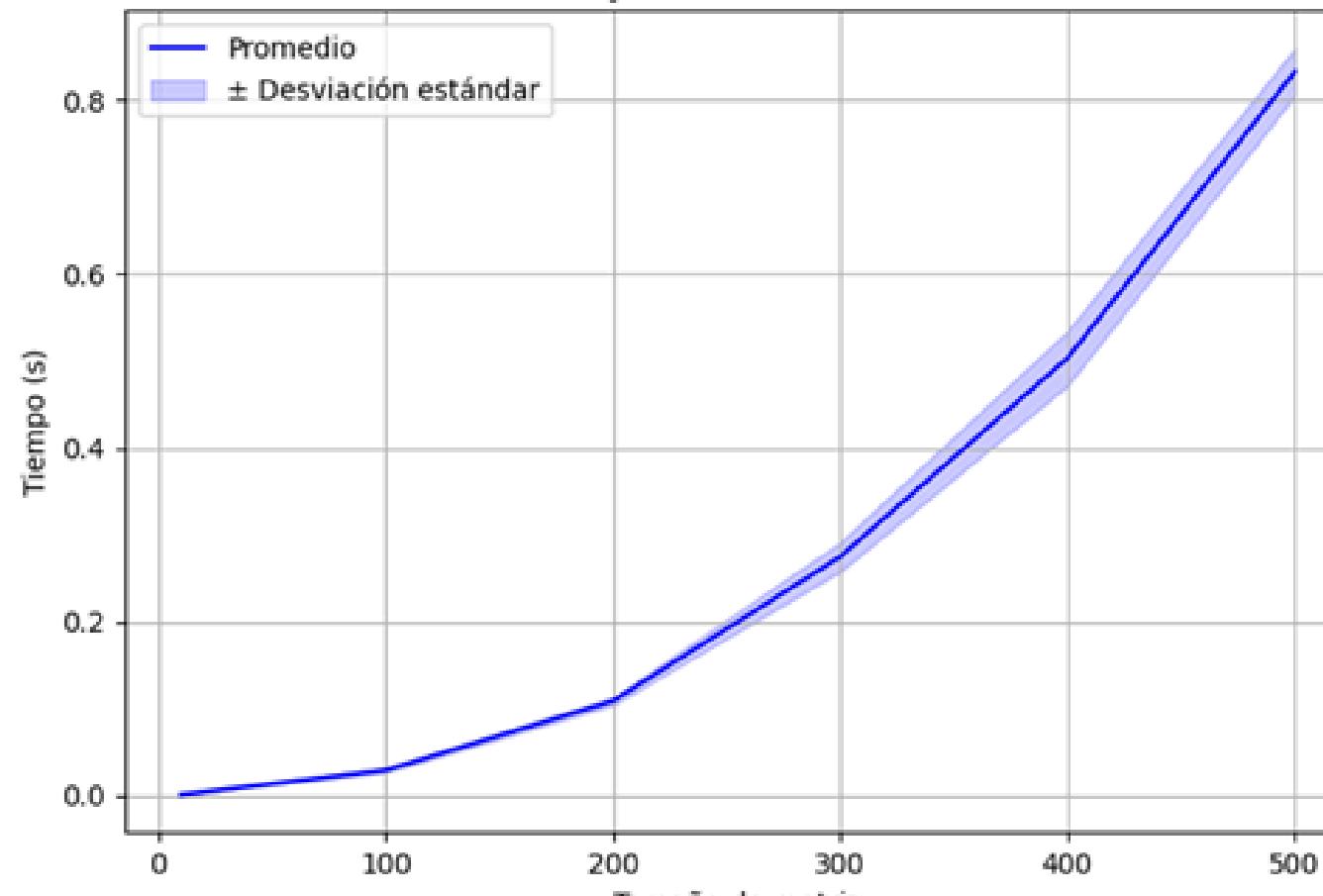
Comparación de tiempos de ejecución (Python vs C, Secuencial vs Paralelo)



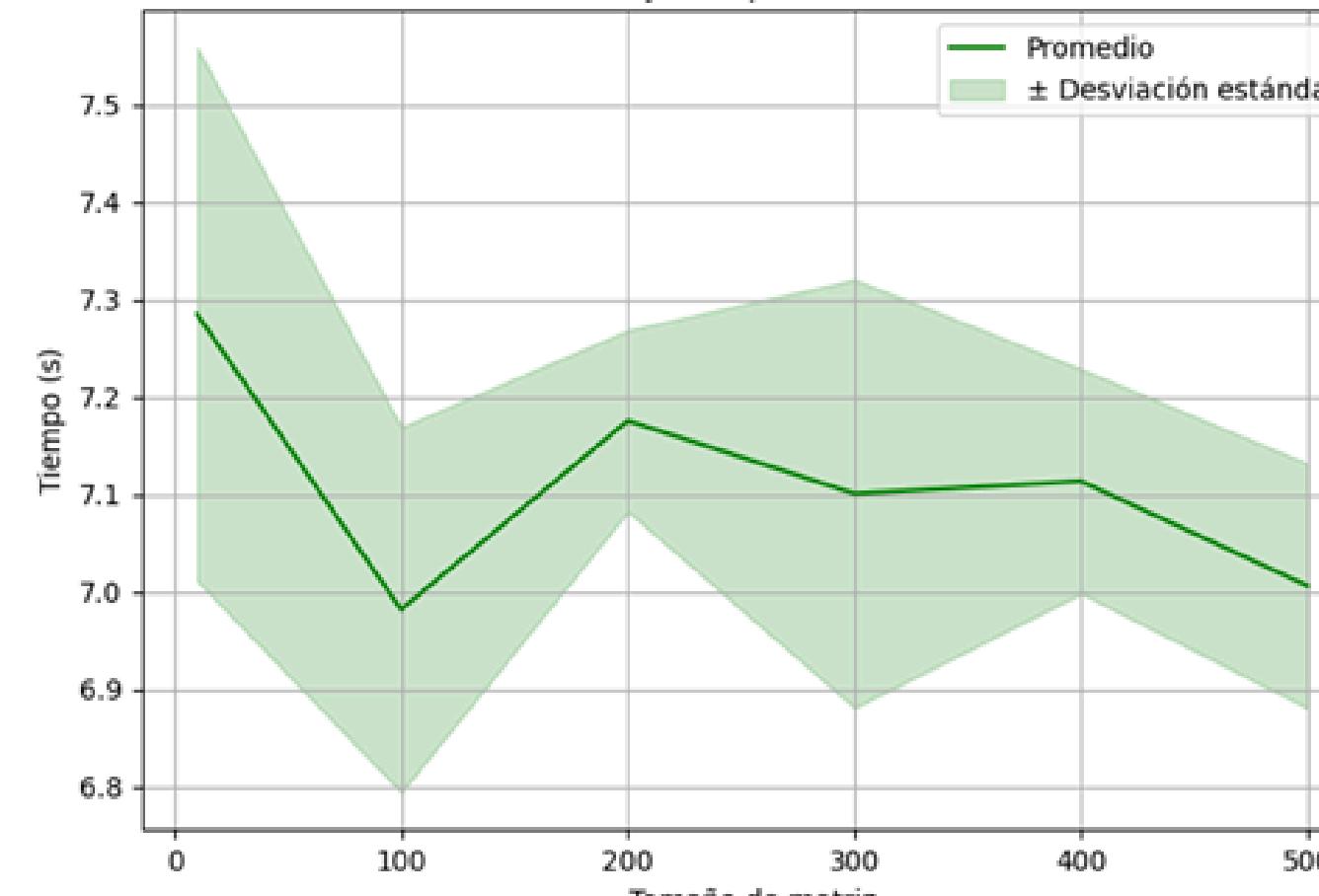
Reducción Gauss

Comparación de tiempos de ejecución (Gaussiano, hasta tamaño 500)

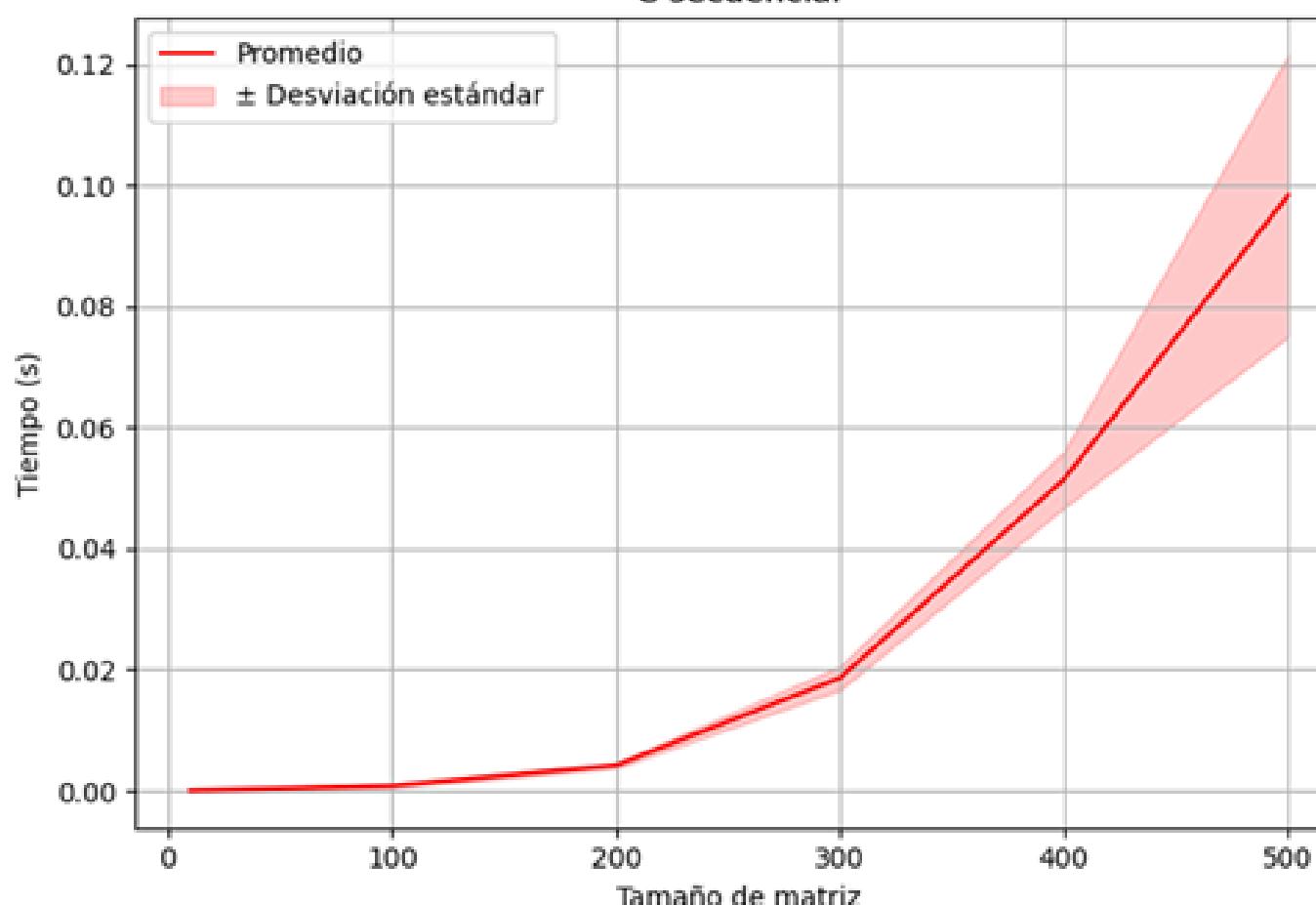
Python secuencial



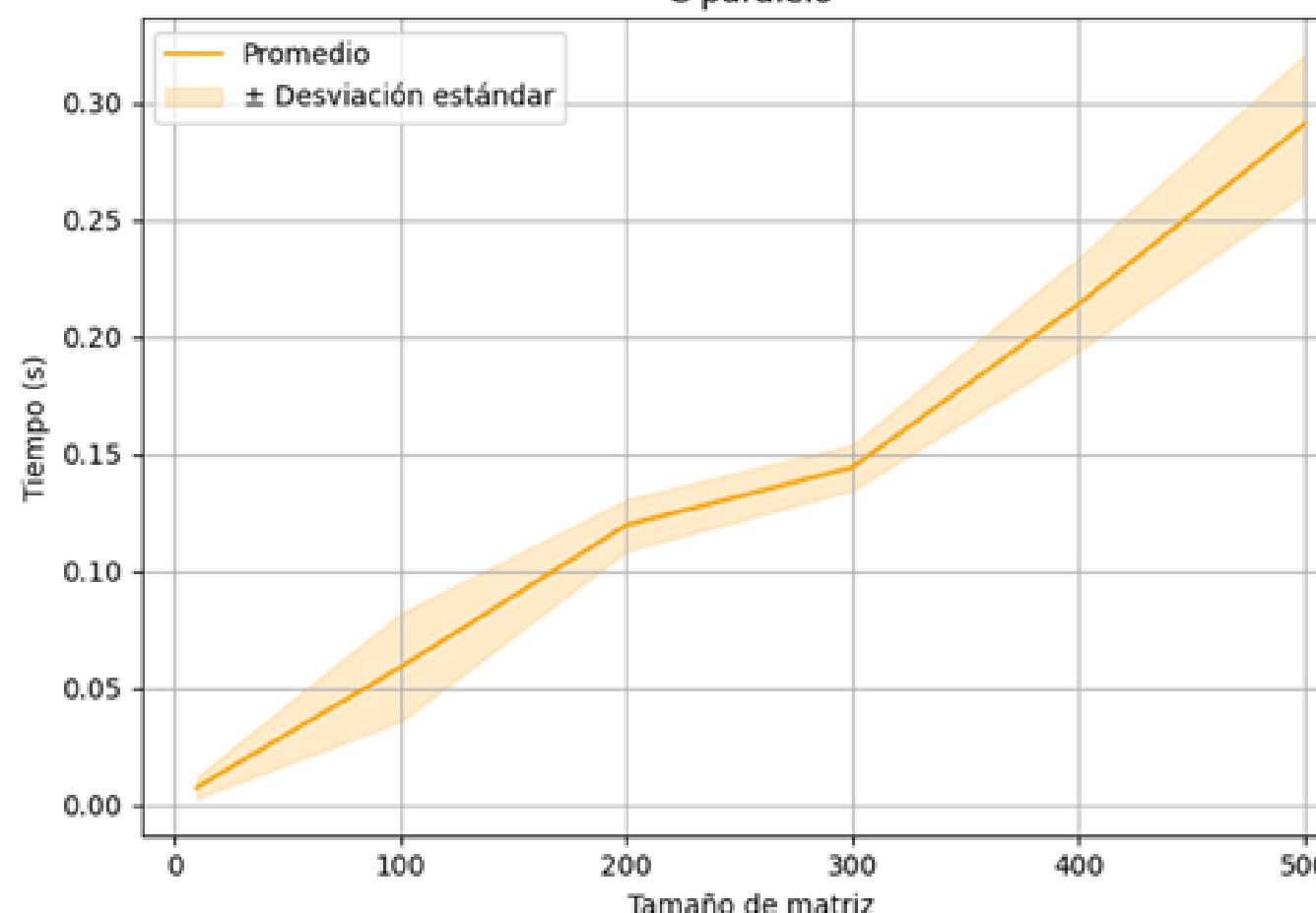
Python paralelo



C secuencial

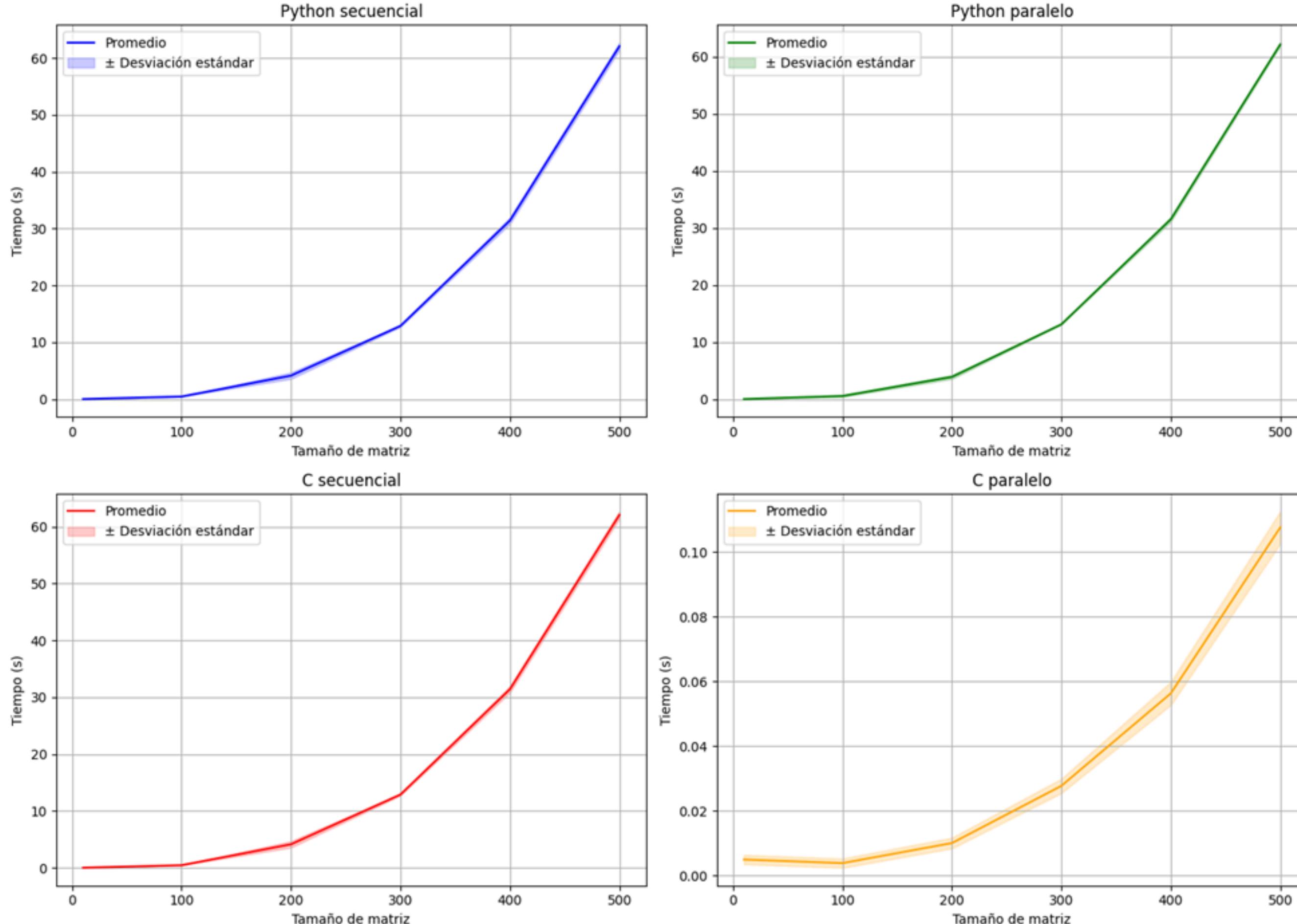


C paralelo

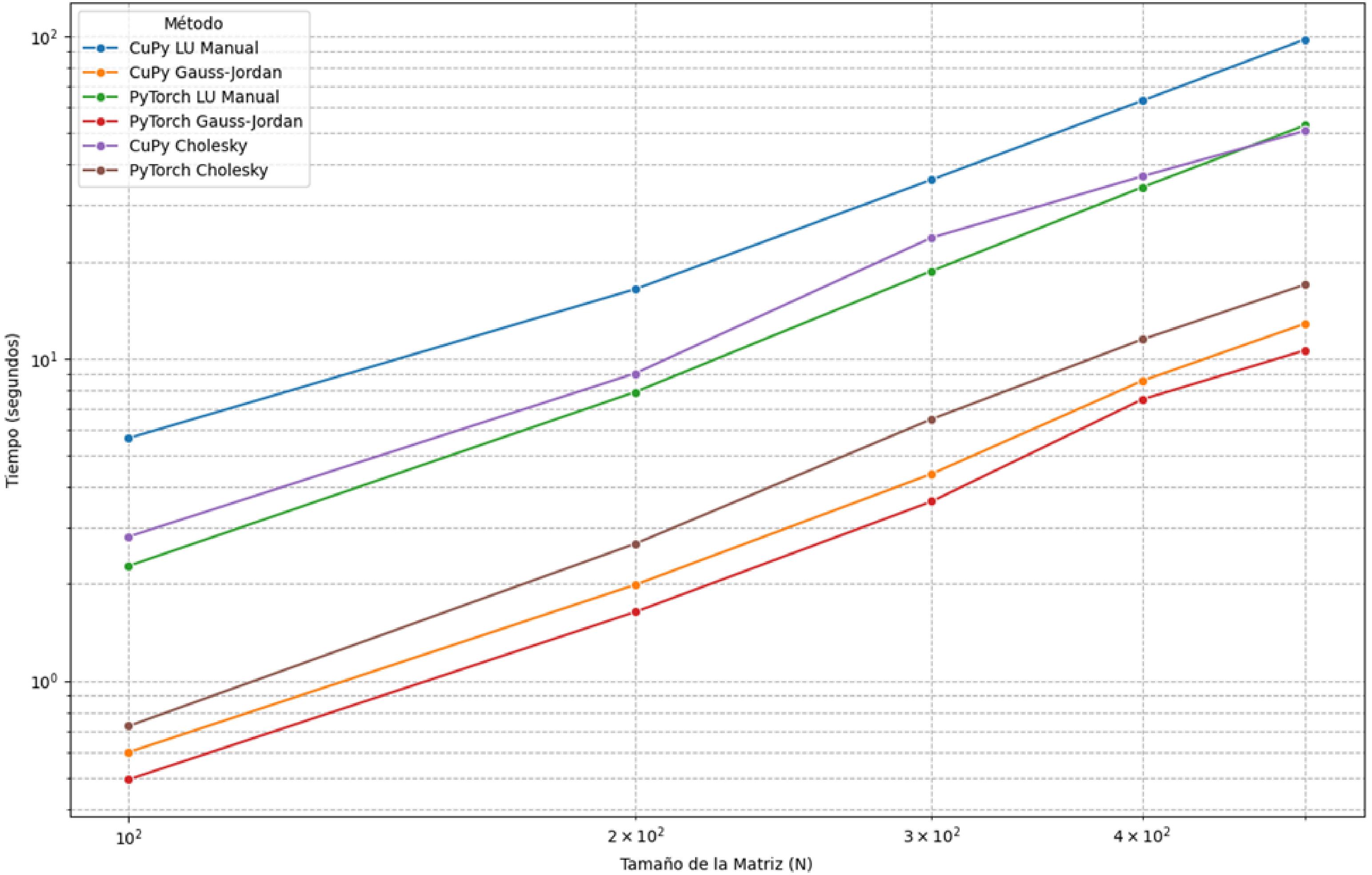


Descomposición Cholesky

Comparación de tiempos de ejecución (Cholesky - Python vs C, Secuencial vs Paralelo)



Tiempo de Ejecución de Inversión de Matrices por Tamaño y Algoritmo (GPU)



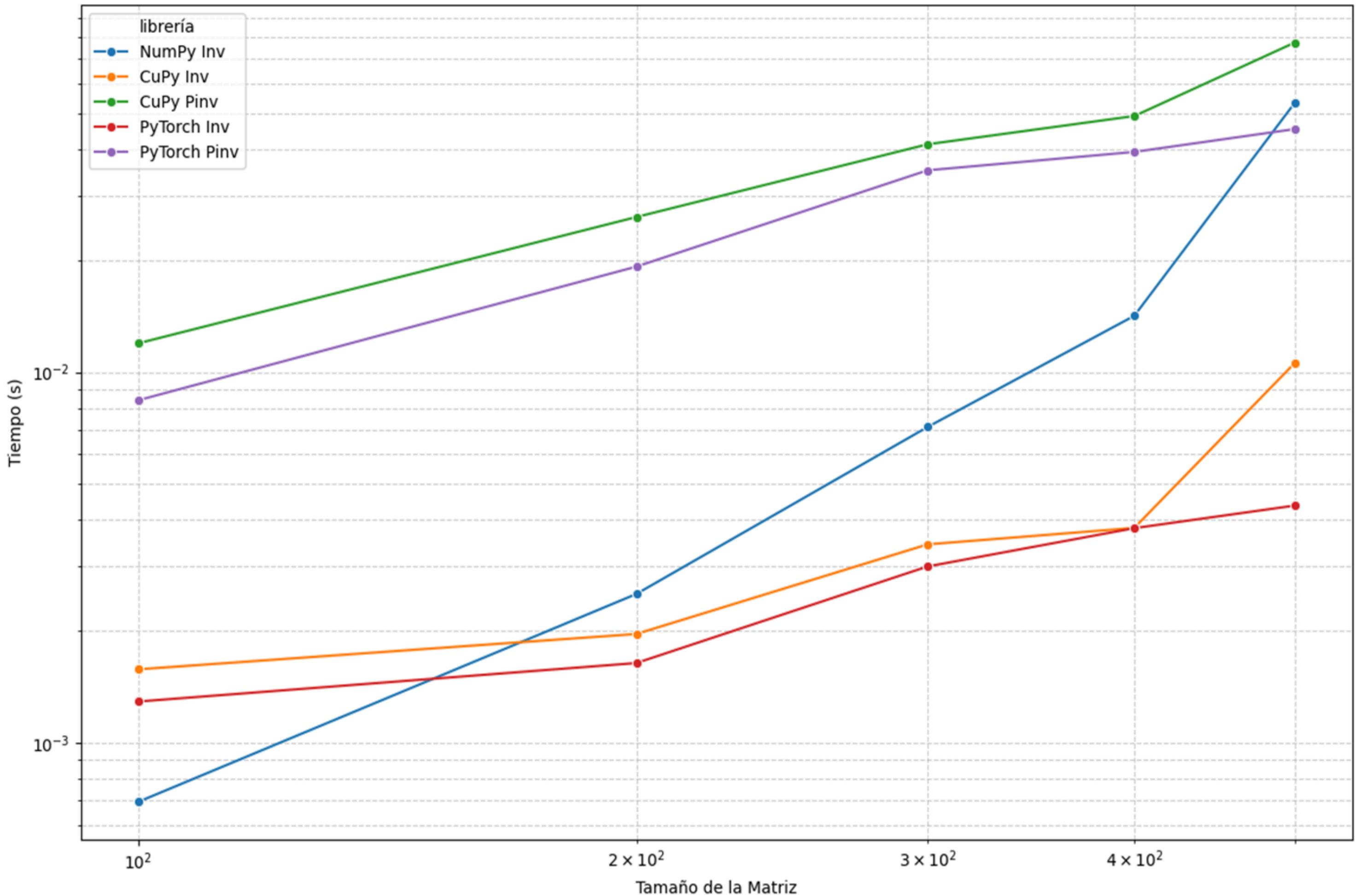
CUDA [CuPy-PyTorch]



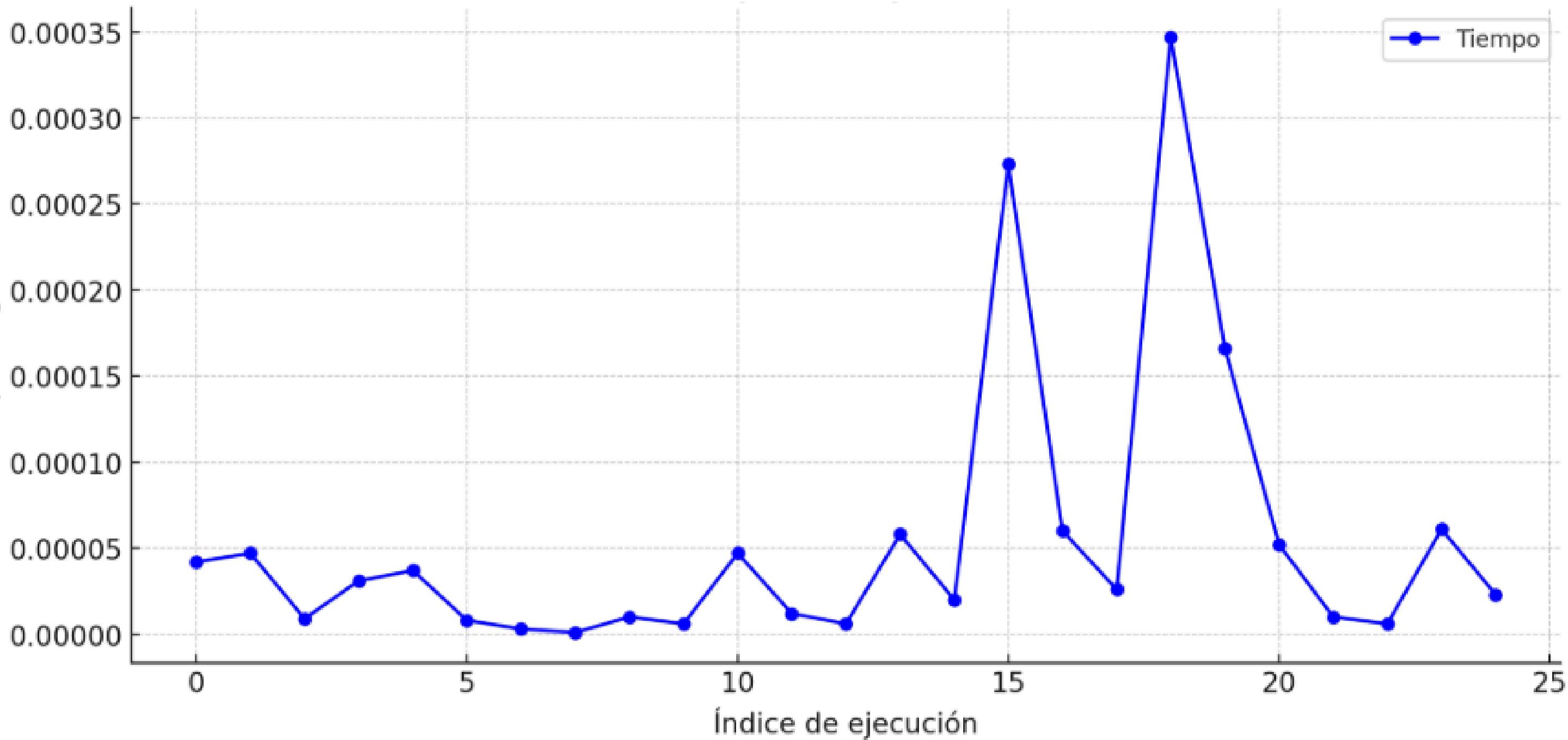
Adicional - Paquetes Alg. Lineal

CUDA [Inverse-Pseudo]

Tiempo de Inversión por Tamaño y Librería



Tiempos de ejecución



Conclusiones

Cumplimiento de objetivos

El proyecto logró cumplir con éxito el objetivo de comparar distintos algoritmos de inversión de matrices en diversos entornos, evaluando su eficiencia, escalabilidad y desempeño en CPU y GPU.

Análisis de rendimiento

Las implementaciones con CUDA demostraron el mejor rendimiento, seguidas por las versiones en C, las cuales obtuvieron mejores tiempos que Python debido al control detallado de memoria y procesos.

Limitaciones del paralelismo

Se evidenció que el paralelismo no siempre es la mejor opción en matrices pequeñas, ya que la sobrecarga por la creación y sincronización de hilos puede afectar negativamente el rendimiento.

Aplicación de conocimientos

El proyecto permitió aplicar conceptos clave de Sistemas Operativos, como la gestión de memoria, procesos e hilos, y profundizar en el uso de computación paralela y aceleración por hardware, destacando la importancia de seleccionar la solución adecuada según los recursos disponibles y el tamaño del problema.