

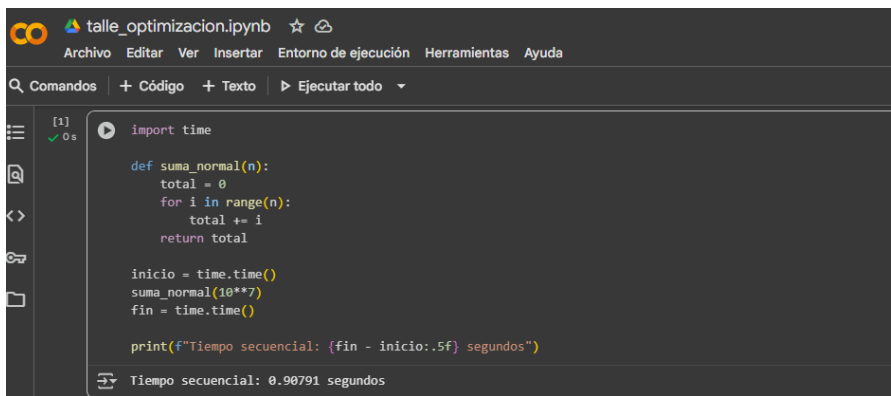
DESARROLLO TALLER ARQUITECTURA COMPUTACIONAL.

PRESENTADO POR:

- ESTEBAN GIOVANNY MENESES
- GIOVANNY ANDRÉS ZAMBRANO

1. ¿Qué hace lento este código? ¿Cómo acceder a una memoria? ¿Qué pasaría si lo dividimos?

- Lo más lento está en el bucle for con la suma total += i, porque Python procesa cada número de manera individual y secuencial; la memoria se accede de forma dispersa al crear tantos objetos, y si lo dividimos en varias partes o usamos una fórmula directa, la ejecución sería mucho más rápida.



The screenshot shows a Jupyter Notebook interface with a file named 'talle_optimizacion.ipynb'. The code in the cell is as follows:

```
import time

def suma_normal(n):
    total = 0
    for i in range(n):
        total += i
    return total

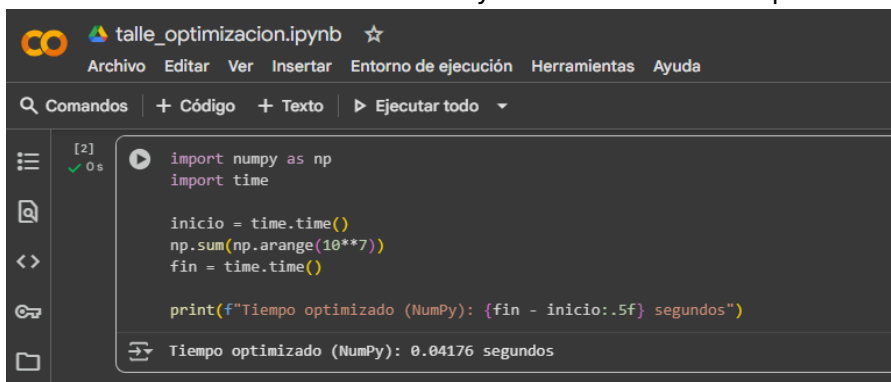
inicio = time.time()
suma_normal(10**7)
fin = time.time()

print(f"Tiempo secuencial: {fin - inicio:.5f} segundos")
```

The output of the cell is: `Tiempo secuencial: 0.90791 segundos`.

2. ¿Por qué NumPy es más rápido? ¿Qué ocurre a nivel de CPU? (Uso de instrucciones vectorizadas, menor CPI).

- Es más rápido porque hace el trabajo “de un tirón” en código compilado (C) y usa instrucciones del procesador que operan sobre muchos datos a la vez; así evita el bucle anterior y reduce mucho el tiempo.



The screenshot shows the same Jupyter Notebook interface. The code in the cell is as follows:

```
import numpy as np
import time

inicio = time.time()
np.sum(np.arange(10**7))
fin = time.time()

print(f"Tiempo optimizado (NumPy): {fin - inicio:.5f} segundos")
```

The output of the cell is: `Tiempo optimizado (NumPy): 0.04176 segundos`.

3. ¿Qué relación tiene esto con arquitecturas multinúcleo? ¿Cómo se distribuye la carga?

- Lo que pasa es que divide la lista en trozos y los reparte entre varios núcleos, cada uno calcula su parte en paralelo y luego se juntan los resultados; así se aprovecha el hardware multinúcleo y baja el tiempo total.

```

from multiprocessing import Pool
import time

def cuadrado(x):
    return x * x

if __name__ == '__main__':
    numeros = list(range(10**6))

    inicio = time.time()
    with Pool() as p:
        p.map(cuadrado, numeros)
    fin = time.time()

    print(f"Tiempo con procesamiento paralelo: {fin - inicio:.5f} segundos")

```

Tiempo con procesamiento paralelo: 0.45149 segundos

4. ¿Por qué la segunda es más lenta? Cada iteración evalúa la condición, lo que aumenta el número de ciclos necesarios por instrucción (CPI).

- Porque la versión con la condición dentro del bucle es más lenta porque verifica lo mismo millones de veces; poner la condición fuera evita ese costo repetido.

```

# Condicional dentro del bucle
def version_2():
    data = range(10**7)
    total = 0
    for i in data:
        if len(data) > 0:
            total += i
    return total

for version in [version_1, version_2]:
    start = time.perf_counter()
    result = version()
    end = time.perf_counter()
    print(f"{version.__name__}: {end - start:.5f} segundos")

```

version_1: 0.61911 segundos
version_2: 1.19024 segundos

5. En la mayoría de las arquitecturas, las matrices se almacenan en memoria fila a fila. Acceder por columnas genera más fallos de caché, lo que afecta negativamente el rendimiento. ¿Por qué se presenta esto? Investiga y responde brevemente.

- Recorrer por filas suele ser más rápido que por columnas porque los datos están guardados seguidos en memoria; al ir en el orden normal se aprovecha mejor la caché y se evitan saltos costosos.

CO **talle_optimizacion.ipynb** ☆

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Q Comandos + Código + Texto ▶ Ejecutar todo ▼

```

[6] ✓ 0 s
for i in range(1000):
    for j in range(1000):
        total += matriz[i][j]
    return total

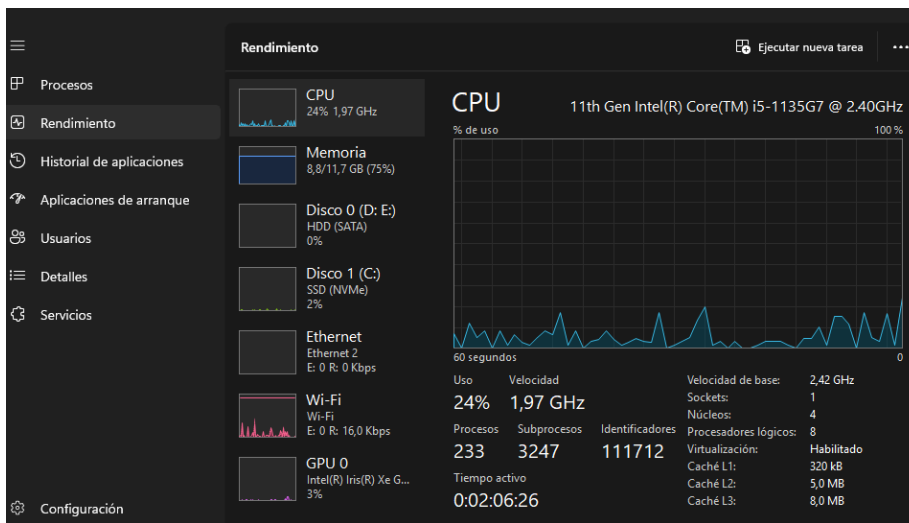
def acceso_columnas():
    total = 0
    for j in range(1000):
        for i in range(1000):
            total += matriz[i][j]
    return total

for version in [acceso_filas, acceso_columnas]:
    start = time.perf_counter()
    version()
    end = time.perf_counter()
    print(f"(version.__name__): {end - start:.5f} segundos")

acceso_filas: 0.35713 segundos
acceso_columnas: 0.38873 segundos

```

2DA PARTE



Solución

$$C_{pi} = \frac{720 \times 1,97 \times 10^9}{100 \times 10^9} = 14,184$$

$$C_{pi} = \frac{14,184 \times 10^9}{100 \times 10^9}$$

C_{pi} estimado ≈ 14,18
ciclos de reloj