Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática

Docente: Fred Torres Cruz

Autor: Mary Luz Nina Palacios

Trabajo Encargado - N° 002
Enunciado de la tarea:

1- Modificar el programa para generar arrays aleatorios en (a) y (b)
2. Realizar la modificación para el calculode una suma ordinaria y una suma paralela
3. Evidenciar la optimización de tiempo entre ambos algoritmos.

Algoritmo Concurrente

a) Codigo en Python

```
import random
import time

def suma_ordinaria(a, b):
    c = [0] * len(a)
    for i in range(len(a)):
        c[i] = a[i] + b[i]
    for i in range(len(c)):
        print(f"c[{i}]={c[i]}")
    return c

if __name__ == "__main__":
    # Generar arrays aleatorios
    a = [random.randint(1, 100) for _ in range(5)]
    b = [random.randint(1, 100) for _ in range(5)]

    start_time = time.time()
    c_ordinaria = suma_ordinaria(a, b)
    end_time = time.time()
    tiempo_ordinaria = end_time - start_time

    print(f"Tiempo de suma ordinaria: {tiempo_ordinaria:.6f} segundos")
```

Explicación:

1. La función `suma_ordinaria` toma dos listas `a` y `b` como argumentos.
2. Dentro de la función, se crea una nueva lista `c` del mismo tamaño que `a` y `b`, inicializada con ceros.
3. Luego, se itera sobre los índices de las listas `a` y `b` utilizando un bucle `for`.
4. En cada iteración, se suma el elemento correspondiente de `a` y `b`, y se asigna el resultado al elemento correspondiente en la lista `c`.
5. Después de completar la iteración, la función retorna la lista resultante `c`.
6. En el bloque `if __name__ == "__main__":`, se generan dos listas aleatorias `a` y `b` de longitud 5 con números enteros entre 1 y 100.
7. Se toma el tiempo actual antes de llamar a la función `suma_ordinaria`.
8. Se llama a la función `suma_ordinaria` pasando las listas `a` y `b` como argumentos, y se almacena el resultado en `c_ordinaria`.
9. Se toma el tiempo actual después de ejecutar la suma.
10. Se calcula el tiempo transcurrido restando el tiempo inicial del tiempo final.
11. Se imprime el tiempo de ejecución de la suma ordinaria.
12. Se imprime el resultado de la suma ordinaria en el formato solicitado (`c[i]=resultado`).

Este algoritmo secuencial realiza la suma de dos listas de forma iterativa, elemento por elemento. Su complejidad temporal es O(n), donde n es la longitud de las listas, ya que debe iterar sobre todos los elementos de las listas para realizar la suma.

b) Salida

```
PS C:\Users\Usuario\Desktop\COMPUTACION PARALELA> & C:/Users/Usuario/AppData/Local/Programs/Python/Python311/python.exe
 "c:/Users/Usuario/Desktop/COMPUTACION PARALELA/Suma_ordinaria.py"
c[0]=9
c[1]=118
c[2]=132
c[3]=39
c[4]=155
Tiempo de suma ordinaria: 0.001000 segundos
```

Algoritmo Paralelo

a) Codigo en Python

```
import multiprocessing
import random
import time

def worker(tid, a, b, c):
    c[tid] = a[tid] + b[tid]
    print(f"c[{tid}]={c[tid]}")

def suma_paralela(a, b):
    c = multiprocessing.Array('i', len(a))  # Shared array

    processes = []
    for tid in range(len(a)):
        process = multiprocessing.Process(target=worker, args=(tid, a, b, c))
        processes.append(process)
        process.start()

    for process in processes:
        process.join()

    return list(c)

if __name__ == "__main__":
    # Generar arrays aleatorios
    a = [random.randint(1, 100) for _ in range(5)]
    b = [random.randint(1, 100) for _ in range(5)]

    start_time = time.time()
    c_paralela = suma_paralela(a, b)
    end_time = time.time()
    tiempo_paralela = end_time - start_time

    print(f"Tiempo de suma paralela: {tiempo_paralela:.6f} segundos")
```

Explicación:

1. Se importan los módulos necesarios: `multiprocessing` para la creación y gestión de procesos, `random` para generar números aleatorios y `time` para medir el tiempo de ejecución.

2. Se define la función `worker`, que recibe el identificador de proceso (`tid`), las listas `a` y `b`, y el arreglo compartido `c`. Esta función suma los elementos correspondientes de `a` y `b` en la posición `tid` del arreglo `c` e imprime el resultado.

3. Se define la función `suma_paralela`, que toma las listas `a` y `b` como entrada. Dentro de esta función:

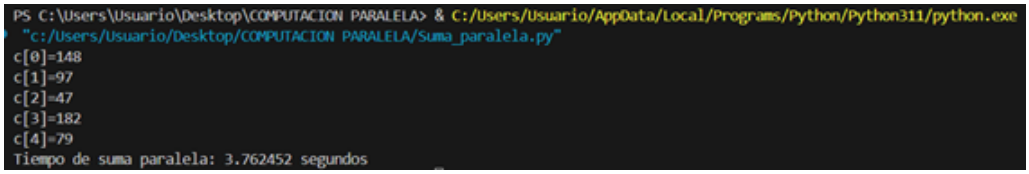
   - Se crea un arreglo compartido `c` utilizando `multiprocessing.Array` para almacenar los resultados.
   - Se crea una lista vacía `processes` para almacenar los procesos creados.
   - Se itera sobre los índices de las listas `a` y `b`.
   - Por cada índice `tid`, se crea un proceso utilizando `multiprocessing.Process` y se le asigna la función `worker` con los argumentos `tid`, `a`, `b` y `c`.
   - Los procesos se añaden a la lista `processes` y se inician con `process.start()`.
   - La función espera a que todos los procesos terminen utilizando `process.join()`.
   - Finalmente, se convierte el arreglo compartido `c` a una lista y se devuelve.

4. En el bloque `if __name__ == "__main__"::`

   - Se generan dos listas aleatorias `a` y `b` de longitud 5 con números enteros entre 1 y 100.
   - Se toma el tiempo actual antes de llamar a la función `suma_paralela`.
   - Se llama a la función `suma_paralela` pasando las listas `a` y `b` como argumentos, y se almacena el resultado en `c_paralela`.
   - Se toma el tiempo actual después de ejecutar la suma paralela.
   - Se calcula el tiempo transcurrido restando el tiempo inicial del tiempo final.
   - Se imprime el tiempo de ejecución de la suma paralela.

Este código paralelo divide el trabajo de sumar las listas entre múltiples procesos, lo que puede resultar en un rendimiento mejorado en comparación con el algoritmo secuencial, especialmente cuando se trabaja con grandes cantidades de datos y se cuenta con múltiples núcleos de CPU disponibles.

b) Salida



Reportes de tiempos de ejecución

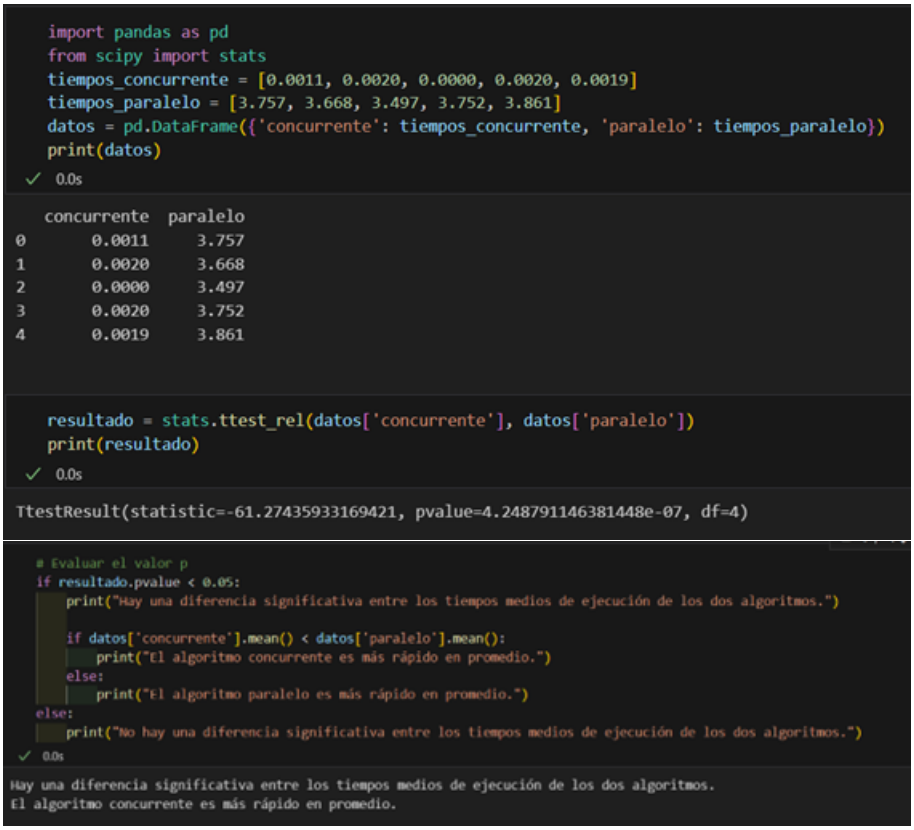
Cuadro 1: Tabla de tiempos de ejecucion en segundos

N° DE PRUEBA	ALGORITMO CONCURRENTE	ALGORITMO PARALELO
1	0.0011	3.757
2	0.0020	3.668
3	0.0000	3.497
4	0.0020	3.752
5	0.0019	3.861

Prueba estadistica - t Student
Codigo en RStudio

```
import pandas as pd
from scipy import stats

# Datos de los tiempos de ejecución
tiempos_concurrente = [0.0011, 0.0020, 0.0000, 0.0020, 0.0019]
tiempos_paralelo = [3.757, 3.668, 3.497, 3.752, 3.861]

# Crear el DataFrame
datos = pd.DataFrame({'concurrente':
tiempos_concurrente, 'paralelo': tiempos_paralelo})

print(datos)

# Realizar la prueba t de muestras pareadas
resultado = stats.ttest_rel(datos['concurrente'], datos['paralelo'])

print(resultado)

# Evaluar el valor p
if resultado.pvalue < 0.05:
    print("Hay una diferencia significativa entre los tiempos medios
```

```
        de ejecución de los dos algoritmos.")

        if datos['concurrente'].mean() < datos['paralelo'].mean():
            print("El algoritmo concurrente es más rápido en promedio.")
        else:
            print("El algoritmo paralelo es más rápido en promedio.")
    else:
        print("No hay una diferencia significativa entre los tiempos medios
        de ejecución de los dos algoritmos.")
```



```
import pandas as pd
from scipy import stats
tiempos_concurrente = [0.0011, 0.0020, 0.0000, 0.0020, 0.0019]
tiempos_paralelo = [3.757, 3.668, 3.497, 3.752, 3.861]
datos = pd.DataFrame({'concurrente': tiempos_concurrente, 'paralelo': tiempos_paralelo})
print(datos)
```

✓ 0.0s

```
   concurrente  paralelo
0       0.0011     3.757
1       0.0020     3.668
2       0.0000     3.497
3       0.0020     3.752
4       0.0019     3.861
```

```
resultado = stats.ttest_rel(datos['concurrente'], datos['paralelo'])
print(resultado)
```

✓ 0.0s

```
TtestResult(statistic=-61.27435933169421, pvalue=4.248791146381448e-07, df=4)
```

```
# Evaluar el valor p
if resultado.pvalue < 0.05:
    print("Hay una diferencia significativa entre los tiempos medios de ejecución de los dos algoritmos.")

    if datos['concurrente'].mean() < datos['paralelo'].mean():
        print("El algoritmo concurrente es más rápido en promedio.")
    else:
        print("El algoritmo paralelo es más rápido en promedio.")
else:
    print("No hay una diferencia significativa entre los tiempos medios de ejecución de los dos algoritmos.")
```

✓ 0.0s

```
Hay una diferencia significativa entre los tiempos medios de ejecución de los dos algoritmos.
El algoritmo concurrente es más rápido en promedio.
```

GITHUB - LUZ052002