Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática
Docente: Fred Torres Cruz
Autor: Cliver Wimar Vilca Tinta

Trabajo Encargado - N° 002

Análisis Comparativo de la Eficiencia de la Suma Paralela y Concurrente en Python: Aplicación de Pruebas Estadísticas para Evaluar el Rendimiento

1. Suma Paralelo

El paralelismo y la concurrencia son técnicas habituales para optimizar el rendimiento en el procesamiento de datos, distribuyendo la carga de trabajo entre varios hilos o procesos. En este estudio, se comparan dos métodos para sumar listas de números: uno paralelo, que utiliza multiprocesamiento en Python, y otro concurrente, pero secuencial. El propósito es establecer cuál de los dos métodos es más eficiente para esta tarea específica, analizando los tiempos de ejecución y realizando un análisis estadístico para identificar diferencias significativas en el rendimiento.

1.1. Codigo Python - Paralelo

```
import multiprocessing
import random
import time
def worker(tid, a, b, c):
    c[tid] = a[tid] + b[tid]
    print(f"c[{tid}]={c[tid]}")
if __name__ == "__main__":
    n = int(input("Ingrese la cantidad de números que desea sumar: "))
    a = [random.randint(1, 10) for _ in range(n)]
    b = [random.randint(1, 10) for _ in range(n)]
    c = multiprocessing.Array('i', n)

    num_cores = multiprocessing.cpu_count()
    print(f"Número de núcleos de tu computadora: {num_cores}")
    print(f"Número de hilos de tu computadora: {num_cores * 2}")
    processes = []
    print("Lista a:", a)
    print("Lista b:", b)
    start_time = time.time()
    for tid in range(min(n, num_cores)):
        process = multiprocessing.Process(target=worker, args=(tid, a, b, c))
```

```
        processes.append(process)
        process.start()
    for process in processes:
        process.join()
    end_time = time.time()
    execution_time = end_time - start_time
    print(f"Tiempo de ejecución: {execution_time} segundos")
```

Salida para un tamaño 5:

```
Ingrese la cantidad de números que desea sumar: 5
Número de núcleos de tu computadora: 16
Número de hilos de tu computadora: 32

Lista a: [9, 1, 8, 6, 5]
Lista b: [9, 6, 10, 10, 2]
c[0]=18
c[1]=7
c[2]=18
c[3]=16
c[4]=7
Tiempo de ejecución: 0.13062262535095215 segundos
```

Salida para un tamaño 10000:

```
c[0]=15
c[1]=5
c[2]=13
c[3]=17
c[4]=17
c[5]=16
c[6]=9
c[7]=15
c[8]=9
c[9]=15
c[10]=17
c[11]=8
c[12]=7
c[13]=12
c[14]=6
c[15]=3
Tiempo de ejecución: 1.2445449829101562 segundos
```

2. Suma Secuencial

Al sumar listas de números de forma concurrente, se realizan múltiples operaciones de suma de manera simultánea, aunque no necesariamente todas las listas se procesen al mismo tiempo. Este método implica alternar entre las diferentes sumas para mantener la concurrencia, lo que puede causar ciertos retrasos debido a la necesidad de sincronización de las operaciones. Sin embargo, la ejecución concurrente suele ser menos compleja que la paralela con múltiples hilos, ya que no requiere gestionar directamente los hilos de ejecución.

2.1. Codigo Python - Secuencial

```
import random
import time
def worker(a, b):
    c = [a[i] + b[i] for i in range(len(a))]
    return c
if __name__ == "__main__":
    n = int(input("Ingrese la cantidad de números que desea sumar: "))
    a = [random.randint(1, 10) for _ in range(n)]
    b = [random.randint(1, 10) for _ in range(n)]
    print("Lista a:\n", a)
    print("Lista b:\n", b)
    start_time = time.time()
    c = worker(a, b)
    end_time = time.time()
    execution_time = end_time - start_time
    print(f"Resultado de la suma en forma secuencial: {c}\n")
    print(f"Tiempo de ejecución en forma secuencial: {execution_time} segundos")
```

Salida para un tamaño 5:

```
Ingrese la cantidad de números que desea sumar: 5
Lista a:
 [4, 3, 8, 7, 8]
Lista b:
 [9, 2, 8, 7, 3]
Resultado de la suma en forma secuencial: [13, 5, 16, 14, 11]

Tiempo de ejecución en forma secuencial: 0.0 segundos
```

Salida para un tamaño 10000:

```
Ingrese la cantidad de números que desea sumar: 10000
Tiempo de ejecución en forma secuencial: 0.0009999275207519531 segundos
```

3. Análisis de Rendimiento

Con el fin de determinar cuál de los enfoques (paralelo o concurrente) es más óptimo y eficiente para el problema de la suma de dos listas, se realizaron múltiples ejecuciones de ambos códigos con diferentes tamaños de listas, esto en un equipo con 16 nucleos y 32 hilos. Los resultados obtenidos se muestran en la siguiente tabla:

Como se puede observar en el Cuadro 1, el enfoque concurrente presenta tiempos de ejecución mucho más rápidos que el enfoque paralelo para todos los tamaños de listas evaluados. Esta diferencia se hace más significativa a medida que aumenta el tamaño del problema.

Tamaño de las listas	Tiempo de ejecución paralelo (s)	Tiempo de ejecución concurrente
5	0.1386	0.0000
10	0.2590	0.0000
100	0.3818	0.0000
1000	1.1569	0.0000
10000	1.2229	0.0009

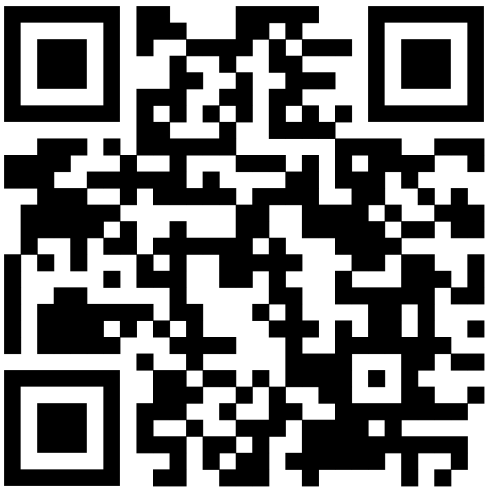
Cuadro 1: Tiempos de ejecución promedios para diferentes tamaños de listas.

La decisión entre la ejecución paralela y secuencial varía en función del problema, el tamaño de los datos y la complejidad de la tarea. En este escenario específico, la suma secuencial puede ser más efectiva que la paralela debido al costo adicional asociado con la paralelización en una tarea simple con una carga de trabajo reducida. Además, el desempeño se ve afectado por las especificaciones del procesador y del equipo utilizados. Por ejemplo, en un sistema con 16 núcleos y 32 hilos, la eficiente gestión de múltiples hilos es esencial para lograr un rendimiento óptimo en el procesamiento de datos.

3.1. Discusión

Los resultados indican que la suma secuencial superó en eficiencia a la paralela, posiblemente debido al costo adicional de la paralelización y la sincronización entre hilos en el enfoque paralelo. Además, el tamaño del problema y la capacidad del hardware, como un sistema con 16 núcleos y 32 hilos, también tuvieron un impacto en los resultados. En resumen, elementos como el costo adicional de la paralelización, la sincronización entre hilos y el tamaño del problema influyeron en la eficacia de la ejecución paralela en comparación con la secuencial en esta tarea de procesamiento de datos.

Código QR Repositorio GitHub



Link Repositorio GitHub https://github.com/CliverVilca/Parallel-Computing.git