Evaluación de una función mediante paralelización

Diego Castro Elvira
Instituto Politécnico Nacional, UPIIT
dcastroe2100@alumno.ipn.mx

1. Problema

Se debe desarrollar un programa que pueda evaluar una función de manera paralela utilizando programación funcional. Este programa debe cumplir con los siguientes requisitos:

- **Balanceo de cargas:** Se debe implementar un mecanismo para distribuir equitativamente la carga de trabajo entre los diferentes procesos o hilos de ejecución que evaluarán la función. Esto garantiza una utilización eficiente de los recursos disponibles y un tiempo de ejecución optimizado.
- **Evaluación de funciones lambda:** El programa debe ser capaz de recibir cualquier función en notación de cálculo lambda. Esto significa que el usuario podrá ingresar la función que desea evaluar utilizando una sintaxis funcional compacta y expresiva.
- **Post-procesamiento de resultados:** Una vez que se haya evaluado la función en paralelo, el programa debe reunir todos los resultados en una sola lista. Esto facilita la posterior manipulación o análisis de los datos obtenidos.

2. Solución paralela

```
def evaluador(datos, num_procesadores,funcion_lambda):
    t = time()
    nivel_cargas = nivelacion_cargas(D=datos, n_p=num_procesadores)
    #print("Nivelacion de cargas ->", nivel_cargas)
    with Pool(num_procesadores) as pool:
        resultados = pool.map(funcion_lambda, nivel_cargas)

    resultado = [items for mini_lista in resultados for items in
        mini_lista]
    tiempo_total = time() - t
    return resultado, tiempo_total
```

1. Se creó una función llamada $evaluador$ donde toma como parámetros
   - **datos**: son los valores a los que se les desea aplicar la función lambda
   - **num_procesadores**: Son el número de procesadores a los cuales se les dea aplicar las operaciones de paralelización
   - **funcio_lambda**: Es la función lambda la cual se desea evaluar
2. Los valores de 'datos' y 'num_procesadores' son pasados a la función $nivelacion_cargas$ donde se calcula la división equitativa de la carga de trabajo entre los procesadores
3. Luego, se crea un pool de procesos con el número especificado de procesadores. Después se ocupa el método map del pool para aplicar la función lambda a cada elemento de la lista de nivelación de cargas en paralelo.
4. Los resultados obtenidos de la evaluación en paralelo se combinan en una lista plana para el post-procesamiento

3. Pruebas

1. Aplicando la siguiente función lambda con los siguientes parámetros:

$$\lambda \to x^2 \tag{1}$$
$$datos \to 9000000$$
$$num\_procesadores \to 4$$

Comparándolo los tiempos de ejecución en segundos usando la técnica de paralelización y sin paralelización obtenemos:

Tabla 1: Comparación para la ecuación 1

Prueba	Paralelo	Sin paralelización
1	1.3294	2.0380
2	1.3970	2.1310
3	1.3922	2.1277

2. Aplicando la siguiente función lambda con los siguientes parámetros:

$$\lambda \to x^2 + \frac{x}{2} - x^3 \tag{2}$$
$$datos \to 50000000$$
$$num\_procesadores \to 4$$

Comparándolo los tiempos de ejecución en segundos usando la técnica de paralelización y sin paralelización obtenemos:

Tabla 2: Comparación para la ecuación 2

Prueba	Paralelo	Sin paralelización
1	6.8007	35.6026
2	6.9708	36.0671
3	6.1671	34.4207

3. Aplicando la siguiente función lambda con los siguientes parámetros:

$$\lambda \to x^2 - x^3 - x \cdot 2 + x^4 - \frac{x}{6} \tag{3}$$
$$datos \to 50000000$$
$$num\_procesadores \to 4$$

Comparándolo los tiempos de ejecución en segundos usando la técnica de paralelización y sin paralelización obtenemos:

Tabla 3: Comparación para la ecuación 3

Prueba	Paralelo	Sin paralelización
1	7.2642	62.6652
2	7.0019	62.9264
3	6.5493	64.3655

4. Conclusiones

Analizando los resultados obtenidos, queda claro que la técnica de paralelización ofrece un rendimiento significativamente superior en comparación con la ejecución no paralela. Este beneficio se vuelve aún más evidente al aplicar ecuaciones más complejas, donde el uso de recursos es mayor, al realizar la evaluación de manera paralela, observamos cómo el sistema aprovecha de forma más eficiente los recursos disponibles, distribuyendo la carga de trabajo entre múltiples procesos o hilos de ejecución.