CC3069 – Computación Paralela y Distribuida
Sección: 21

2do Ciclo – 2024

Corto #13

- Patrones de Programación Paralela -

Diego Alberto Leiva Pérez
Carné: 21752

Parte 1: Preguntas de Concepto

1. **Explica con tus propias palabras la diferencia entre descomposición funcional y descomposición de datos. Da un ejemplo para cada una.**
   La descomposición funcional divide un problema en tareas o funciones individuales. Cada tarea es un bloque de operaciones que debe ejecutarse, lo cual facilita la asignación de distintas funciones a distintos procesadores o hilos. Por ejemplo, en el procesamiento de una imagen, una función puede encargarse de la detección de bordes, otra del ajuste de brillo y una tercera de la detección de color. En cambio la descomposición de datos divide el conjunto de datos en partes y las procesa en paralelo. Un ejemplo sería procesar una gran matriz dividiéndola en varias sub-matrices y realizando operaciones en cada una por separado.

2. **¿Qué ventajas y desventajas tiene el paralelismo manual en comparación con el paralelismo automático?**

Ventajas	Desventajas
Paralelismo Manual	
Mayor control sobre la división del trabajo y la sincronización entre tareas; puede ser más eficiente en entornos específicos.	Más complejo de implementar y mantener; mayor probabilidad de errores, como condiciones de carrera o deadlocks.
Paralelismo Automático	
Menor carga de trabajo para el programador; el código suele ser más sencillo y menos propenso a errores de sincronización.	Puede ser menos eficiente que el manual debido a las limitaciones en el análisis automático; suele depender de hardware y software compatibles.

3. **En el patrón de diseño 'Fork-Join', ¿qué representa la fase de 'fork' y qué representa la fase de 'join'? Proporciona un ejemplo práctico.**
    - La fase de fork representa el momento en que una tarea se divide en sub-tareas independientes, que se ejecutan en paralelo.
    - La fase de join es cuando los resultados de las sub-tareas se combinan para formar el resultado final.
    - Ejemplo Práctico: En una búsqueda en un arreglo grande, el arreglo se puede dividir en varias sub-arreglos (fork) y realizar la búsqueda en cada sub-arreglo en paralelo. Los resultados de cada búsqueda se combinan al final para determinar si el elemento está presente (join).

4. **Describe el funcionamiento del patrón de diseño 'Stencil'. ¿En qué tipo de aplicaciones es más común su uso?**
   Este patrón actualiza elementos de una estructura de datos basándose en los valores de sus vecinos cercanos. Cada actualización se realiza sobre una celda de datos considerando sus celdas adyacentes. Es común en simulaciones científicas, como modelos de difusión térmica y dinámica de fluidos.

Parte 2: Aplicación Práctica

5. **Dado un conjunto de datos de 100 millones de registros, diseña un plan de paralelización utilizando el patrón 'Map-Reduce' para procesar estos datos. Explica cómo dividirías los datos y cómo los resultados finales serían combinados.**
    - Map: Divide los 100 millones de registros en bloques más pequeños (por ejemplo, bloques de un millón de registros) que pueden ser procesados en paralelo en múltiples nodos. Cada nodo procesará su bloque y generará un resultado intermedio.
    - Reduce: Una vez que los nodos hayan procesado sus bloques, se combinan los resultados intermedios en un paso de reducción, consolidando los resultados parciales en un resultado final.

6. **Imagina que estás trabajando en una simulación física donde se actualizan los estados de millones de partículas basadas en las posiciones de sus vecinas. ¿Qué patrón de diseño aplicarías y por qué? Explica cómo dividirías el trabajo entre múltiples hilos o procesadores.**
   El patrón Stencil es adecuado aquí, ya que el estado de cada partícula depende de las posiciones de sus vecinas. Dividiría el espacio en zonas y asignaría cada zona a un hilo. Para evitar conflictos en los límites de las zonas, se podrían utilizar zonas de buffer para procesar las interacciones entre partículas en diferentes zonas.

7. **Implementa en pseudocódigo un algoritmo que use el patrón 'Maestro/Esclavo' para realizar una tarea simple, como la suma de una lista de números distribuidos en múltiples procesadores.**

```
funcion MaestroEsclavo (lista):
    dividir lista en sublistas
    asignar cada sublista a un esclavo para el procesamiento en paralelo

    iniciar sumas parciales = []

    for cada esclavo:
       suma_parcial = esclavo.sumarSublista()
       agregar suma_parcial a sumas parciales

    return sumar(sumas parciales)
```

Parte 3: Proyecto Corto

Implementa una pequeña simulación en el lenguaje de tu preferencia (Python, C, C++, etc.) utilizando el patrón Fork-Join. La simulación debe calcular la suma de los elementos de una lista dividida en sublistas, donde cada sublista es procesada en paralelo, y luego los resultados parciales son combinados.

Requisitos del proyecto:

- El código debe ejecutarse en paralelo utilizando al menos 4 hilos o procesadores.
- Documenta cómo lograste dividir las tareas y cómo manejas la combinación de los resultados.
- Entrega el código junto con un reporte en PDF que explique la arquitectura utilizada y los resultados obtenidos.

Arquitectura

La simulación está basada en el patrón de diseño Fork-Join, que implica dividir una tarea en sub-tareas que se ejecutan en paralelo (fork) y luego combinar los resultados de estas sub-tareas para obtener el resultado final (join).

Fork (Division de Tareas)

- El programa divide la lista de elementos en sublistas de tamaño aproximadamente igual. En este caso, se utiliza un arreglo de 10, 100 y 1000 elementos y se asignan 4 hilos para procesar estas sublistas.
- Cada hilo recibe una porción de la lista y calcula la suma de los elementos en esa porción de forma independiente. Esta división es posible gracias al uso de *#pragma omp parallel*, que permite ejecutar múltiples hilos en paralelo utilizando la librería OpenMP.

Join (Combinación de Resultados):

- Después de que cada hilo termina de calcular su suma parcial, utiliza una sección crítica (#pragma omp critical) para agregar su resultado parcial a la variable suma_total. Este enfoque asegura que solo un hilo a la vez modifique suma_total, previniendo así condiciones de carrera.
- Finalmente, una vez que todos los hilos han agregado sus sumas parciales, suma_total contiene la suma completa de los elementos en la lista.

División de Tareas

La lista se divide en 4 partes, asignando a cada hilo una sublista específica según su ID. Cada hilo calcula la suma parcial de su sublista utilizando la función suma_parcial y luego imprime su resultado. La partición de la lista se define en función del número de hilos y el tamaño total de la lista (n/num_hilos), de manera que cada hilo reciba una porción equilibrada. En caso de que el tamaño de la lista no sea divisible exactamente entre el número de hilos, el último hilo toma la sección restante.

Documentación del Código

Inicialización de la Lista:

- Se crea un vector lista y se llena con los números del 1 al n, facilitando una suma de sus elementos.

Paralelización con OpenMP:

- #pragma omp parallel permite que los cálculos se realicen en paralelo.
- Cada hilo calcula su suma parcial con suma_parcial, luego imprime el resultado parcial y finalmente lo combina en suma_total mediante una sección crítica.

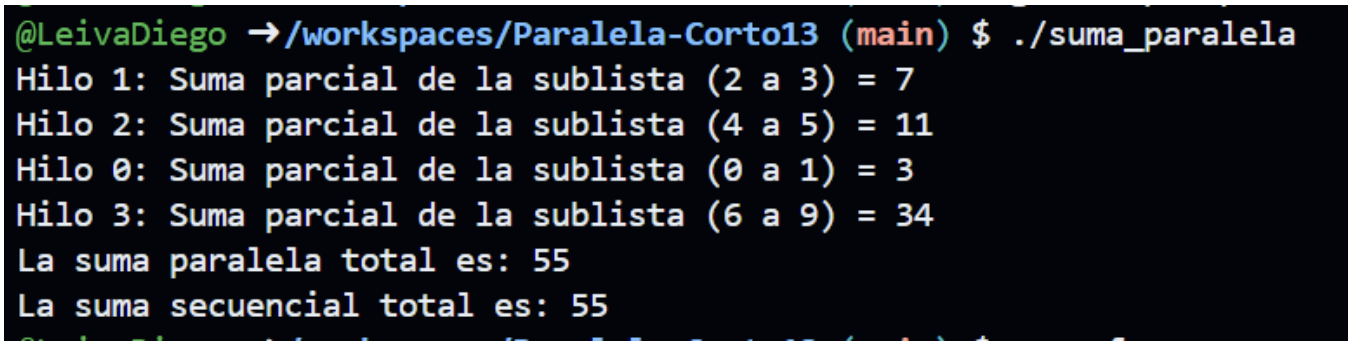
Resultados Parciales y Totales:

- Cada hilo imprime la suma parcial de su sublista, lo cual ayuda a visualizar cómo se distribuyeron y calcularon las tareas.
- Al final, se imprime suma_total, que contiene la suma total de la lista.

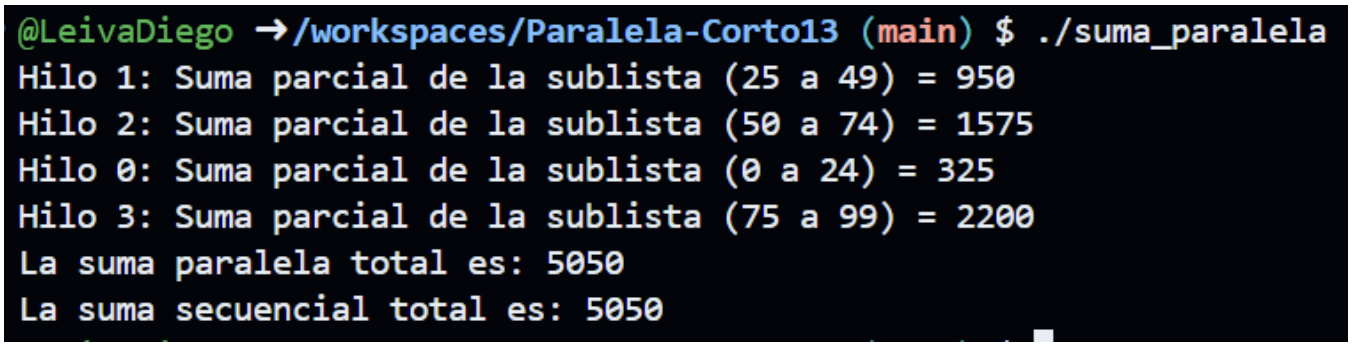
Resultados de la Simulación

Como ejemplo se hizo una simulación sencilla con una lista de 10, 100 y 1000.

Suma de lista de 10 elementos



Suma de lista de 100 elementos



Suma de lista de 1000 elementos

```
@LeivaDiego →/workspaces/Paralela-Corto13 (main) $ ./suma_paralela
Hilo 1: Suma parcial de la sublista (250 a 499) = 93875
Hilo 2: Suma parcial de la sublista (500 a 749) = 156375
Hilo 3: Suma parcial de la sublista (750 a 999) = 218875
Hilo 0: Suma parcial de la sublista (0 a 249) = 31375
La suma paralela total es: 500500
La suma secuencial total es: 500500
```

En los 3 casos de ejemplo la sumatoria de elementos de la lista de manera paralela son el mismo valor que la secuencial.

Repositorio:

https://github.com/LeivaDiego/Paralela-Corto13.git