UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3069 - Computación Paralela y Distribuida

Sección 21

Ing. Juan Luis García Zarceño

Examen Práctico - Corto No. 13

Patrones de Programación Paralela

José Pablo Orellana 21970

Guatemala, 07 de octubre del 2024

Examen Corto 13

Instrucciones:

Responde las siguientes preguntas y completa las actividades durante el fin de semana. Se permite usar cualquier recurso visto en clase o en la presentación. Deben entregarse el lunes 7 de octubre del 2024 antes de las 23:59. Subir las respuestas a la plataforma en un archivo PDF.

Parte 1: Preguntas de Concepto (20 puntos)

1. **(5 puntos) Explica con tus propias palabras la diferencia entre descomposición funcional y descomposición de datos. Da un ejemplo para cada una.**

    - La descomposición funcional es una técnica que lo que hace es dividir un problema en diferentes funciones o tareas independientes. Cada función realizará una parte específica del trabajo, y luego las tareas pueden ejecutarse en paralelo. Un ejemplo de ello puede ser el procesamiento de imágenes, una descomposición funcional puede dividir las tareas en filtrado, ajustar brillo, etc.
    - Mientras que la descomposición de datos lo que se haría sería dividir el conjunto de datos en varias partes para que puedan ser procesadas de manera simultánea por distintos hilos o procesadores. Y utilizando el ejemplo de imágenes, en este caso sería como procesar una imagen grande y cada procesador se encargaría de una sección de la imagen.

2. **(5 puntos) ¿Qué ventajas y desventajas tiene el paralelismo manual en comparación con el paralelismo automático?**

    - **Ventajas** del paralelismo **Manual** podría ser que se tiene mayor control de cómo se distribuyen las tareas y datos y también puede ser más eficiente en situaciones donde nosotros como programadores conocemos bien las particularidades del problema.
      La **desventaja** puede ser que es más propenso a errores y requiere de un esfuerzo adicional para gestionar bien todo.

    - En cuanto al paralelismo **automático**, la **ventaja** es que simplifica el trabajo de los programadores y reduco probabilidad de errores.
      Y la **desventaja** es que el compilador no podría tomar decisiones óptimas, lo que puede llevar a menor eficiencia.

3. **(5 puntos) En el patrón de diseño 'Fork-Join', ¿qué representa la fase de 'fork' y qué representa la fase de 'join'? Proporciona un ejemplo práctico.**

    - Fase Fork sería el momento en el que una tarea se divide en varias tareas pequeñas que se pueden ejecutar de forma paralela Y el Fase Join sería el punto donde las tareas paralelas completadas se sincronizan y sus resultados se combinan para continuar el flujo.
    - Un ejemplo práctico, sería que por ejemplo nosotros realicemos una suma de un gran array con números. En la Fase Fork, se divide el array en partes pequeñas y luego la Fase Join sería cuando las sumas parciales se combinan para tener una suma total.

4. **(5 puntos) Describe el funcionamiento del patrón de diseño 'Stencil'. ¿En qué tipo de aplicaciones es más común su uso?**

    - En cuanto al funcionamiento, cada elemento en una estructura es actualizado usando un vecindario de elementos circundantes. Este vecindario es fijo y la actualización de los elementos puede pasar de manera paralela.

    - El patrón sería común en simulaciones de física, análisis numérico o bien en procesamiento de imágenes.

Parte 2: Aplicación Práctica (30 puntos)

5. **(10 puntos) 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.**

    - Dividiría los 100 millones en bloques más pequeños, si tengo 10 nodos de procesamiento, dividiría los datos en 10 partes de 10 millones de registros cada una.
    - Cada nodo de procesamiento ejecuta la fase de Map, que sería aplicar una función a cada uno de los registros de su bloque de datos. La función sería cualquier operación que se quiera realizar sobre los datos.
    - La Fase Map produciría una serie de pares clave-valor para cada bloque de datos.
    - Una vez que cada nodo ya haya procesado su bloque, los pares clave-valor producidos en la fase de Map son enviados a la fase de Reduce.
    - En la fase de Reduce, los resultados se agrupan por la clave y se aplicarían las operaciones finales.
    - Por último, los resultados de cada nodo se combinan dando resultados finales.

6. **(10 puntos) 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.**

    - Usaría el diseño Stencil porque podría actualizar el estado de las partículas basadas en sus posiciones vecinas. Sería ideal porque son simulaciones donde las operaciones sobre un elemento dependen de sus vecinos.
    - En este caso, dividiría el espacio de simulación en bloques y asignaría cada bloque a un hilo distinto, cada bloque tendría un número de partículas que pueden ser actualizadas de forma paralela.
    - También haría un intercambio de información, porque las partículas de un bloque pueden depender de las posiciones de las partículas de bloques vecinos.
    - Este intercambio podría hacerse luego de cada paso de la simulación, permitiendo que cada procesador actualice su bloque para compartir los datos.

7. **(10 puntos) 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.**

    - ```
      function maestro(lista_de_numeros, num_esclavos):
          # Dividir la lista en partes iguales para cada esclavo
          tamanio_parte = longitud(lista_de_numeros) / num_esclavos
          resultados_parciales = []

          for i = 0 to num_esclavos - 1:
              # Asignar una parte de la lista a cada esclavo
                  parte = obtener_parte(lista_de_numeros, i * tamanio_parte, (i + 1) * tamanio_parte)
              resultado = esclavo(parte)  # Llamar a la función esclavo
              resultados_parciales.append(resultado)

          # El maestro combina los resultados parciales
          suma_total = 0
          for resultado in resultados_parciales:
              suma_total += resultado
          return suma_total

      function esclavo(parte):
          # Cada esclavo suma los números de su parte
          suma_parcial = 0
          for numero in parte:
              suma_parcial += numero
          return suma_parcial
      ```
      ```

Parte 3: Proyecto Corto (50 puntos)

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.

Documentación

División de la lista en sublistas

Calcular el tamaño de las sublistas:
Dado que la lista original contiene N elementos y se usan T hilos, se calcula el tamaño de cada sublista dividiendo la longitud de la lista (N) entre el número de hilos (T). El tamaño de cada sublista equivale a lo siguiente:

$$tamaño = \frac{N+T-1}{T}$$

Con este cálculo se puede garantizar que todas las sublistas sean de un tamaño similar y que los elementos sobrantes se distribuyen de manera equitativa entre los hilos.

Manejo de la combinación de resultados

Una vez que los hilos competaron la suma de sus sublistas, es necesario combinar los resultados parciales para obtener el resultado final.

Reducción de sumas parciales:
OpenMP facilita la combinación de los resultados parciales a través de la cláusula reduction. Esta cláusula asegura que la variable suma_total se acumule correctamente entre todos los hilos. Cada hilo mantiene una copia local de suma_total, y al final del ciclo paralelo, los valores locales de todos los hilos se suman para obtener el valor final de suma_total.

Luego de esto se procede a imprimir los resultados donde se muestran las sumas parciales de los 4 hilos indicados en las instrucciones, así como la suma paralela y la suma secuencial para realizar la comparación. Y los resultados son los siguientes:

Resultado

```
C:\Users\Pablo Orellana\Documents\GitHub\Corto13>g++ -std=c++11 -fopenmp corto13.cpp -o corto13

C:\Users\Pablo Orellana\Documents\GitHub\Corto13>corto13.exe
-------------------------------------------------
|        Sumatoria Parcial de cada Hilo         |
-------------------------------------------------
Hilo 0: Suma del rango (0 a 249) = 31375
Hilo 1: Suma del rango (250 a 499) = 93875
Hilo 2: Suma del rango (500 a 749) = 156375
Hilo 3: Suma del rango (750 a 999) = 218875
-------------------------------------------------
|        Sumatoria Paralela y Secuencial        |
-------------------------------------------------
Suma total calculada en paralelo: 500500
Suma total calculada secuencialmente: 500500
```

```
C:\Users\Pablo Orellana\Documents\GitHub\Corto13>g++ -std=c++11 -fopenmp corto13.cpp -o corto13

C:\Users\Pablo Orellana\Documents\GitHub\Corto13>corto13.exe
-------------------------------------------------
|        Sumatoria Parcial de cada Hilo         |
-------------------------------------------------
Hilo 0: Suma del rango (0 a 24) = 325
Hilo 2: Suma del rango (50 a 74) = 1575
Hilo 1: Suma del rango (25 a 49) = 950
Hilo 3: Suma del rango (75 a 99) = 2200
-------------------------------------------------
|        Sumatoria Paralela y Secuencial        |
-------------------------------------------------
Suma total calculada en paralelo: 5050
Suma total calculada secuencialmente: 5050

C:\Users\Pablo Orellana\Documents\GitHub\Corto13>
```

Enlace Repositorio: https://github.com/JPOrellana/Corto13