Examen Práctico: Patrones de Programación Paralela

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 el proceso de dividir un sistema o proceso complejo en funciones o tareas más pequeñas y manejables. Esto permite la comprensión, el diseño, la implementación y el mantenimiento de sistemas al permitir que cada componente funcional se aborde de manera individual. Por ejemplo, si se está desarrollando una aplicación de comercio electrónico, esta se puede descomponer en las siguientes funciones principales:

- Gestión de usuarios: registro, inicio de sesión, perfil de usuario.
- Catálogo de productos: visualización de productos, categorías, búsqueda.
- Carrito de compras: agregar o eliminar productos, ver el carrito.
- Procesamiento de pagos: integración con pasarelas de pago, validación de transacciones.
- Envíos y entregas: cálculo de costos de envío, seguimiento de pedidos.

La descomposición de datos hace referencia al proceso de dividir conjuntos de datos grandes en partes más pequeñas para facilitar su almacenamiento, procesamiento o análisis. Esto es especialmente útil en sistemas que manejan grandes volúmenes de información o que requieren procesamiento paralelo. La descomposición de datos ayuda a optimizar el rendimiento y la escalabilidad del sistema. Por ejemplo si se tiene una base de datos gigantesca de registros de transacciones financieras, para poder mejorar el rendimiento y facilitar el análisis, se descomponer los datos de la siguiente manera:

- Fragmentación horizontal: dividir la tabla de transacciones en varias tablas más pequeñas basadas en rangos de fechas o regiones geográficas.
- Fragmentación vertical: separar las columnas de la tabla en tablas distintas.
- Distribución en clústeres: repartir los fragmentos de datos en diferentes servidores o nodos para permitir el procesamiento paralelo y reducir la carga en un solo sistema.

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

El paralelismo manual permite al programador tener un control detallado sobre cómo se divide y sincroniza el trabajo en un programa. Esto facilita la optimización específica para el hardware o la arquitectura utilizada y permite explotar oportunidades de paralelismo que las herramientas automáticas podrían pasar por alto. Sin embargo, este enfoque aumenta la complejidad del código, incrementa el riesgo de errores como condiciones de carrera e interbloqueos, y requiere más tiempo y experiencia para su desarrollo y mantenimiento.

Caso contrario, el paralelismo automático simplifica el proceso al delegar en el compilador o el entorno de ejecución la tarea de paralelizar el código. Esto reduce la carga sobre el programador, minimiza los errores de concurrencia y mejora la mantenibilidad del software. No obstante, puede resultar en un rendimiento subóptimo ya que no todas las oportunidades de paralelismo son detectadas automáticamente, y ofrece menos control sobre el proceso de paralización, dificultando la aplicación de optimizaciones específicas.

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.

La fase de fork representa la división de una tarea grande en subtareas más pequeñas que pueden ejecutarse en paralelo. Este enfoque es útil cuando un problema puede descomponerse en partes independientes, permitiendo que cada parte se procese simultáneamente para aprovechar múltiples núcleos de procesamiento. Por ejemplo, en el procesamiento de imágenes, una imagen grande se puede dividir en bloques más pequeños para aplicar un filtro en cada uno de dichos bloques de manera paralela.

Mientras que la fase de join ocurre cuando todas las subtareas han terminado su procesamiento y sus resultados se combinan para obtener el resultado final. Es decir, cuando se terminó de procesar cada bloque por separado, se juntan todos los bloques procesados para formar la imagen completa. Esto permite completar tareas grandes de forma más eficiente y rápida, beneficiándose del procesamiento paralelo.

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

Este patrón de diseño se utiliza en aplicaciones que trabajan con datos organizados en rejillas o mallas, donde el valor de cada punto se actualiza iterativamente en función de los valores de sus vecinos cercanos, siguiendo una plantilla predefinida. Este patrón es común en simulaciones científicas como la modelación de fenómenos físicos, ecuaciones diferenciales, análisis de calor y fluidos, así como en procesamiento de imágenes. Es particularmente útil en problemas donde los cálculos dependen de un entorno local dentro de una estructura espacial.

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.

Lo primero a hacer sería dividir los datos en bloques manejables, por ejemplo, en fragmentos de 1 millón de registros cada uno. En la fase de Map, cada fragmento sería asignado a un nodo de procesamiento que aplicaría una función de mapeo para transformar o extraer información importante de estos datos. Luego de esto se generan los pares clave-valor intermedios que representan resultados parciales. En la fase de Shuffle, estos pares se reorganizan y se agrupan por claves similares para prepararlos para la fase de reducción. Por último, en la fase de Reduce, los nodos toman los grupos de pares clave-valor y combinan los resultados parciales para producir el resultado final.

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.

En una simulación física donde las partículas se actualizan basándose en sus vecinas, el patrón de diseño más adecuado sería Stencil. Este patrón es ideal para situaciones donde el valor de una entidad, como una partícula, depende de los valores de sus vecinos inmediatos. El patrón Stencil permite actualizar los valores de las partículas en paralelo, lo que es particularmente eficiente en simulaciones basadas en cuadrículas o sistemas con interacciones locales entre partículas.

Para dividir el trabajo, organizaría las partículas en subconjuntos, asignando a cada hilo o procesador una región específica del espacio simulado. Las actualizaciones dentro de estas regiones pueden realizarse en paralelo, asegurando que no haya dependencia directa entre los elementos que están siendo procesados al mismo tiempo. Para evitar conflictos en las fronteras de cada región, podría implementarse un sistema de sincronización que actualice las fronteras entre subconjuntos de manera secuencial o con una estrategia de barrera.

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.

En este enfoque, el proceso Maestro divide la lista de números en segmentos y los distribuye entre varios procesos Esclavos. Cada esclavo calcula la suma parcial de su segmento y devuelve el resultado al maestro, que luego suma todas las sumas parciales para obtener el total final.

```
// Proceso Maestro
función maestro(lista_numeros):
    dividir lista_numeros en N segmentos
    para cada esclavo en esclavos_disponibles:
        enviar segmento a esclavo
    sumas_parciales = []
    para cada esclavo en esclavos_disponibles:
        recibir suma_parcial de esclavo
        agregar suma_parcial a sumas_parciales
    suma_total = sumar todos los valores en sumas_parciales
    retornar suma_total

// Proceso Esclavo
función esclavo():
    recibir segmento del maestro
    suma_parcial = sumar todos los números en segmento
    enviar suma_parcial al maestro
```

En este pseudocódigo, el maestro coordina la distribución y recolección de datos, mientras que los esclavos realizan el cálculo intensivo, siguiendo el patrón Maestro/Esclavo para paralelizar la tarea de suma.

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.

El código divide una lista de un millón de números aleatorios en cuatro sublistas, distribuyendo el trabajo entre cuatro procesos. Cada proceso ejecuta una función worker que calcula la suma de los elementos de su sublista y almacena el resultado en una cola

compartida. Esto se logra dividiendo la lista original de forma equitativa entre los procesos mediante el uso de índices calculados según el tamaño de la lista.

La combinación de resultados se realiza recogiendo las sumas parciales desde la cola, donde cada proceso ha puesto su resultado. El proceso principal espera a que todos los procesos secundarios terminen, luego suma los resultados parciales y los compara con el cálculo de suma en un solo hilo para verificar su exactitud.

El código aprovecha el paralelismo para acelerar el procesamiento de grandes cantidades de datos mediante el patrón Fork-Join. Cada proceso calcula una parte de la tarea de forma independiente y, al finalizar, el proceso principal combina estos resultados. El resultado final de la suma en paralelo es idéntico a la suma calculada en un solo hilo, confirmando que el enfoque distribuido es correcto.

Código:

https://github.com/angelcast2002/Corto-13-Paralelas.git