UNIVERSIDAD TECNOLÓGICA DE SANTIAGO, UTESA.

Asignatura:

INF-025-001 ALGORITMOS PARALELOS

Tema:

Tarea Semana 2

Presentado por:

Georges de Jesús Gil Pichardo

Matricula:

1-18-2363

Presentado a:

M.A. IVAN MENDOZA

Procesamiento Paralelo

El procesamiento paralelo es una técnica en informática donde múltiples procesadores o núcleos de CPU trabajan juntos para realizar tareas de manera simultánea. En lugar de depender de un solo procesador para realizar todas las operaciones, el procesamiento paralelo divide la tarea en subproblemas más pequeños que se pueden resolver simultáneamente, lo que lleva a un aumento significativo en la velocidad y eficiencia del procesamiento. Hay varias formas de implementar el procesamiento paralelo:

Paralelismo a Nivel de Tarea: En este enfoque, diferentes tareas o procesos se ejecutan simultáneamente en varios procesadores o núcleos. Esto se puede lograr utilizando técnicas de programación concurrente o mediante el uso de sistemas operativos y planificadores que distribuyen tareas en paralelo.

Paralelismo a Nivel de Datos: En este tipo de paralelismo, una tarea se divide en subproblemas que se procesan simultáneamente. Cada subproblema se procesa utilizando los mismos conjuntos de instrucciones, pero en diferentes conjuntos de datos. Este enfoque se utiliza comúnmente en aplicaciones que involucran grandes conjuntos de datos, como el procesamiento de imágenes y videos, donde los píxeles pueden ser procesados en paralelo.

Paralelismo a Nivel de Instrucción: En este nivel de paralelismo, múltiples instrucciones de un programa se ejecutan simultáneamente en diferentes unidades de ejecución dentro de un procesador. Los procesadores modernos a menudo tienen pipelines y ejecución fuera de orden para lograr el paralelismo a nivel de instrucción.

Computación en Malla o Grid Computing: En este modelo, múltiples computadoras y servidores están interconectados en una red y trabajan juntas para resolver problemas

complejos. Las tareas se dividen y distribuyen a través de la red, y los resultados se combinan para obtener la solución final.

Computación en Clúster: En un entorno de clúster, varias computadoras independientes se agrupan para trabajar como un solo sistema. Cada computadora en el clúster tiene su propio conjunto de recursos, como CPU, memoria y almacenamiento, y las tareas se distribuyen entre estas computadoras para realizar el procesamiento de manera paralela.

Computación en la Nube: La computación en la nube permite acceder a recursos informáticos a través de Internet. Los proveedores de servicios en la nube utilizan el procesamiento paralelo para manejar grandes volúmenes de datos y ofrecer servicios escalables a través de múltiples servidores.

Modelos De Cómputo Paralelos

Existen varios modelos de cómputo paralelo que describen cómo los procesadores o nodos en un sistema paralelo interactúan y realizan operaciones en paralelo. Estos modelos proporcionan abstracciones y reglas para la ejecución de algoritmos en sistemas paralelos. Algunos de los modelos de cómputo paralelos más comunes son:

Modelo PRAM (Parallel Random Access Machine): En el modelo PRAM, se supone que todos los procesadores pueden acceder a la memoria compartida al mismo tiempo. Existen variantes del modelo PRAM, como EREW (Exclusive Read Exclusive Write), CREW (Concurrent Read Exclusive Write), CRCW (Concurrent Read Concurrent Write), que difieren en las políticas de acceso a la memoria. Este modelo proporciona una visión idealizada de la computación paralela, donde las operaciones se realizan en tiempo constante y no hay conflictos de acceso a la memoria.

Modelo BSP (Bulk Synchronous Parallel): En el modelo BSP, la computación se divide en rondas, y cada ronda consiste en una fase de cómputo y una fase de comunicación. Durante la fase de cómputo, los procesadores realizan operaciones en paralelo, y durante la fase de comunicación, se pueden intercambiar mensajes entre procesadores. Este modelo proporciona una estructura ordenada para la sincronización y la comunicación en sistemas paralelos.

Modelo de Memoria Distribuida: En este modelo, cada procesador tiene su propia memoria local y se comunica con otros procesadores a través de mensajes. Los procesadores en este modelo no comparten la memoria física, lo que implica que las comunicaciones se realizan explícitamente mediante el intercambio de mensajes a través de una red de interconexión.

Modelo de Paso de Mensajes: En este modelo, los procesadores se comunican intercambiando mensajes a través de canales de comunicación. Los mensajes contienen datos y órdenes para los procesadores receptores. Este modelo refleja la forma en que los sistemas paralelos distribuidos interactúan en el mundo real.

Modelo de Actores: En el modelo de actores, los elementos de cómputo (llamados "actores") interactúan enviándose mensajes entre ellos. Cada actor tiene su propio estado y procesamiento, y las interacciones ocurren mediante el envío y la recepción de mensajes. Este modelo es utilizado en lenguajes de programación concurrente y paralela.

Modelo de Computación Heterogénea: En este modelo, diferentes tipos de procesadores con características y capacidades variadas trabajan juntos en un sistema paralelo. Esto puede incluir combinaciones de CPUs, GPUs (Unidades de Procesamiento Gráfico) y FPGAs (Arreglos de Compuertas Programables en Campo), aprovechando las fortalezas específicas de cada tipo de procesador para tareas específicas.

Complejidad de la comunicación

Se refiere a la cantidad de datos que necesitan ser transferidos entre procesadores o nodos de un sistema paralelo durante la ejecución del algoritmo. Esta medida es fundamental para evaluar la eficiencia y el rendimiento de un algoritmo en un entorno paralelo. La complejidad de la comunicación puede dividirse en dos aspectos principales:

Ancho de Banda:

El ancho de banda de comunicación se refiere a la cantidad máxima de datos que pueden ser transferidos entre procesadores por unidad de tiempo. Una complejidad de comunicación alta debido a un ancho de banda limitado significa que los procesadores tienen dificultades para enviar o recibir datos rápidamente, lo que puede ralentizar la ejecución del algoritmo paralelo.

Latencia:

La latencia es el tiempo que tarda un mensaje en viajar desde el remitente hasta el receptor. Una complejidad de comunicación alta debido a una latencia significativa implica que hay demoras considerables cada vez que los procesadores intentan comunicarse entre sí. Esto puede afectar negativamente el rendimiento del algoritmo, especialmente en situaciones donde la comunicación frecuente es necesaria.

Desempeño computacional de algoritmos paralelos

El desempeño computacional de algoritmos paralelos se refiere a la medida de eficiencia y velocidad con la que un algoritmo puede resolver un problema utilizando múltiples procesadores o núcleos de CPU en paralelo. Los algoritmos paralelos se utilizan para aprovechar la capacidad de procesamiento de sistemas con múltiples núcleos, clústeres de computadoras o supercomputadoras para resolver problemas complejos de manera más rápida.

Existen varios factores que pueden afectar el desempeño computacional de algoritmos paralelos:

Escalabilidad: La escalabilidad se refiere a la capacidad del algoritmo para manejar un aumento en el número de procesadores sin degradación significativa en el rendimiento. Los algoritmos paralelos deben estar diseñados para escalar eficientemente a medida que se agregan más recursos de hardware.

Balance de Carga: En un sistema paralelo, es esencial distribuir la carga de trabajo de manera equitativa entre los procesadores para evitar que algunos se sobrecarguen mientras otros permanecen inactivos. Un desequilibrio en la carga puede conducir a un su aprovechamiento de los recursos disponibles.

Comunicación y Sincronización: La comunicación entre procesadores y la sincronización de tareas son operaciones costosas en sistemas paralelos. Los algoritmos deben minimizar la necesidad de comunicación y sincronización para evitar el tiempo de espera innecesario.

Granularidad: La granularidad se refiere al tamaño de las tareas que se dividen para la ejecución paralela. Si las tareas son demasiado pequeñas, el costo de la comunicación y la sincronización puede superar el beneficio de la paralelización. Por otro lado, si las tareas son demasiado grandes, algunos procesadores pueden estar inactivos mientras esperan que se completen las tareas más grandes.

Algoritmos y Estructuras de Datos: Algunos algoritmos son inherentemente paralelizables, mientras que otros pueden ser difíciles de paralelizar debido a dependencias de datos complejas. Además, la elección de estructuras de datos adecuadas puede influir en el desempeño de los algoritmos paralelos.

Overhead: Existen costos asociados con la administración y coordinación de tareas en un entorno paralelo. Este overhead puede afectar el desempeño, especialmente para problemas pequeños que no justifican la paralelización.

Optimización

La optimización en el contexto de algoritmos paralelos se refiere a mejorar la eficiencia y el rendimiento de estos algoritmos. Hay varias estrategias de optimización que pueden aplicarse para mejorar el desempeño de algoritmos paralelos, especialmente en términos de la complejidad de la comunicación:

Minimización de la Comunicación:

Reducción de la Cantidad de Datos: Reducir la cantidad de datos que se deben transferir entre procesadores puede ayudar a disminuir la complejidad de la comunicación. Esto puede lograrse mediante técnicas como la computación local y el uso eficiente de estructuras de datos.

Uso de Comunicación Asíncrona: Implementar técnicas de comunicación asincrónica puede permitir que los procesadores continúen su trabajo sin esperar a que todos los demás completen las operaciones de comunicación.

Optimización de la Topología de Red:

Selección de una Topología Eficiente: Elegir una topología de red adecuada para el sistema paralelo puede reducir la latencia y mejorar el ancho de banda de la comunicación. Por ejemplo, las topologías de red en malla o toro suelen ser eficientes para ciertas aplicaciones.

Uso Eficiente de la Memoria Compartida o Distribuida:

Balance de Carga de Memoria: En sistemas con memoria compartida, asegurar un equilibrio en la carga de memoria puede evitar cuellos de botella y mejorar el rendimiento general del sistema.

Optimización del Algoritmo:

Paralelización Efectiva: Diseñar algoritmos que puedan ser paralelizados de manera eficiente, minimizando las dependencias entre tareas, puede mejorar significativamente el rendimiento en sistemas paralelos.

División de Tareas: Determinar la granularidad adecuada para dividir las tareas entre procesadores puede evitar la sobrecarga de comunicación y optimizar el uso de los recursos disponibles.

Uso de Técnicas Específicas de Optimización:

Prefetching y Almacenamiento en Caché: Utilizar técnicas de prefetching y optimizar el acceso a la memoria caché puede reducir el tiempo de acceso a los datos, mejorando así el rendimiento.

Programación Híbrida: Combinar técnicas de programación paralela a nivel de hilo y a nivel de datos (como OpenMP y MPI) puede aprovechar lo mejor de ambos enfoques, optimizando la ejecución en sistemas paralelos complejos.

Monitoreo y Ajuste Dinámico:

Monitoreo del Desempeño: Implementar sistemas de monitoreo para evaluar el desempeño del algoritmo en tiempo real y realizar ajustes según sea necesario.

Ajuste Dinámico: Implementar técnicas que permitan ajustar dinámicamente la distribución de tareas y la comunicación en función del estado actual del sistema puede optimizar la ejecución en entornos cambiantes.