García López Erik
Ramirez Medina Daniel

Algoritmos de ordenamiento paralelos

Los algoritmos de ordenamiento paralelos son una clase de algoritmos que se utilizan para ordenar grandes conjuntos de datos en paralelo, es decir, distribuyendo la carga de trabajo entre múltiples procesadores o núcleos de una computadora o incluso entre múltiples computadoras. Estos algoritmos se utilizan comúnmente en entornos de procesamiento de datos de alta velocidad, como bases de datos y sistemas de procesamiento de transacciones en línea.

En general, los algoritmos de ordenamiento paralelos se dividen en dos categorías principales: los algoritmos basados en comparación y los algoritmos no basados en comparación. Los algoritmos basados en comparación, como Quicksort y Mergesort, comparan elementos en pares y reordenan los elementos en función de sus relaciones de orden. Los algoritmos no basados en comparación, como Counting Sort y Radix Sort, utilizan técnicas diferentes para ordenar los elementos y no comparan los elementos directamente.

A medida que aumenta el tamaño de los conjuntos de datos a ordenar, los algoritmos de ordenamiento paralelos se vuelven cada vez más útiles, ya que permiten procesar grandes cantidades de datos de manera más rápida y eficiente que los algoritmos secuenciales. Sin embargo, el diseño y la implementación de estos algoritmos pueden ser complejos debido a la necesidad de coordinar múltiples procesadores o nodos de computadora para realizar el trabajo de manera eficiente y sin conflictos.

Bitonic Sort

Bitonic sort paralelo es un algoritmo de ordenamiento paralelo basado en comparación que se utiliza para ordenar conjuntos de datos de tamaño potencia de 2. Este algoritmo se basa en la estructura de ordenamiento "bitonic", que es una secuencia de elementos que se ordenan en orden ascendente y luego en orden descendente.

El algoritmo de Bitonic sort paralelo comienza dividiendo el conjunto de datos en dos mitades iguales y ordenando cada mitad en orden ascendente. Luego, se realiza una operación "bitonic merge" en la que las dos mitades se combinan en una secuencia bitónica completa. El proceso se repite recursivamente en cada mitad de la secuencia bitónica, hasta que se obtiene una secuencia completamente ordenada.

La operación de bitonic merge se realiza en paralelo mediante el uso de múltiples procesadores o núcleos de computadora. Cada procesador o núcleo se encarga de comparar y ordenar una subsecuencia de la secuencia bitónica. Durante la

operación de bitonic merge, los elementos se comparan y se intercambian de tal manera que la secuencia resultante siga siendo bitónica.

El rendimiento del algoritmo de Bitonic sort paralelo mejora significativamente con el aumento del número de procesadores o núcleos de la computadora utilizados. Sin embargo, el algoritmo es limitado por el hecho de que el tamaño del conjunto de datos debe ser una potencia de 2, y puede ser menos eficiente que otros algoritmos de ordenamiento paralelo para conjuntos de datos que no son potencias de 2.

Snake sort

Snake sort paralelo es un algoritmo de ordenamiento paralelo que se utiliza para ordenar conjuntos de datos bidimensionales en forma de matriz. Este algoritmo se basa en el algoritmo de ordenamiento secuencial llamado "Snake Sort", que ordena los elementos de una matriz moviéndose en una "serpiente" a través de la matriz.

El algoritmo de Snake sort paralelo comienza dividiendo la matriz en varias secciones rectangulares iguales, cada una de las cuales se ordena utilizando el algoritmo de Snake sort secuencial. Luego, las secciones ordenadas se combinan en una sola matriz ordenada.

Durante la operación de Snake sort paralelo, los procesadores o núcleos de la computadora se utilizan para ordenar cada sección rectangular de la matriz de manera independiente, utilizando el algoritmo de Snake sort secuencial. Luego, las secciones ordenadas se combinan utilizando una operación de "fusionado de serpientes" que imita el movimiento de la serpiente a través de la matriz.

El rendimiento del algoritmo de Snake sort paralelo mejora significativamente con el aumento del número de procesadores o núcleos de la computadora utilizados. Además, el algoritmo es capaz de aprovechar la localidad espacial de los datos en la matriz, lo que puede resultar en una mayor eficiencia en el acceso a memoria y en el rendimiento en general.

Sin embargo, el algoritmo de Snake sort paralelo puede ser menos eficiente que otros algoritmos de ordenamiento paralelo para conjuntos de datos que no son bidimensionales o para matrices con un gran número de filas y columnas. Además, el algoritmo puede requerir una cantidad significativa de comunicación y coordinación entre los procesadores o núcleos de la computadora, lo que puede aumentar la complejidad de la implementación.

El algoritmo es particularmente útil para matrices de gran tamaño que no caben en la memoria caché del procesador.

- Procesamiento de imágenes y video: las matrices de píxeles que representan imágenes y video pueden ser ordenadas con el algoritmo de Snake sort. Esto permite realizar operaciones de procesamiento de imágenes y video más eficientes.
- Simulaciones numéricas: muchas simulaciones numéricas, como simulaciones de dinámica de fluidos, utilizan matrices de gran tamaño para representar datos. El algoritmo de Snake sort puede utilizarse para ordenar estas matrices y mejorar la eficiencia del cálculo.
- Análisis de datos: Se puede utilizar en el análisis de datos en diversas áreas, como la bioinformática y la minería de datos. Por ejemplo, se puede utilizar para ordenar matrices de datos genéticos para identificar patrones y relaciones entre diferentes muestras.
- Procesamiento de señales: Se utiliza en el procesamiento de señales digitales para mejorar la eficiencia del cálculo de algoritmos de filtrado y transformación de señales.

Parallel Quick Sort

El Parallel Quicksort (Ordenamiento Rápido Paralelo) es una variante del algoritmo de ordenamiento rápido (Quick Sort) que utiliza técnicas de paralelismo para mejorar el rendimiento en sistemas multi-core o distribuidos.

El algoritmo de ordenamiento rápido es conocido por su eficiencia en la mayoría de los casos y su complejidad promedio de tiempo de ejecución de O(n log n). Sin embargo, el Quick Sort tradicional es un algoritmo secuencial, lo que significa que utiliza un único hilo de ejecución para ordenar los elementos de la lista.

En contraste, el Parallel Quick Sort divide la lista en subconjuntos más pequeños y luego aplica el algoritmo de Quick Sort a cada subconjunto de manera simultánea utilizando múltiples hilos o procesadores. Estos subconjuntos se pueden procesar en paralelo, lo que permite un mejor aprovechamiento de los recursos disponibles y acelera el proceso de ordenamiento.

La clave para lograr el paralelismo en el Quick Sort es elegir un pivote adecuado y dividir la lista en subconjuntos que se puedan procesar de forma independiente. Una vez que se hayan ordenado los subconjuntos de manera individual, se puede realizar una etapa de fusión o combinación para obtener la lista finalmente ordenada.

García López Erik
Ramirez Medina Daniel

Parallel Bucket Sort

Bucket Sort u “Ordenamiento por Casillero” es un algoritmo bastante fácil de implementar cuando se habla de algoritmos paralelos. Su lógica principal es reservar diferentes partes de una matriz en diferentes buckets, luego ordenarlos al mismo tiempo con otro algoritmo y volver a juntarlos en una gran matriz de resultados.

En el Parallel Bucket Sort, los elementos se distribuyen inicialmente en cubetas o contenedores utilizando múltiples hilos o procesadores de manera paralela. Cada hilo o procesador se encarga de una parte de los elementos y los coloca en las cubetas correspondientes según su valor. Esta distribución paralela de los elementos en las cubetas es una de las características clave que diferencian al Parallel Bucket Sort del bucket sort secuencial.

Una vez que los elementos se distribuyen en las cubetas, cada cubeta se ordena de forma independiente utilizando cualquier algoritmo de ordenamiento adecuado, como el ordenamiento por inserción o el ordenamiento rápido. Esta etapa de ordenamiento de las cubetas se puede realizar de manera paralela, donde cada hilo o procesador trabaja en una o varias cubetas simultáneamente.

Después de que todas las cubetas se hayan ordenado por separado, se realiza una combinación final para obtener la lista ordenada completa. Esta etapa de combinación suele ser secuencial, ya que se necesita un proceso de fusión para juntar las cubetas en el orden correcto.

¿Cuándo conviene paralelizar un algoritmo?

Existen varios casos en los que conviene paralelizar un algoritmo de ordenamiento. Algunos de ellos son:

1. Grandes conjuntos de datos: Si se tiene un gran conjunto de datos que necesitan ser ordenados, el proceso puede llevar mucho tiempo en una sola CPU. La paralelización puede ayudar a acelerar el proceso y reducir el tiempo total de ejecución.

2. Hardware con múltiples núcleos o procesadores: La mayoría de las computadoras modernas tienen múltiples núcleos o procesadores, lo que significa que pueden realizar varias tareas al mismo tiempo. La paralelización de un algoritmo de ordenamiento puede aprovechar esta capacidad para mejorar la eficiencia.

3. Escalabilidad: Si se espera que el tamaño del conjunto de datos aumente en el futuro, es posible que la solución en serie no sea suficiente a largo plazo.

La paralelización permite que el algoritmo de ordenamiento se adapte a conjuntos de datos más grandes en el futuro.

4. Restricciones de tiempo: Si se tiene un límite de tiempo para la finalización del proceso de ordenamiento, la paralelización puede ayudar a acelerar el proceso y cumplir con la restricción de tiempo.

En resumen, la paralelización de un algoritmo de ordenamiento puede mejorar la eficiencia y el rendimiento del proceso de ordenamiento, especialmente cuando se tienen grandes conjuntos de datos, hardware con múltiples núcleos o procesadores, y se espera una escalabilidad futura.

Bibliografía

https://slideplayer.es/slide/3173643/
https://iq.opengenus.org/parallel-merge-sort/
https://www.youtube.com/watch?v=32NLIL_6WJg
https://www.geeksforgeeks.org/bitonic-sort/
http://users.atw.hu/parallelcomp/ch09lev1sec4.html
https://www.smaizys.com/programing/bucket-sort-parallel-algorithm-using-c-openmpi/
http://www.egr.unlv.edu/~matt/teaching/CSC789/assignment6.pdf