Departamento de Matemática y Ciencia de la Computación

Algoritmo paralelo en memoria compartida para determinar si un grafo dirigido es fuertemente conectado

Jorge Aziel Rebolledo Araya
jorge.rebolledo.ar@usach.cl

Algoritmos distribuidos- 22629 2° Semestre 2024
Licenciatura en Ciencia de la Computación

1 Introducción

En este trabajo se aborda la implementación de un algoritmo paralelo en memoria compartida para determinar si un grafo dirigido es fuertemente conectado. Un grafo dirigido se considera fuertemente conectado si existe un camino desde cualquier vértice hacia todos los demás vértices. Para evaluar la conectividad fuerte del grafo, el algoritmo utiliza una estructura de pila dinámica en conjunto con la técnica de búsqueda en profundidad (*Depth-First Search*, DFS) paralelizada. La ejecución del programa se realiza mediante el siguiente comando:

```
./t1.exe k -O < data.txt
```

donde:

- `k` representa el número de hilos a utilizar.
- `O` especifica el modo de ejecución, donde `V` es el modo verboso y `S` el modo silencioso.

El archivo de datos tiene un formato en el cual la primera línea corresponde al número de vértices del grafo (`n`), seguido de una matriz de adyacencia de `n` x `n` elementos que define las conexiones entre vértices. Cada elemento a_{ij} en esta matriz es igual a 1 si existe una arista dirigida del vértice `i` al vértice `j`, y 0 en caso contrario.

El algoritmo se ha diseñado para funcionar en sistemas de memoria compartida, utilizando la biblioteca `pthread` para el manejo de hilos y sincronización mediante un mecanismo de mutex para evitar condiciones de carrera. De este modo, se logra realizar una verificación de conectividad eficiente, aprovechando la capacidad de procesamiento paralelo en sistemas multinúcleo.

2 Procedimiento

El desarrollo de este algoritmo comenzó con la implementación de una versión secuencial del mismo, que sirvió como punto de partida para analizar la solución del problema. Esta versión inicial permitía verificar si un grafo dirigido era fuertemente conectado al recorrer todos sus vértices y comprobar si todos los nodos eran alcanzables desde cualquier nodo del grafo. El enfoque secuencial no aprovechaba las ventajas del paralelismo, lo que limitaba su rendimiento en grafos grandes.

Análisis del Problema

Tras la implementación de la versión secuencial, se procedió a un análisis más profundo del problema, que incluyó:

- **Identificación de cuellos de botella**: Se observó que la operación más costosa del algoritmo secuencial era la exploración de todos los vértices, especialmente en grafos grandes. La necesidad de recorrer todos los nodos y sus conexiones repetidamente incrementaba considerablemente el tiempo de ejecución.
- **Exploración de posibles soluciones paralelas**: Se analizó la viabilidad de paralelizar el algoritmo, con el objetivo de reducir el tiempo de ejecución dividiendo la tarea de exploración entre múltiples hilos. La exploración en profundidad (DFS) resultó ser una operación adecuada para paralelizar, ya que permite dividir el grafo en subgrafos independientes, cada uno procesado por un hilo diferente.

Implementación de la Solución Paralela

Con base en el análisis, se diseñó una solución paralela en memoria compartida que utiliza varios hilos para explorar el grafo de forma concurrente. Para ello:

1. **División del trabajo**: El conjunto de vértices se dividió en bloques que fueron asignados a diferentes hilos. Cada hilo ejecuta una búsqueda en profundidad (DFS) sobre un subconjunto de vértices.
2. **Control de la ejecución**: Se introdujo una variable global, `isConnectedGlobal`, que permite a los hilos finalizar su ejecución tempranamente si se detecta que algún nodo no es alcanzable. Esto optimiza el proceso y evita trabajo innecesario.
3. **Sincronización**: Para garantizar la integridad de los datos, se utilizó un mecanismo de bloqueo (`pthread_mutex`) para proteger las variables compartidas entre los hilos y evitar condiciones de carrera.

Este enfoque paralelizado permite una exploración más eficiente del grafo, aprovechando la arquitectura de memoria compartida para ejecutar múltiples hilos simultáneamente y reducir el tiempo de ejecución, especialmente en grafos grandes. El diseño de la solución también permite controlar el nivel de paralelismo, ajustando el número de hilos según la capacidad del sistema y el tamaño del grafo.

3 Algoritmo

El algoritmo implementado tiene como objetivo verificar si un grafo dirigido es fuertemente conectado utilizando procesamiento paralelo en memoria compartida. La estrategia principal consiste en dividir el trabajo de exploración del grafo entre varios hilos, donde cada uno de ellos ejecuta una versión de la búsqueda en profundidad (*Depth-First Search*, DFS) sobre una porción del grafo. A continuación, se describen las funciones clave del algoritmo: `dfs` y `check_connectivity`.

Función `dfs`

La función `dfs` es responsable de explorar el grafo desde un nodo inicial para determinar si todos los nodos alcanzables desde ese nodo han sido visitados. Cada hilo ejecuta esta función de forma recursiva para realizar un recorrido DFS de manera iterativa utilizando una pila. El pseudocódigo de la función es el siguiente:

```
dfs(node, num_nodes, graph, visited):
    visited[node] = 1
    Para cada nodo i de 0 a num_nodes - 1:
        Si hay una conexión entre node e i y i no ha sido visitado:
            Llamar recursivamente a dfs(i)
```

En este pseudocódigo: - `node` es el nodo actual que se está explorando. - `num_nodes` es el número total de nodos en el grafo. - `graph` es la matriz de adyacencia del grafo. - `visited` es un arreglo que lleva un registro de los nodos visitados.

Función `check_connectivity`

La función `check_connectivity` coordina la ejecución de varios hilos y verifica si el grafo es fuertemente conectado. Cada hilo explora un subconjunto de nodos del grafo y, mediante DFS, determina si todos los nodos en ese subconjunto son alcanzables. Si algún nodo no es alcanzable desde cualquier nodo dentro del subconjunto, se detiene la ejecución de los hilos y se marca el grafo como no fuertemente conexo.

El pseudocódigo de esta función es el siguiente:

```
check_connectivity(data):
    Crear un arreglo local_visited para los nodos visitados
    Para cada nodo en el rango [start, end]:
        Si stop es verdadero, salir
        Inicializar local_visited a 0 para todos los nodos
        Ejecutar dfs(i) para cada nodo i en el rango [start, end]
        Si algún nodo no fue visitado, establecer stop a 1 y salir
```

En este pseudocódigo: - `start` y `end` indican el rango de nodos que cada hilo debe explorar. - `stop` es una variable global que se establece en 1 si algún nodo no es alcanzable.

Este código implementa un algoritmo eficiente y paralelo para verificar la conectividad fuerte de un grafo dirigido, utilizando múltiples hilos en un sistema de memoria compartida. Cada hilo explora una porción del grafo, y el algoritmo determina si el grafo es fuertemente conexo evaluando si todos los nodos son alcanzables desde cualquier nodo dado.

4 Conclusión

En este informe se presentó el desarrollo e implementación de un algoritmo para determinar si un grafo dirigido es fuertemente conectado, utilizando un enfoque paralelo en memoria compartida. Durante la fase de análisis del problema, se exploraron diversas hipótesis y soluciones posibles que podrían haber llevado a una resolución eficiente del mismo.

Una de las principales ideas analizadas fue la utilización del algoritmo de Kosaraju, que aprovecha el grafo traspuesto para identificar componentes fuertemente conexos. Esta estrategia tiene la ventaja de ser intuitiva y eficiente en ciertos contextos, pero no fue implementada en esta solución debido a la necesidad de adaptar el grafo y realizar múltiples recorridos, lo cual podría complicar la paralelización.

Otra hipótesis interesante surgió al comparar el grafo dirigido con redes de Petri, un enfoque que permitió explorar la posibilidad de modelar la conectividad del grafo como un problema de verificación de la "vivacidad" de la red. En este modelo, se planteaba que un grafo fuertemente conexo podría ser representado mediante una red de Petri activa, lo que permitiría verificar si todas las partes del sistema están interconectadas y operativas. Esta idea fue descartada en favor de la implementación directa de un recorrido DFS paralelo, pero sirvió como base para reflexionar sobre diferentes formas de representar y verificar la conectividad de grafos complejos.

Un aspecto clave que se analizó durante el proceso fue la división del trabajo entre los hilos. En un principio, se consideró dividir la matriz de adyacencia de forma vertical o horizontal, asignando diferentes segmentos a los hilos para que realizaran la búsqueda en profundidad de manera paralela. Sin embargo, se concluyó que no tiene sentido partir la matriz de esta manera, ya que la estructura del grafo no se distribuye de forma regular en bloques independientes que permitan un procesamiento paralelo eficiente. La conectividad de un vértice puede depender de otros vértices dispersos por toda la matriz, lo que hace que la partición de la matriz no sea adecuada para paralelizar el recorrido DFS de manera efectiva. Por esta razón, esta estrategia no fue implementada.

Finalmente, tras analizar diversas alternativas, se optó por una solución paralela utilizando DFS, que permitió distribuir el trabajo de exploración de vértices entre varios hilos de forma más eficiente. Cada hilo realiza un recorrido DFS sobre un subconjunto de vértices, sin necesidad de dividir la matriz de

adyacencia. Esta implementación resultó ser una solución adecuada, eficiente y escalable para el problema de determinar si un grafo es fuertemente conexo.

En resumen, la fase de análisis permitió considerar diferentes enfoques, cada uno con sus ventajas y desventajas. La implementación final se centró en la paralelización del algoritmo DFS, lo cual resultó ser una solución eficiente y escalable para el problema planteado.

5 Bibliografía

Carvajal, R. (2024). norm-prog-c .Departamento de Matemática y Ciencia de la Computación, Universidad de Santiago de Chile.