Facultad de Ciencias

Computación Distribuida

PRACTICA 01

Semestre 2024 – 1

Profesor:
Luis Germán Pérez Hernández

Ayudantes:
Daniel Michel Tavera
Yael Antonio Calzada Martín

Autor

Marco Silva Huerta

Edgar Montiel Ledesma

Carlos Cortés

26 de Septiembre de 2023

Algotitmo Dijkstra Distribuido

Forma de compilar

Funcionamiento

Este código implementa el algoritmo de Dijkstra para encontrar el camino más corto en un grafo ponderado con múltiples procesos utilizando MPI

1. Incluye las bibliotecas necesarias, incluyendo ‘¡stdio.h¿‘, ‘¡stdlib.h¿‘, ‘¡limits.h¿‘, ‘¡time.h¿‘ y ‘¡mpi.h¿‘.
2. Define la constante ‘INFINITY‘ como ‘INT MAX‘, que se utiliza para representar una distancia infinita en el algoritmo de Dijkstra.
3. Define una función ‘randomDelay‘ que genera un valor de retraso aleatorio entre 1 y 1000.
4. Define una función ‘minDelay‘ que encuentra el vértice con el retraso mínimo no visitado en el grafo.
5. Define una función ‘printDijkstra‘ para imprimir los resultados del algoritmo de Dijkstra, incluyendo el retraso mínimo y la ruta desde el nodo 0 hasta cada nodo.
6. La función ‘dijkstra‘ implementa el algoritmo de Dijkstra en paralelo. Cada proceso MPI calcula las distancias parciales desde el nodo 0 hasta un subconjunto de nodos. Luego, se sincronizan los resultados entre los procesos para obtener la distancia mínima global.
7. La función ‘main‘ es la entrada principal del programa. Inicializa MPI, obtiene el rango y el tamaño del comunicador MPI, y solicita al usuario el número de nodos para el grafo.
8. Utiliza MPI para transmitir el número de nodos a todos los procesos.
9. Crea una matriz ‘graph‘ para representar el grafo y llena sus valores con retrasos aleatorios. Esto se hace solo en el proceso con rango 0.
10. Utiliza MPI para transmitir la matriz ‘graph‘ a todos los procesos.
11. Llama a la función ‘dijkstra‘ para calcular el camino más corto desde el nodo 0 hasta todos los demás nodos en paralelo.
12. Libera la memoria asignada para la matriz ‘graph‘ y finaliza MPI.

Pseudocódigo del Algotitmo

1. `Inicializar todas las distancias en D con un valor infinito relativo, ya que son desconocidas al principio, exceptuando la de a, qué se debe colocar en 0, pues la distancia de a a si mismo sería 0. C es copia de V`

2. `Para todo vértice i en C se establece [PI]= a.`

3. Se obtiene el vértice s en C tal que no existe otro vértice w en C
   tal que (D[w] < D[s]).
   Para esto se envía un mensaje al nodo correspondiente y se regresa
   un mensaje de respuesta en donde se toma el tiempo y se le asigna a
   su distancia correspondiente. De manera concurrente el nodo destino
   realiza el mismo procedimiento para calcular su distancia a sus nodos
   vecinos que no han sido visitados.

   // En lugar de buscar el vértice con la distancia más corta
   // iterativamente, ahora se utiliza una heap para mantener una lista
   // de vértices no visitados, ordenada por la distancia más corta. Así
   // encontrar el vértice n la distancia más corta en tiempo logarítmico.

4. Se elimina de C el vértice s. El vértice u se elimina del conjunto C.

5. Para cada arista e en E de longitud l,
   que une el vértice s con algún otro vértice t en C,
   Para cada arista que sale del vértice u, se verifica si la distancia
   a través del vértice u es menor que la distancia actual del vértice t.

   - Si l + D[s] < D[t], entonces:
     // Si la distancia a través del vértice u es menor que la distancia
     // actual del vértice t, entonces se actualiza la distancia del vértice t.
   - Se establece D[t] := l + D[s].
     // La distancia del vértice t se establece en la suma de la distancia
     // del vértice u y el peso de la arista.
   - Se establece P[t] := s.
   // El predecesor del vértice t se establece en el vértice u.

6. Se regresa al paso 4.
   // El algoritmo regresa al paso 4 y repite el proceso hasta que todos
   // los vértices hayan sido visitados.

Desarrollo