**Parallelize Dijkstra’s Algorithm for finding the Shortest Path**

**Raghu Gudipati (36521939)**

**Snehal Gandham (79007757)**

**Problem Statement**

Given a Graph G = (V, E) and Source s ∈ V, compute all distances δ(s, v), where v ∈ V .

**What is Dijkstra’s Algorithm???**

Dijkstra’s algorithm solves the single source shortest-path (SSSP) problem from a given vertex to all other vertices in a graph. Dijkstra’s algorithm is used over directed graphs with non-negative weights. The algorithm identifies two types of vertices: (1) Solved and (2) Unsolved vertices. It initially sets the source vertex as a solved vertex and checks all the other edges (through unsolved vertices) connected to the source vertex for shortest-paths to the destination. Once the algorithm identifies the shortest edge, it adds the corresponding vertex to the list of solved vertices. The algorithm iterates until all vertices are solved.

**Time Complexity:**

Dijkstra’s algorithm achieves a time complexity of O(n2). One advantage of the algorithm is that it does not need to investigate all edges. This is particularly useful when the weights on some of the edges are expensive. The disadvantage is that the algorithm deals only with non-negative weighted edges.

**Dijkstra’s Sequential:**

**Dijkstra sequential implementation:**

 The input data is a source vertex, s and a graph G. The vertices are split into two sets: S, the set of settled vertices, for which the shortest path distance d(s,u),u ∈ S and shortest path π(s,u),u ∈ S is known, and Q, the set of unsettled vertices, for which we only have a current best estimate of the shortest path. Initially all vertices are unsettled. A pseudo-code representation of the algorithm is given here. d(s,u) is represented as an element in an array of costs, d[u]. The path, π(s,u), is represented as a linked list of references to previous vertices, each element being in previous[u]. The edge weights are represented as w[u,v], giving the weight for the edge between vertices u and v. The algorithm proceeds as follows:

V = set of all vertices

 for each v in V:

d[v] = infinity

previous[v] = undefined

d[s] = 0 S = empty

set Q = V

while Q is not an empty set:

u = extract-min(Q)

S = S union {u}

for each edge (u,v) incident with u:

if d[v] > d[u] + w[u,v]:

d[v] = d[u] + w[u,v]

previous[v] = u

At each step, the unsettled vertex with the shortest distance to the source is made settled as the best path has been found to it. The vertices adjacent to that one is then tested to see if there is a path via this newly settled vertex to the source that is shorter than their previous shortest. If so, the best-known distance and previous reference are updated. This is known as the relaxation step - it is like the end of a piece of elastic (vertex v) relaxing back into place after the other end (vertex u) has been pulled away (by vertex v being notified that there is a shorter path via u).

**Dijkstra sequential implementation:**

The parallel implementation works the same way as sequential implementation. The only difference is that OpenMP has been used for parallelization. The following terms has been used for parallelization.

* #pragma omp parallel – This is the keyword from where the threads are generated and

parallelization starts.

* omp\_get\_wtime - This has been used to calculate the time from the start to the end of

“#pragma omp parallel”.

* #pragma omp parallel - The threads have been set using this command.

for num\_threads

(“Number of threads”)

**Results:**

The graphs have been plotted by varying the number of vertices:

**The results achieved are as follows:**

**Serialized Code:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # of Threads | 20000 Vertices | 5000 Vertices | 1000 Vertices | 500 Vertices |
| 1 | 34176.558 | 1975.273 | 81.781 | 1.2 |

**Parallelized Code:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # of Threads | 20000 Vertices | 5000 Vertices | 1000 Vertices | 500 Vertices |
| 2 | 34885.553 | 1581.93 | 71.25 | 2.8 |
| 3 | 36447.935 | 1394.136 | 67.18 | 3.4 |
| 4 | 32164.298 | 1273.711 | 65.47 | 4.83 |
| 5 | 15941.969 | 1352.152 | 66.24 | 4.91 |
| 6 | 15425.874 | 1270.552 | 70.03 | 6.15 |
| 7 | 15946.046 | 1235.408 | 72.17 | 8.84 |
| 8 | 15866.751 | 1266.826 | 76.33 | 8.59 |
| 9 | 15987.106 | 1260.062 | 78.62 | 8.3 |
| 10 | 15945.235 | 1238.173 | 80.32 | 7.26 |