Parallel Implementation of Graph Algorithms

Group 27

Bhagyashri Bhamare	(181IT111)
Chinmayi C. Ramakrishna	(181IT113)
K. Keerthana	(181IT221)
Utkarsh Meshram	(181IT250)

Introduction

- The **Single-Source Shortest Path** (SSSP) problem consists of finding the shortest paths between a given vertex v and all other vertices in the graph.
- Four algorithms for parallel implementation of SSSP problem : Dijkstra's algorithm, Bellman Ford algorithm, Floyd Warshall algorithm and Prim's algorithm.
- Parallel implementation of these algorithms using OpenMP.
- Dijkstra’s algorithm is a graph search algorithm that solves single-source shortest path for a graph with non-negative weights.
- The Floyd Warshall is a classic dynamic programming algorithm that solves the all-pairs shortest path (APSP) problem on directed weighted graphs.Floyd Warshall can be applied to graphs with negative weight edges to determine whether the graph has negative cycles or not.
- Prim's algorithm is a popular greedy algorithm that finds a minimum spanning tree for a weighted undirected graph.

Problem Statement

Solving Single Source Shortest Path Problem using:

- ❑ Dijkstra's algorithm
- ❑ Bellman Ford algorithm
- ❑ Floyd Warshall algorithm
- ❑ Prim's algorithm

Objectives

- ❑ To parallelise the Shortest Source Path Problem using four algorithms: Dijkstra's algorithm, Bellman Ford algorithm, Floyd Warshall algorithm and Prim's algorithm.
- ❑ To compare sequential and parallel execution of each algorithm.
- ❑ To compare the algorithms.
- ❑ To find the best algorithm for different size of graphs.

Literature Survey

S. No	References	Work Done
1.	F. Busato and N. Bombieri An efficient implementation of the Bellman-Ford algorithm for Kepler GPU architectures.	Use of frontier data structure to implement Bellman-Ford algorithm.
2.	H. Ortega-Arranz et. al. A new GPU-based approach to the shortest path problem.	Parallelizing internal working of sequential Dijkstra algorithm.
3.	Vladimir Loncˇar, Srdjan Škrbic´and Antun Balazˇ Parallelization of Minimum Spanning Tree Algorithms Using Distributed Memory Architectures	Parallelizing Minimum Spanning Tree using Prim's algorithm and Kruskal algorithm.
4.	An Implementation of Parallel Floyd-Warshall Algorithm Based on Hybrid MPI and OpenMP	Parallelizing Floyd-Warshall Algorithm to exploit the parallelism inside a multi-core node computer.

Serial Implementation

Dijkstra’s Algorithm

```
function Dijkstra(Graph, source):
     for each vertex v in Graph:
          dist[v] := infinity
          previous[v] := undefined
     dist[source] := 0
     Q := the set of all nodes in Graph
     while Q is not empty:
          u := node in Q with smallest dist[ ]
          remove u from Q
          for each neighbor v of u:
               alt := dist[u] + dist_between(u, v)
               if alt < dist[v]
                    dist[v] := alt`
                    previous[v] := u
     return previous[ ]
```

Bellman Ford Algorithm

```
function bellmanFord(G, S)
  for each vertex V in G
    distance[V] <- infinite
      previous[V] <- NULL
  distance[S] <- 0
  for each vertex V in G
    for each edge (U,V) in G
      tempDistance <- distance[U] + edge_weight(U, V)
      if tempDistance < distance[V]
        distance[V] <- tempDistance
        previous[V] <- U
  for each edge (U,V) in G
    If distance[U] + edge_weight(U, V) < distance[V}
      Error: Negative Cycle Exists
   return distance[], previous[]
```

Floyd Warshall Algorithm

```
function FloydWarshall(Ak,n):
     n = no of vertices
     A = matrix of dimension n*n
     for k = 1 to n
          for i = 1 to n
                  for j = 1 to n
                        A[i, j] = min (Ak-1[i, j], Ak-1[i, k] + Ak-1[k, j])
     return A
```

Prim’s Algorithm

```
function Prim:
        T = ∅;
    U = { 1 }
    while (U ≠ V)
        let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U
                T = T ∪ {(u, v)}
                U = U ∪ {v}
```

Parallel Implementation

Dijkstra’s Algorithm

```
#pragma omp parallel for schedule(runtime) private(i)
          for(i = 0; i < V; i++)
                    if(vertices[i].visited == FALSE)
                              int c = findEdge( u, vertices[i], edges, weights);
                              len[vertices[i].title] = minimum(len[vertices[i].title],
len[u.title] + c);
```

Bellman Ford Algorithm

```
function bellmanFord:
            omp_set_num_threads(p)
      while(!queue.empty() and no negative cycle):
            u <- queue.front()
            queue.pop()
            In_queue[u] = false
      #pragma omp parallel for
           for (int v = 0; v < n; v++):
               weight = mat[u * n + v];
               if (weight < INF):
                   new_dist = weight + dist[u];
                   if (new_dist < dist[v]):
                       dist[v] = new_dist;
                       enqueue_counter[v]++;
                       if (in_queue[v] == false)
                           in_queue[v] = true;
                           if (enqueue_counter[v] >= n)
                               *has_negative_cycle = true;
      #pragma omp critical
                        queue.push(v)
```

Floyd Warshall Algorithm

```
function FloydWarshall(Ak,n):
     for (nthreads = 1; nthreads <=10 ;nthreads++) //Parallel Region
     omp_setnum_threads(nthreads);
     #pragma omp parallel
          for (k = 0 ; k <N; k++)
                    dm = Ak[k]
              #pragma omp parallel
                    for (i=0 ; i<N ; i++)
                              ds = Ak[i]
                              for (j=0; j<N;j++)
                                 ds[j] = min (ds[j],ds[k]+dm[j])
```

Prim’s Algorithm

```
function minKey(key[], visited[]):
     min = INT_MAX, index, i;
     #pragma omp parallel
          index_local = index;
          min_local = min
     #pragma omp for nowait
          for (i to n)
               if (visited[i] == false and key[i] < min_local)
                    min_local = key[i]
                    index_local = i
     #pragma omp critical
          if (min_local < min)
          min = min_local
          index = index_local
     return index
```

Individual Contribution

Bhagyashri Bhamare (181IT111)	Bellman Ford algorithm
K. Keerthana (181IT221)	Dijkstra's algorithm
Utkarsh Meshram (181IT250)	Floyd Warshall algorithm
Chinmayi C. Ramakrishna (181IT113)	Prim's algorithm