National University of Computer & Emerging Sciences

Parallel and Distributed Computing

21i-0572 Kissa Zahra
Section: Z

The Traveling salesman problem (TSP) involves finding the shortest possible route that visits a set of cities and returns to the starting city, without visiting any city more than once. Our **main goal** is to find the shortest path and return to the original destination.

Sequential approach:

In the sequential version of the TSP algorithm, we used BFS to visit all possible paths from the starting city to all other cities and return to the starting city. The algorithm involves the following steps:

1. Choose a starting point.
2. Apply the BFS algorithm to find the shortest path.
3. During BFS traversal, explore neighboring nodes to find the shortest path, making sure no node is visited more than once.
4. After visiting all nodes, add up the edge connecting the last node with the starting node to complete the tour.

Output of the sequential:

```
kissa@kissasium:~$ gcc q1.c -o q1
kissa@kissasium:~$ ./q1
Shortest Path from node 0:
Node 0 -> Node 1 -> Node 2 -> Node 3 -> Node 4 -> Node 5 -> Node 6 -> Node 7 -> Node 8 -> Node 9 -> Node 0
Total cost of visiting all cities keeping starting and ending city same: 18
```

parallelized approach:

In a parallel approach we divide our graph among the number of processes that we are creating (e.g: if there are 3 processes so we are dividing the 10 nodes among them giving process 0 -> 3 cities, process 1-> 3 cities and process 2 -> 4 cities).
Every process applies a BFS algorithm on its data set and finds the shortest path. After all the processes have executed parallelly we call MPI_reduce() function to add up the shortest path of all the processes to get the OVERALL shortest path. At last we need to connect the last node with the starting node to return to the original position.

```c
// adding them all
int total_shortest_path;
MPI_Reduce(&total_cost, &total_shortest_path, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
```

Output of the parallel:

```
Total cost: 147
```

BFS code used in both sequential and parallel:

```
//bfs
void BFS(int start, int end, int* distance, int* parent)
{
    int visited[N] = { 0 };
    initializeQueue();

    for (int i = 0; i < N; ++i)
    {
        distance[i] = INT_MAX;
        parent[i] = -1;
    }

    distance[start] = 0;
    parent[start] = start;
    visited[start] = 1;
    enqueue(start);

    while (!isQueueEmpty())
    {
        int current = dequeue();
        for (int i = 0; i < N; ++i)
        {
            if (graph[current][i] && !visited[i])
            {
                visited[i] = 1;
                distance[i] = distance[current] + 1;
                parent[i] = current;
                enqueue(i);
            }
        }
    }
}
```

Performance metrics:

Parallel approach is better than sequential as it takes less time and at the same time utilizes more resources as many processes are executing at same time making a bigger problem divided into subproblems to be solved. It makes the overall process a lot faster and more efficient. But there is one drawback with parallel implementation and that is **communication overhead.** If there is a bigger graph with millions of nodes and if we divide it among the processes the communication will become very hard to manage and not efficient.