Parallel & Distributed Computing |CS-3006

PROJECT REPORT

Submitted by: Muhammad Yahya,
Muhammad Omer Nasir
Roll#: 21I-2592, 21I-2476

Course Instructor: Mr. M. Farrukh Bashir
Submitted on: Sunday, April 28, 2024
Section: BCS-D

Report: Top K Shortest Path Problem with MPI and OpenMP

Introduction: The goal of this project was to implement a parallel algorithm to find the top K shortest paths in a graph using MPI (Message Passing Interface) and OpenMP (Open Multi-Processing) in C. Two versions of the code were developed: one without optimization and one with optimization using parallelization techniques.

Challenges Faced:

1. **Preprocessing:**
    - Reading the graph from a CSV file posed challenges in parsing the file efficiently while handling memory allocation for nodes and edges dynamically.
    - Identifying and handling node duplicates efficiently was crucial to ensure accurate representation of the graph.
2. **Implementation:**
    - Implementing Dijkstra's algorithm for finding the top K shortest paths required careful consideration of the data read in the adjacency list. Making sure that the data is correctly pre-processed for the parameters of `findKShortest` function.
    - Integrating MPI and OpenMP for parallel processing introduced complexities in data distribution and synchronization.
    - Ensuring load balancing and avoiding race conditions in parallel regions were challenges in parallelization.
3. **Testing:**
    - Testing the correctness and efficiency of the parallel algorithm required rigorous experimentation and validation against sequential implementations.
    - Debugging issues related to incorrect path calculations or synchronization errors in the parallelized code was time-consuming.

Optimizations Applied:

1. **Memory Management:**
    - Implemented dynamic memory allocation to efficiently handle large graphs without overcommitting memory.
    - Employed strategies to avoid memory leaks and ensure proper deallocation of resources.
2. **Parallelization:**
    - Utilized OpenMP directives to parallelize critical sections of the code, such as modified Dijkstra's algorithm for finding shortest K paths for random 10 pairs of nodes.
    - Implemented OpenMP for parallelizing loops using the `#omp parallel for` directive withing each MPI process.

- Implemented MPI to enable distributed computing, allowing multiple processes to collaborate in finding shortest paths.

Experimental Results:

- **doctorwho.csv:**
  - Sequential Execution Time:

```
Maximum Number of Nodes: 694
Maximum Number of Edges: 7065

Average time taken: 22.264579 seconds
/project $
```

  - Parallel Execution Time (MPI + OpenMP):

```
Average time taken: 1.062625 seconds
/project $
```

- **new-who.csv:**
  - Sequential Execution Time:

```
Maximum Number of Nodes: 335
Maximum Number of Edges: 3288

Average time taken: 3.326126 seconds
/project $
```

  - Parallel Execution Time (MPI + OpenMP):

```
Average time taken: 0.161630 seconds
/project $
```

- **classic-who.csv**
  - Sequential Execution Time:

```
Maximum Number of Nodes: 377
Maximum Number of Edges: 3793

Average time taken: 2.304142 seconds
/project $
```

  - Parallel Execution Time (MPI + OpenMP):

```
Average time taken: 0.149388 seconds
/project $
```