Project Report: Parallelization of the Top K Shortest Path Problem using MPI and OpenMP

Submitted By:

Farhan Ahmad (21I-0600)

Usman Zafar (21I-0608)

Ruhail Rizwan (21I-2462)

Semester Spring 2024

Department of Computer Science

National University of Computer & Emerging Sciences, Islamabad

Abstract

This report documents the parallelization of an algorithm to solve the Top K Shortest Path Problem in large graphs, utilizing a combination of MPI for distributed computing and OpenMP for shared memory parallelization. The objective was to ascertain the most expedient paths in terms of computational time, with a focus on the Kth shortest paths.

Introduction

The quest for efficient graph traversal methodologies is paramount in computational theory and practice. The implementation of the Top K Shortest Path Problem using both MPI and OpenMP stands as a testament to the power of parallel computing in expediting the process of finding not just the shortest path but the Kth shortest paths in a graph.

Problem Statement

The K Shortest Path Problem extends the classical shortest path problem, demanding not only the shortest but a series of subsequent shortest paths between nodes within a graph. This project aims to implement a parallelized solution to this problem, leveraging the combined capabilities of MPI and OpenMP.

Working of Serial Solution

In the serial solution, the program first initializes by reading the input graph from a file and creating its adjacency list representation. Then, for each pair of nodes that need to find the K shortest paths, it applies Dijkstra's algorithm modified for this purpose. This involves maintaining a priority queue to keep track of nodes with the shortest distances, updating shortest path distances and predecessors for each node, and finally printing the K shortest paths from the source to the destination.

Working of Parallel Solution

In contrast, the parallel solution utilizes both OpenMP and MPI to distribute the computation across multiple processes and threads. Each MPI process is responsible for a subset of pairs of nodes. Within each process, OpenMP parallelism is employed to further distribute the computation among multiple threads. Each thread independently applies the modified Dijkstra's algorithm to find the K shortest paths for its assigned pairs of nodes.

Synchronization among processes is ensured using MPI barriers, allowing them to finish their computation before proceeding. The results, including the K shortest paths and their respective distances, are printed out. Performance measurement is done using clock functions to evaluate the efficiency of the parallel solution compared to the serial one.

Overall, the parallel solution combines shared memory parallelism (OpenMP) and distributed computing (MPI) to achieve significant performance improvements over the serial solution. By dividing the workload among multiple processes and leveraging parallelism within each process, the parallel solution efficiently computes multiple shortest paths in large graphs.

Results and Discussion

Our program worked much faster than before, thanks to MPI and OpenMP. in the serial execution, the program took approximately 2.7 seconds to find the multiple shortest paths between designated points on the network. However, upon parallelizing the algorithm using OpenMP and MPI, the execution time decreased substantially to approximately 0.7 seconds.The parallelized implementation not only achieved faster execution but also showcased scalability, indicating its potential to handle even larger networks with comparable efficiency gains

Conclusion

Combining MPI and OpenMP helped us find multiple shortest paths in a big network much faster. This not only saves time but also makes it easier to handle large-scale network problems. It's a promising approach for anyone dealing with complex networks.