

**FAST School of Computer and Emerging Sciences**

**NATIONAL UNIVERSITY OF COMPUTERS AND EMERGING SCIENCES**

**ISLAMABAD CAMPUS**

**Parallel and Distributed Computing – Spring-2024**

CS-3006

# Final Project Report

**Top K Shortest Path Problem with MPI and OpenMP**

## PreProcessing

The input files were in different format so we first had to convert the files into specific format, for the doctor who file which was a csv file, the file was read into a python program and then mapped to integers, these mapped values and their keys were also stored, and the updated file was used, for the other two datasets the file each row had two values and no weights so weights were added into the files, weights for all the edges were assumed to be 1 in our case.

## Implementation

For the implementation we followed the same approach as that of the link state algorithm used for networking, all the nodes had their own information on certain nodes, and all the processes are provided with unique nodes. The algorithm start with starting value in priorityQueue(PQ), its neighbors are determined and their costs are sent to the respective owners(processes), the processes receive the information about their nodes and updated the information where needed, whenever there is an update the node is pushed into PQ as it may lead to a shorter path than one determined, the code only stops when all the processes have no values in PQ, as even if one the PQ is empty it may receive another node from its nodes in later iterations, all processes go through equal number of iterations, and thus the final path is obtained, this algorithm basically calculates single source to all node shortest paths, and when the algorithm stops all the nodes have their shortest path stored in their respective parents.

## Testing

For Testion we initially used the graph provided in the GFG, it was a small graph so it was easier to debug the solution and identify any errors, when we were sure that the algorithm is working correctly for the smaller gives we used the dataset provided and compared it with the shortest paths provided by the serial method to confirm our solution.

## Optimizations

### openMPI

For optimization we had two options for the algorithm, one was to take 10 pairs, distribute these pairs among the processes and find the shortest path, the other one was to improvise the link state algorithm and obtain the result. The latter seemed more interesting as it was actually parallelizing the algorithm and not the nodes. E.g. if the number of pairs of nodes was 1 and the graph was really large where only the master node had the capacity to contain the whole matrix, this solution works the best as the graph is distributed between nodes, however for a small graph with many pairs the other technique works better, we chose Scattering the matrix to all nodes as it seemed more challenging,

### openMP

For parallelizing using threads we used OPENMP, where each process distributed the loops into multiple threads, in our case thread number was defined to be 10 and to avoid race conditions we also used critical section, and scheduling for the loops was dynamic for all the loops.

## Results

As the whole graph resulted in sparse adjacency matrix, it took a lot of time to iterate over useless values, however implementing a vectorized approach in C was much harder (cpp wasn't allowed), so our machine only works for the small graph of doctor who which has about 8000 edges and around 700 nodes for other datasets we need a better machine or an actual distributed environment.

### Speedup On the doctor who dataset where k value is 2

Time for serial execution: 0.1 sec

Time for parallel execution: 5 sec

Speedup = Serial Time / Parallel Time

= 0.1 / 5

= 0.02

The scalability of the parallelized solver appears to be constrained, evident from the limited speedup observed. It implies that the overhead incurred by MPI-based parallelization outweighs the potential gains, particularly for smaller problem instances. Another reason for this difference could be that the serial solution is working on the adjacency list and the parallel solution on the adjacency matrix.