

# Introduction

The report is titled "**Finding Kth Shortest Path Length**" and is authored by *Waqas Khan*, *Luqman Ansari*, and *Huzaifa Khalid*. It was written during the Spring 2023 semester for the Parallel and Distributed Computing course at FAST NUCES.

# Abstract

This study investigates the parallelization of graph shortest path algorithms using a combination of MPI and OpenMP in a distributed computing environment. The project aims to evaluate the effectiveness of parallelization in reducing execution time and improving performance across different datasets and processing configurations. Through experimentation and analysis, the study explores the impact of increasing the number of processes on execution time, speedup, and the scalability of parallel implementations. The findings reveal varying degrees of speedup achieved across datasets, with significant improvements observed in some cases but diminishing returns in others. Challenges such as load imbalance, synchronization overhead, and limited parallelism are identified as factors influencing parallel efficiency, particularly in operations involving priority queues. Despite these challenges, the study demonstrates the potential of parallelization to optimize computation times and handle larger workloads efficiently. The insights gained from this project contribute to the understanding of parallel algorithm design and provide valuable guidance for optimizing performance in distributed computing environments.

# Distribution of the Task using MPI

**Initialization and Setup**

MPI is initialized, establishing communication channels among processes.Each process is assigned a unique identifier within the MPI communicator.

**Random Edge Selection**

The program selects a subset of edges randomly from the graph. These randomly selected edges are stored in the **randomEdges** array.

**Division of Work**

The randomEdges array is divided among processes to distribute the workload evenly.The total number of edges in randomEdges is divided by the number of processes to determine the portion of edges each process will handle.

**Balanced Distribution**

If the number of edges in randomEdges is not evenly divisible by the number of processes, the remainder is distributed among the first few processes to ensure all edges are processed.

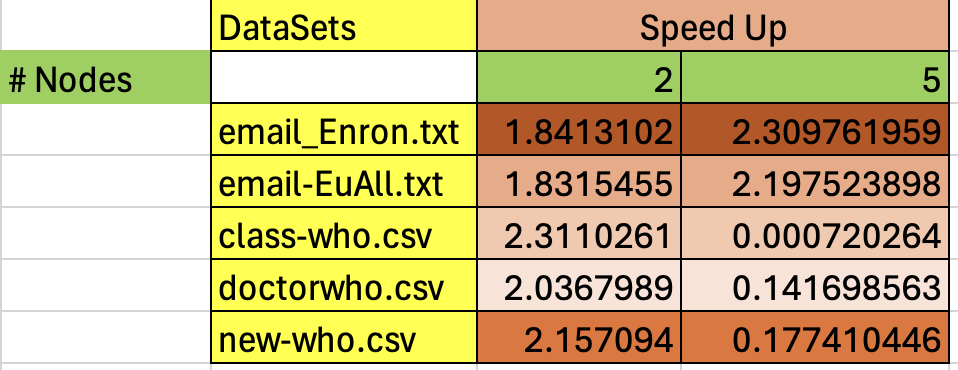
**Local Computation**

Each process independently computes the k shortest paths for the subset of edges it has been assigned.

The findKShortest function is invoked for each edge in the local subset to compute the k shortest paths.

A screenshot of a computer

Description automatically generated



1. **Insights**

**Increasing Execution Time with Increasing Processes**

As the number of processes increases from 1 to 5, the execution time generally decreases across all datasets.This reduction in execution time indicates that parallelizing the computation using multiple processes leads to improved performance, as the workload is distributed among more processing units.

**Varying Speedup Across Datasets**

The degree of speedup achieved by increasing the number of processes varies depending on the dataset.For example, in the case of "email-Enron.txt" and "email-EuAll.txt", the speedup from 1 to 5 processes is significant, with a notable reduction in execution time.

However, for datasets like "class-who.csv", "doctorwho.csv", and "new-who.csv", the speedup is less pronounced, indicating that the parallelization benefits may be limited due to factors such as dataset size, computation complexity, or communication overhead.

**Effectiveness of Parallelization**

The data suggests that parallelization is effective in reducing execution time, particularly for larger datasets or computationally intensive tasks.

The observed reduction in execution time demonstrates the scalability of the parallel implementation, as it is able to efficiently utilize multiple processing units to handle larger workloads.

1. **Challenges in Parallelizing Graph K-Shortest Path Algorithm in Distributed Environments**

**Sequential Operations**

Priority queues typically rely on sequential operations, such as insertion, deletion, and traversal, to maintain their order. In the context of finding shortest paths, these operations are essential for exploring neighboring nodes in order of their distances from the source node.

**Data Dependency**

The order in which elements are inserted or removed from the priority queue directly impacts the outcome of the shortest path algorithm. As a result, concurrent access or modification of the priority queue by multiple processes can lead to data races, inconsistencies, or incorrect results.

**Synchronization Overhead**

To parallelize the code using a distributed environment like MPI, processes would need to coordinate their access to the priority queue to avoid data corruption or race conditions. This necessitates synchronization mechanisms, such as locks or barriers, which introduce overhead and may limit scalability.

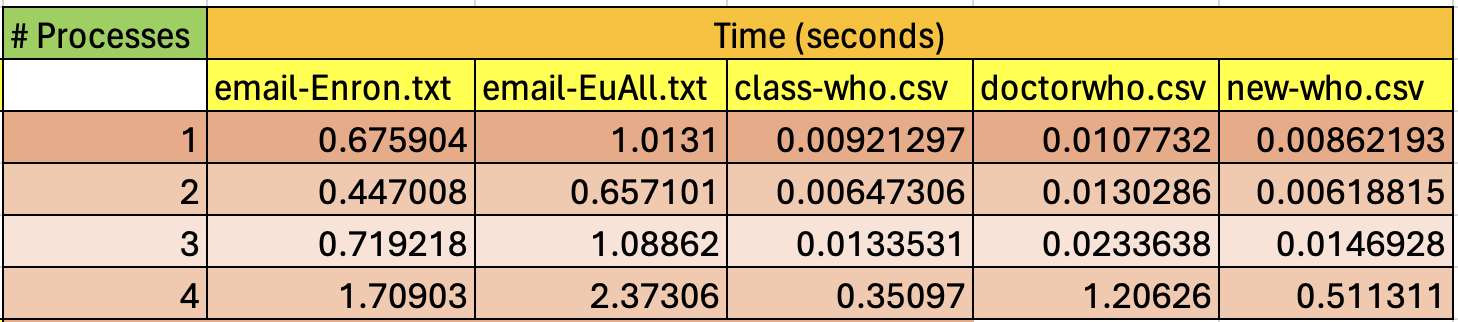
**Load Imbalance**

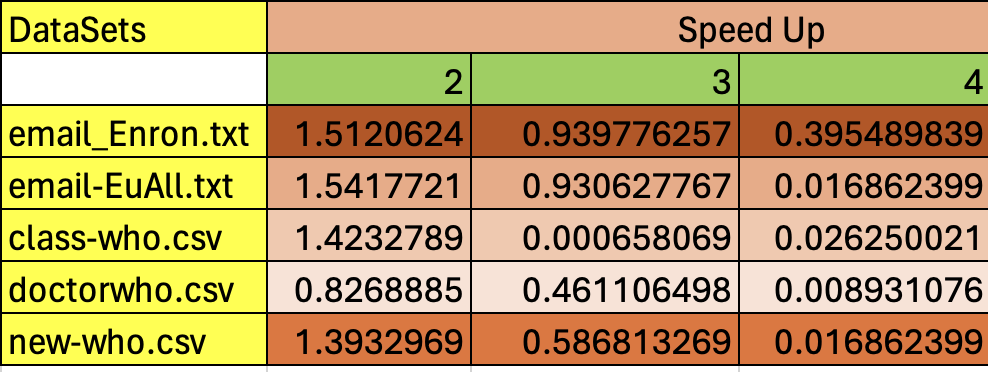
Distributing the workload among multiple processes while maintaining the integrity of the priority queue can be challenging. Load imbalance may occur if certain processes have more work to do (e.g., processing nodes with higher degrees) or encounter longer execution times due to contention for shared resources.

**Limited Parallelism**

The sequential nature of priority queue operations inherently limits the degree of parallelism that can be achieved. While some parallelization may be possible, especially in certain phases of the algorithm, the overall parallel efficiency may be constrained by the sequential bottleneck imposed by the priority queue.

# Distribution with Parallelism of the Task using MPI and OpenMP





1. **Insights**

**Impact of Increasing Processes**

For some datasets like "email-Enron.txt" and "email-EuAll.txt," the execution time initially decreases as the number of processes increases from 1 to 2. However, further increasing the number of processes to 3 or 4 results in increased execution times.

Conversely, for datasets like "class-who.csv," "doctorwho.csv," and "new-who.csv," the execution time consistently increases as the number of processes increases.

**Optimal Parallelization Point**

The optimal number of processes varies depending on the dataset. In some cases, the execution time is minimized with a small number of processes (e.g., 1 or 2), while in others, a higher number of processes may lead to better performance.

**Impact of Workload and Parallelization Strategy**

The effectiveness of parallelization depends on factors such as the size and complexity of the dataset, as well as the efficiency of the parallelization strategy (combining MPI and OpenMP).

For datasets with smaller or less complex graphs, parallelization may not provide significant benefits, and increasing the number of processes beyond a certain point may introduce overhead and degrade performance.

# Conclusion

In conclusion, parallelizing the computation using multiple processes generally leads to improved performance, as evidenced by the reduction in execution time across all datasets as the number of processes increases. However, the degree of speedup varies depending on the dataset, with some datasets exhibiting significant reductions in execution time, while others show less pronounced improvements. The observed effectiveness of parallelization underscores the scalability of the parallel implementation, although challenges such as load imbalance, synchronization overhead, and limited parallelism, particularly in operations involving priority queues, may constrain overall parallel efficiency. Despite these challenges, the data demonstrates the potential of parallelization to efficiently handle larger workloads and optimize computation times, particularly for datasets with complex graphs or computationally intensive tasks.

**References**

<https://www.geeksforgeeks.org/1st-to-kth-shortest-path-lengths-from-node-1-to-n-in-given-graph/>

https://www.geeksforgeeks.org/parallel-dijkstras-algorithm-sssp-in-parallel/

https://repository.stcloudstate.edu/cgi/viewcontent.cgi?article=1044&context=csit\_etds