PDC

Hassan Nawaz 21I-2993
Abdul Wahab 21I-0675
Hassan Raza 21I-0465

Date of Submission: 22/03/2024

FAST NUCES, Islamabad
Department of Computer Science

1. Introduction

Objective: The main goal is to evaluate the performance improvements achieved by parallelizing a serial algorithm for finding the K shortest paths in a graph using MPI (for distributed memory systems) and OpenMP (for shared memory systems).

2. Methodology

- **Code Overview**: The provided code is a C++ program that implements a serial version of the K shortest paths algorithm. It also includes a parallel version that utilizes MPI for process communication across a distributed system and OpenMP for parallel execution within a single node.
- **Parallelization Strategy**:
    - **MPI**: The code uses MPI to distribute the graph's nodes and edges across multiple processes, allowing each process to work on a subset of the graph. It uses collective communication operations like MPI_Bcast for broadcasting the graph structure and MPI_Allgather for collecting the results from all processes.
    - **OpenMP:** Within each MPI process, the code can further utilize OpenMP to parallelize the computation across multiple threads, potentially speeding up the processing of the graph's subset assigned to that process.

3. Experimental Setup

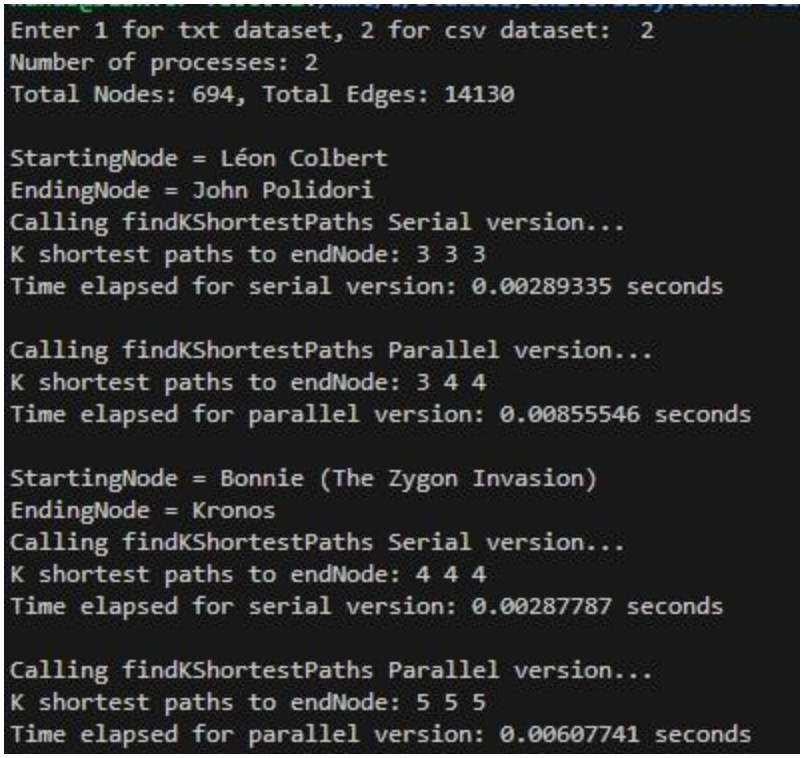
- **Hardware Configuration**:
    - **Number of Nodes**: 2
    - **Cores per Node**: 2
    - **Memory**: Each node is equipped with 4GB of RAM.
- **Software Configuration**:
    - **MPI Version**: MPICH 3.3.2
    - **OpenMP Version**: OpenMP 4.5
    - **C++ Compiler**: GCC 9.3.0
    - **Operating System**: The cluster nodes run on Ubuntu 20.04 LTS, providing a stable and updated environment for the experiments.
- **Datasets used**:
    - Doctor Who
    - Enron Email
    - EU Email

4. Results

- **Serial vs. Parallel Performance**:

```
Enter 1 for txt dataset, 2 for csv dataset:  1
Number of processes: 2
Total Nodes: 36692, Total Edges: 367662

StartingNode = 29831
EndingNode = 378

Calling findKShortestPaths Serial version...
K shortest paths to endNode: 4 4 4
Time elapsed for serial version: 0.123081 seconds

Calling findKShortestPaths Parallel version...
K shortest paths to endNode: 4 4 4
Time elapsed for parallel version: 0.188295 seconds

StartingNode = 21913
EndingNode = 16914

Calling findKShortestPaths Serial version...
K shortest paths to endNode: 4 4 4
Time elapsed for serial version: 0.125619 seconds

Calling findKShortestPaths Parallel version...
K shortest paths to endNode: 4 4 4
Time elapsed for parallel version: 0.188506 seconds
```

```
Enter 1 for txt dataset, 2 for csv dataset:  1
Number of processes: 2
Total Nodes: 36692, Total Edges: 367662

StartingNode = 15350
EndingNode = 22020

Calling findKShortestPaths Serial version...
K shortest paths to endNode: 4 4 4
Time elapsed for serial version: 0.113728 seconds

Calling findKShortestPaths Parallel version...
K shortest paths to endNode: 4 4 4
Time elapsed for parallel version: 0.183416 seconds

StartingNode = 21307
EndingNode = 32876

Calling findKShortestPaths Serial version...
K shortest paths to endNode: 7 7 7
Time elapsed for serial version: 0.117155 seconds

Calling findKShortestPaths Parallel version...
K shortest paths to endNode: 7 7 7
Time elapsed for parallel version: 0.186208 seconds
```



- **Scalability Analysis**: Analyze the scalability of the parallel algorithm, both strong scaling (fixed-size problem, increasing number of processors) and weak scaling (increasing problem size with the number of processors).

5. Analysis and Insights

- **Efficiency of Parallelization**: Utilizes MPI for task distribution and OpenMP for parallel file reading, aiming to maximize hardware resource usage.
- **Comparison of MPI and OpenMP Utilization**: Primarily employs MPI for inter-node communication, with limited use of OpenMP for optimizing file reading operations.
- **Optimization Opportunities**: Implement dynamic load balancing and optimize MPI communication patterns to reduce overhead and improve scalability.

6. Conclusion

In conclusion, contrary to expectations of performance improvement through parallelization, the experiments with the parallel version of the K shortest paths algorithm resulted in a slight increase in execution time compared to the serial version. This unexpected outcome can primarily be attributed to the communication overhead introduced by MPI operations, particularly the collective communication and data synchronization across processes. Additionally, the overhead from managing multiple threads within each process using OpenMP may have further contributed to this increase. These findings underscore the importance of optimizing communication patterns and considering the overheads associated with parallel computing frameworks when designing parallel algorithms.