**Project Report: Parallel Dynamic SSSP using MPI and OpenMP Course: Parallel and Distributed Computing**

Group members: Ali haider (22i-0936) , Saif ur Rehman (22i-0933) , Muhammad Owais (22i-0945)

**1. Introduction**

This project focuses on the development of an optimized parallel update method for Single-Source Shortest Paths (SSSP) in dynamic, large-scale graphs. The approach uses MPI for communication across processes and OpenMP for parallelism within a single node. The algorithm follows the template outlined in the paper *A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks*.

**2. Methodology**

**2.1 Datasets**

Two datasets were utilized for evaluation:

* **Amazon Dataset**: 334,863 nodes and 925,872 edges
* **DBLP Dataset**: 317,080 nodes and 1,048,066 edges

**2.2 Technologies**

* **Programming Language**: C++17
* **Libraries**: METIS for graph partitioning, OpenMPI, OpenMP
* **Approaches Implemented**:
  + Sequential Dijkstra's algorithm
  + MPI-only (4 processes)
  + OpenMP-only (4 threads)
  + Hybrid MPI + OpenMP (optional extension)

**2.3 Performance Metrics**

The following metrics were used for evaluating the performance of the parallelized algorithms:

* **Execution Time**
* **Speedup**
* **Efficiency**
* **Scalability (Strong Scaling)**

**3. Experimental Results**

**3.1 Execution Time**

* **Amazon Dataset**:
  + Sequential: 8.003s
  + MPI (4 proc): 2.268s
  + OpenMP (4 threads): 1.281s
* **DBLP Dataset**:
  + Sequential: 18.846s
  + MPI (4 proc): 2.525s
  + OpenMP (4 threads): 1.382s

**3.2 Speedup**

* **Amazon**:
  + MPI: 3.53×
  + OpenMP: 6.25×
* **DBLP**:
  + MPI: 7.46×
  + OpenMP: 13.63×

**3.3 Efficiency**

Efficiency for MPI and OpenMP (4 processors/threads) are as follows:

* **Amazon Dataset**:
  + MPI: 88.25%
  + OpenMP: 156.25%\*
* **DBLP Dataset**:
  + MPI: 186.5%
  + OpenMP: 340.75%\*

*(Note: Efficiency exceeding 100% for OpenMP is due to favorable cache usage and reduced overhead in comparison to MPI, indicating super-linear behavior typically seen when optimized shared memory code outperforms sequential execution.)*

**3.4 Scalability (Strong Scaling)**

Scalability was evaluated by observing how execution time decreases as the number of processors is increased:

* **Amazon Dataset**: Speedup ranges from ~3.5× to 6.2×, showing moderate scalability.
* **DBLP Dataset**: Speedup ranges from ~7.4× to 13.6×, indicating excellent scalability. The larger and denser graph allows for better utilization of parallel resources and reduced relative overhead.

**3.5 Visualization**

* **Execution Time Comparison** (seconds):
  + **Amazon**: Sequential (8.003), MPI (2.268), OpenMP (1.281)
  + **DBLP**: Sequential (18.846), MPI (2.525), OpenMP (1.382)

**4. Observations**

* OpenMP provides significantly better performance compared to MPI due to its reduced overhead and better data locality in shared memory environments.
* The larger DBLP dataset saw more significant improvements in speedup due to its scalability.
* METIS partitioning overhead was minimal (~0.7s).
* Consistency in output statistics across all implementations (e.g., WCC, SCC, clustering coefficient, triangles) validated the correctness of the parallel algorithms.

**5. Conclusion**

This project demonstrates that parallel techniques such as MPI and OpenMP are effective for updating SSSP in large dynamic graphs. MPI delivers solid performance for distributed systems, while OpenMP achieves high performance in shared-memory systems. Future work may involve integrating both MPI and OpenMP into a hybrid model to exploit both architectures.

**6. Future Work**

* Full integration of the hybrid MPI + OpenMP execution.
* Support for batch processing of edge deletions in addition to insertions.
* Dynamic graph partitioning strategies for improved performance.
* Testing on real multi-node clusters with 8–64 cores.
* Visualization of affected subgraphs and incremental updates to SSSP.