**Parallel SSSP Update Algorithm – Project Report**

Project Title: Parallel Implementation of SSSP Update Algorithm

Authors: Talha Azim & Ahsan Raza

Repository: https://github.com/Talha-Azim/PDC-Project

Problem Statement

This project aimed to develop an efficient parallel algorithm for updating the Single-Source Shortest Paths (SSSP) in large, dynamically changing networks. The work was inspired by the research paper \*"A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks."\* The primary challenge was to efficiently process graph updates (such as edge weight changes or node additions/deletions) while comparing different parallel computing strategies to determine the most effective approach.

Approach

The implementation was divided into three versions to evaluate performance:

1. Sequential Version (Baseline)

- Used Dijkstra's algorithm to compute shortest paths.

- Updates were processed one after another without any parallelization.

- Provided a reference point for comparing speedup in parallel versions.

2. MPI-Based Parallel Version

- Leveraged MPICH (Message Passing Interface) for distributed computing.

- The graph was partitioned using METIS, a graph partitioning tool, to distribute workload across processes.

- Communication between processes was managed using MPI functions like `MPI\_Bcast`, `MPI\_Gather`, and `MPI\_Allgather`.

- While this version reduced computation time, communication overhead (especially with many processes) sometimes negated performance gains.

3. Hybrid MPI + OpenMP Version

- Combined MPI (for inter-node parallelism) and OpenMP (for intra-node multithreading).

- Each MPI process handled a graph partition, while OpenMP threads accelerated computations within each node.

- This hybrid approach outperformed the other two versions, as it maximized CPU utilization while minimizing idle time.

Performance Evaluation

The team compared execution times (excluding MPI communication overhead) against the number of updates:

- The sequential version showed linear time growth—more updates meant longer processing times.

- The MPI version improved computation speed but suffered from communication bottlenecks, especially with excessive use of collective operations like `MPI\_Bcast`.

- The Hybrid (MPI + OpenMP) version was the fastest, efficiently balancing computation and communication.

Key Findings

- Hybrid parallelism (MPI + OpenMP) works best for large-scale graphs, offering better scalability.

- Communication overhead is a major challenge in distributed computing—optimizing message-passing is crucial.

- Static graph partitioning (using METIS) helps with load balancing, but dynamic adjustments might be needed for highly irregular graphs.

Lessons Learned

- Profiling is essential—identifying bottlenecks before and after parallelization ensures meaningful optimizations.

- More processes ≠ faster execution—there’s an optimal number of processes based on workload and graph size.

- MPI offers fine-grained control but requires careful design to minimize communication delays.

- OpenMP is simpler to integrate and highly effective for shared-memory parallelism.

Conclusion

This project demonstrated that parallel computing, especially hybrid models (MPI + OpenMP), significantly speeds up SSSP updates in dynamic networks. However, performance depends heavily on balancing computation and communication. Future work could explore dynamic load balancing and more efficient communication patterns to further enhance scalability.

