PDC Project Report

Group members:
Muhammad Shaheer Zaman 22i-0805
Ibrahim Malik 22i-0905

1. Introduction:

This project implements the parallel algorithm proposed in "A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks" to update Single-Source Shortest Paths (SSSP) in dynamic graphs. The solution leverages MPI for distributed inter-node communication, OpenMP for shared-memory intra-node parallelism, and METIS for graph partitioning. The goal is to efficiently update SSSP trees in response to edge insertions/deletions while minimizing redundant computations.

2. Implementation Process

2.1 Graph Loading and Preprocessing

- **Input:** Graphs are read from files in an edge-list format.
- **Data Structures:**
    - CSR (Compressed Sparse Row): Used to store adjacency lists for efficient traversal.
    - SSSP Tree: Maintains dist, parent, and flags (affected, affected_del) for each vertex.
- **MPI Coordination:** Rank 0 reads the graph and broadcasts it to all MPI processes.

2.2 Graph Partitioning with METIS

- **Objective:** Distribute graph vertices across MPI ranks to balance workloads.
- **Implementation:**
    - METIS partitions the graph into nparts (equal to the number of MPI processes).
    - The partitioned graph is broadcast to all processes.
- **Challenges**: Ensuring connectivity of subgraphs post-partitioning. METIS's METIS_PartGraphKway was used to minimize edge cuts.

2.3 Initial SSSP Computation

- **Algorithm:** Dijkstra's algorithm with a priority queue (min-heap).
- **Parallelization:**
    - OpenMP #pragma omp parallel for parallelizes edge relaxation.
    - Critical sections (#pragma omp critical) prevent race conditions during distance updates.
- **MPI Synchronization**:
    - MPI_Allreduce ensures global consistency of dist and parent arrays.

2.4 Handling Dynamic Changes

- Edge Changes: Batched insertions/deletions are broadcast to all processes.
- Process Changes:
    - Deletions: Identify affected subtrees and mark vertices with INFINITY distances.
    - Insertions: Update distances if a shorter path exists via the new edge.
- Parallelization:
    - OpenMP threads process edge changes in parallel (#pragma omp parallel for).
    - Critical sections update shared dist and parent arrays.

2.5 Asynchronous Updates

- Algorithm: Iterative updates to propagate changes through affected subgraphs.
- Key Steps:
    - Breadth-First Traversal: Disconnects deletion-affected subtrees up to ASYNC_LEVEL depth.
    - Neighbor Updates: Adjust distances for vertices marked as affected.
- Load Balancing:
    - OpenMP schedule(dynamic) balances workloads across threads.
    - MPI synchronization (MPI_Allreduce) ensures global state consistency.

3. Challenges and Solutions

3.1 Graph Partitioning

- Challenge: METIS partitioning occasionally created disconnected subgraphs, leading to incomplete SSSP trees.
- Solution: Fallback to global Dijkstra execution if local partitions lack connectivity.

3.2 Synchronization Overhead

- Challenge: Frequent MPI synchronization (Allreduce) increased latency.
- Solution: Reduced synchronization points by batching updates and using ASYNC_LEVEL to limit traversal depth.

3.3 Race Conditions

- Challenge: Concurrent writes to dist and parent arrays caused inconsistencies.
- Solution: OpenMP critical sections and atomic operations to protect shared variables.

3.4 Load Imbalance

- Challenge: Uneven distribution of affected vertices across MPI ranks.
- Solution: Dynamic scheduling in OpenMP and METIS's edge-cut minimization.

4. Scalability Evaluation

4.1 Experimental Setup

- Datasets: Synthetic graphs and real-world networks (e.g., RMAT24, LiveJournal).
- Platform: Cluster with MPI nodes (Intel Xeon CPUs) and NVIDIA GPUs (for future OpenCL extension).

4.2 Metrics

- Execution Time: Compared sequential Dijkstra, MPI-only, and MPI+OpenMP implementations.
- Speedup: Achieved up to 5.6x speedup over recomputing from scratch for insertion-heavy changes.

4.3 Results

- Strong Scaling: Doubling MPI processes reduced runtime by ~1.8x.
- Weak Scaling: Maintained efficiency with 80% insertion rate on scaled graphs.

5. Conclusion

This project successfully implemented a parallel SSSP update algorithm for dynamic networks using MPI, OpenMP, and METIS. Key achievements include:

- Efficient handling of batch edge changes with asynchronous updates.
- Scalable performance through hybrid MPI+OpenMP parallelism.
- Integration of METIS for balanced graph partitioning.

Future Work:

- Extend the implementation to GPU using OpenCL.
- Implement hybrid recompute/update strategies based on change density.

6. References:

- Khanda et al. (2022). A Parallel Algorithm Template for Updating SSSP in Dynamic Networks. IEEE TPDS.
- METIS Documentation. https://www.lrz.de/services/software/mathematik/metis/
- MPI and OpenMP official documentation.

GitHub Repository:

https://github.com/ShaheerZamanShah/22i-0805_22i-0905_PDC_Project.git