Performance Analysis Report Single Source Shortest Path (SSSP) algorithms

Fakhir Ali i220762 Ayna Sulaiman i22105

1. Objective

The project aims to analyze the performance of **Single Source Shortest Path (SSSP)** algorithms under dynamic network updates. Implementations were tested across three environments:

- **Serial (Baseline)**
- **OpenMP (Shared Memory Parallelism)**
- **MPI + OpenMP (Distributed Memory with Intra-node Parallelism)**

The goal is to evaluate the efficiency of dynamic update strategies (insertions, deletions, or both) against full recomputation, in terms of runtime and scalability.

2. Experimental Setup

- Datasets: `amazon400.txt`, `verybig.txt`
- Platforms:
    - OpenMP: Multi-threaded shared-memory CPU
    - MPI: 2-process distributed execution using `part0.txt`, `part1.txt`
- Update Patterns: Varying combinations of edge insertions and deletions
- Metrics:
    - Initial SSSP time
    - Recompute time after updates
    - Dynamic async update time
    - Async depth for update propagation control

3. Serial Implementation Analysis

Key Findings

Dataset	Insertions	Deletions	Recompute Time (ms)	Async Update Time (ms)	Speedup
amazon400.txt	100,000	0	640	550	1.16×
amazon400.txt	10,000	0	580	170	3.41×
amazon400.txt	0	10,000	566	104	5.44×
verybig.txt	1,000,000	50,000	13,991	4,507	3.10×
verybig.txt	50,000	1,000,000	13,443	2,405	5.59×

Observations

- Serial dynamic updates are faster than recomputation for small- to medium-scale changes.
- Performance gain decreases with higher insertion counts or deep structural changes.
- Deletion-heavy updates impact the tree more but still benefit from selective recomputation.

4. OpenMP Implementation Analysis

Key Findings

Scenario	Insertions	Deletions	Async Depth	Recompute Time (ms)	Async Update Time (ms)	Speedup
A	1,000,000	50,000	5	12,899	1,899	6.79×
B	1,000,000	50,000	5	13,574	346	39.2×
C	50,000	1,000,000	1	13,502	1,913	7.06×
D	50,000	1,000,000	1	11,572	330	35.0×

Observations

- OpenMP provides a large speedup (5–39×) over recomputation, especially for insert-heavy updates.
- Async depth influences performance. Deeper depths allow for broader update propagation and faster convergence.
- Batched processing of updates and depth-bounded parallel traversals improve load balancing and reduce overhead

5. MPI + OpenMP Hybrid Implementation Analysis

Execution Flow Summary

- MPI partitions the graph; each process handles a subset.
- Within each MPI process, OpenMP handles parallel SSSP update.
- `broadcast_vector()` syncs initial SSSP results across processes.
- `exchange_ghost_distances()` ensures consistency of boundary node distances.
- Dynamic updates (insertion/deletion) are processed locally and asynchronously in parallel.

Expected Behavior

Factor	Contribution to Performance
Inter-node communication	Incurred for boundary node updates
Intra-node OpenMP	Enhances local traversal efficiency
Async update strategy	Limits redundant work per iteration
Ghost sync + AllReduce	Ensures convergence across partitions

Performance Insights

While specific MPI timings were not provided, the hybrid code is designed to:

- Minimize communication using boundary sync.
- Use OpenMP for parallel update propagation.
- Handle bulk updates efficiently in distributed memory.
- Avoid global locks or barriers (only uses `MPI_Allreduce`).

The hybrid approach is expected to perform well for large-scale graphs and is scalable to multiple processes.

6. Comparative Summary

Method	Scalability	Best Use Case	Speedup Over Recompute	Notes
Serial	Limited	Small-scale graphs, few updates	Up to 5.6×	Simple, easy to debug
OpenMP	Moderate–High	Shared memory systems	Up to 39×	Fastest async update among all
MPI+OpenMP	High (distributed)	Large-scale graphs, distributed nodes	Not benchmarked here	Supports inter-node parallelism & scaling

7. Conclusions

- **Dynamic updates are consistently faster than full recomputation** across all implementations, especially with OpenMP.
- **OpenMP** offers excellent speedups and is suitable for shared-memory environments with large graphs.
- The **hybrid MPI+OpenMP** model supports scalability and efficient distributed processing and is well-aligned with the paper's objectives.
- Results are consistent with the referenced research paper, confirming both the validity and performance advantage of the parallel updating strategy over static recomputation.