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** is used to partition the graph across two processes (`rank 0` and `rank 1`), each responsible for a subset of nodes and edges.
- **OpenMP** is used within each MPI process to parallelize dynamic updates (insertions, deletions) using multithreading.
- The initial **Single-Source Shortest Path (SSSP)** is computed centrally on `rank 0`, then distributed using `broadcast_vector()` to all other ranks.
- **exchange_ghost_distances()** synchronizes distances at partition boundaries to maintain correctness.
- Dynamic edge insertions and deletions are processed in parallel using an **asynchronous propagation model**, bounded by an asynchrony depth `A = 5`.

Test Configuration and Results

Parameter	Value
Input Graph	`amazon400.txt`
MPI Ranks	2 (Rank 0 and Rank 1)
Threads per Rank	Based on system setting (`OMP_NUM_THREADS`)
Insertions	100,000
Deletions	50,000
Asynchrony Depth (A)	5
Initial SSSP Time	716 ms (Rank 0 only)
Dynamic Update Time	443 ms (both ranks)

Performance Insights

- **Concurrent Update Execution:** Both ranks completed the dynamic update phase in exactly **443 ms**, confirming simultaneous processing and proper hybrid parallelization.
- **Inter-Process Communication:** Overhead from boundary exchange is minimized via `MPI_Sendrecv`, which syncs ghost node distances. Convergence is handled through `MPI_Allreduce`, avoiding full barriers.
- **Multithreaded Efficiency:** OpenMP parallelizes update propagation inside `async_update()`, reducing per-rank workload time.
- **Load Distribution:** While both ranks complete at the same time, underlying load balance may vary based on the distribution of affected nodes.

Scalability and Suitability

- The hybrid model effectively combines **distributed memory (MPI)** and **shared memory (OpenMP)** strategies, making it suitable for **large-scale graphs** and **multi-node clusters**.
- Though only tested on two MPI ranks, the code is architected to scale with additional processes by extending ghost sync mechanisms.
- Initial SSSP centralization on Rank 0 avoids global duplication, while local updates maintain high parallel efficiency.

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	Up to 5×	Ran on virtual machines

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.