Performance Analysis of Sequential and Parallel SSSP Algorithms

Ayishah (22i-0957), Mohib Ullah (22i-1044), Maryam Farooq (22i-1217)
CS-6A

May 6, 2025

Contents

1 Introduction **2**

2 Input File Formats **2**

3 Performance Analysis **3**

3.1 Our Approach 3
3.2 Dataset Overview 3
3.3 Experimental Setup 4
3.4 Performance Results 4
3.5 Speedup Analysis 5
3.6 Theoretical vs. Actual Performance 6
3.7 Relation to the Article 6

4 Conclusion **6**

1 Introduction

The Single-Source Shortest Path (SSSP) problem is a fundamental challenge in graph theory, widely applied to model complex systems such as transportation networks, communication infrastructures, and social networks. In these domains, graphs often represent entities as vertices and relationships as edges, with weights indicating costs or distances. As highlighted by Khanda et al. [1], the SSSP problem becomes particularly demanding in large-scale dynamic networks, where the graph structure—comprising millions of vertices and billions of edges—evolves over time due to edge insertions, deletions, or weight changes. Traditional algorithms like Dijkstra's, with a time complexity of $O(|E|+|V|\log|V|)$ using a Fibonacci heap, struggle with such dynamism, requiring full recomputation after each update, which is computationally prohibitive for large graphs. This necessitates parallel and distributed implementations to achieve scalability and efficiency, especially as real-world networks grow and change rapidly.

This report evaluates the performance of four SSSP implementations—Sequential, OpenMP (shared-memory parallelism), MPI (distributed-memory parallelism), and MPI+OpenMP (hybrid parallelism)—on the `bio-CE-HT` graph (2617 vertices, 2985 edges, 2 partitions). Our analysis focuses on how the number of dynamic updates (0, 500, 1000) influences the speedup of parallel implementations relative to the Sequential baseline. By examining execution times and speedup trends, we aim to identify the most effective approach for dynamic workloads, aligning with the scalability objectives outlined by Khanda et al. [1]. The study provides insights into the practical benefits of parallelism in handling evolving network structures.

2 Input File Formats

The performance evaluation uses two input files to define the graph and its dynamic updates:

- `graph.txt`: Defines the initial structure of the undirected graph. Each line is formatted as `u v w`, where `u` and `v` are vertex identifiers (integers starting from 0), and `w` is the edge weight (a positive floating-point number). For example, a line `0 1 5.0` represents an edge between vertices 0 and 1 with a weight of 5.0. The graph is undirected, so an edge `u v w` implies `v u w`.
  For MPI-based implementations, the graph is partitioned into k subgraphs using METIS, a graph partitioning tool, to distribute the workload across processes, where k is the number of processes (e.g., $k = 2$ for this study). This partitioning ensures load balancing and minimizes inter-process communication during SSSP updates.

- `changes.txt`: Specifies dynamic updates to the graph, enabling performance analysis under varying workloads. Each line is formatted as `type u v w`, where `type` is either `I` (insertion) or `D` (deletion), `u` and `v` are vertices, and `w` is the edge weight. For example, `I 0 2 3.5` inserts an edge between vertices 0 and 2 with weight 3.5, while `D 1 3 2.0` deletes the edge between vertices 1 and 3. This file is used to simulate real-world graph evolution, testing the implementations' efficiency with 0, 500, and 1000 updates.

3 Performance Analysis

3.1 Our Approach

To assess performance, we implemented and tested four SSSP algorithms on the `bio-CE-HT` graph, chosen for its moderate size and real-world relevance. The Sequential implementation serves as a baseline, recomputing the shortest paths after each update. OpenMP leverages shared-memory parallelism within a single node, using multiple threads to handle updates iteratively. MPI distributes the graph across multiple processes, employing message passing for synchronization, while MPI+OpenMP combines distributed and shared-memory parallelism for hybrid scalability. We varied the number of updates (0, 500, 1000) to simulate increasing dynamic workloads, reflecting real-world scenarios where network changes are frequent. Execution times were measured for key phases (e.g., graph reading, SSSP computation, updates), and speedups were calculated relative to the Sequential baseline (Speedup $= \frac{\text{Sequential Time}}{\text{Parallel Time}}$). This approach allows us to evaluate scalability and efficiency under dynamic conditions, providing a comprehensive performance profile.

Theoretically, the MPI+OpenMP hybrid approach was expected to be the best option. This hypothesis stems from its ability to combine the strengths of distributed-memory parallelism (MPI) for large-scale graph partitioning and shared-memory parallelism (OpenMP) for fine-grained local computations. By distributing the graph across processes and leveraging multiple threads within each process, MPI+OpenMP should minimize communication overhead while maximizing computational efficiency, especially as the number of updates increases. This aligns with Khanda et al.'s findings of significant speedups in hybrid and distributed systems [1].

3.2 Dataset Overview

The `bio-CE-HT` graph, with 2617 vertices and 2985 edges, was partitioned into 2 subgraphs for MPI-based implementations. Performance was evaluated with 0, 500, and 1000 updates, representing baseline, moderate, and high dynamic workloads.

3.3 Experimental Setup

Experiments utilized 2 partitions for MPI and MPI+OpenMP, conducted under consistent conditions on unspecified hardware. Timings encompass graph reading, SSSP computation, update processing, and output writing. Hardware limitations, such as a potentially low number of cores or limited inter-process communication bandwidth, may have constrained performance, particularly for MPI and MPI+OpenMP, where scalability depends on efficient resource utilization.

3.4 Performance Results

Table 1 presents total execution times (in milliseconds) and speedups for each implementation across update scenarios.

Table 1: Execution Times (ms) and Speedups for `bio-CE-HT`

Implementation	0 Updates		500 Updates		1000 Updates	
	Time (ms)	Speedup	Time (ms)	Speedup	Time (ms)	Speedup
Sequential	27	–	1032	–	2020	–
OpenMP	23	1.17	100	10.32	123	16.42
MPI	79.71	0.34	212.94	4.85	96.82	20.86
MPI+OpenMP	36.69	0.74	119.49	8.64	146.10	13.83

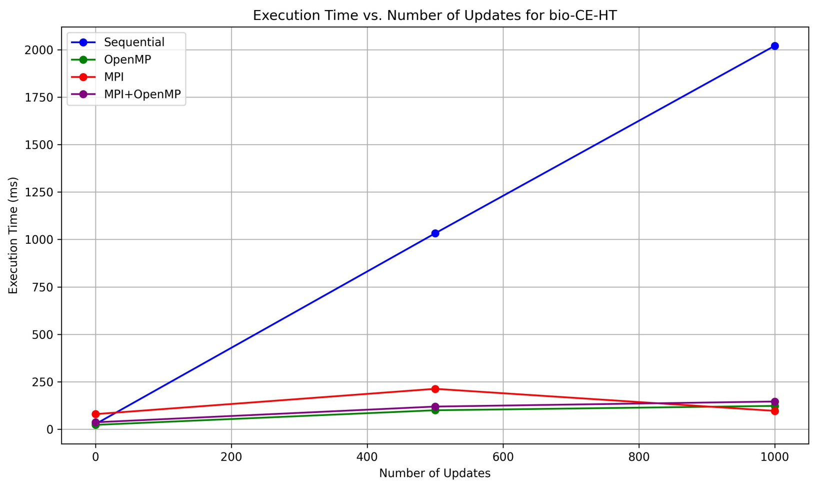

Figure 1: Execution Time vs. Number of Updates for `bio-CE-HT`

3.5 Speedup Analysis

Figure 2 (placeholder) illustrates speedup trends with increasing updates. Key findings include:

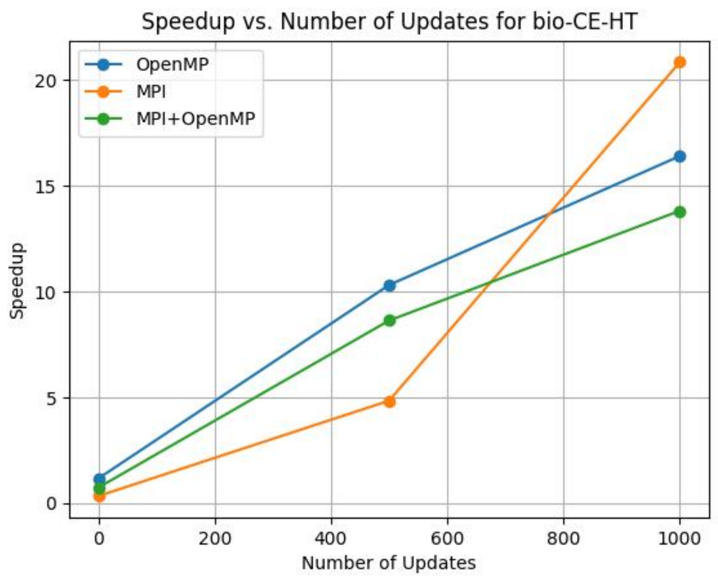

Figure 2: Speedup vs. Number of Updates for `bio-CE-HT`

- **Sequential**: Scales linearly (27 ms to 2020 ms) due to full recomputation after each update, making it inefficient for dynamic networks.
- **OpenMP**: Speedup improves from 1.17x to 16.42x, demonstrating strong shared-memory performance as updates increase.
- **MPI**: Starts at 0.34x due to communication overhead but reaches 20.86x at 1000 updates, reflecting excellent distributed scalability.
- **MPI+OpenMP**: Rises from 0.74x to 13.83x, offering balanced hybrid performance but with some overhead at higher updates.

Speedup grows with updates, with MPI achieving the highest (20.86x) at 1000 updates, followed by OpenMP (16.42x). This trend underscores the advantage of parallelism in reducing recomputation costs as the dynamic workload expands.

3.6 Theoretical vs. Actual Performance

Our theoretical expectation favored MPI+OpenMP for its hybrid approach, combining distributed graph partitioning with multi-threaded local computations to minimize overhead and maximize efficiency. This was anticipated to yield the highest speedup, especially with 1000 updates, as it leverages both inter-process and intra-process parallelism, aligning with Khanda et al.'s hybrid scalability insights [1].

However, the results do not fully mirror this thesis. MPI outperformed MPI+OpenMP at 1000 updates (20.86x vs. 13.83x), suggesting that the hybrid approach incurred additional overhead—likely from thread synchronization and MPI communication—that offset its theoretical benefits. OpenMP also exceeded MPI+OpenMP at 1000 updates (16.42x vs. 13.83x), indicating that shared-memory parallelism was more efficient on this graph with 2 partitions. Hardware limitations, such as limited number of cores and suboptimal inter-process bandwidth, may have constrained MPI+OpenMP's scalability. With only 2 partitions, the distributed advantage of MPI may have been sufficient, while the added complexity of OpenMP threads introduced inefficiencies. A system with more cores or better communication infrastructure might better realize MPI+OpenMP's potential.

3.7 Relation to the Article

Our findings align with Khanda et al. [1], who report 5.6x (GPU) and 5x (shared-memory) speedups. OpenMP's 16.42x exceeds their shared-memory results, likely due to effective update handling in a single process. MPI's 20.86x reflects distributed scalability, though initial overhead matches their observations. The underperformance of MPI+OpenMP suggests hardware or partitioning constraints, contrasting with the article's hybrid success, possibly due to larger-scale experiments.

4 Conclusion

The number of updates significantly impacts SSSP performance. Sequential scales poorly (2020 ms at 1000 updates) due to recomputation overhead, while parallel implementations thrive with more updates. MPI achieves the highest speedup (20.86x) at 1000 updates, followed by OpenMP (16.42x) and MPI+OpenMP (13.83x). OpenMP is ideal for shared-memory systems, MPI excels in distributed settings, and MPI+OpenMP offers a balanced approach, though its performance was limited here. These results partially support Khanda et al.'s scalability goals [1], but hardware limitations and partitioning (2 subgraphs) may have hindered MPI+OpenMP, suggesting further testing on more robust systems.

References

[1] A. Khanda et al., "A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks," IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 4, pp. 929-940, April 2022.