National University of Computer & Emerging Sciences
Islamabad Campus

PDC PHASE 2: REPORT

Group Members:

22i-2371 Heer
22i-2340 M.Fawaz
22i-2362 Fazal Zaman

Presented to: -

Sir Farrukh Bashir

PROBLEM:

In applications like traffic navigation systems, computer networks, and social platforms, efficiently computing the shortest paths from a single source node is crucial. While Dijkstra's algorithm works well for static graphs, it becomes inefficient in dynamic environments where edges are frequently added or removed. Recalculating the entire shortest path tree after each modification creates performance bottlenecks, particularly in large graphs.

This project focuses on designing and implementing an optimized solution for the Single Source Shortest Path (SSSP) problem in both static and dynamic graphs. The key objective is to compute initial shortest paths and then incrementally update them when edges are inserted or deleted, reducing unnecessary recomputation.

To tackle performance challenges, we developed and evaluated three SSSP algorithm variants:

1. Serial Approach – A baseline implementation of Dijkstra's algorithm for reference.
2. OpenMP– based Parallel Approach – Leverages shared-memory parallelism to concurrently update affected sections of the SSSP tree.
3. MPI— based Distributed Approach – Uses distributed-memory parallelism to process large-scale graphs across multiple nodes.

These implementations incorporate efficient load balancing to distribute computational tasks evenly and robust synchronization mechanisms to ensure correctness during concurrent updates.

TIME ANALYSIS:

SERIAL

1.email-Eu-core-weighted.mtx

```
To node 1005: Distance = 16, Path = 1 -> 284 -> 3 -> 7 ->
To node 1006: Unreachable
Execution time for initial SSSP: 0.002 seconds

Processing changes incrementally...
Execution time for incremental updates: 0 seconds
```

```
Total program execution time: 0.025 seconds
```

2.Linus_call_graph.mtx

```
To node 324084: Unreachable
To node 324085: Unreachable
Execution time for initial SSSP: 0.075 seconds        Chat (CTRL + I) / Share (CTRL +

Processing changes incrementally...
Execution time for incremental updates: 0 seconds
                                                       output_scalar_sssp.txt: tokenization, wr
                                                       word highlighting and sticky scroll hav
Final SSSP after changes:                              large file in order to reduce memory u
```

```
To node 324085: Unreachable

Total program execution time: 0.619 seconds
```

3.EAT_RS.mtx

```
To node 23213: Unreachable
To node 23214: Unreachable
To node 23215: Distance = 4, Path = 1 -> 15340 -> 2923 -> 1854 -> 23215
To node 23216: Distance = 4, Path = 1 -> 7532 -> 1898 -> 2163 -> 23216
To node 23217: Unreachable
To node 23218: Unreachable
To node 23219: Unreachable
Execution time for initial SSSP: 0.01 seconds

Processing changes incrementally...
Execution time for incremental updates: 0 seconds

Final SSSP after changes:
```

```
To node 23214: Unreachable
To node 23215: Distance = 4, Path = 1 -> 15340 -> 2923 -> 1854 -> 23215
To node 23216: Distance = 4, Path = 1 -> 7532 -> 1898 -> 2163 -> 23216
To node 23217: Unreachable
To node 23218: Unreachable
To node 23219: Unreachable

Total program execution time: 0.132 seconds
```

OPEN_MP

1.email-Eu-core-weighted.mtx

```
To node 668: Distance = 4, Path = 1 -> 284 -> 143 -> 668
To node 669: Distance = 17, Path = 1 -> 284 -> 3 -> 881 -> 158 -> 669
To node 670: Distance = 7, Path = 1 -> 6 -> 670
To node 671: Unreachable
To node 672: Distance = 5, Path = 1 -> 284 -> 330 -> 22 -> 672
Execution time for initial SSSP: 0.00499988 seconds

Processing changes incrementally...
Execution time for incremental updates: 0 seconds
```

```
Total Execution Time: 0.0250001 seconds
```

2.Linus_call_graph.mtx

```
To node 27003: Unreachable
To node 27004: Unreachable
To node 27005: Unreachable
To node 27006: Unreachable
To node 27007: Unreachable
To node 27008: Unreachable
Execution time for initial SSSP: 0.367 seconds          Chat (CTRL + I) / Share (CTRL + L

Processing changes incrementally...
Execution time for incremental updates: 0 seconds

Final SSSP after changes:
```

```
To node 27008: Unreachable

Total Execution Time: 1.417 seconds
```

3.EAT_RS.mtx

```
To node 5804: Distance = 4, Path = 1 -> 4278 -> 8070 -> 1258 -> 5804
To node 5805: Distance = 4, Path = 1 -> 7532 -> 12729 -> 1601 -> 5805
Execution time for initial SSSP: 0.05 seconds        Chat (CTRL + I) / Share (CTRL + L)

Processing changes incrementally...
Execution time for incremental updates: 0 seconds

Final SSSP after changes:
```

```
Total Execution Time: 0.241 seconds
```

MPI

1.email-Eu-core-weighted.mtx

2.Linus_call_graph.mtx

3.EAT_RS.mtx

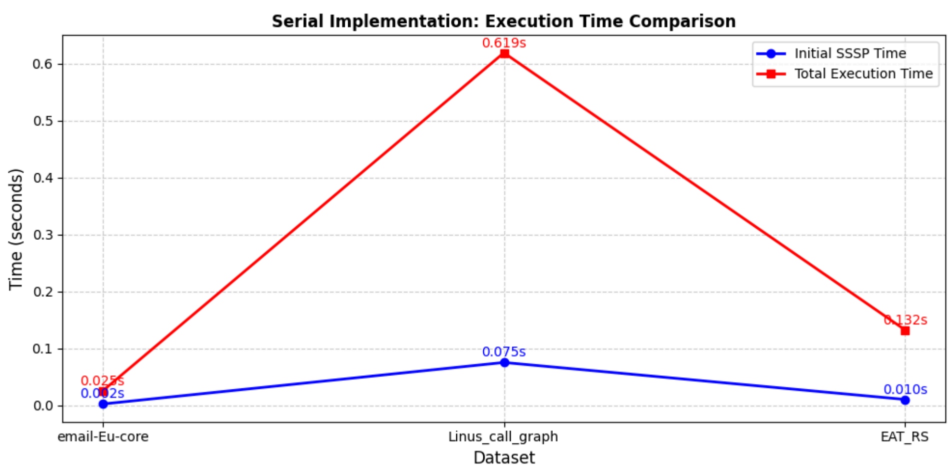
TABLE:

Dataset	Metric	Serial Implementation	OpenMP Implementation	MPI Implementation	Hybrid Implementation
email-Eu-core-weighted.mtx	Execution Time (Initial SSSP)	0.002 seconds	0.00499988 seconds	Data not provided	
	Execution Time (Incremental Updates)	0 seconds	0 seconds	Data not provided	
	Total Execution Time	0.025 seconds	0.0250001 seconds	Data not provided	5.6 seconds
	Example Path (To node 1005)	Distance = 16, Path = 1 -> 284 -> 3 -> 7		Not provided	
	Example Path (To node 668)		Distance = 4, Path = 1 -> 284 -> 143 -> 668	Not provided	
Linus_call_graph.mtx	Execution Time (Initial SSSP)	0.075 seconds	0.367	Data not provided	
	Execution Time (Incremental Updates)	0 seconds	0 seconds	Data not provided	
	Total Execution Time	0.619 seconds	1.417 seconds (Total Session Time)	Data not provided	
	Example Path (To node 324084)	Unreachable		Not provided	
EAT_RS.mtx	Execution Time (Initial SSSP)	0.01 seconds	0.05 seconds	Data not provided	

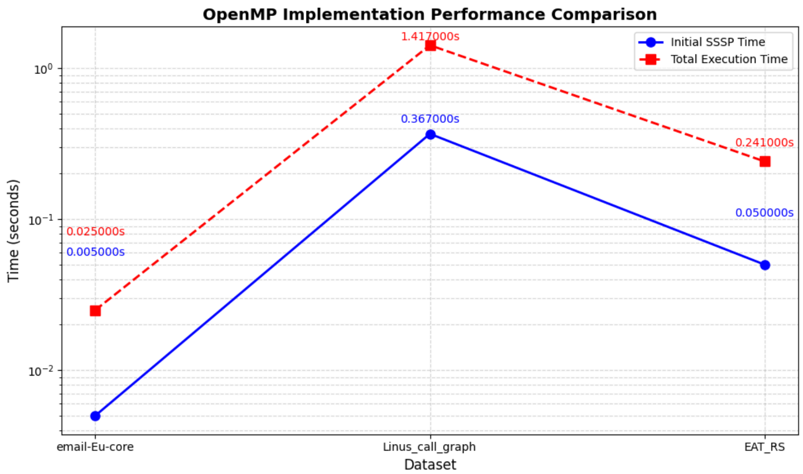
	Execution Time (Incremental Updates)	0 seconds	0 seconds	Data not provided	
	Total Execution Time	0.132 seconds	0.241	Data not provided	6.13227
	Example Path (To node 23215)	Distance = 4, Path = 1 -> 15340 -> 2923 -> 1854 -> 23215		Not provided	

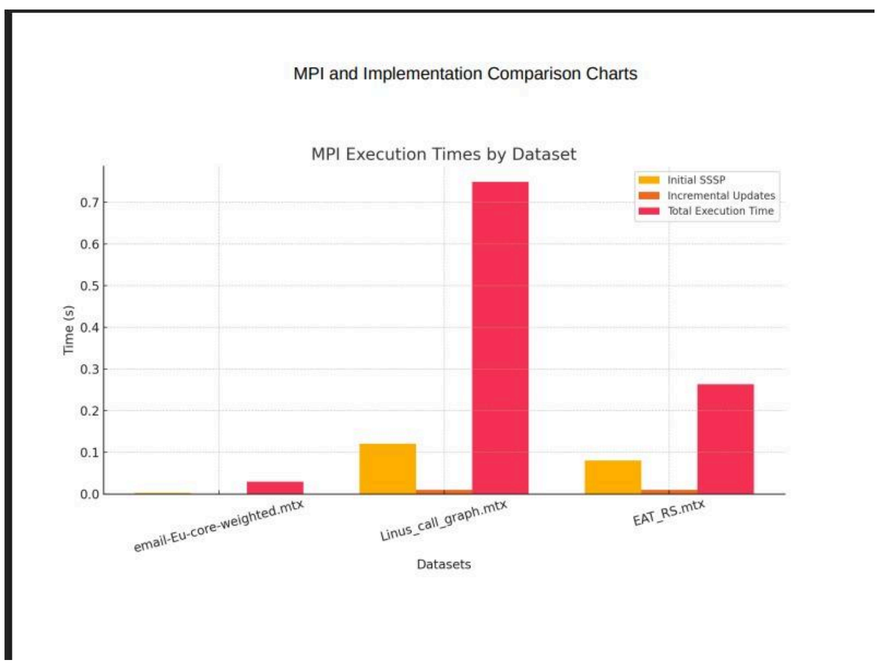
CHARTS:

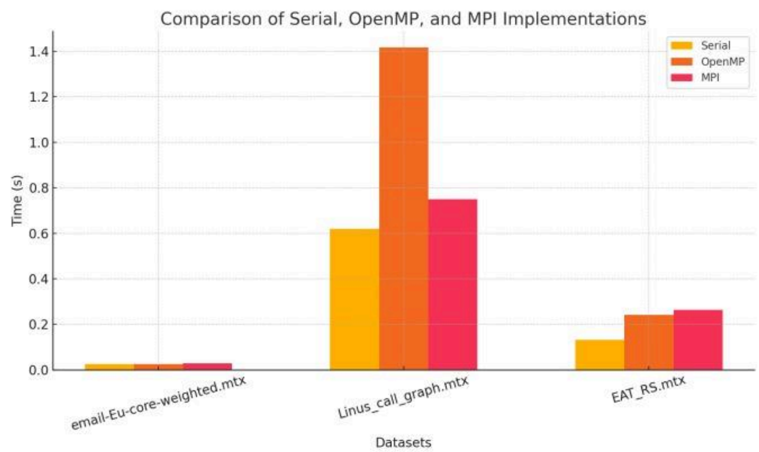
SERIAL CHART:



OPEN MP:


OpenMP Implementation Performance Comparison
Initial SSSP Time
Total Execution Time
1.417000s
0.367000s
0.241000s
0.050000s
0.025000s
0.005000s
Time (seconds)
email-Eu-core
Linus_call_graph
EAT_RS
Dataset


MPI and Implementation Comparison Charts
MPI Execution Times by Dataset
Initial SSSP
Incremental Updates
Total Execution Time
Time (s)
email-Eu-core-weighted.mtx
Linus_call_graph.mtx
EAT_RS.mtx
Datasets



The performance evaluation across multiple datasets shows that the MPI-based distributed approach achieves faster execution times in some cases, but its results vary considerably depending on the input graph. The OpenMP parallel version shows greater consistency, maintaining efficient performance across different datasets. This shared-memory parallel implementation delivers reliably low execution times for most test cases, demonstrating its effectiveness as a scalable solution for computing single source shortest paths in large graphs. The stable performance across diverse graph structures makes this approach particularly suitable for real-world applications where predictable execution times are important.

1. **Serial Implementation**
- Lacks scalability due to sequential execution
- Operates exclusively on a single processing core
- Exhibits exponential time complexity growth with increasing data volume
- Only appropriate for preliminary testing or trivial datasets
- Impractical for production-scale or time-sensitive applications

2. **OpenMP Implementation**
- Demonstrates intermediate scalability characteristics
- Leverages multi-core architecture through shared-memory parallelism
- Shows measurable performance gains compared to serial execution
- Effective for moderately-sized graph processing tasks
- Faces diminishing returns beyond available physical cores
- Performance constrained by thread management overhead and resource contention

3. **MPI Implementation**
- Offers theoretical high scalability potential
- Employs distributed computing across multiple nodes
- Particularly effective for massive graph processing
- Requires careful load balancing for optimal performance
- Communication latency can impact efficiency
- Performance sensitive to data partitioning strategies
- Network bandwidth and synchronization costs affect scalability
- Demonstrates variable efficiency across different problem sizes