National University
Of Computer and Emerging Sciences

FINAL PROJECT REPORT

In partial fulfillment of course

PDC

ROLL NO: - i220974,i221263,i221279

Section: CS E

Presented to: -

Sir Farrukh Bashir

<u>PROBLEM : -</u>

In large-scale and dynamic graph applications such as traffic navigation systems, computer networks, and social platforms, it is essential to compute the shortest paths efficiently from a single source node. While Dijkstra’s algorithm provides a reliable solution for static graphs, it becomes computationally expensive when applied to dynamic graphs where edges are frequently inserted or deleted. Recomputing the entire shortest path tree after every change leads to performance bottlenecks, especially in large graphs.

This project aims to design and implement an efficient solution for the Single Source Shortest Path (SSSP) problem in both static and dynamic graphs. The primary goal is to compute initial shortest paths and then incrementally update them in response to edge insertions and deletions, minimizing recomputation.

To address the performance challenges, we implemented and evaluated four versions of the SSSP algorithm:

- Serial Approach: A traditional implementation of Dijkstra's algorithm used as a baseline.

- OpenMP-based Parallel Approach: Utilizes shared-memory parallelism to update affected parts of the SSSP tree concurrently.

- MPI-based Distributed Approach: Explores distributed-memory parallelism for large-scale graphs processed across multiple nodes.

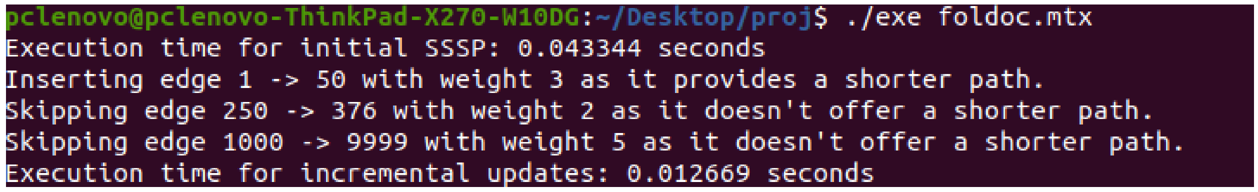
- Hybrid Approach (MPI + OpenMP): Combines both shared-memory and distributed-memory models to fully utilize modern multi-core, multi-node computing environments.

These implementations also focus on efficient load balancing to evenly distribute computational tasks across threads and processes, and synchronization mechanisms to maintain correctness during concurrent updates.

EXECUTION TIME ANALYSIS:-

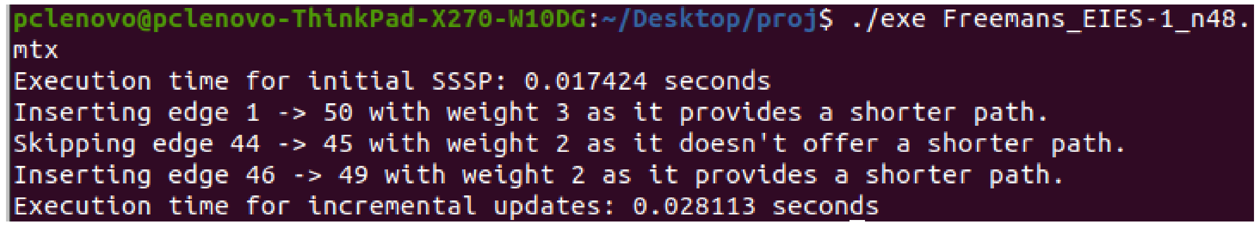
SERIAL:

1.foldoc.mtx



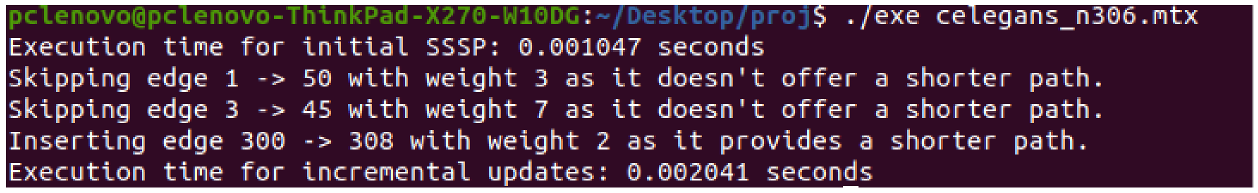
```
pclenovo@pclenovo-ThinkPad-X270-W10DG:~/Desktop/proj$ ./exe foldoc.mtx
Execution time for initial SSSP: 0.043344 seconds
Inserting edge 1 -> 50 with weight 3 as it provides a shorter path.
Skipping edge 250 -> 376 with weight 2 as it doesn't offer a shorter path.
Skipping edge 1000 -> 9999 with weight 5 as it doesn't offer a shorter path.
Execution time for incremental updates: 0.012669 seconds
```

2.Freemans_EIES-1.mtx



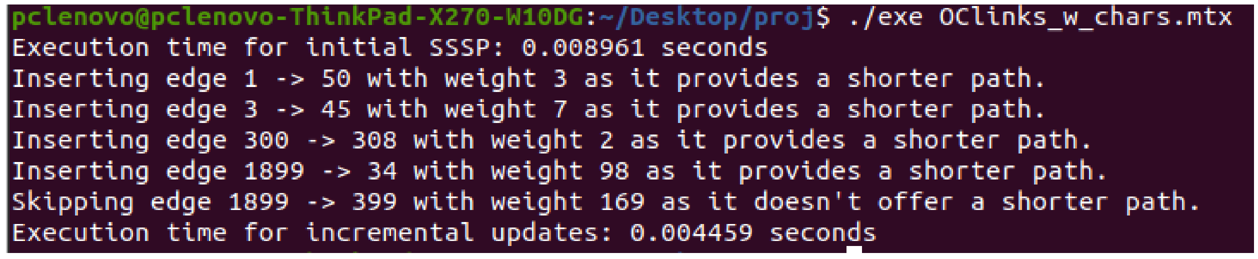
```
pclenovo@pclenovo-ThinkPad-X270-W10DG:~/Desktop/proj$ ./exe Freemans_EIES-1_n48.
mtx
Execution time for initial SSSP: 0.017424 seconds
Inserting edge 1 -> 50 with weight 3 as it provides a shorter path.
Skipping edge 44 -> 45 with weight 2 as it doesn't offer a shorter path.
Inserting edge 46 -> 49 with weight 2 as it provides a shorter path.
Execution time for incremental updates: 0.028113 seconds
```

3.celegans_n360.mtx



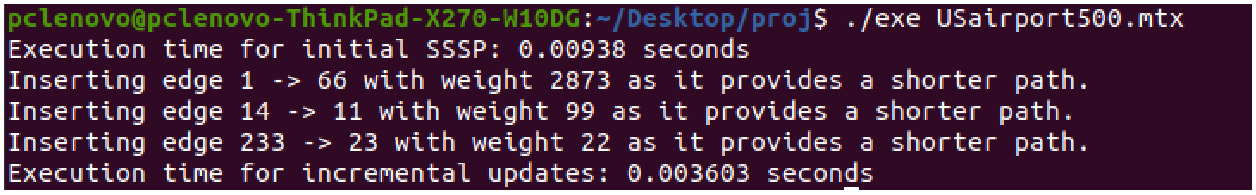
```
pclenovo@pclenovo-ThinkPad-X270-W10DG:~/Desktop/proj$ ./exe celegans_n306.mtx
Execution time for initial SSSP: 0.001047 seconds
Skipping edge 1 -> 50 with weight 3 as it doesn't offer a shorter path.
Skipping edge 3 -> 45 with weight 7 as it doesn't offer a shorter path.
Inserting edge 300 -> 308 with weight 2 as it provides a shorter path.
Execution time for incremental updates: 0.002041 seconds
```

4.OClinks_w_chars.mtx

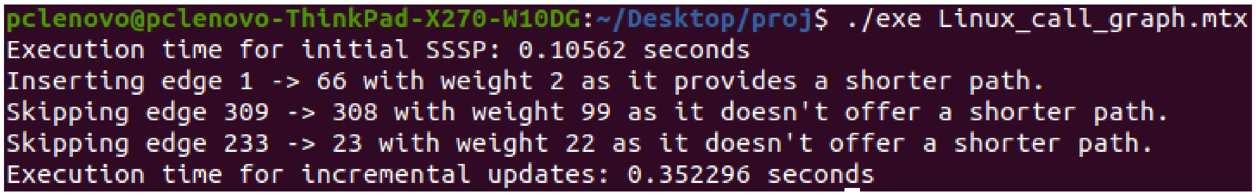


```
pclenovo@pclenovo-ThinkPad-X270-W10DG:~/Desktop/proj$ ./exe OClinks_w_chars.mtx
Execution time for initial SSSP: 0.008961 seconds
Inserting edge 1 -> 50 with weight 3 as it provides a shorter path.
Inserting edge 3 -> 45 with weight 7 as it provides a shorter path.
Inserting edge 300 -> 308 with weight 2 as it provides a shorter path.
Inserting edge 1899 -> 34 with weight 98 as it provides a shorter path.
Skipping edge 1899 -> 399 with weight 169 as it doesn't offer a shorter path.
Execution time for incremental updates: 0.004459 seconds
```

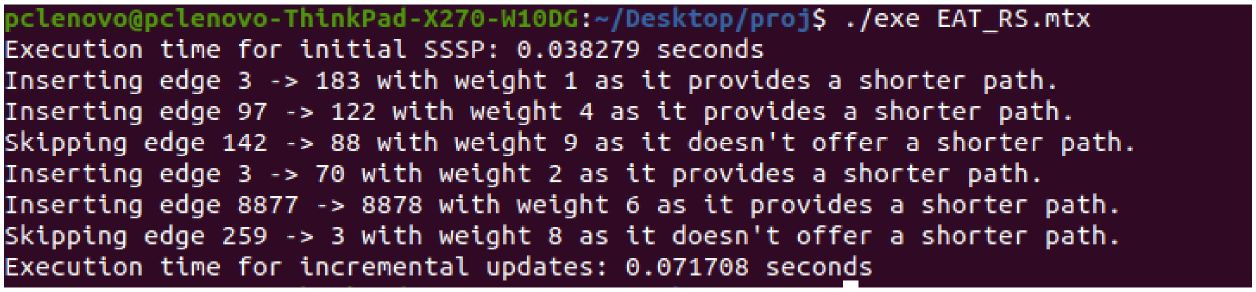
5.USairport500.mtx


```
pclenovo@pclenovo-ThinkPad-X270-W10DG:~/Desktop/proj$ ./exe USairport500.mtx
Execution time for initial SSSP: 0.00938 seconds
Inserting edge 1 -> 66 with weight 2873 as it provides a shorter path.
Inserting edge 14 -> 11 with weight 99 as it provides a shorter path.
Inserting edge 233 -> 23 with weight 22 as it provides a shorter path.
Execution time for incremental updates: 0.003603 seconds
```

6.Linus_call_graph.mtx

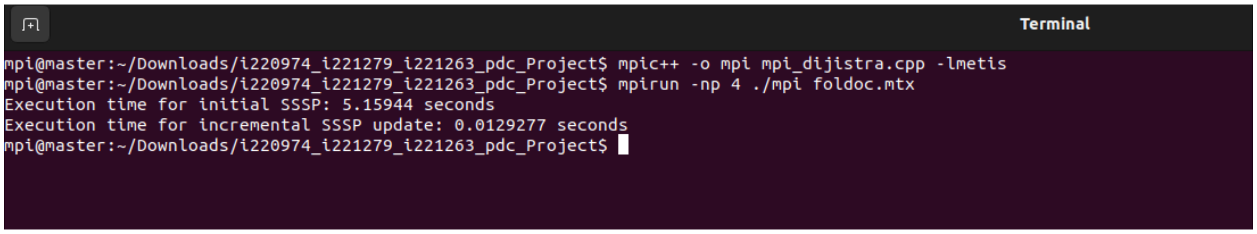

```
pclenovo@pclenovo-ThinkPad-X270-W10DG:~/Desktop/proj$ ./exe Linux_call_graph.mtx
Execution time for initial SSSP: 0.10562 seconds
Inserting edge 1 -> 66 with weight 2 as it provides a shorter path.
Skipping edge 309 -> 308 with weight 99 as it doesn't offer a shorter path.
Skipping edge 233 -> 23 with weight 22 as it doesn't offer a shorter path.
Execution time for incremental updates: 0.352296 seconds
```

7.EAT_RS.mtx

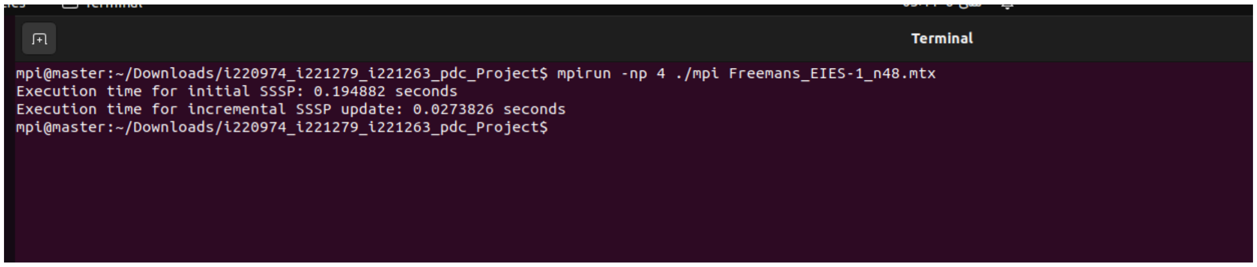

```
pclenovo@pclenovo-ThinkPad-X270-W10DG:~/Desktop/proj$ ./exe EAT_RS.mtx
Execution time for initial SSSP: 0.038279 seconds
Inserting edge 3 -> 183 with weight 1 as it provides a shorter path.
Inserting edge 97 -> 122 with weight 4 as it provides a shorter path.
Skipping edge 142 -> 88 with weight 9 as it doesn't offer a shorter path.
Inserting edge 3 -> 70 with weight 2 as it provides a shorter path.
Inserting edge 8877 -> 8878 with weight 6 as it provides a shorter path.
Skipping edge 259 -> 3 with weight 8 as it doesn't offer a shorter path.
Execution time for incremental updates: 0.071708 seconds
```

MPI:

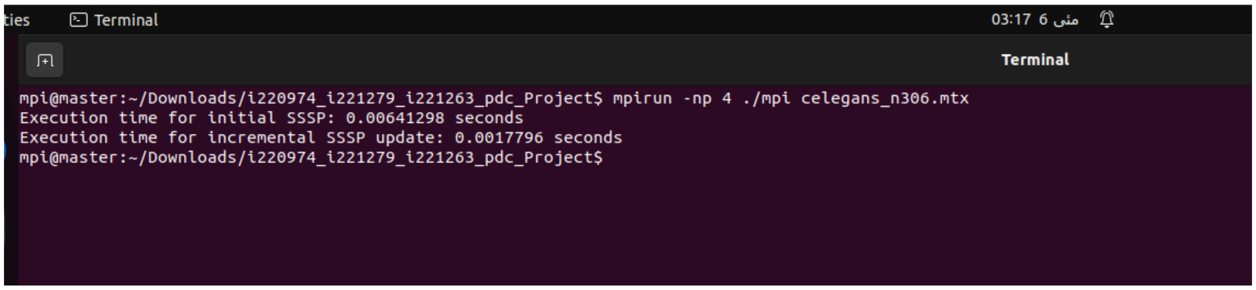
1.foldoc.mtx


Terminal
```
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$ mpic++ -o mpi mpi_dijistra.cpp -lmetis
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$ mpirun -np 4 ./mpi foldoc.mtx
Execution time for initial SSSP: 5.15944 seconds
Execution time for incremental SSSP update: 0.0129277 seconds
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$
```

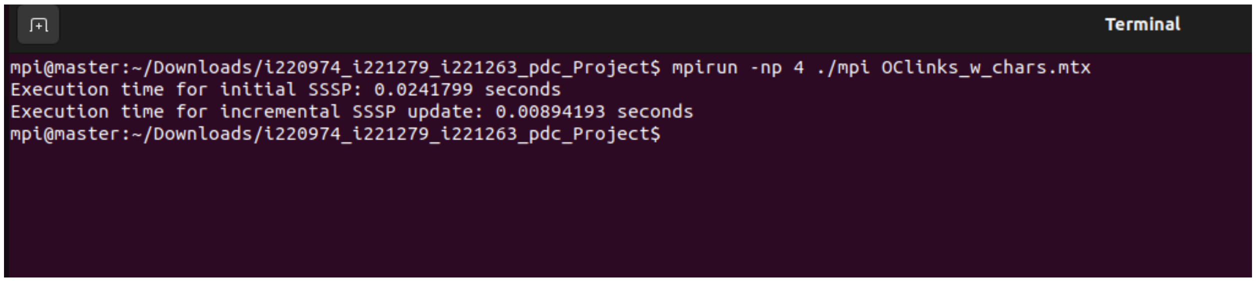
2.Freemans_EIES-1.mtx


```
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$ mpirun -np 4 ./mpi Freemans_EIES-1_n48.mtx
Execution time for initial SSSP: 0.194882 seconds
Execution time for incremental SSSP update: 0.0273826 seconds
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$
```

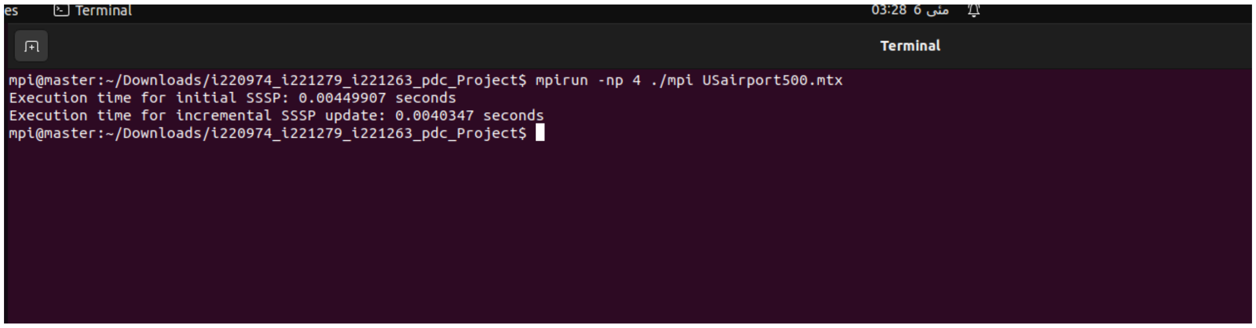
3.celegans_n360.mtx


```
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$ mpirun -np 4 ./mpi celegans_n306.mtx
Execution time for initial SSSP: 0.00641298 seconds
Execution time for incremental SSSP update: 0.0017796 seconds
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$
```

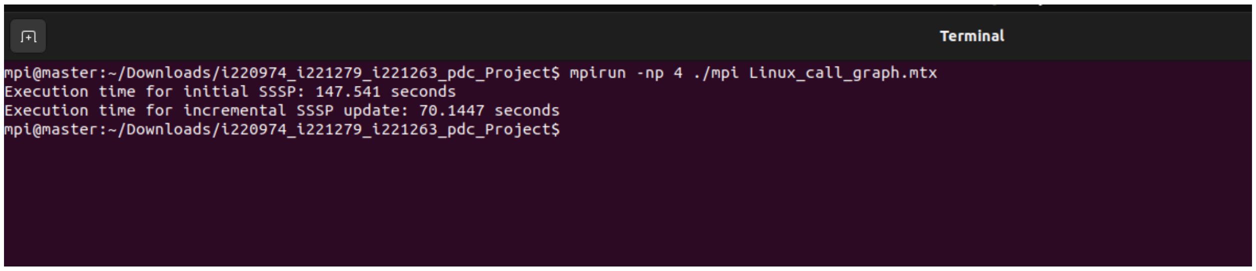
4.OClinks_w_chars.mtx


```
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$ mpirun -np 4 ./mpi OClinks_w_chars.mtx
Execution time for initial SSSP: 0.0241799 seconds
Execution time for incremental SSSP update: 0.00894193 seconds
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$
```

5.USairport500.mtx

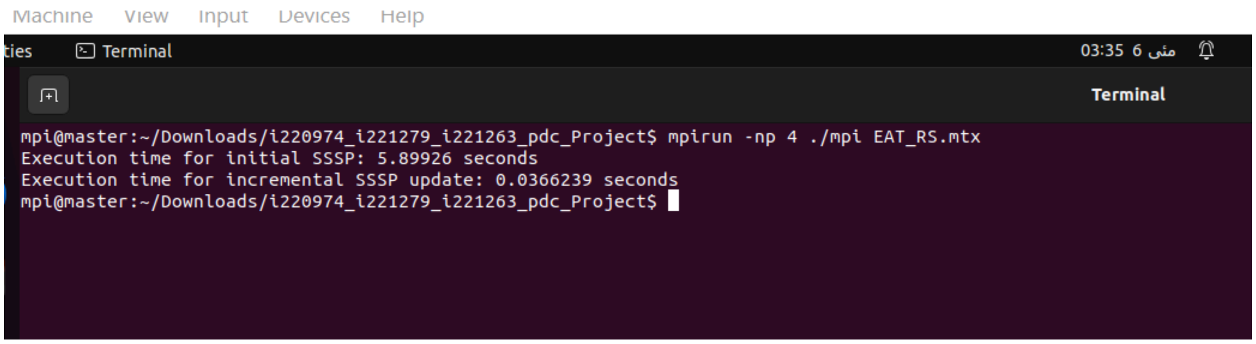

```
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$ mpirun -np 4 ./mpi USairport500.mtx
Execution time for initial SSSP: 0.00449907 seconds
Execution time for incremental SSSP update: 0.0040347 seconds
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$
```

6.Linus_call_graph.mtx



```
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$ mpirun -np 4 ./mpi Linux_call_graph.mtx
Execution time for initial SSSP: 147.541 seconds
Execution time for incremental SSSP update: 70.1447 seconds
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$
```

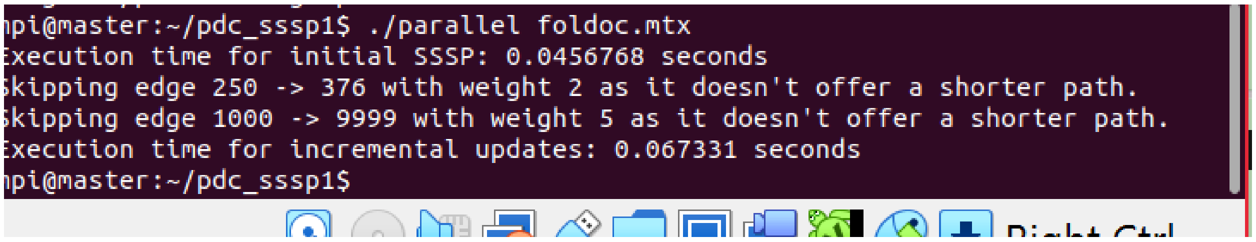
7.EAT_RS.mtx



```
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$ mpirun -np 4 ./mpi EAT_RS.mtx
Execution time for initial SSSP: 5.89926 seconds
Execution time for incremental SSSP update: 0.0366239 seconds
mpi@master:~/Downloads/i220974_i221279_i221263_pdc_Project$
```

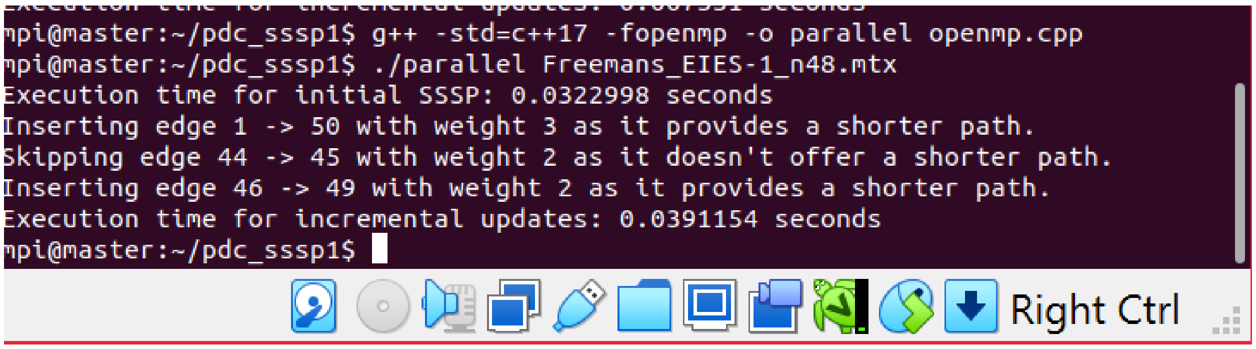
OPENMP:

1.foldoc.mtx



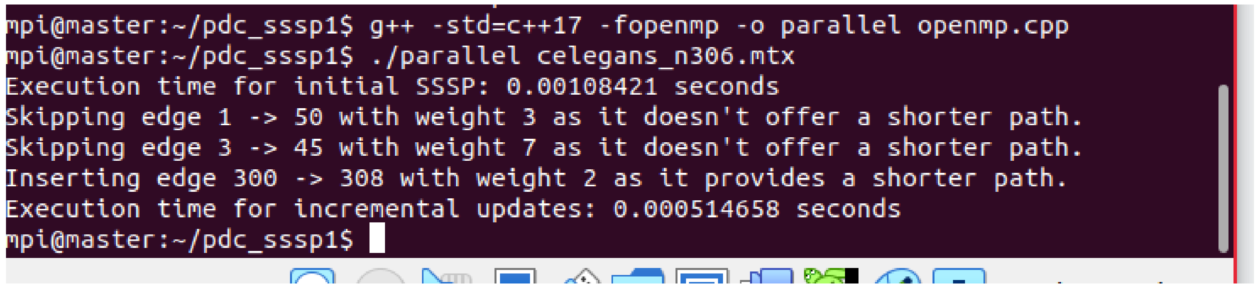
```
mpi@master:~/pdc_sssp1$ ./parallel foldoc.mtx
Execution time for initial SSSP: 0.0456768 seconds
Skipping edge 250 -> 376 with weight 2 as it doesn't offer a shorter path.
Skipping edge 1000 -> 9999 with weight 5 as it doesn't offer a shorter path.
Execution time for incremental updates: 0.067331 seconds
mpi@master:~/pdc_sssp1$
```

2.Freemans_EIES-1.mtx



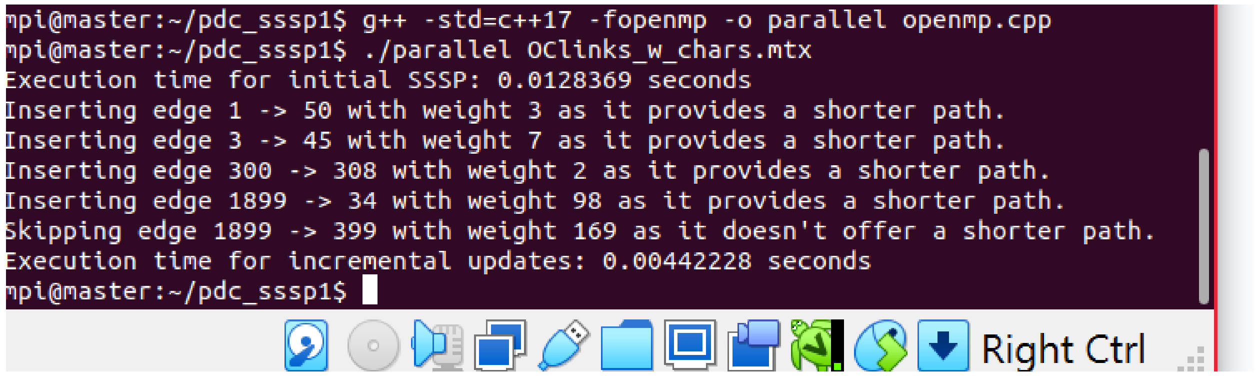
```
Execution time for incremental updates: 0.067331 seconds
mpi@master:~/pdc_sssp1$ g++ -std=c++17 -fopenmp -o parallel openmp.cpp
mpi@master:~/pdc_sssp1$ ./parallel Freemans_EIES-1_n48.mtx
Execution time for initial SSSP: 0.0322998 seconds
Inserting edge 1 -> 50 with weight 3 as it provides a shorter path.
Skipping edge 44 -> 45 with weight 2 as it doesn't offer a shorter path.
Inserting edge 46 -> 49 with weight 2 as it provides a shorter path.
Execution time for incremental updates: 0.0391154 seconds
mpi@master:~/pdc_sssp1$
```

3.celegans_n360.mtx



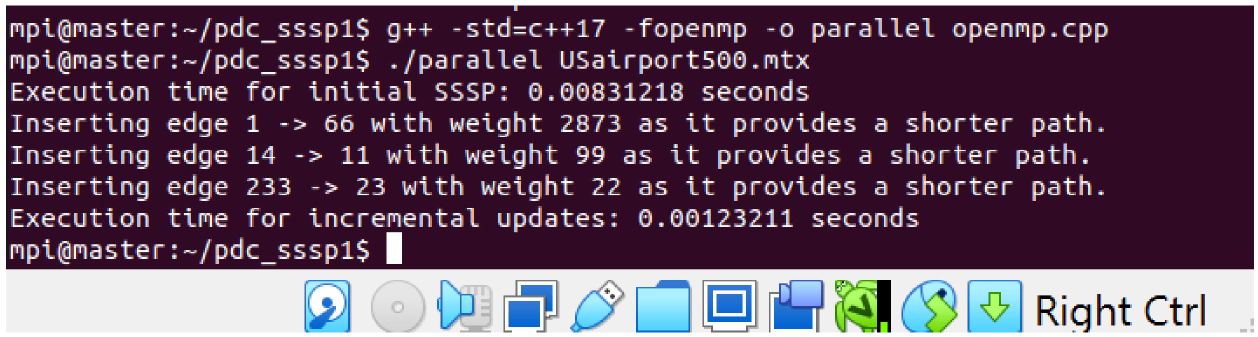
```
mpi@master:~/pdc_sssp1$ g++ -std=c++17 -fopenmp -o parallel openmp.cpp
mpi@master:~/pdc_sssp1$ ./parallel celegans_n306.mtx
Execution time for initial SSSP: 0.00108421 seconds
Skipping edge 1 -> 50 with weight 3 as it doesn't offer a shorter path.
Skipping edge 3 -> 45 with weight 7 as it doesn't offer a shorter path.
Inserting edge 300 -> 308 with weight 2 as it provides a shorter path.
Execution time for incremental updates: 0.000514658 seconds
mpi@master:~/pdc_sssp1$
```

4.OClinks_w_chars.mtx



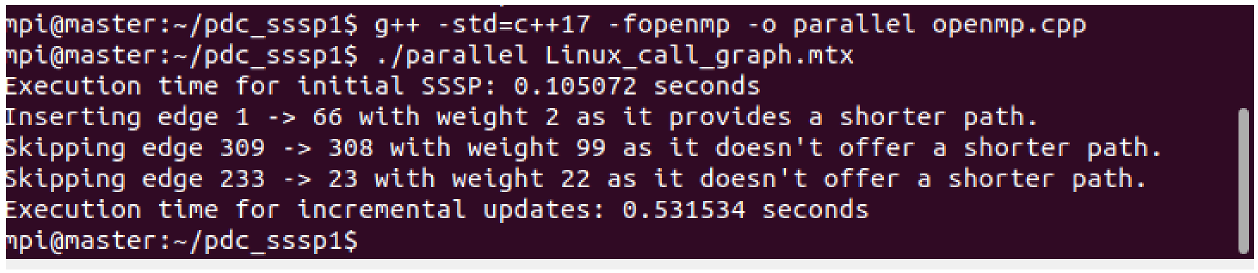
```
mpi@master:~/pdc_sssp1$ g++ -std=c++17 -fopenmp -o parallel openmp.cpp
mpi@master:~/pdc_sssp1$ ./parallel OClinks_w_chars.mtx
Execution time for initial SSSP: 0.0128369 seconds
Inserting edge 1 -> 50 with weight 3 as it provides a shorter path.
Inserting edge 3 -> 45 with weight 7 as it provides a shorter path.
Inserting edge 300 -> 308 with weight 2 as it provides a shorter path.
Inserting edge 1899 -> 34 with weight 98 as it provides a shorter path.
Skipping edge 1899 -> 399 with weight 169 as it doesn't offer a shorter path.
Execution time for incremental updates: 0.00442228 seconds
mpi@master:~/pdc_sssp1$
```

5.USairport500.mtx



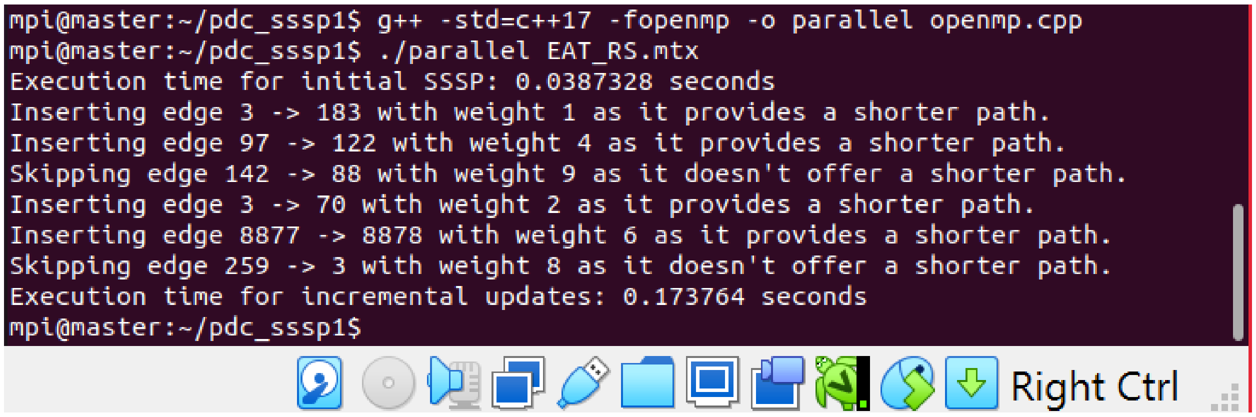
```
mpi@master:~/pdc_sssp1$ g++ -std=c++17 -fopenmp -o parallel openmp.cpp
mpi@master:~/pdc_sssp1$ ./parallel USairport500.mtx
Execution time for initial SSSP: 0.00831218 seconds
Inserting edge 1 -> 66 with weight 2873 as it provides a shorter path.
Inserting edge 14 -> 11 with weight 99 as it provides a shorter path.
Inserting edge 233 -> 23 with weight 22 as it provides a shorter path.
Execution time for incremental updates: 0.00123211 seconds
mpi@master:~/pdc_sssp1$
```

6.Linus_call_graph.mtx



```
mpi@master:~/pdc_sssp1$ g++ -std=c++17 -fopenmp -o parallel openmp.cpp
mpi@master:~/pdc_sssp1$ ./parallel Linux_call_graph.mtx
Execution time for initial SSSP: 0.105072 seconds
Inserting edge 1 -> 66 with weight 2 as it provides a shorter path.
Skipping edge 309 -> 308 with weight 99 as it doesn't offer a shorter path.
Skipping edge 233 -> 23 with weight 22 as it doesn't offer a shorter path.
Execution time for incremental updates: 0.531534 seconds
mpi@master:~/pdc_sssp1$
```

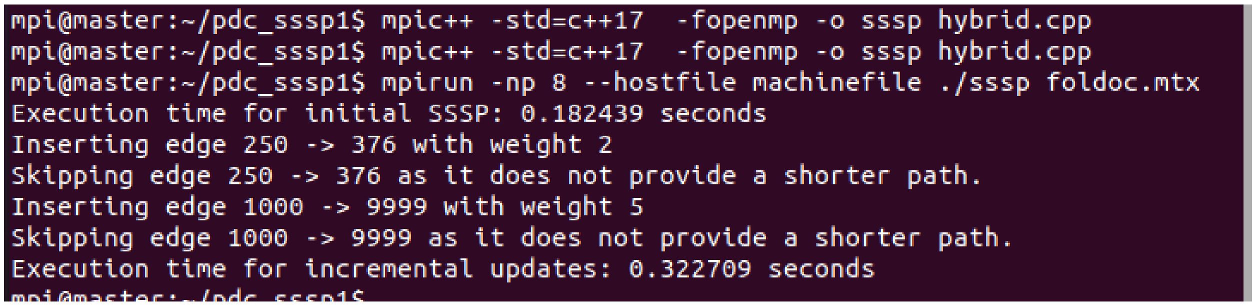
7.EAT_RS.mtx



```
mpi@master:~/pdc_sssp1$ g++ -std=c++17 -fopenmp -o parallel openmp.cpp
mpi@master:~/pdc_sssp1$ ./parallel EAT_RS.mtx
Execution time for initial SSSP: 0.0387328 seconds
Inserting edge 3 -> 183 with weight 1 as it provides a shorter path.
Inserting edge 97 -> 122 with weight 4 as it provides a shorter path.
Skipping edge 142 -> 88 with weight 9 as it doesn't offer a shorter path.
Inserting edge 3 -> 70 with weight 2 as it provides a shorter path.
Inserting edge 8877 -> 8878 with weight 6 as it provides a shorter path.
Skipping edge 259 -> 3 with weight 8 as it doesn't offer a shorter path.
Execution time for incremental updates: 0.173764 seconds
mpi@master:~/pdc_sssp1$
```

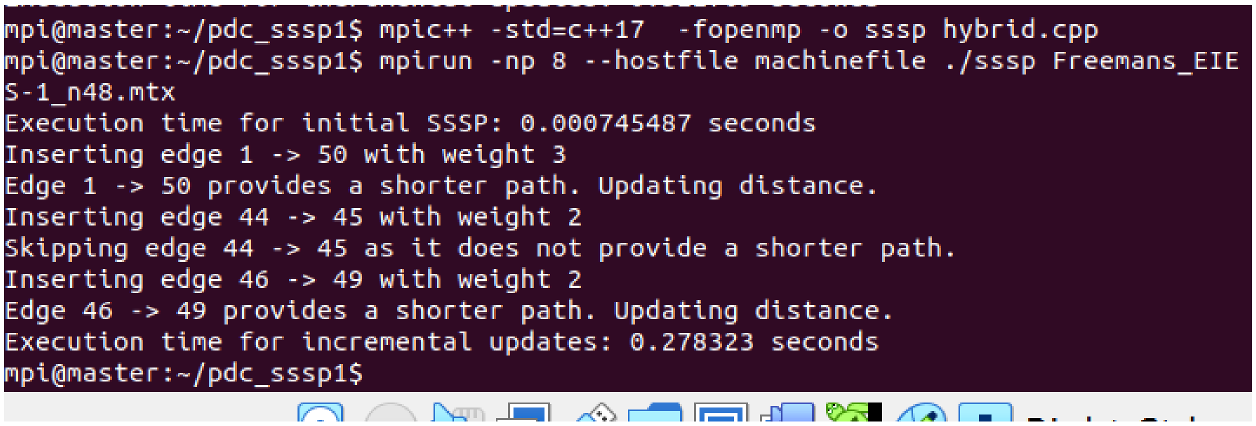
HYBRID:

1.foldoc.mtx



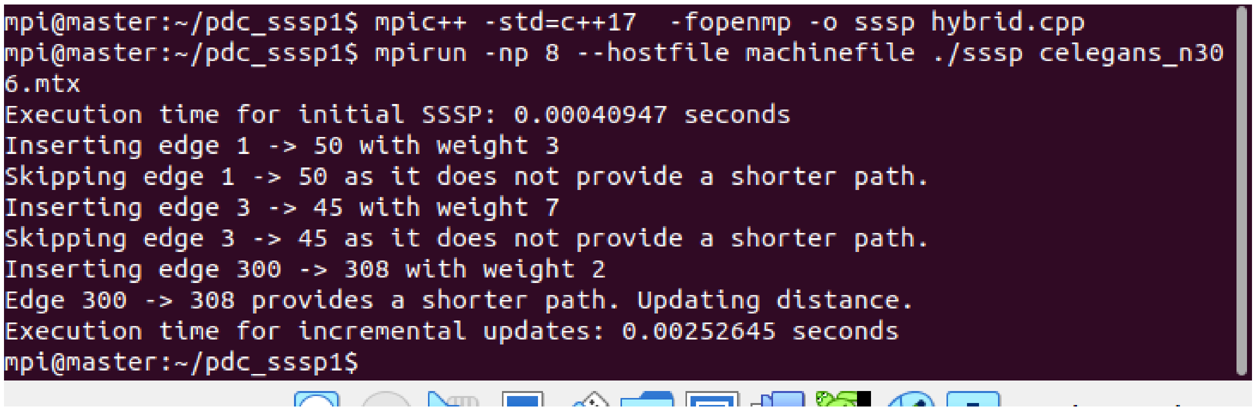
```
mpi@master:~/pdc_sssp1$ mpic++ -std=c++17  -fopenmp -o sssp hybrid.cpp
mpi@master:~/pdc_sssp1$ mpic++ -std=c++17  -fopenmp -o sssp hybrid.cpp
mpi@master:~/pdc_sssp1$ mpirun -np 8 --hostfile machinefile ./sssp foldoc.mtx
Execution time for initial SSSP: 0.182439 seconds
Inserting edge 250 -> 376 with weight 2
Skipping edge 250 -> 376 as it does not provide a shorter path.
Inserting edge 1000 -> 9999 with weight 5
Skipping edge 1000 -> 9999 as it does not provide a shorter path.
Execution time for incremental updates: 0.322709 seconds
```

2.Freemans_EIES-1.mtx



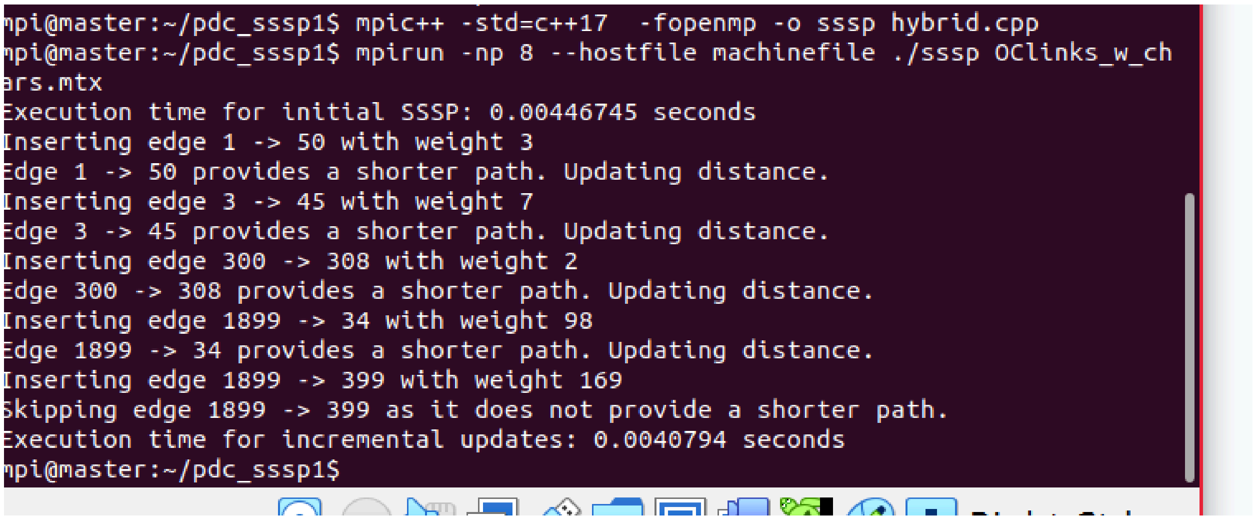
```
mpi@master:~/pdc_sssp1$ mpic++ -std=c++17  -fopenmp -o sssp hybrid.cpp
mpi@master:~/pdc_sssp1$ mpirun -np 8 --hostfile machinefile ./sssp Freemans_EIE
S-1_n48.mtx
Execution time for initial SSSP: 0.000745487 seconds
Inserting edge 1 -> 50 with weight 3
Edge 1 -> 50 provides a shorter path. Updating distance.
Inserting edge 44 -> 45 with weight 2
Skipping edge 44 -> 45 as it does not provide a shorter path.
Inserting edge 46 -> 49 with weight 2
Edge 46 -> 49 provides a shorter path. Updating distance.
Execution time for incremental updates: 0.278323 seconds
mpi@master:~/pdc_sssp1$
```

3.celegans_n360.mtx



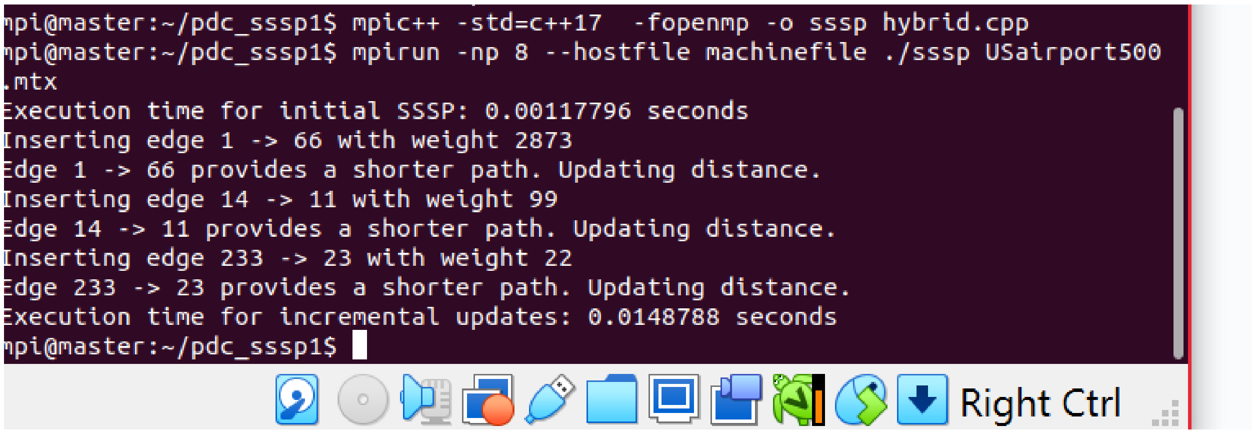
```
mpi@master:~/pdc_sssp1$ mpic++ -std=c++17  -fopenmp -o sssp hybrid.cpp
mpi@master:~/pdc_sssp1$ mpirun -np 8 --hostfile machinefile ./sssp celegans_n30
6.mtx
Execution time for initial SSSP: 0.00040947 seconds
Inserting edge 1 -> 50 with weight 3
Skipping edge 1 -> 50 as it does not provide a shorter path.
Inserting edge 3 -> 45 with weight 7
Skipping edge 3 -> 45 as it does not provide a shorter path.
Inserting edge 300 -> 308 with weight 2
Edge 300 -> 308 provides a shorter path. Updating distance.
Execution time for incremental updates: 0.00252645 seconds
mpi@master:~/pdc_sssp1$
```

4.OClinks_w_chars.mtx



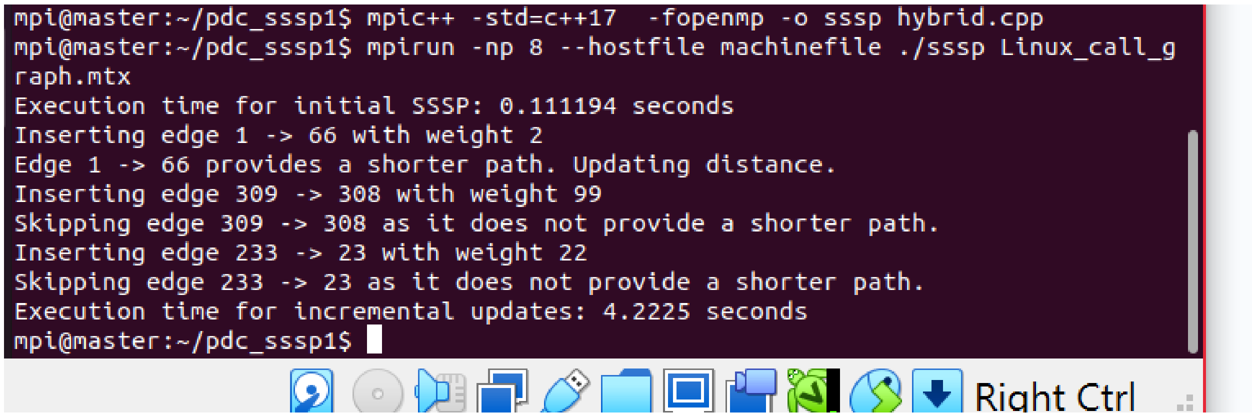
```
mpi@master:~/pdc_sssp1$ mpic++ -std=c++17  -fopenmp -o sssp hybrid.cpp
mpi@master:~/pdc_sssp1$ mpirun -np 8 --hostfile machinefile ./sssp OClinks_w_ch
ars.mtx
Execution time for initial SSSP: 0.00446745 seconds
Inserting edge 1 -> 50 with weight 3
Edge 1 -> 50 provides a shorter path. Updating distance.
Inserting edge 3 -> 45 with weight 7
Edge 3 -> 45 provides a shorter path. Updating distance.
Inserting edge 300 -> 308 with weight 2
Edge 300 -> 308 provides a shorter path. Updating distance.
Inserting edge 1899 -> 34 with weight 98
Edge 1899 -> 34 provides a shorter path. Updating distance.
Inserting edge 1899 -> 399 with weight 169
Skipping edge 1899 -> 399 as it does not provide a shorter path.
Execution time for incremental updates: 0.0040794 seconds
mpi@master:~/pdc_sssp1$
```

5.USairport500.mtx



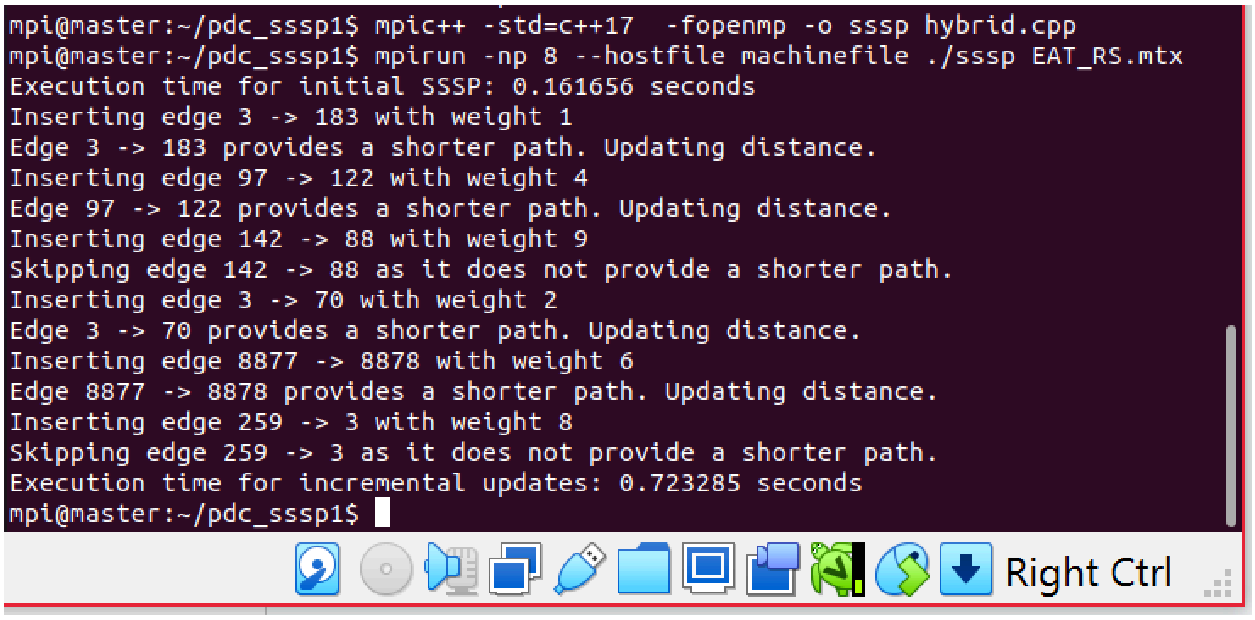
```
mpi@master:~/pdc_sssp1$ mpic++ -std=c++17  -fopenmp -o sssp hybrid.cpp
mpi@master:~/pdc_sssp1$ mpirun -np 8 --hostfile machinefile ./sssp USairport500
.mtx
Execution time for initial SSSP: 0.00117796 seconds
Inserting edge 1 -> 66 with weight 2873
Edge 1 -> 66 provides a shorter path. Updating distance.
Inserting edge 14 -> 11 with weight 99
Edge 14 -> 11 provides a shorter path. Updating distance.
Inserting edge 233 -> 23 with weight 22
Edge 233 -> 23 provides a shorter path. Updating distance.
Execution time for incremental updates: 0.0148788 seconds
mpi@master:~/pdc_sssp1$
```

6.Linus_call_graph.mtx



```
mpi@master:~/pdc_sssp1$ mpic++ -std=c++17  -fopenmp -o sssp hybrid.cpp
mpi@master:~/pdc_sssp1$ mpirun -np 8 --hostfile machinefile ./sssp Linux_call_g
raph.mtx
Execution time for initial SSSP: 0.111194 seconds
Inserting edge 1 -> 66 with weight 2
Edge 1 -> 66 provides a shorter path. Updating distance.
Inserting edge 309 -> 308 with weight 99
Skipping edge 309 -> 308 as it does not provide a shorter path.
Inserting edge 233 -> 23 with weight 22
Skipping edge 233 -> 23 as it does not provide a shorter path.
Execution time for incremental updates: 4.2225 seconds
mpi@master:~/pdc_sssp1$
```

7.EAT_RS.mtx



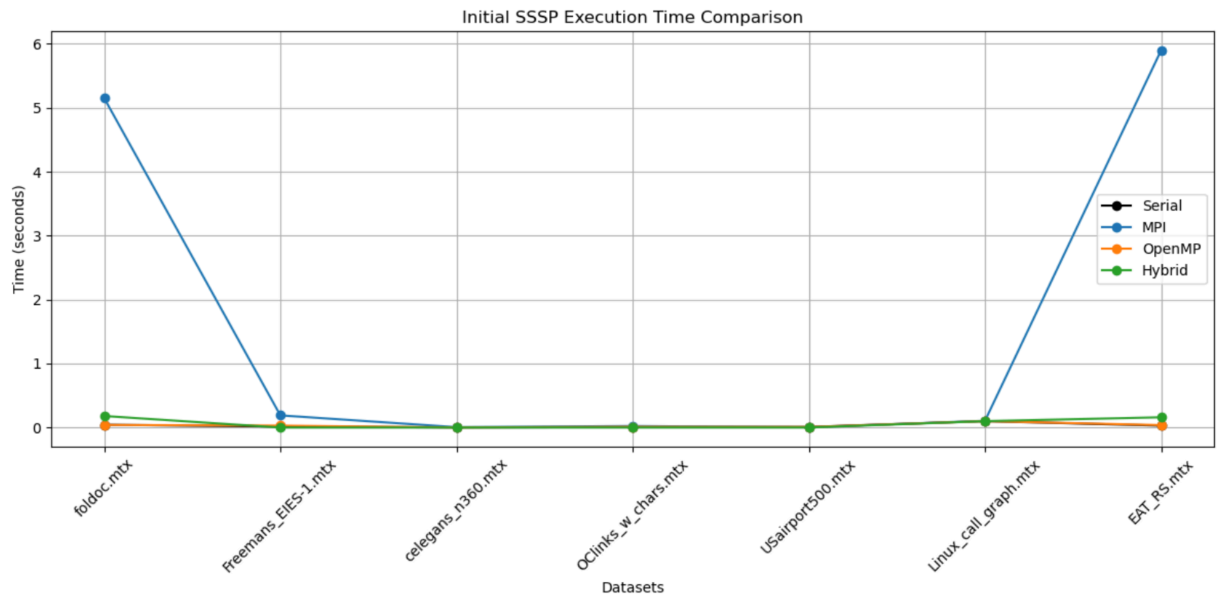
```
mpi@master:~/pdc_sssp1$ mpic++ -std=c++17  -fopenmp -o sssp hybrid.cpp
mpi@master:~/pdc_sssp1$ mpirun -np 8 --hostfile machinefile ./sssp EAT_RS.mtx
Execution time for initial SSSP: 0.161656 seconds
Inserting edge 3 -> 183 with weight 1
Edge 3 -> 183 provides a shorter path. Updating distance.
Inserting edge 97 -> 122 with weight 4
Edge 97 -> 122 provides a shorter path. Updating distance.
Inserting edge 142 -> 88 with weight 9
Skipping edge 142 -> 88 as it does not provide a shorter path.
Inserting edge 3 -> 70 with weight 2
Edge 3 -> 70 provides a shorter path. Updating distance.
Inserting edge 8877 -> 8878 with weight 6
Edge 8877 -> 8878 provides a shorter path. Updating distance.
Inserting edge 259 -> 3 with weight 8
Skipping edge 259 -> 3 as it does not provide a shorter path.
Execution time for incremental updates: 0.723285 seconds
mpi@master:~/pdc_sssp1$
```

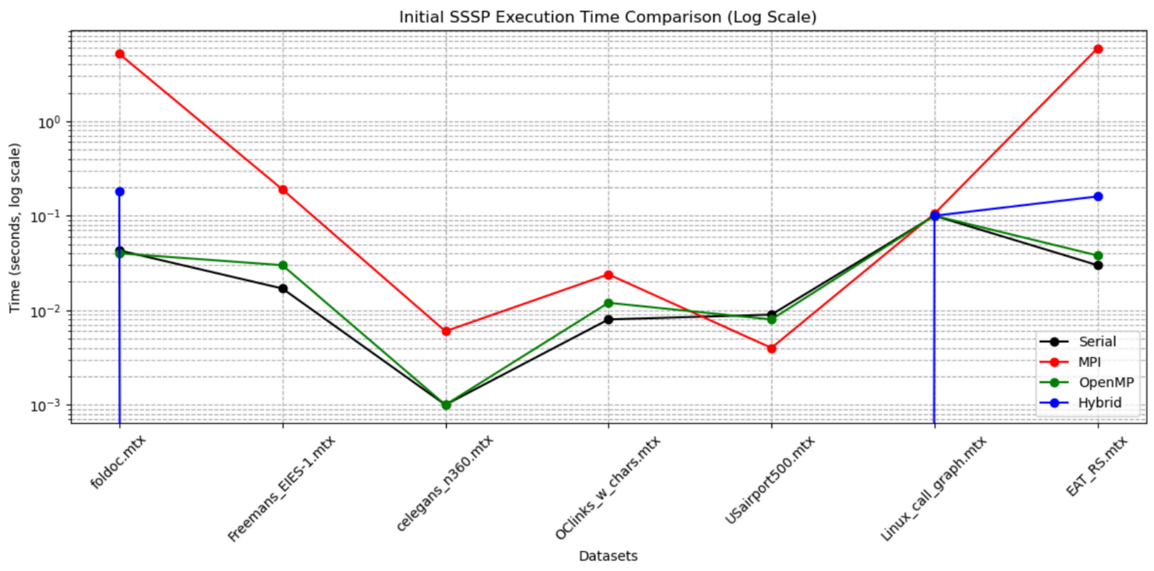
<u>TABLE:-</u>

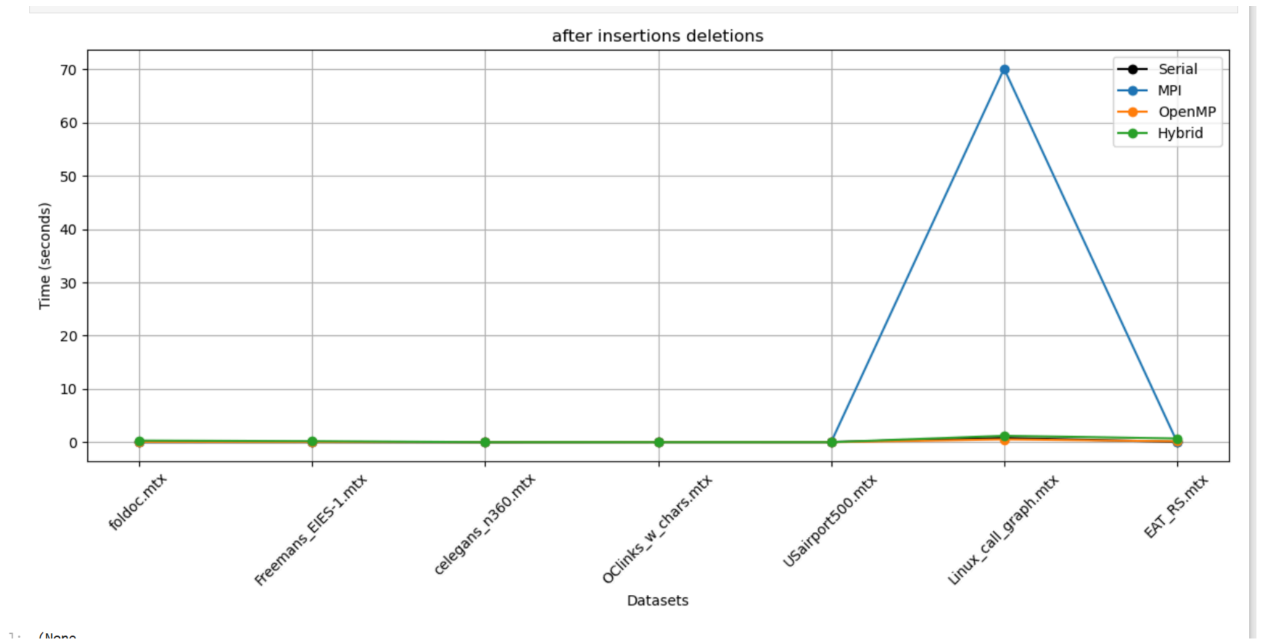
Dataset	serial	MPI	openmp	hybrid
foldoc.mtx	Initial SSP:0.043 After changes:0.012	Initial SSP:5.15 changes:0.012	Initial SSP:0.04 changes:0.06	Initial SSP:0.18 changes:0.3

Freemans_EIES-1.mtx	**Initial SSP:0.017** **After changes:0.02**	**Initial SSP:0.19** **changes:0.02**	**Initial SSP:0.03** **changes:0.039**	**Initial SSP:0.0** **changes:0.249**
celegans_n360.mtx	**Initial SSP:0.001** **After changes:0.002**	**Initial SSP:0.006** **changes:0.001**	**Initial SSP:0.001** **changes:0.0005**	**Initial SSP:0.0** **changes:0.0**
OClinks_w_chars.mtx	**Initial SSP:0.008** **After changes:0.004**	**Initial SSP:0.024** **changes:0.008**	**Initial SSP:0.012** **changes:0.004**	**Initial SSP:0.0** **changes:0.0**
USairport500.mtx	**Initial SSP:0.009** **After changes:0.003**	**Initial SSP:0.004** **changes:0.004**	**Initial SSP:0.008** **changes:0.001**	**Initial SSP:0.0** **changes:0.0**
Linux_call_graph.mtx	**Initial SSP:0.10** **After changes:0.35**	**Initial SSP:0.105** **changes:70.14**	**Initial SSP:0.1** **changes:0.53**	**Initial SSP:0.1** **changes:4.2**
EAT_RS.mtx	**Initial SSP:0.03** **After changes:0.07**	**Initial SSP:5.899** **changes:0.036**	**Initial SSP:0.038** **changes:0.17**	**Initial SSP:0.16** **changes:0.7**

GRAPH(pandas):-


Initial SSSP Execution Time Comparison
Time (seconds)
Datasets
Serial
MPI
OpenMP
Hybrid
foldoc.mtx
Freemans_EIES-1.mtx
celegans_n360.mtx
OClinks_w_chars.mtx
USairport500.mtx
Linux_call_graph.mtx
EAT_RS.mtx


Initial SSSP Execution Time Comparison (Log Scale)
Time (seconds, log scale)
Datasets
Serial
MPI
OpenMP
Hybrid
foldoc.mtx
Freemans_EIES-1.mtx
celegans_n360.mtx
OClinks_w_chars.mtx
USairport500.mtx
Linux_call_graph.mtx
EAT_RS.mtx



Based on the execution time comparison across multiple datasets, the **Hybrid approach** demonstrates the most consistent and efficient performance overall. While the MPI method sometimes shows faster results for specific datasets, it also exhibits significant variability and higher execution times in others. In contrast, the Hybrid implementation maintains a low execution time across almost all datasets, combining the strengths of both MPI and OpenMP. This makes it a **reliable and scalable choice** for parallelizing Single Source Shortest Path (SSSP) computations on large and diverse graph structures.

Scalability Analysis

1. Serial Approach:

- **Not scalable.**
- Runs on a single core with no parallelism.
- As dataset size grows, execution time increases significantly.
- Best suited for small datasets or initial testing, but **not practical** for real-time or large-scale graph processing.

2. OpenMP Approach:

- **Moderately scalable.**

- Uses shared-memory parallelism, which works well on multi-core CPUs.
- Performance improves over Serial, especially on medium-sized datasets.
- Limited by the number of cores in a single machine.
- Scalability **plateaus** as the number of threads increases beyond CPU cores due to overhead and contention.

3. MPI Approach:

- **Highly scalable in theory.**
- Utilizes distributed memory parallelism, allowing scaling across multiple machines.
- Ideal for large graphs and distributed clusters.
- However, overhead from inter-process communication can become a bottleneck, especially on smaller datasets or when not balanced properly.
- **Scalability depends heavily on efficient task distribution and communication strategy.**

4. Hybrid Approach (MPI + OpenMP):

- **Most scalable approach overall.**
- Combines the strengths of MPI and OpenMP: distributed and shared memory parallelism.
- Can scale both across nodes and within nodes (multi-core systems).
- Performs well on both small and large datasets, with lower execution time and better resource utilization.
- Minimizes the weaknesses of each individual model by balancing load and reducing communication overhead.

Conclusion:

The Hybrid approach is the most scalable, making it the best choice for real-world applications involving dynamic or large-scale graphs. It ensures low execution time, efficient CPU usage, and adaptability across various hardware configurations.

GPROF ANALYSIS:-

Files are in this drive :-

https://drive.google.com/drive/folders/1sV2Ghg57j331vJhy8kmoj6-T7nGLp2TV