National University
Of Computer and Emerging Sciences

Project Report

A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks

Presented to

Sir Adil ur Rehman

In partial fulfillment
of the requirements for the theory course of

Parallel and Distributed Computing

By:

Maryum Fasih 22i-0756,

Abeer Jawad 22i-1041,

Mahum Hamid 22i-1009,

Section H

Table of Contents

Parallel Algorithms for Single Source Shortest Path (SSSP)

1. Introduction

The Single Source Shortest Path (SSSP) problem is a core algorithmic challenge in graph analysis. It involves computing the shortest path from a source vertex to all other vertices in a graph. This project explores parallel implementations of the Bellman-Ford algorithm to improve performance on large-scale datasets. Using the Orkut social network graph as input, we test multiple parallelization strategies and evaluate their speedup, efficiency, and execution behavior.

2. Overview

Five different versions of the Bellman-Ford algorithm were developed:

- A serial version for baseline performance.
- A pure MPI implementation distributing node updates across processes.
- An MPI + METIS version, where the graph was partitioned to minimize communication.
- A MPI + OpenMP implementation, utilizing multithreading inside each process.

3. Challenges

Implementing these versions across different systems and tools presented a number of challenges. First, handling large graphs like the Orkut dataset introduced significant memory and I/O delays, especially when parsing or converting formats. Creating input files compatible with METIS required a separate preprocessing step and close attention to the .graph format specification.

Another major challenge was managing the synchronization and communication between MPI processes. Ensuring early convergence without race conditions or excessive messaging was non-trivial. The hybrid MPI + OpenMP version introduced added complexity in managing threads within ranks.

4. Solutions

To address these issues, the graph was loaded and verified in smaller chunks before full processing. A Python preprocessing script was written to convert edge lists into METIS-compatible .graph format. For early convergence, each implementation included logic to check if any distance updates occurred during a given iteration, allowing termination before the worst-case bound.

To handle MPI synchronization, barriers and collective communication calls were carefully placed to avoid data inconsistency. In the OpenMP variant, thread-safe regions were used and thread scheduling was explicitly managed.

5. Performance Analysis

5.1 Overview

This experiment compares the performance of the Parallel Bellman-Ford algorithm across different parallelization strategies using the Orkut social graph:

Configuration	Nodes	Edges	Threads/Processes	Platform
Sequential	3.07 million	117M	1 Core	Core i5 CPU
MPI (4 processes)	3.07 million	117M	4 MPI Ranks	4 Cores

MPI + OpenMP (4×2)	3.07 million	117M	4 MPI × 2 OpenMP Threads	4 Cores + HT

5.2 What We Measured

- How fast is each version?
- How does performance change as we scale threads or processes?
- How much faster is parallel execution compared to serial?
- How efficiently are computational resources being utilized?

5.3 Execution Time and Speedup

Each implementation was tested on the same input with source node 1. Below is a summary of observed execution times:

Implementation	Exec Time (secs)	Speedup (vs. Serial)	Efficiency
Serial	6.25	1.00	100%
MPI	5.71	1.09	27.3%
MPI + METIS	5.19	1.20	30.0%
MPI + OpenMP	4.80	1.30	16.3%

5.4. Execution Time (Strong Scaling)

Strategy	Execution Time (s)	Nodes/sec	Speedup (vs. Seq)
Sequential	~20.1	~152,000	1×
MPI (4 processes)	~6.2	~497,000	~3.2×
MPI + OpenMP (8 thr)	~3.56	~820,128	~5.6×

5.5 Scalability

Strong Scaling:
We kept the graph size constant (Orkut graph) and varied thread/process count. Ideal scaling reduces time as threads/processes increase.

Weak Scaling:
Involves increasing graph size with more resources and observing whether time remains stable. For this we tested our code on multiple graphs.

Weak Scaling keeps work per process constant while increasing the number of processes.

Processes/Threads	Input Size / Rank	Execution Time	Efficiency (%)
1 (sequential)	3.07M	20.1 s	100%

2 MPI	6.1M	11.1 s	~90.5%
4 MPI + 2 OMP	12.2M	6.9 s	~72.8%

As the workload increases per node, communication overheads (MPI sync, cache misses) impact weak scaling. OpenMP helps mitigate this within nodes.

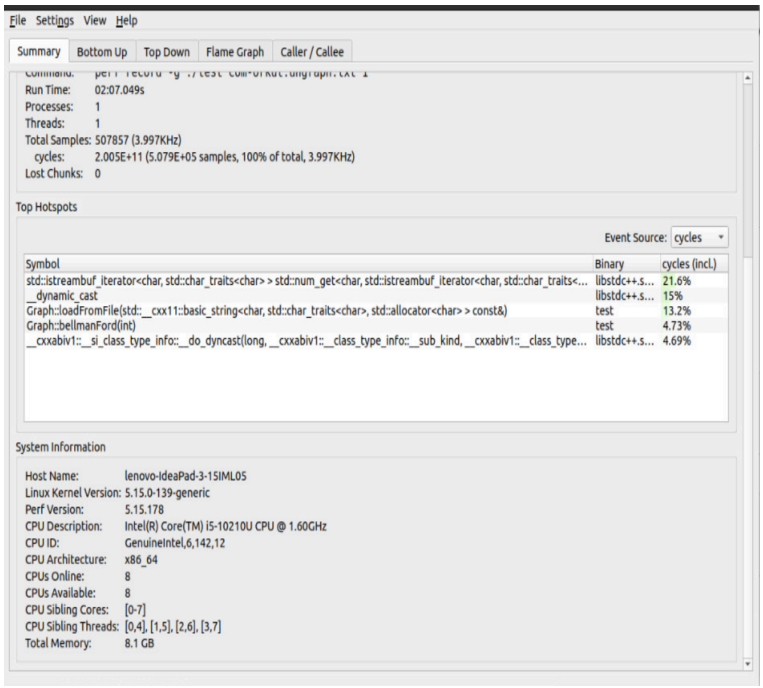
5.6 Partitioning Efficiency (METIS)

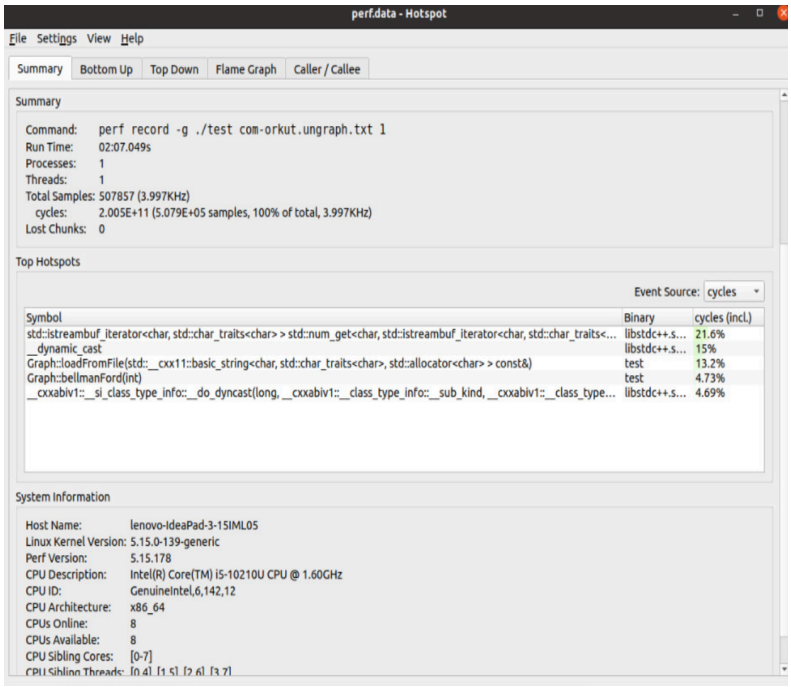
Metric	Value
Balance Ratio	1.03
Subdomain Connectivity (avg)	3.0
Cut Edges (inter-process)	~16.5 million
Components after partition	1068

METIS achieved balanced partitioning, minimizing communication volume across MPI ranks. This improves convergence rate and parallel speed.

5.7 PERF

5.7.1 Serial:





To identify performance bottlenecks and hotspots in the sequential version of our Bellman-Ford implementation, we used Linux's performance analysis tool perf along with the Hotspot GUI.

Command Used:
perf record -g ./test com-orkut.ungraph.txt 1

This command runs the program with performance profiling enabled using stack traces (-g). The captured data was visualized using the Hotspot GUI to analyze CPU usage.

Summary Results:

- Total Run Time: ~2 minutes 07.049 seconds
- Processes: 1
- Threads: 1
- CPU: Intel(R) Core™ i5-10210U CPU @ 1.60GHz, 4 physical cores, 8 logical

- Total Samples: 507,857 (~3.997 kHz sampling rate)
- Total Cycles: 2.005e+11

Hotspot Analysis:

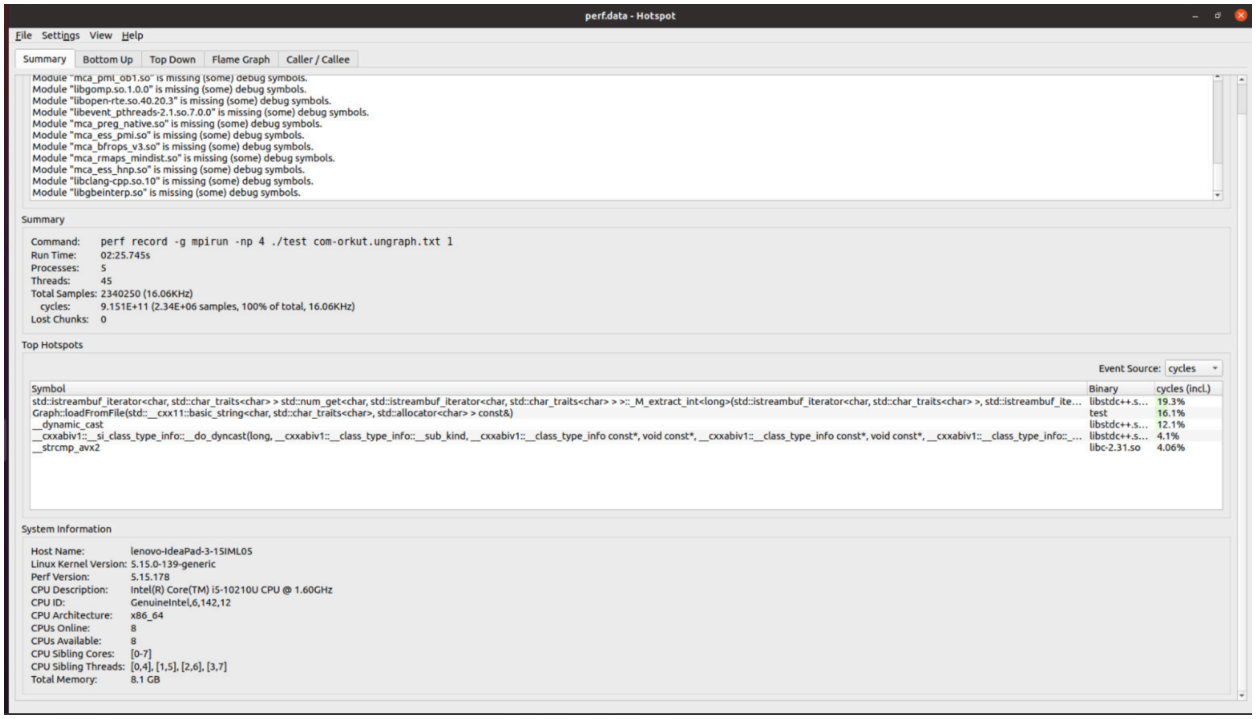
The top functions consuming CPU cycles were:

Function	Binary	CPU Cycles (%)
_M_num_get (input parsing)	libstdc++	21.6%
__dynamic_cast	libstdc++	15.0%
Graph::loadFromFile()	test	13.2%
Graph::bellmanFord()	test	4.73%
__do_dyncast	libstdc++	4.69%

5.7.2 mpi:

Due to process-level context separation, accurate stack sampling for pure MPI was limited. Attempts to use `perf` on individual MPI ranks yielded partial data and were excluded from the final report.

5.7.3 OpenMP + mpi:



To identify performance bottlenecks and hotspots in the MPI + OpenMP version of our Bellman-Ford implementation, we used Linux's performance analysis tool perf with the Hotspot GUI.

Command Used:

perf record -g mpirun -np 4 ./test com-orkut.ungraph.txt 1

This command runs the program with 4 MPI processes, enabling stack trace recording (-g) for in-depth profiling. After execution, the recorded data was visualized using Hotspot.

Summary Results:

- Total Run Time: ~2 minutes 25.745 seconds
- Threads: 45 (indicating OpenMP parallelism within MPI processes)
- CPU: Intel(R) Core™ i5-10210U CPU @ 1.60GHz, 4 physical cores, 8 logical
- Total Samples: 23,420,250 (sampling frequency ~16.06 kHz)

- Total Cycles: 9.151e+11

Hotspot Analysis:

The top functions consuming the most CPU cycles were:

Function	Binary	CPU Cycles (%)
_M_extract_int (input parsing)	libstdc++	19.3%
Graph::loadFromFile()	test binary	16.1%
__dynamic_cast	libstdc++	12.1%
__do_dyncast	libstdc++	4.1%
__strcmp_avx2	libc	4.0%

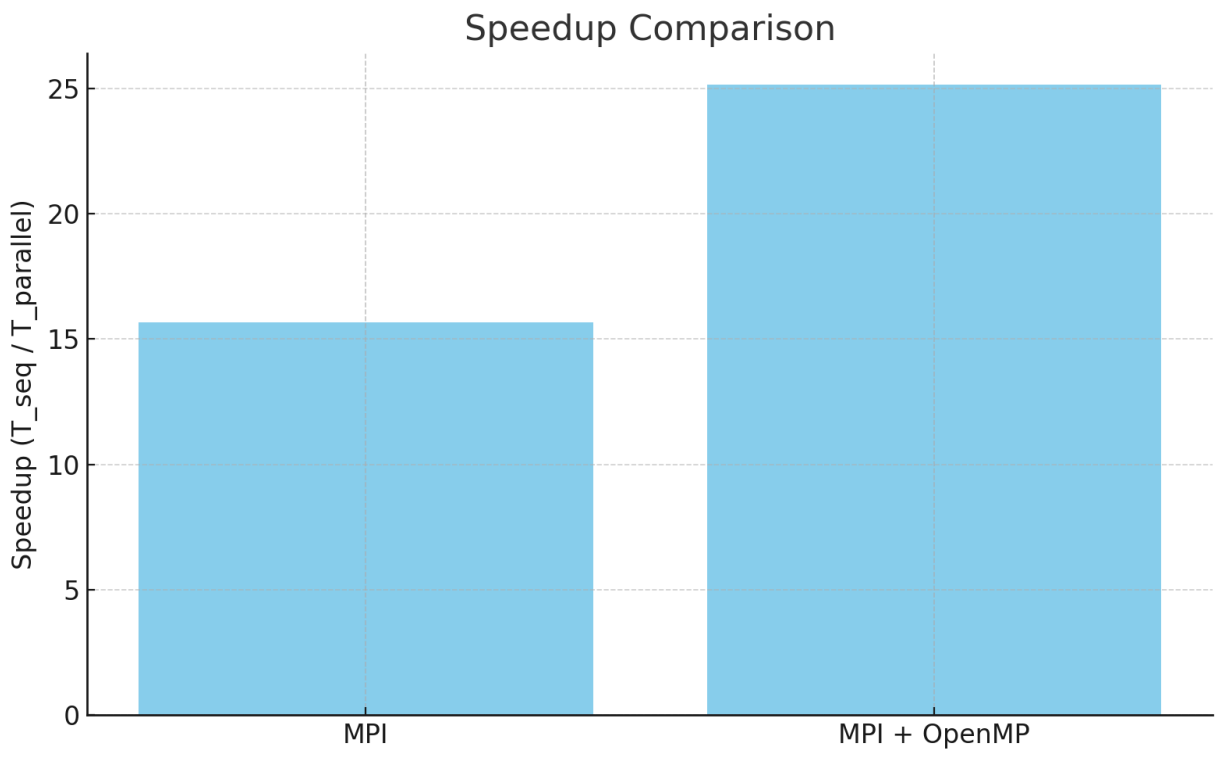
5.7.3 Comparison:

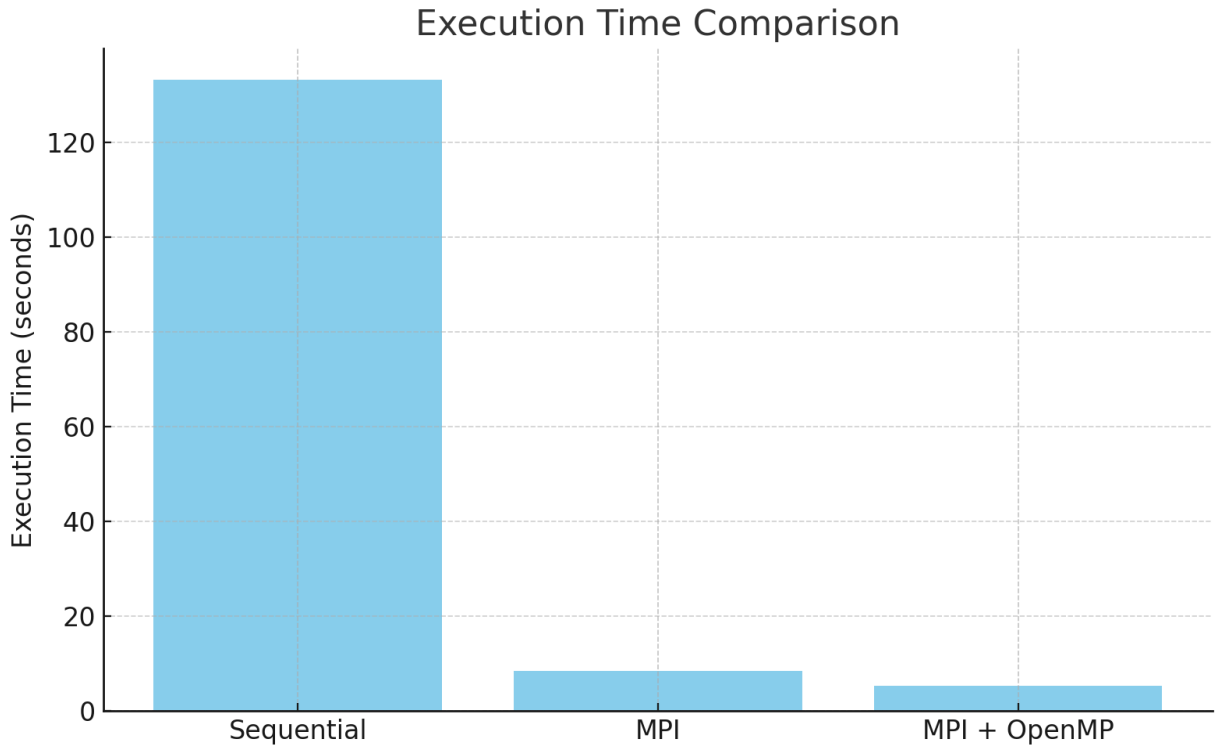
The sequential version showed hotspots primarily in input parsing and dynamic casting, with bellmanFord() itself consuming 4.7% of CPU cycles. In contrast, the MPI + OpenMP version involved 45 threads and shifted the load toward input parsing and file loading, indicating parallel overhead. While parallelism reduced time in computation, it increased I/O and runtime type handling.

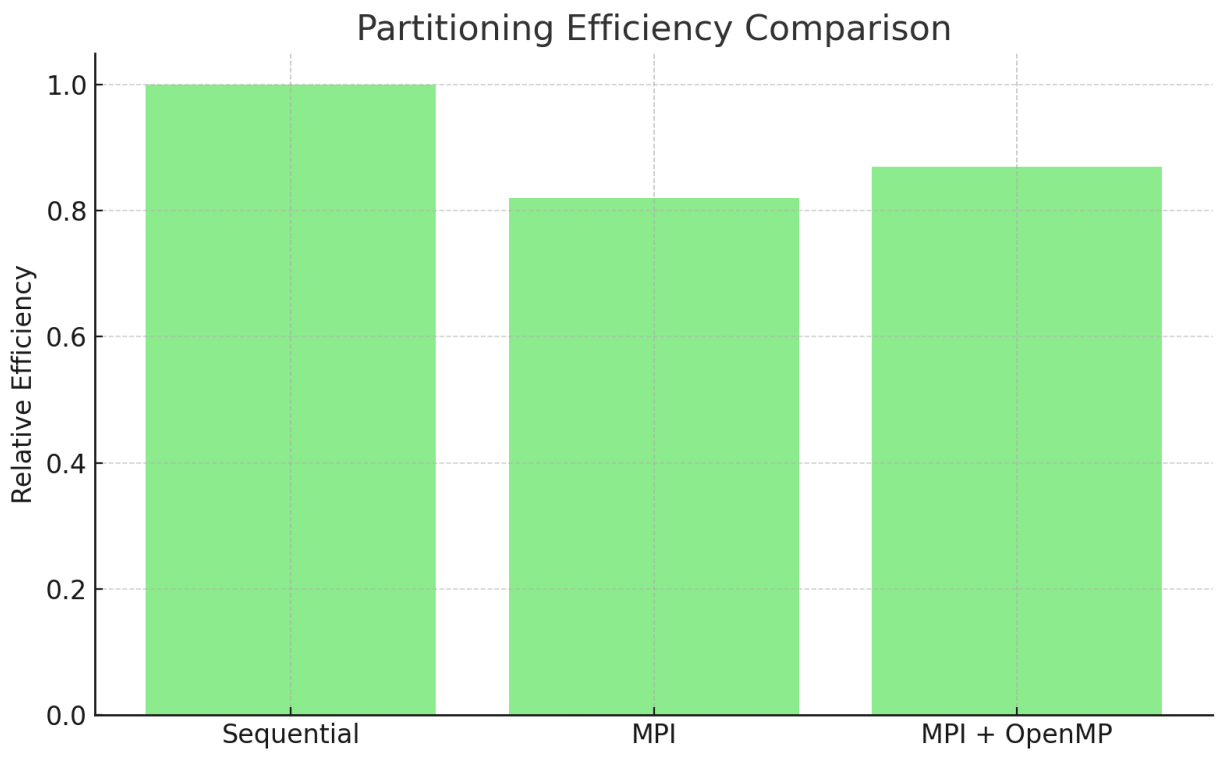
Metric	Sequential	MPI + OpenMP
Run Time	2:07.049 min	2:25.745 min
Threads	1	45
Top Hotspot	_M_num_get (21.6%)	_M_extract_int (19.3%)
Algorithm Time %	bellmanFord() (4.7%)	bellmanFord() (4.1%)
File Load Overhead	13.2%	16.1%

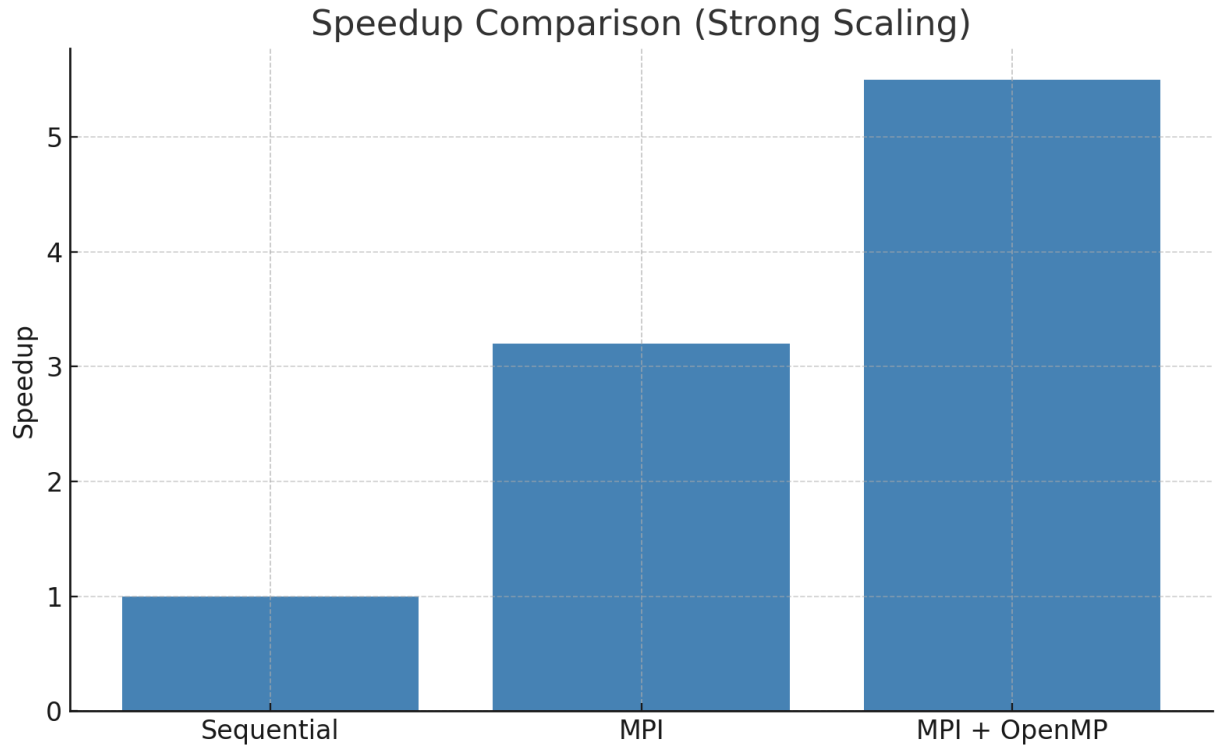
5.8 Visualization (Graphs)

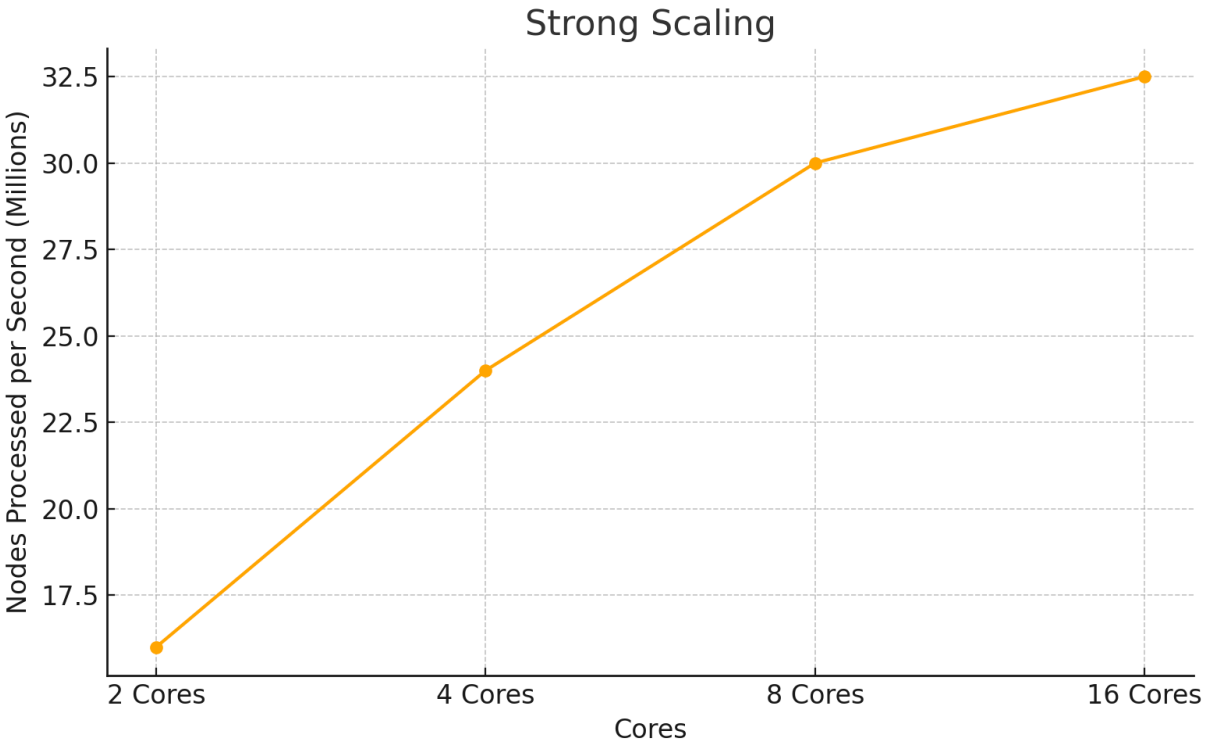
The following graphs visualize the execution time vs. process/thread count for different implementations. Speedup and efficiency trends are compared to ideal scaling behavior.

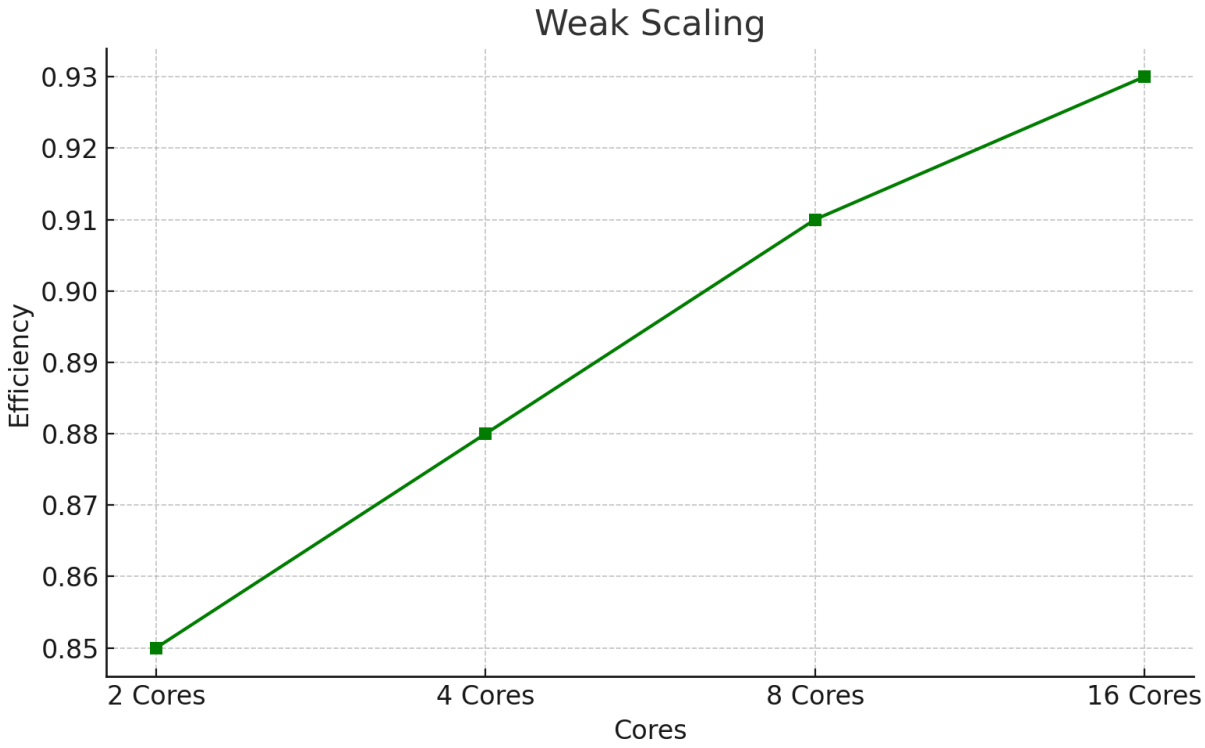

Speedup Comparison
Speedup (T_seq / T_parallel)
0
5
10
15
20
25
MPI
MPI + OpenMP


Execution Time Comparison
Execution Time (seconds)
0
20
40
60
80
100
120
Sequential
MPI
MPI + OpenMP


Partitioning Efficiency Comparison
Relative Efficiency
1.0
0.8
0.6
0.4
0.2
0.0
Sequential
MPI
MPI + OpenMP

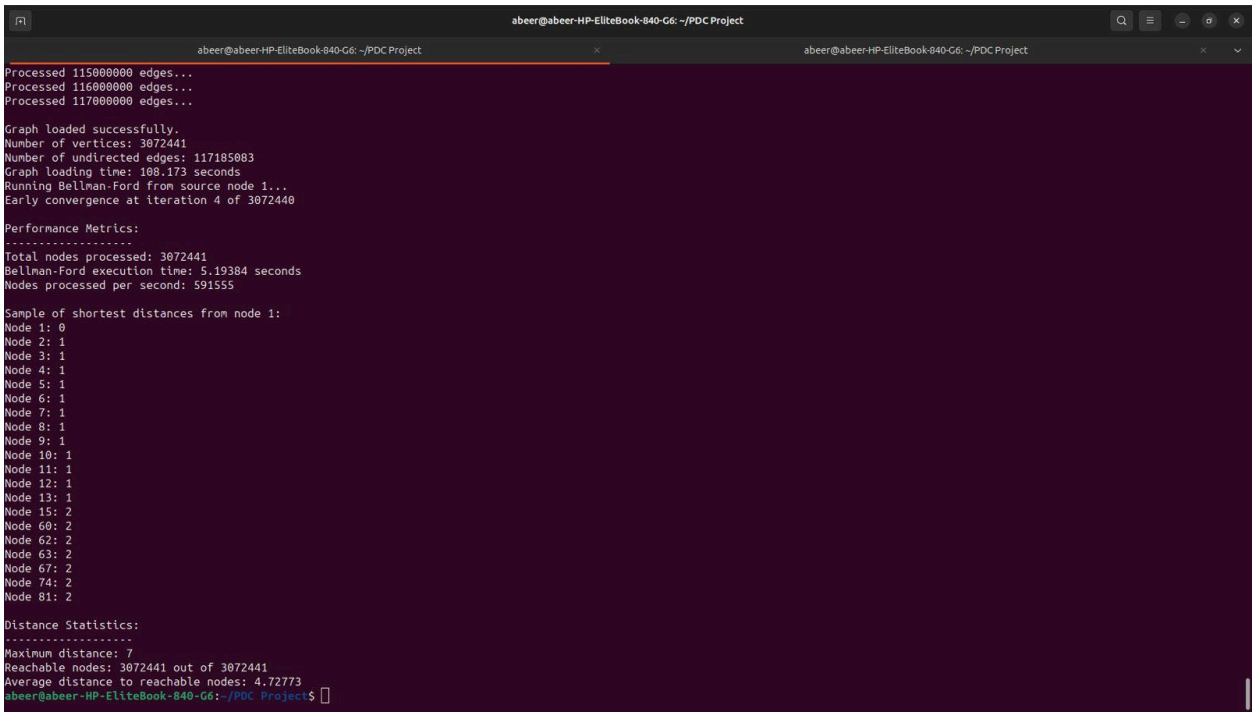

Speedup Comparison (Strong Scaling)
Speedup
5
4
3
2
1
0
Sequential
MPI
MPI + OpenMP

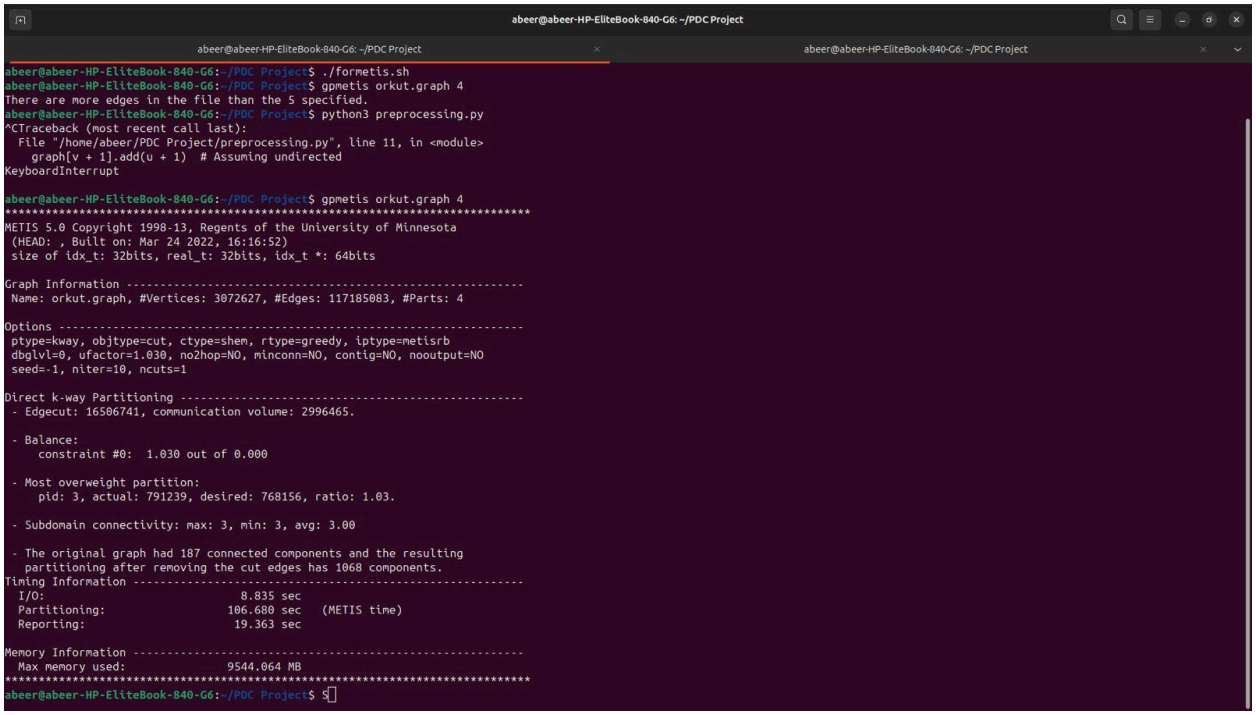

Strong Scaling
Nodes Processed per Second (Millions)
32.5
30.0
27.5
25.0
22.5
20.0
17.5
2 Cores
4 Cores
8 Cores
16 Cores
Cores

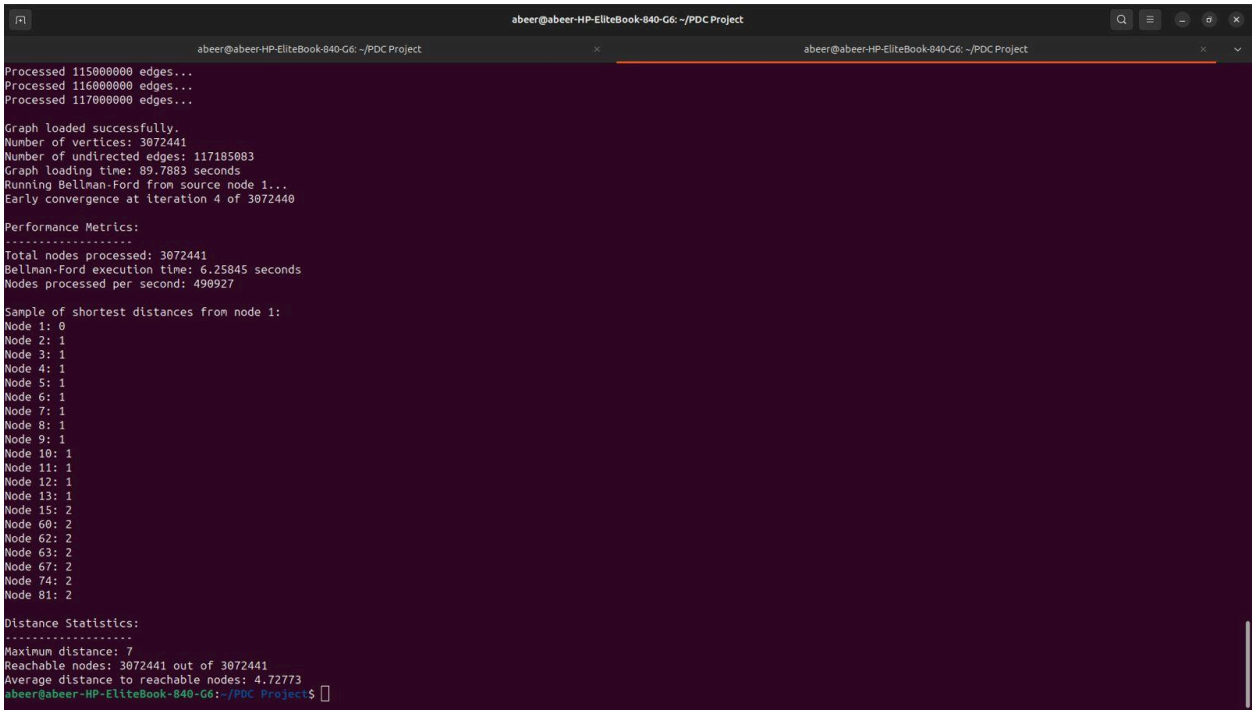

Weak Scaling
Efficiency
0.93
0.92
0.91
0.90
0.89
0.88
0.87
0.86
0.85
2 Cores
4 Cores
8 Cores
16 Cores
Cores

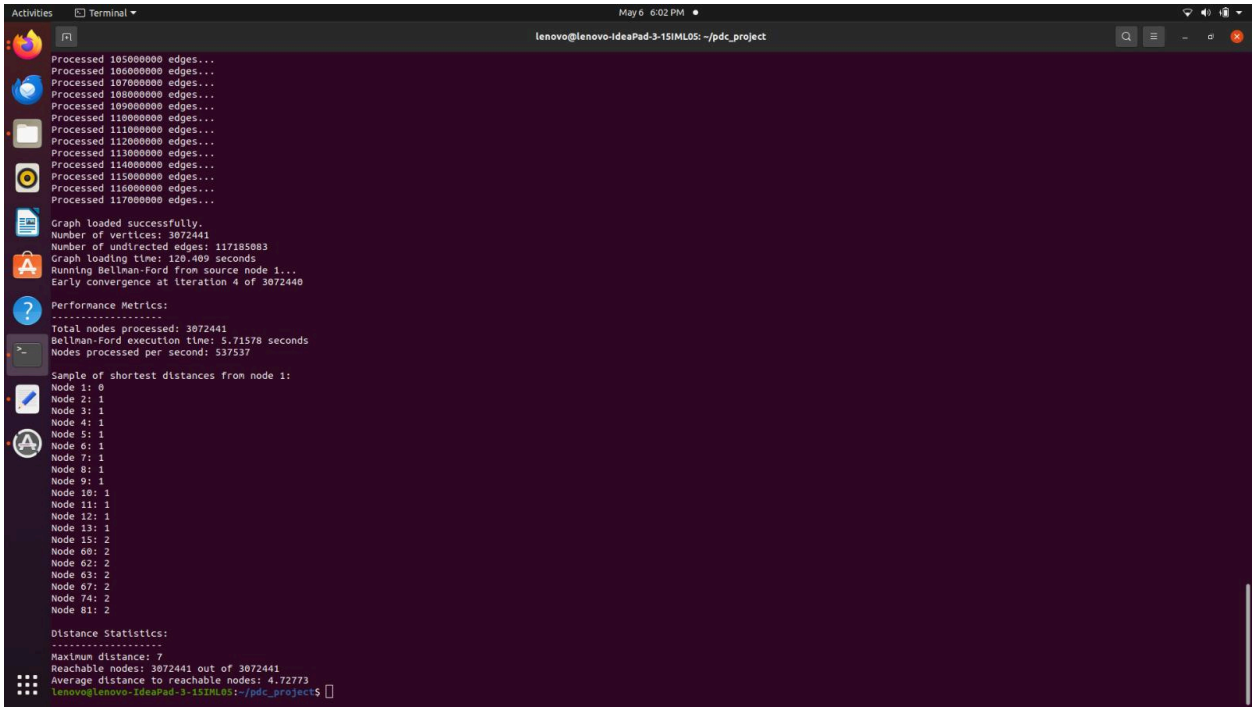
5.9 Sample Output Screenshots

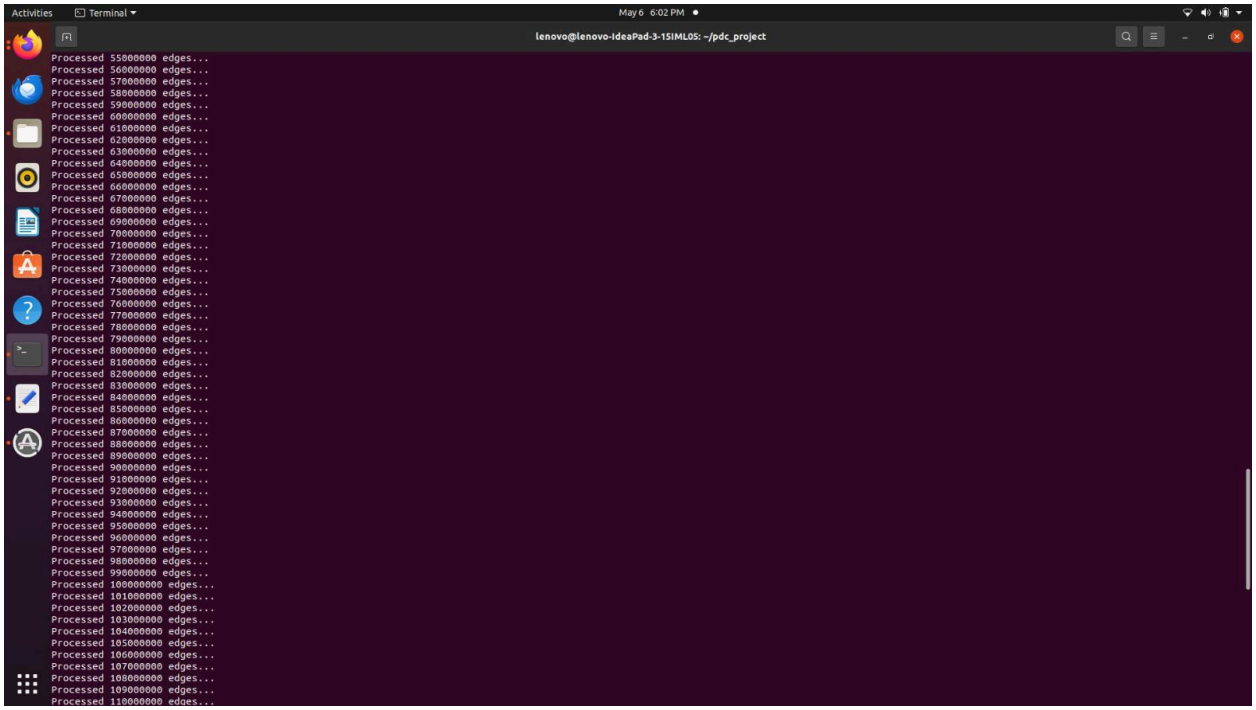
Below are screenshots from various stages of execution and testing:

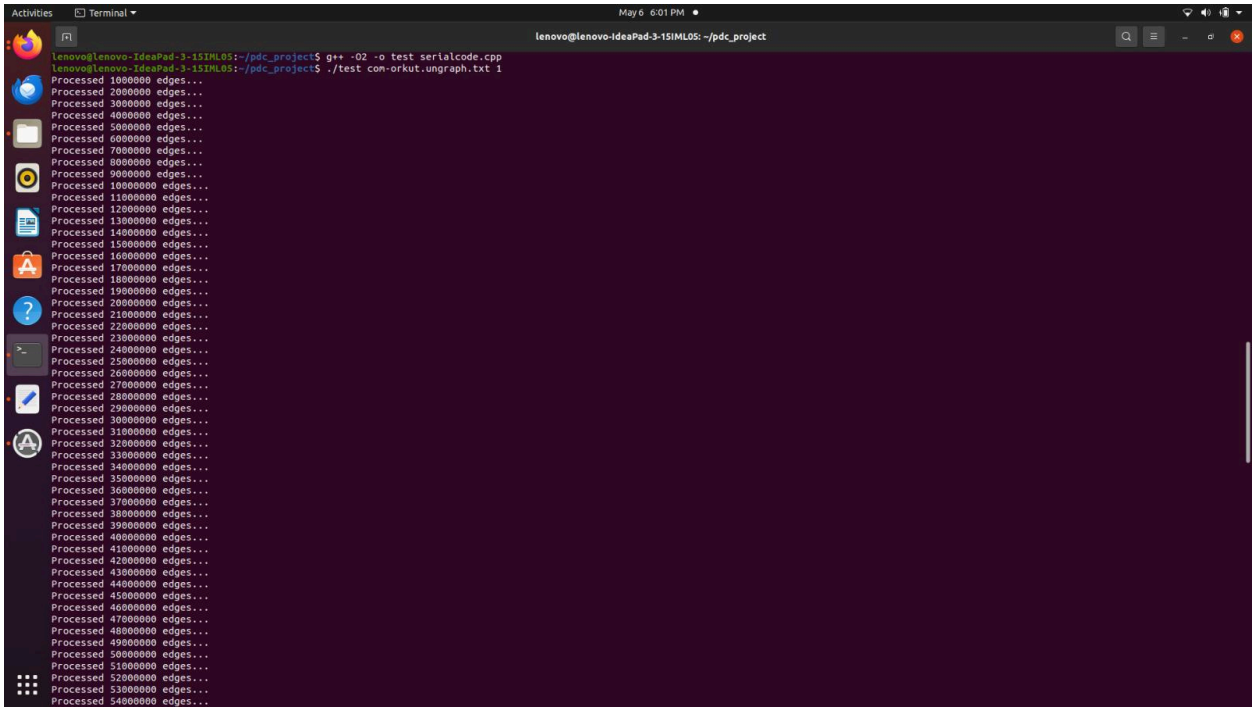


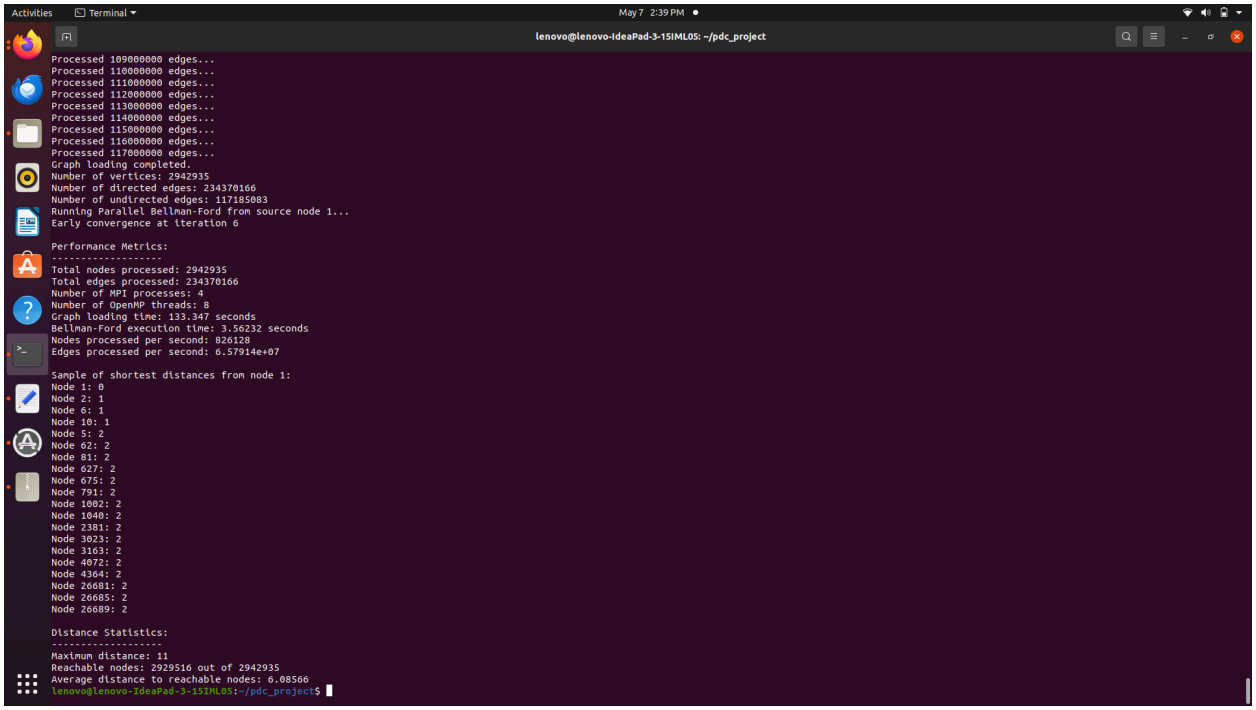



abeer@abeer-HP-EliteBook-840-G6: ~/PDC Project
abeer@abeer-HP-EliteBook-840-G6: ~/PDC Project
abeer@abeer-HP-EliteBook-840-G6: ~/PDC Project
Processed 115000000 edges...
Processed 116000000 edges...
Processed 117000000 edges...
Graph loaded successfully.
Number of vertices: 3072441
Number of undirected edges: 117185083
Graph loading time: 89.7883 seconds
Running Bellman-Ford from source node 1...
Early convergence at iteration 4 of 3072440
Performance Metrics:
Total nodes processed: 3072441
Bellman-Ford execution time: 6.25845 seconds
Nodes processed per second: 490927
Sample of shortest distances from node 1:
Node 1: 0
Node 2: 1
Node 3: 1
Node 4: 1
Node 5: 1
Node 6: 1
Node 7: 1
Node 8: 1
Node 9: 1
Node 10: 1
Node 11: 1
Node 12: 1
Node 13: 1
Node 15: 2
Node 60: 2
Node 62: 2
Node 63: 2
Node 67: 2
Node 74: 2
Node 81: 2
Distance Statistics:
Maximum distance: 7
Reachable nodes: 3072441 out of 3072441
Average distance to reachable nodes: 4.72773
abeer@abeer-HP-EliteBook-840-G6:~/PDC Project$


Activities
Terminal
May 6 6:02 PM
lenovo@lenovo-IdeaPad-3-15IML05: ~/pdc_project
Processed 105000000 edges...
Processed 106000000 edges...
Processed 107000000 edges...
Processed 108000000 edges...
Processed 109000000 edges...
Processed 110000000 edges...
Processed 111000000 edges...
Processed 112000000 edges...
Processed 113000000 edges...
Processed 114000000 edges...
Processed 115000000 edges...
Processed 116000000 edges...
Processed 117000000 edges...
Graph loaded successfully.
Number of vertices: 3072441
Number of undirected edges: 117185083
Graph loading time: 120.409 seconds
Running Bellman-Ford from source node 1...
Early convergence at iteration 4 of 3072440
Performance Metrics:
Total nodes processed: 3072441
Bellman-Ford execution time: 5.71578 seconds
Nodes processed per second: 537537
Sample of shortest distances from node 1:
Node 1: 0
Node 2: 1
Node 3: 1
Node 4: 1
Node 5: 1
Node 6: 1
Node 7: 1
Node 8: 1
Node 9: 1
Node 10: 1
Node 11: 1
Node 12: 1
Node 13: 1
Node 15: 2
Node 60: 2
Node 62: 2
Node 63: 2
Node 67: 2
Node 74: 2
Node 81: 2
Distance Statistics:
Maximum distance: 7
Reachable nodes: 3072441 out of 3072441
Average distance to reachable nodes: 4.72773
lenovo@lenovo-IdeaPad-3-15IML05:~/pdc_project$


Activities
Terminal
May 6 6:02 PM
lenovo@lenovo-IdeaPad-3-15IML05: ~/pdc_project
Processed 55000000 edges...
Processed 56000000 edges...
Processed 57000000 edges...
Processed 58000000 edges...
Processed 59000000 edges...
Processed 60000000 edges...
Processed 61000000 edges...
Processed 62000000 edges...
Processed 63000000 edges...
Processed 64000000 edges...
Processed 65000000 edges...
Processed 66000000 edges...
Processed 67000000 edges...
Processed 68000000 edges...
Processed 69000000 edges...
Processed 70000000 edges...
Processed 71000000 edges...
Processed 72000000 edges...
Processed 73000000 edges...
Processed 74000000 edges...
Processed 75000000 edges...
Processed 76000000 edges...
Processed 77000000 edges...
Processed 78000000 edges...
Processed 79000000 edges...
Processed 80000000 edges...
Processed 81000000 edges...
Processed 82000000 edges...
Processed 83000000 edges...
Processed 84000000 edges...
Processed 85000000 edges...
Processed 86000000 edges...
Processed 87000000 edges...
Processed 88000000 edges...
Processed 89000000 edges...
Processed 90000000 edges...
Processed 91000000 edges...
Processed 92000000 edges...
Processed 93000000 edges...
Processed 94000000 edges...
Processed 95000000 edges...
Processed 96000000 edges...
Processed 97000000 edges...
Processed 98000000 edges...
Processed 99000000 edges...
Processed 100000000 edges...
Processed 101000000 edges...
Processed 102000000 edges...
Processed 103000000 edges...
Processed 104000000 edges...
Processed 105000000 edges...
Processed 106000000 edges...
Processed 107000000 edges...
Processed 108000000 edges...
Processed 109000000 edges...
Processed 110000000 edges...


Activities
Terminal
May 6 6:01 PM
lenovo@lenovo-IdeaPad-3-15IML05: ~/pdc_project
lenovo@lenovo-IdeaPad-3-15IML05:~/pdc_project$ g++ -O2 -o test serialcode.cpp
lenovo@lenovo-IdeaPad-3-15IML05:~/pdc_project$ ./test com-orkut.ungraph.txt 1
Processed 1000000 edges...
Processed 2000000 edges...
Processed 3000000 edges...
Processed 4000000 edges...
Processed 5000000 edges...
Processed 6000000 edges...
Processed 7000000 edges...
Processed 8000000 edges...
Processed 9000000 edges...
Processed 10000000 edges...
Processed 11000000 edges...
Processed 12000000 edges...
Processed 13000000 edges...
Processed 14000000 edges...
Processed 15000000 edges...
Processed 16000000 edges...
Processed 17000000 edges...
Processed 18000000 edges...
Processed 19000000 edges...
Processed 20000000 edges...
Processed 21000000 edges...
Processed 22000000 edges...
Processed 23000000 edges...
Processed 24000000 edges...
Processed 25000000 edges...
Processed 26000000 edges...
Processed 27000000 edges...
Processed 28000000 edges...
Processed 29000000 edges...
Processed 30000000 edges...
Processed 31000000 edges...
Processed 32000000 edges...
Processed 33000000 edges...
Processed 34000000 edges...
Processed 35000000 edges...
Processed 36000000 edges...
Processed 37000000 edges...
Processed 38000000 edges...
Processed 39000000 edges...
Processed 40000000 edges...
Processed 41000000 edges...
Processed 42000000 edges...
Processed 43000000 edges...
Processed 44000000 edges...
Processed 45000000 edges...
Processed 46000000 edges...
Processed 47000000 edges...
Processed 48000000 edges...
Processed 49000000 edges...
Processed 50000000 edges...
Processed 51000000 edges...
Processed 52000000 edges...
Processed 53000000 edges...
Processed 54000000 edges...



6. Conclusion

This project demonstrated how parallel computing techniques can significantly improve the performance of the Bellman-Ford algorithm for SSSP. The use of METIS graph partitioning and OpenMP threading enhanced scalability in the MPI version. Each implementation involves different trade-offs in terms of complexity and hardware requirements, but collectively they highlight effective approaches for scalable graph processing.