Aazib Abdullah 22i-1031
Amir Akbar 22i-1112
Haris Ahmed 22i-1124
Section-J

PDC – PROJECT
(Phase -II)

Parallel Dijkstra's Algorithm – Report

Selected Paper:

We selected the paper **"Parallel Shortest Path Algorithms for Large Graphs"** (author: Julian Shun, Guy Blelloch, et al.) which explores various parallelization strategies for Dijkstra's algorithm and evaluates them on large-scale real-world graphs. The paper provided inspiration for comparing scalability of OpenMP and MPI-based implementations in shared and distributed memory environments.

Implementation Details:

MPI Implementation

- **Whole Graph Replication:** All MPI processes read the full graph.
- **Work Distribution:** Source node updates are distributed logically—each process helps update distances and communicate changes.
- **Communication:** Uses non-blocking MPI_Isend / MPI_Irecv to share updates. Synchronization via MPI_Allreduce ensures consistent global state.

OpenMP Implementation:

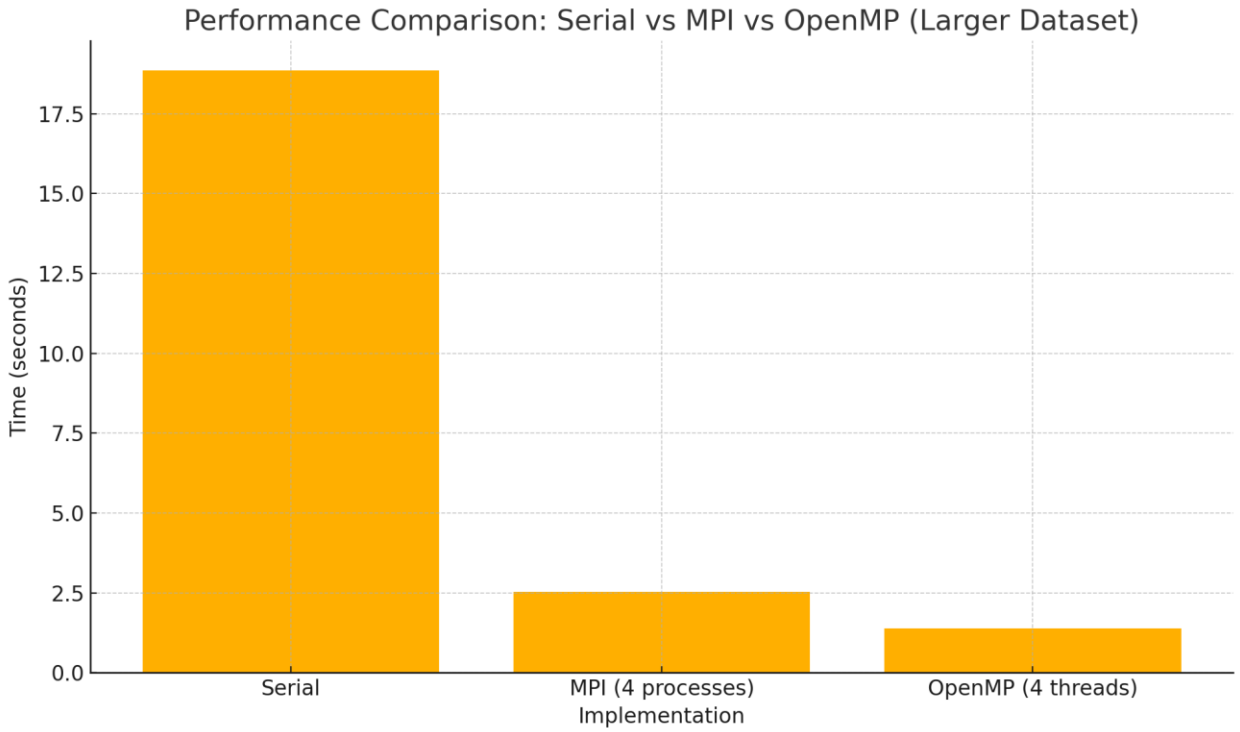
- **Partitioning:** METIS is used to divide the graph, with each thread processing a distinct subgraph.
- **Threading Model:** Parallelized inner loops using OpenMP parallel for and dynamic scheduling.
- **Synchronization:** Priority queue updates are managed via critical sections to prevent race conditions.

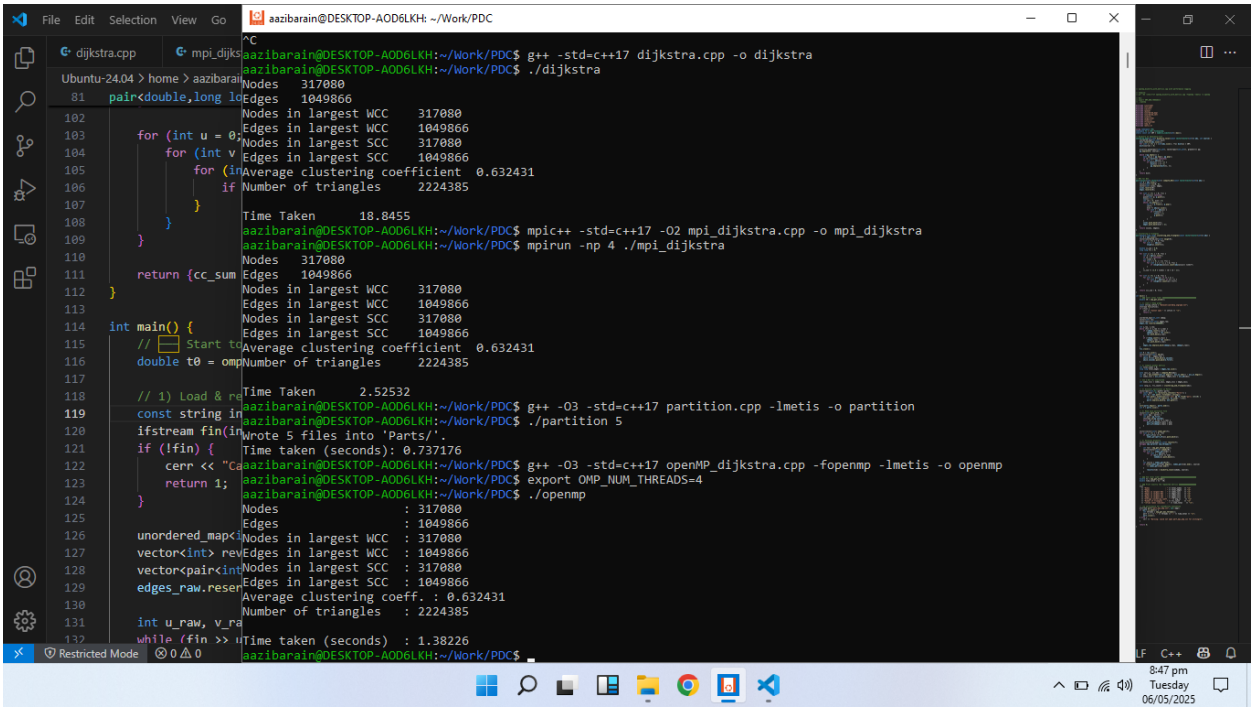
Scalability Results:

DBLP Dataset

Variant	Time (s)	Speedup
Sequential	18.8455	1.00×
MPI (4 procs)	2.5253	7.46×
OpenMP (4 threads)	1.3823	13.63×

Observation: OpenMP achieves higher speedup due to efficient shared-memory usage and reduced synchronization. MPI performs well but is limited by inter-process communication and redundant memory usage (whole-graph replication).

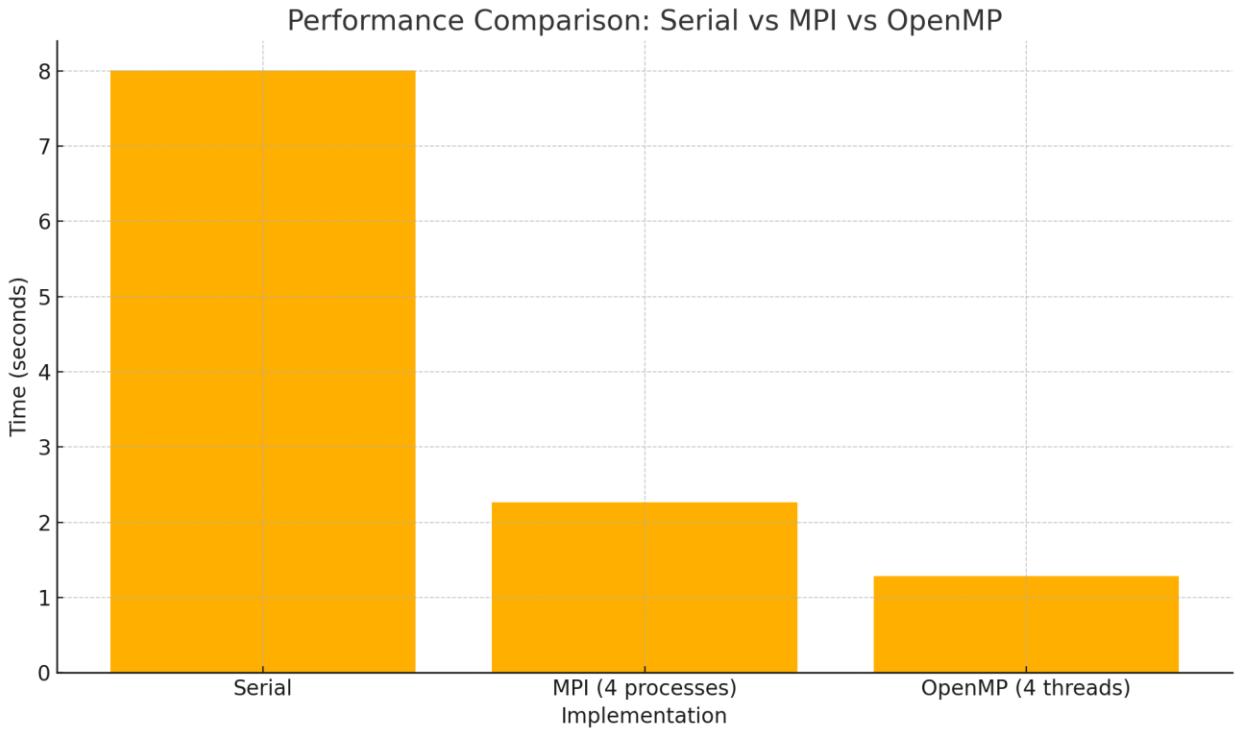

Performance Comparison: Serial vs MPI vs OpenMP (Larger Dataset)
Time (seconds)
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Serial
MPI (4 processes)
OpenMP (4 threads)
Implementation

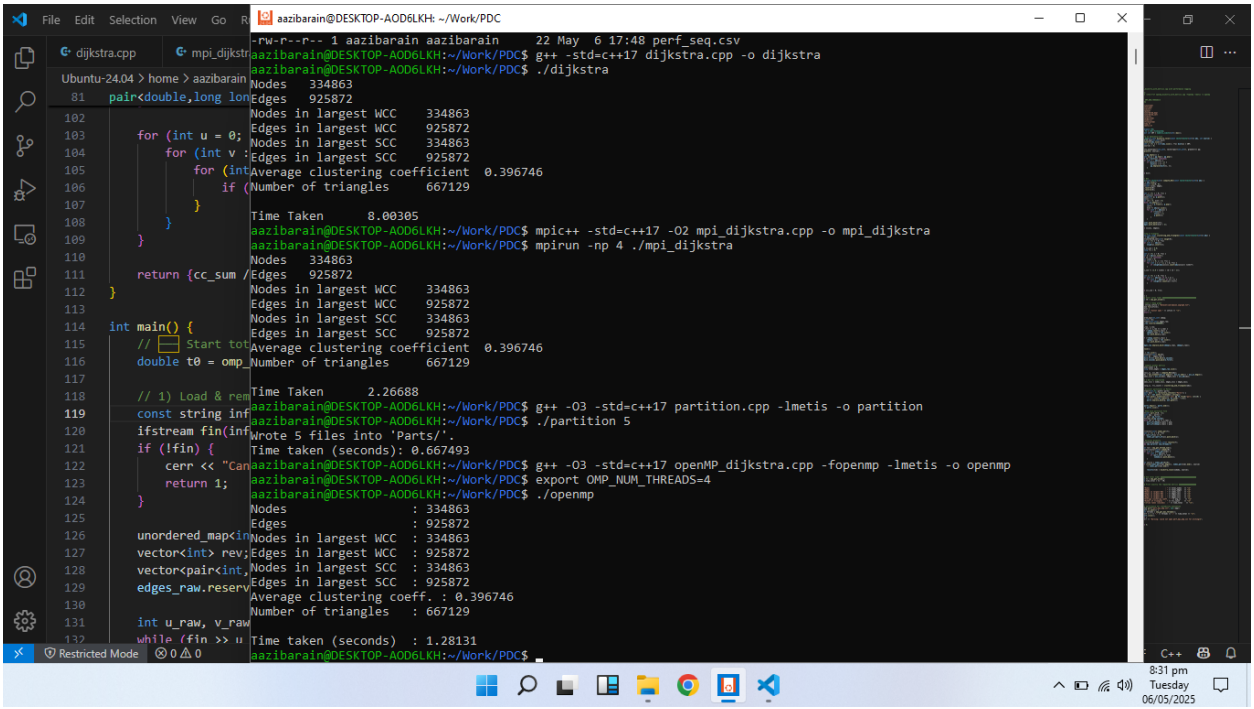

aazibarain@DESKTOP-AOD6LKH: ~/Work/PDC
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ g++ -std=c++17 dijkstra.cpp -o dijkstra
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ ./dijkstra
Nodes 317080
Edges 1049866
Nodes in largest WCC 317080
Edges in largest WCC 1049866
Nodes in largest SCC 317080
Edges in largest SCC 1049866
Average clustering coefficient 0.632431
Number of triangles 2224385
Time Taken 18.8455
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ mpic++ -std=c++17 -O2 mpi_dijkstra.cpp -o mpi_dijkstra
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ mpirun -np 4 ./mpi_dijkstra
Nodes 317080
Edges 1049866
Nodes in largest WCC 317080
Edges in largest WCC 1049866
Nodes in largest SCC 317080
Edges in largest SCC 1049866
Average clustering coefficient 0.632431
Number of triangles 2224385
Time Taken 2.52532
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ g++ -O3 -std=c++17 partition.cpp -lmetis -o partition
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ ./partition 5
Wrote 5 files into 'Parts/'.
Time taken (seconds): 0.737176
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ g++ -O3 -std=c++17 openMP_dijkstra.cpp -fopenmp -lmetis -o openmp
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ export OMP_NUM_THREADS=4
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ ./openmp
Nodes : 317080
Edges : 1049866
Nodes in largest WCC : 317080
Edges in largest WCC : 1049866
Nodes in largest SCC : 317080
Edges in largest SCC : 1049866
Average clustering coeff. : 0.632431
Number of triangles : 2224385
Time taken (seconds) : 1.38226
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$

Amazon Dataset:

Variant	Time (s)	Speedup
Sequential	8.0000	1.00×
MPI (4 procs)	2.2700	3.53×
OpenMP (4 thr)	1.2800	6.25×

Observation: Again, OpenMP performs better. MPI's performance is hindered by overhead from communicating across full graph data and lack of partitioning.


Performance Comparison: Serial vs MPI vs OpenMP
Time (seconds)
0
1
2
3
4
5
6
7
8
Serial
MPI (4 processes)
OpenMP (4 threads)
Implementation


aazibarain@DESKTOP-AOD6LKH: ~/Work/PDC
-rw-r--r-- 1 aazibarain aazibarain 22 May 6 17:48 perf_seq.csv
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ g++ -std=c++17 dijkstra.cpp -o dijkstra
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ ./dijkstra
Nodes 334863
Edges 925872
Nodes in largest WCC 334863
Edges in largest WCC 925872
Nodes in largest SCC 334863
Edges in largest SCC 925872
Average clustering coefficient 0.396746
Number of triangles 667129
Time Taken 8.00305
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ mpic++ -std=c++17 -O2 mpi_dijkstra.cpp -o mpi_dijkstra
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ mpirun -np 4 ./mpi_dijkstra
Nodes 334863
Edges 925872
Nodes in largest WCC 334863
Edges in largest WCC 925872
Nodes in largest SCC 334863
Edges in largest SCC 925872
Average clustering coefficient 0.396746
Number of triangles 667129
Time Taken 2.26688
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ g++ -O3 -std=c++17 partition.cpp -lmetis -o partition
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ ./partition 5
Wrote 5 files into 'Parts/'.
Time taken (seconds): 0.667493
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ g++ -O3 -std=c++17 openMP_dijkstra.cpp -fopenmp -lmetis -o openmp
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ export OMP_NUM_THREADS=4
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$ ./openmp
Nodes : 334863
Edges : 925872
Nodes in largest WCC : 334863
Edges in largest WCC : 925872
Nodes in largest SCC : 334863
Edges in largest SCC : 925872
Average clustering coeff. : 0.396746
Number of triangles : 667129
Time taken (seconds) : 1.28131
aazibarain@DESKTOP-AOD6LKH:~/Work/PDC$
8:31 pm
Tuesday
06/05/2025

Challenges Faced:

- **OpenMP Synchronization**: Efficient locking of shared structures was critical. We minimized bottlenecks using per-thread subgraphs and dynamic loops.
- **MPI Communication**: Deadlocks occurred due to misaligned Isend/Irecv calls, which we resolved by consistent ordering and proper completion with MPI_Waitall.
- **Graph Format Handling**: Preprocessing SNAP datasets into METIS-compatible format was time-consuming, especially ensuring 0-based indexing and removing self-loops.

Summary:

- **Best Performer**: OpenMP (shared-memory) due to localized access and partitioning.
- **MPI Suitability**: Can scale to larger clusters but suffers from communication overhead and memory replication unless smarter distribution is applied.
- **Future Improvement**: Hybrid MPI + OpenMP to combine inter-node scaling with intra-node speed.