# Project Documentation: Parallel Dijkstra’s SSSP with MPI, OpenMP, and METIS

## Team Members

Muhammad Sooban  
Umar Nadeem  
Daniyal Babar

## Phase 2 Requirements

• Implement a parallel algorithm using MPI and either OpenMP or OpenCL  
• Use METIS for graph partitioning  
• Test on multiple datasets, including public datasets  
• Evaluate scalability and performance  
• Visualize and demonstrate results (execution time, speedup, partitioning efficiency)  
• Perform comparisons between:  
 - Sequential Dijkstra  
 - MPI-based Dijkstra  
 - MPI + OpenMP/OpenCL Dijkstra

## Problem Description

Dijkstra's algorithm is inherently sequential due to its greedy approach. To scale it across large graphs:  
- The graph is partitioned using METIS.  
- Each partition is processed by a separate MPI process.  
- Optionally, OpenMP is used within each MPI process to exploit multi-threading for intra-process speedup.

## Technologies Used

MPI: Distribute graph partitions to processes  
OpenMP: Parallelize local work within each process  
METIS: Partition the graph intelligently  
C++: Core implementation language

## Implementation Highlights

Features:  
- Graph parsing from file.  
- Graph partitioning using METIS\_PartGraphKway.  
- Support for disconnected graphs using options[METIS\_OPTION\_CONTIG] = 0.  
- Parallel SSSP with MPI, communicating boundary nodes.  
- Optional OpenMP usage for per-node relaxation in Dijkstra.  
- Time measurement to compare sequential vs. parallel.  
- Compatibility with public graph datasets.

## Resolved Issues

• undefined reference to METIS\_SetDefaultOptions: Resolved by linking with -lmetis  
• METIS contiguous partition error: Fixed with METIS\_OPTION\_CONTIG = 0  
• MPI slower on small datasets: Performance expected to improve with weak scaling

## Evaluation Strategy

Performance Metrics:  
- Execution Time  
- Speedup = T\_seq / T\_parallel  
- Scalability (Strong and Weak Scaling)  
- Partition Quality (edge cuts, balance ratio)  
  
Visualization Tools:  
- Python/Matplotlib for plotting  
- Optional: Graphviz or Gephi for partition visualization

## Datasets Used

• Custom graphs  
• Public datasets:  
 - SNAP (Stanford)  
 - DIMACS Road Networks  
 - OpenStreetMap graphs

## Running the Implementation

Compilation:  
mpic++ -fopenmp -O2 open.cpp -o c -lmetis  
  
Execution:  
mpirun -np <num\_procs> ./c <input\_graph\_file>

## Findings & Observations

To be updated after full testing.  
Initial tests show MPI overhead for small graphs; performance improves with larger graphs and more processes.

## Timeline

April: Implement MPI-based SSSP  
May 3: Fix METIS partitioning issue  
May 4: Integrate OpenMP and test  
May 5: Demo and performance presentation

## Lessons Learned

- MPI overhead on small graphs  
- METIS requires tuning for disconnected graphs  
- OpenMP integration inside MPI processes requires care  
- Scaling strategies must be tailored to dataset size

## Future Work

- Implement hybrid OpenCL version  
- Load balancing across partitions  
- Use asynchronous MPI communication