# **Final Project Report: Parallel SSSP Update Using MPI, OpenMP, and METIS**

## **Title**

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

## **1. Introduction**

This project implements the parallel algorithm described in the paper *"A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks"*. The objective is to update shortest paths efficiently in massive, dynamically changing graphs. Traditional methods recalculate paths from scratch, but this work focuses on incremental updates using a frontier-based model, achieving both scalability and efficiency.

Our implementation uses:

* MPI for distributed communication
* OpenMP for shared-memory multithreading
* METIS for efficient graph partitioning

## **2. Tools and Technologies**

* MPI: Enables inter-node communication in distributed systems.
* OpenMP: Facilitates multi-threading within each node.
* METIS: Provides optimized graph partitioning to reduce communication and balance workloads.
* Dataset: mdual.graph, a representative real-world large graph.

## **3. Problem Statement**

In dynamic networks (e.g., road maps, social networks), frequent changes such as edge insertions or deletions make it impractical to recompute shortest paths from scratch. The goal is to implement a parallel, incremental Single-Source Shortest Path (SSSP) update algorithm that:

* Reduces recomputation.
* Utilizes both inter- and intra-node parallelism.
* Scales well with increasing graph size and number of processes.

## **4. Parallelization Strategy**

### **Graph Partitioning (METIS)**

* Splits the graph into roughly equal subgraphs.
* Minimizes edge cuts to reduce inter-process communication.

### **Inter-node Parallelism (MPI)**

* Each MPI process handles a partition.
* Coordinates with neighbors when the frontier expands across partitions.

### **Intra-node Parallelism (OpenMP)**

* Parallelizes relaxations and updates of local vertices.
* Achieves concurrency within each MPI process.

## **5. Implementation Overview**

1. **Graph Load & Partition:** METIS partitions the input graph; each MPI process loads its part.
2. **Initialization:** Distance and frontier structures are initialized.
3. **Edge Update Handling:** Upon edge insertion/deletion, frontier propagation is triggered.
4. **Frontier Processing:**
   * Uses OpenMP to parallelize local relaxations.
   * Frontier changes are shared with neighbor MPI processes.

## **6. Experimental Setup**

* **Cluster:** 4-node cluster (each node supports OpenMP threads).
* **Graph:** mdual.graph
* **Partition Tool:** METIS
* **Measured Metrics:**
  + Execution Time
  + Speedup (relative to baseline)
  + Partition Balance
  + Strong and Weak Scaling

## **7. Results and Scalability Analysis**

### **Execution Time Comparison**

| MPI Processes | Execution Time (s) | Speedup |
| --- | --- | --- |
| 2 | 7.32 | 1.00× |
| 3 | 4.85 | 1.51× |
| 4 | 3.79 | 1.93× |

### **Observations**

* **Speedup:** Near-linear for up to 4 processes. Diminishing returns beyond that expected due to communication overhead.
* **Partition Balance:** METIS ensured even vertex/edge distribution across processes.
* **Communication Cost:** Increased with more partitions but was mitigated by METIS’s low edge-cut partitions.

### **Scalability**

* **Strong Scaling:** Fixed graph size; reduced execution time with increasing processes.
* **Weak Scaling:** Maintained performance as both graph size and number of processes increased proportionally.

## **8. Challenges and Solutions**

| **Challenge** | **Solution** |
| --- | --- |
| High MPI communication overhead | Reduced frontier sharing via METIS partitions and batching updates |
| Synchronization issues with OpenMP+MPI | Structured computation into barriers and used #pragma omp critical where needed |
| Load imbalance in small graphs | Used METIS with balancing heuristics |

## **9. GitHub Repository**

**Link:** <https://github.com/molarmuaz/PDC_PROJECT/>  
 The repository includes:

* Full source code (MPI, OpenMP, METIS integration)
* Build instructions and test scripts
* Partitioned graph files
* Plots for performance comparison
* Slides and documentation

## **10. Conclusion**

This project successfully demonstrates the feasibility of efficiently updating SSSP paths in large-scale graphs using hybrid parallelism. We validate the claims of the original paper through practical implementation and performance benchmarking.

Our findings:

* The algorithm scales effectively up to 4 processes.
* MPI+OpenMP hybrid model is effective for dynamic SSSP updates.
* METIS partitioning plays a critical role in minimizing overhead.