**Performance Analysis Report for Single-Objective Shortest Path (SOSP) and Multi-Objective Shortest Path (MOSP) in Large Dynamic Networks**

**1. Introduction**

This report checks how well three different versions of our programs work for updating Single-Objective Shortest Path (SOSP) and Multi-Objective Shortest Path (MOSP) in big, changing networks. These programs come from a study called *A Parallel Algorithm for Updating a Multi-objective Shortest Path in Large Dynamic Networks*. We used OpenMP to let multiple tasks run together on one computer and MPI with METIS for teamwork across computers. We looked at how fast the programs run, how they handle more work, and how much computer power they use, with help from tools like *perf* and *flame graphs*.

**2. Tools and Resources**

We used these tools to build and test our programs:

* **METIS**: A tool that splits big graphs into smaller pieces to make teamwork across computers easier and faster.
* **MPI**: A system that lets computers talk to each other for big tasks. For now, we use it on one computer with clustering and a hosts file called hosts.txt.
* **OpenMP**: A tool that helps multiple parts of a program run at the same time on one computer, sharing memory.

**3. Implementation Versions**

We made three versions of the program to see which one works best:

**3.1 Version 1: Serial Implementation with OpenMP (Baseline)**

This version runs the program step-by-step on one thread as a starting point. We added OpenMP to make some parts run together, but it’s mainly a simple version to compare with faster ones.

**3.2 Version 2: MPI-Only Parallel Implementation**

This version uses MPI to split work across tasks on a **single computer**, using clustering and a hosts file named hosts.txt. It handles groups of edges and updates distances to nearby points at the same time, with different tasks sharing the work.

**3.3 Version 3: Hybrid OpenMP and MPI with METIS**

This version mixes OpenMP for teamwork on one computer with MPI for teamwork across multiple computers. METIS splits the graph into parts so each computer gets an equal amount of work.

**4. Performance Metrics and Analysis**

**4.1 Execution Time**

We measured how long each version takes to finish on different network sizes and with different numbers of edge changes (batch sizes). The datasets we used are:

* **Facebook Dataset**: 4,039 nodes, 88,234 edges.
* **Road-usa Dataset**: 23,947,347 nodes, 28,854,312 edges.
* **LiveJournal Dataset**: 4,847,571 nodes, 68,993,773 edges.

The MPI-only version was much faster than the serial version, showing up to 15 times better speed on the biggest network, Road-usa. The hybrid version (OpenMP + MPI) should handle even bigger networks well, but it might take extra time at first because splitting the graph with METIS takes effort.

* **Serial Baseline**: Takes a long time as the network gets bigger, especially for Road-usa.
* **MPI-Only**: Runs faster with more tasks on a single computer, based on our tests.
* **Hybrid OpenMP + MPI**: Early tests show it can handle big networks, but we need more testing to be sure.

**4.2 Scalability Analysis**

Scalability shows how well the program works when we add more tasks (for MPI) or threads (for OpenMP). We tested the MPI version by using different numbers of processes and the hybrid version with different numbers of OpenMP threads, all on the **Facebook dataset**. The results are shown in the screenshot below.

[Insert Screenshot of Scalability Results Here]

The MPI version got faster with more processes on a single computer, especially on the Facebook dataset. For the hybrid version, scalability depends on how well METIS splits the graph, which we’ll test more in a distributed setup.

**4.3 CPU Usage and Profiling**

We used *perf* and *flame graphs* to see how the CPU is used and find slow parts of the program for the **Road-usa dataset**:

* **Serial Version ("serial" Image)**: The flame graph shows that the parallel\_mosp function takes the most time, using about 45% of the CPU. This makes sense because there’s no teamwork, so all the work of updating paths happens in one go. Other functions like readGraphFromFile (10%) and libc\_start\_call\_main (5%) also use some CPU time since loading the big Road-usa dataset takes effort. There’s no extra delay from tasks talking to each other because it’s not parallel.
* **MPI-Only Version ("mpi" Image)**: The flame graph shows parallel\_mosp still takes a lot of time but less than the serial version, around 28% of the CPU. Since MPI splits the work across tasks on a single computer using hosts.txt, the workload is shared better. We also see MPI functions like MPI\_Allreduce (8%) and mca\_btl\_vader\_so (6%) because tasks need to talk to each other to share updates. This talking adds some delay but makes the program faster overall.
* **Hybrid OpenMP + MPI Version ("bothuse" Image)**: The flame graph shows parallel\_mosp using about 22% of the CPU, which is the least of the three versions because both OpenMP and MPI share the work. We see MPI functions like MPI\_Allreduce (7%) and mca\_btl\_vader\_so (5%) for communication across tasks, and OpenMP delays like omp\_parallel (4%) because threads inside each task need to work together. METIS splitting with METIS\_PartGraphRecursive takes a small amount of time, around 3%. Most of the CPU time still goes to updating paths, which matches the study’s finding that SOSP updates take the most effort.

[Insert Screenshot "serial" Here]  
[Insert Screenshot "mpi" Here]  
[Insert Screenshot "bothuse" Here]

**4.4 Memory and Resource Utilization**

We checked how much memory the program uses for things like the graph and SOSP tree. The MPI-only version needs memory for each task to store its part of the graph on a single computer. The hybrid version might use more memory because of MPI communication and METIS splitting, which we’ll keep tracking in future tests.

**5. Comparison of Versions**

Here’s how the three versions stack up:

* **Serial vs. MPI-Only**: The MPI version is much faster because it splits edge groups and updates across tasks on a single computer using clustering and hosts.txt.
* **MPI-Only vs. Hybrid**: The hybrid version (OpenMP + MPI) could be better for very big networks across multiple computers, but the extra work of splitting the graph with METIS might slow it down for smaller setups.
* **METIS Application**: We didn’t use METIS in the serial version since it’s not needed. But METIS is helpful in the hybrid version to share work evenly across computers.

**6. Recommendations**

Based on what we found, here’s what we suggest:

* Use the MPI-only version for systems with multiple tasks on a single computer, as it works well on the networks we tested.
* Test the hybrid OpenMP + MPI version with METIS for bigger networks on distributed systems, making sure METIS splits the graph well.
* Fix the parallel\_mosp and readGraphFromFile functions, as they’re the slowest parts according to the flame graph.

**7. Conclusion**

The MPI-only version is a fast and strong solution for the study, performing much better than the serial version on a single computer with clustering. The hybrid OpenMP + MPI version with METIS could be great for bigger networks across multiple computers, but we need more tests to confirm. Future work should focus on speeding up slow functions and improving teamwork across computers for handling multiple goals.