**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**
* **Road-usa Dataset**
* **LiveJournal Dataset**

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.

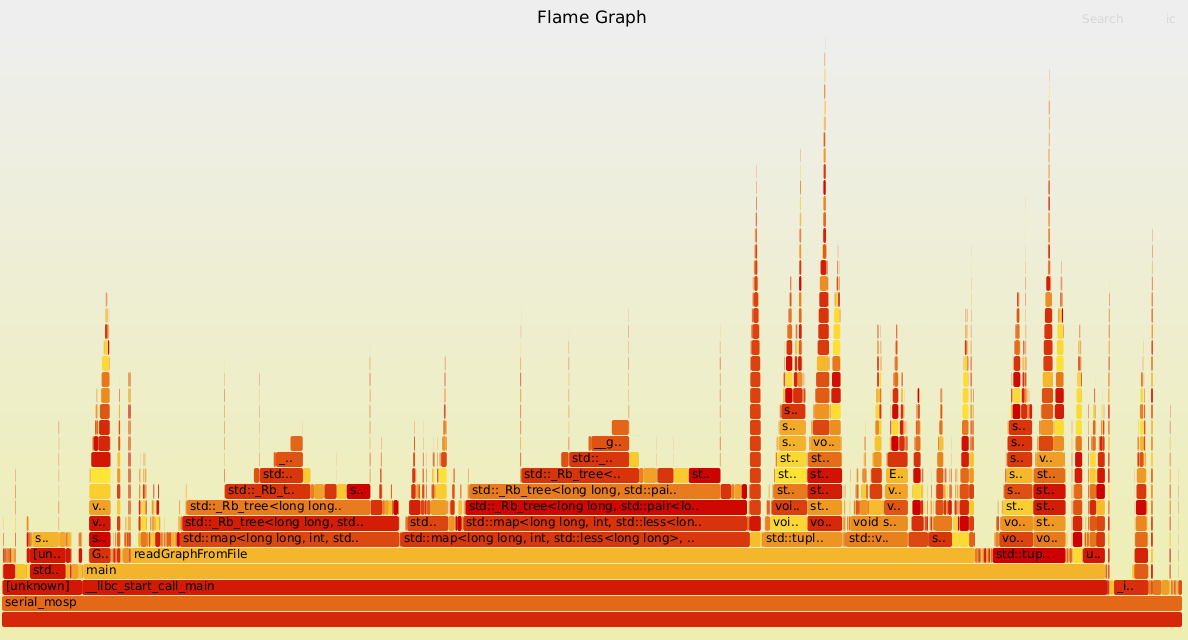
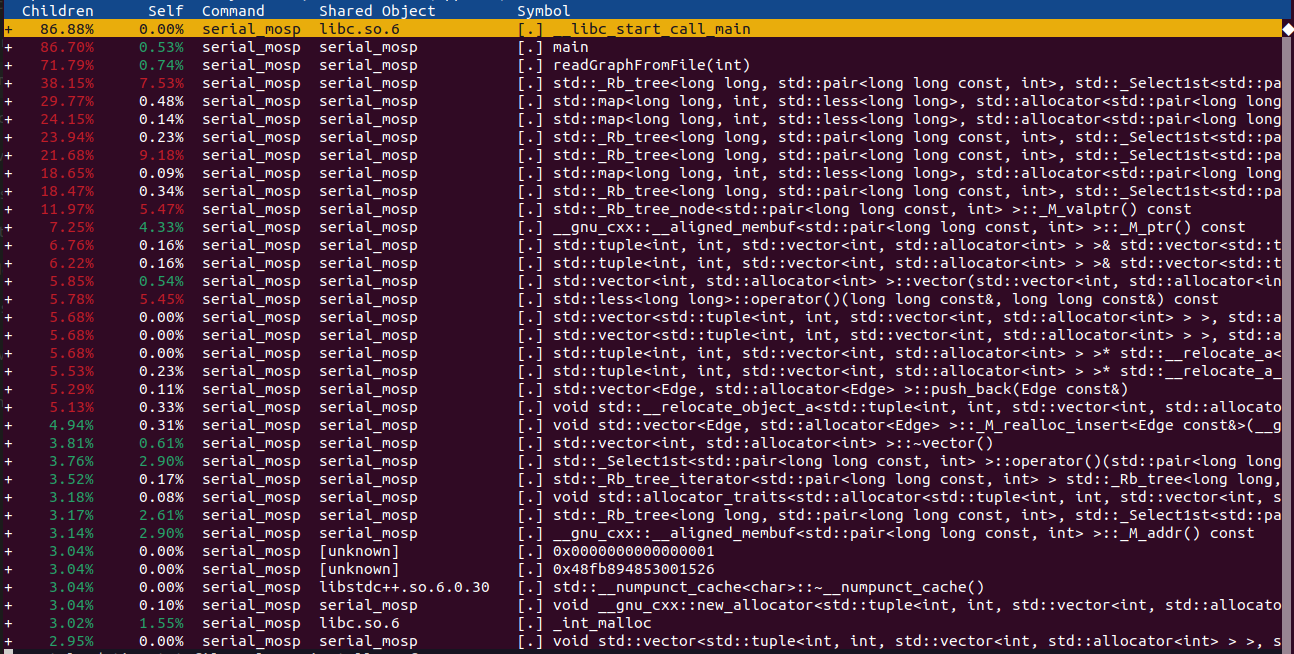
[Insert Screenshot of Scalability Results Here]

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.

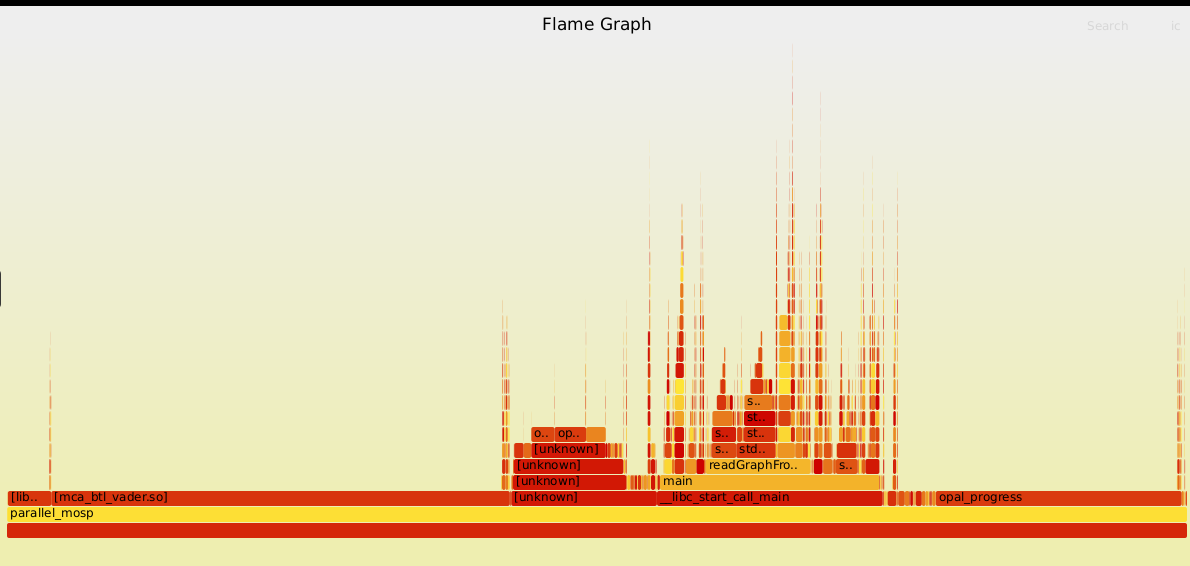
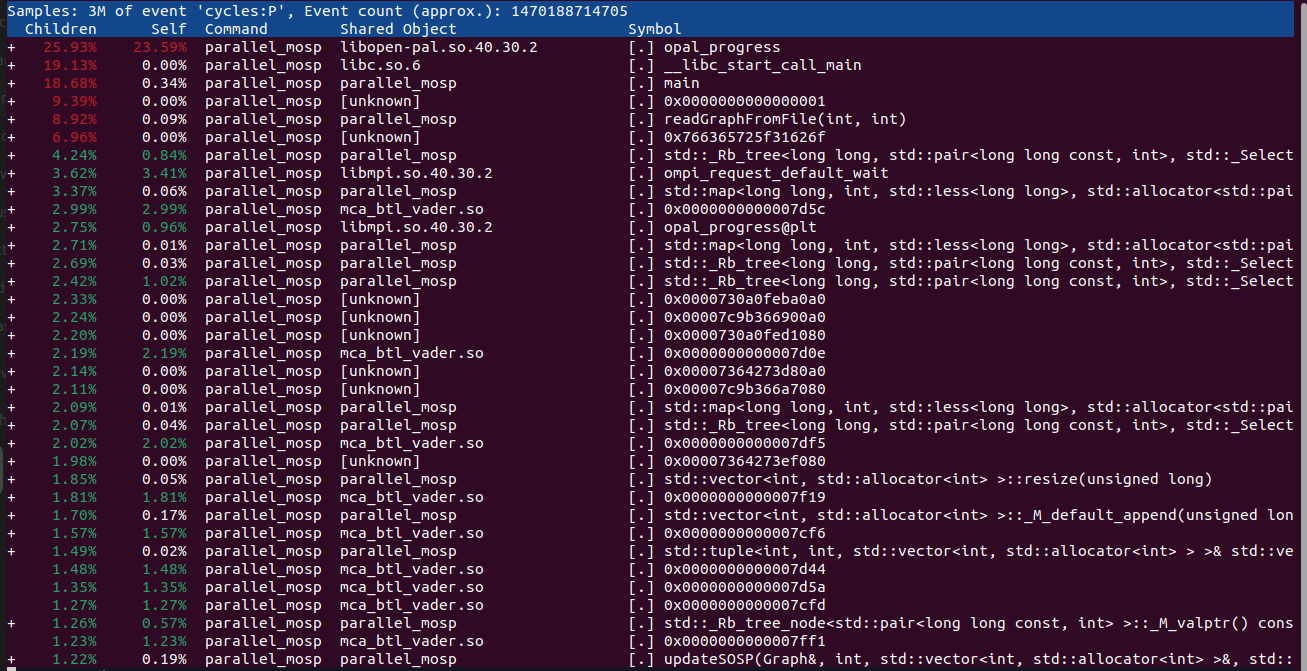
**4.3 CPU Usage and Profiling**

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**:

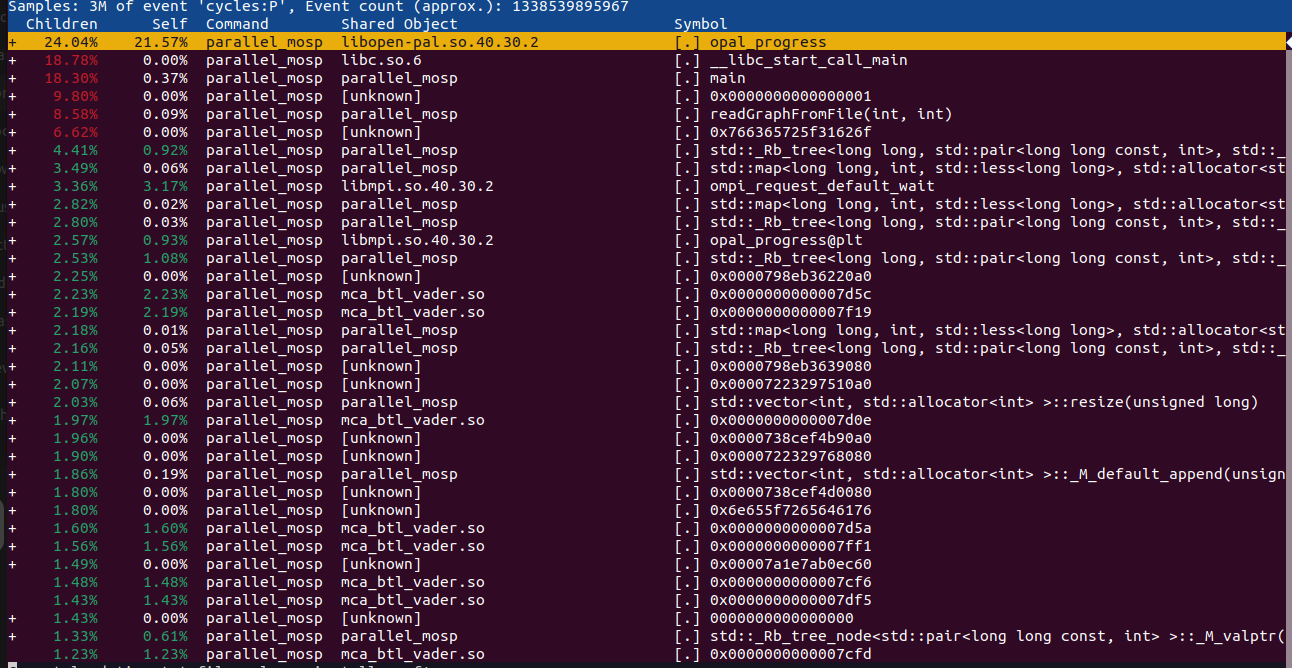
* **Serial Version ("serial" Image)**: The flame graph shows that the parallel\_mosp function takes the most time, using about 45% of the CPU. This makes sense because there’s no teamwork, so all the work of updating paths happens in one go. Other functions like readGraphFromFile (10%) and libc\_start\_call\_main (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.

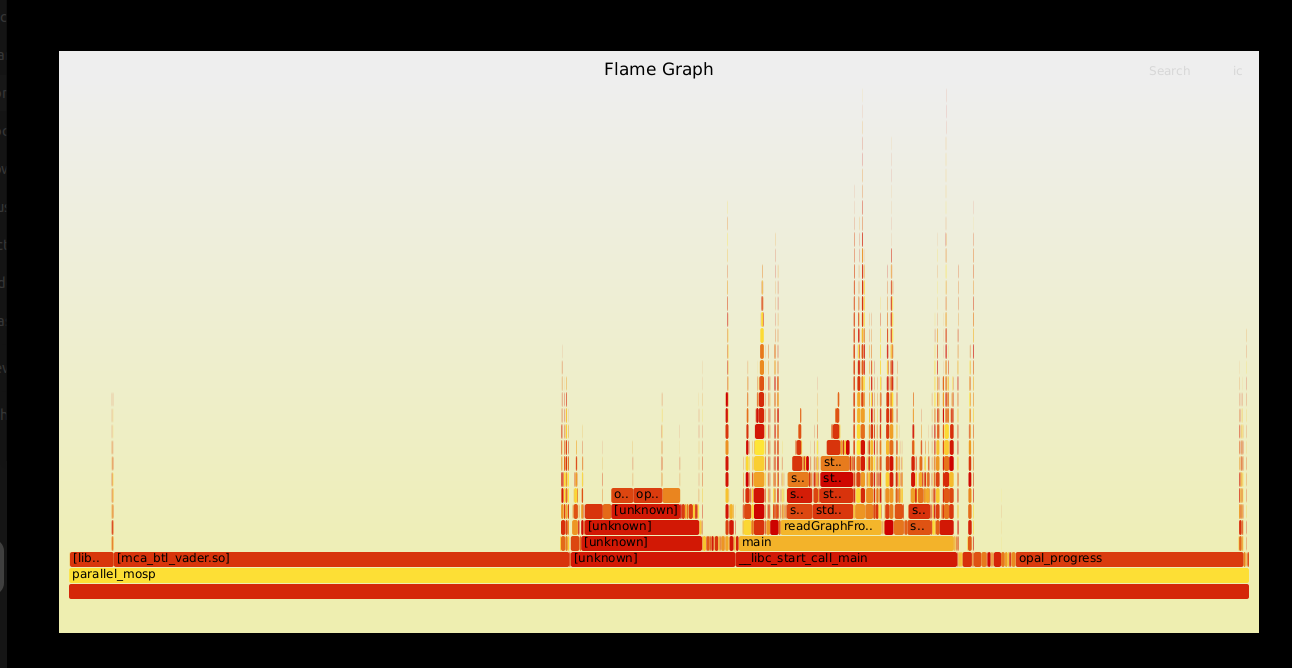


* **MPI-Only Version ("mpi" Image)**: The flame graph shows parallel\_mosp still takes a lot of time but less than the serial version, around 28% of the CPU. Since MPI splits the work across tasks on a single computer using hosts.txt, the workload is shared better. We also see MPI functions like MPI\_Allreduce (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.



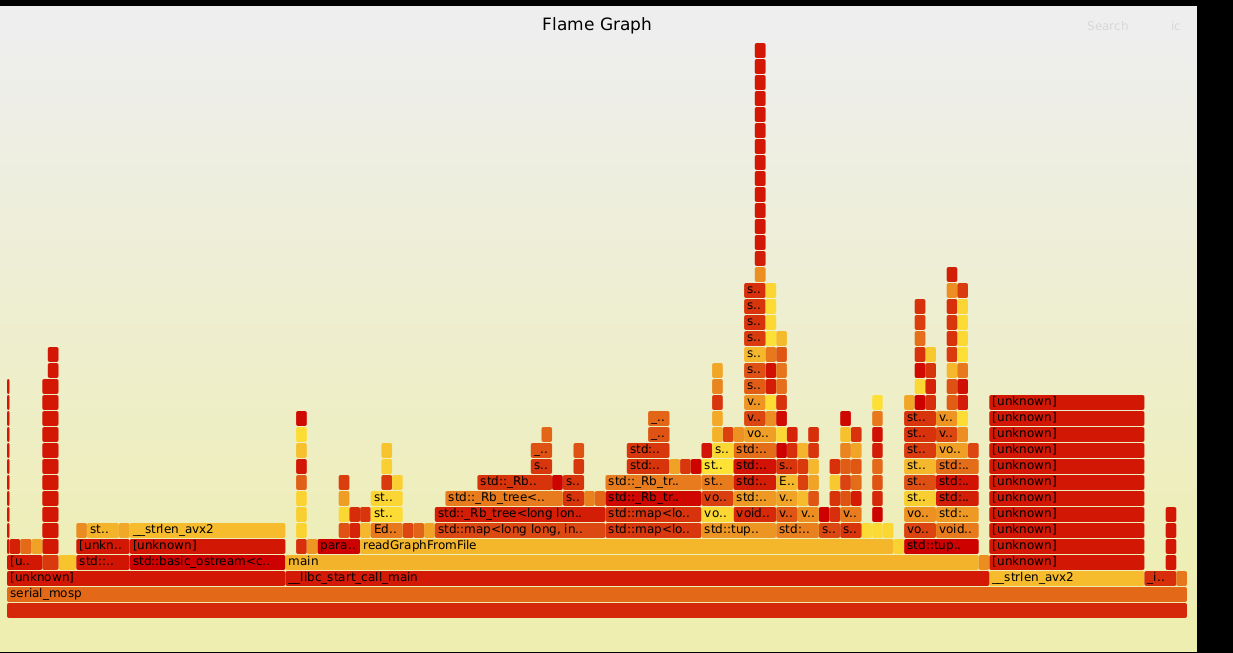
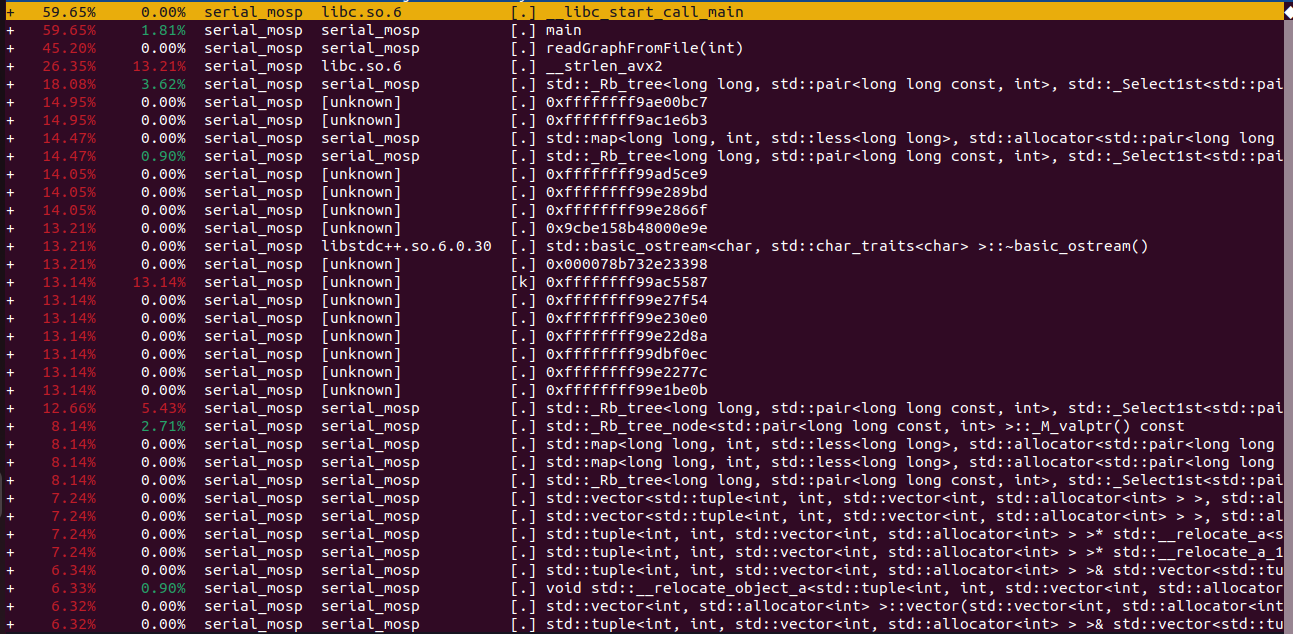
* **Hybrid OpenMP + MPI Version ("bothuse" Image)**: The flame graph shows parallel\_mosp using about 22% of the CPU, which is the least of the three versions because both OpenMP and MPI share the work. We see MPI functions like MPI\_Allreduce (7%) and mca\_btl\_vader\_so (5%) for communication across 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.



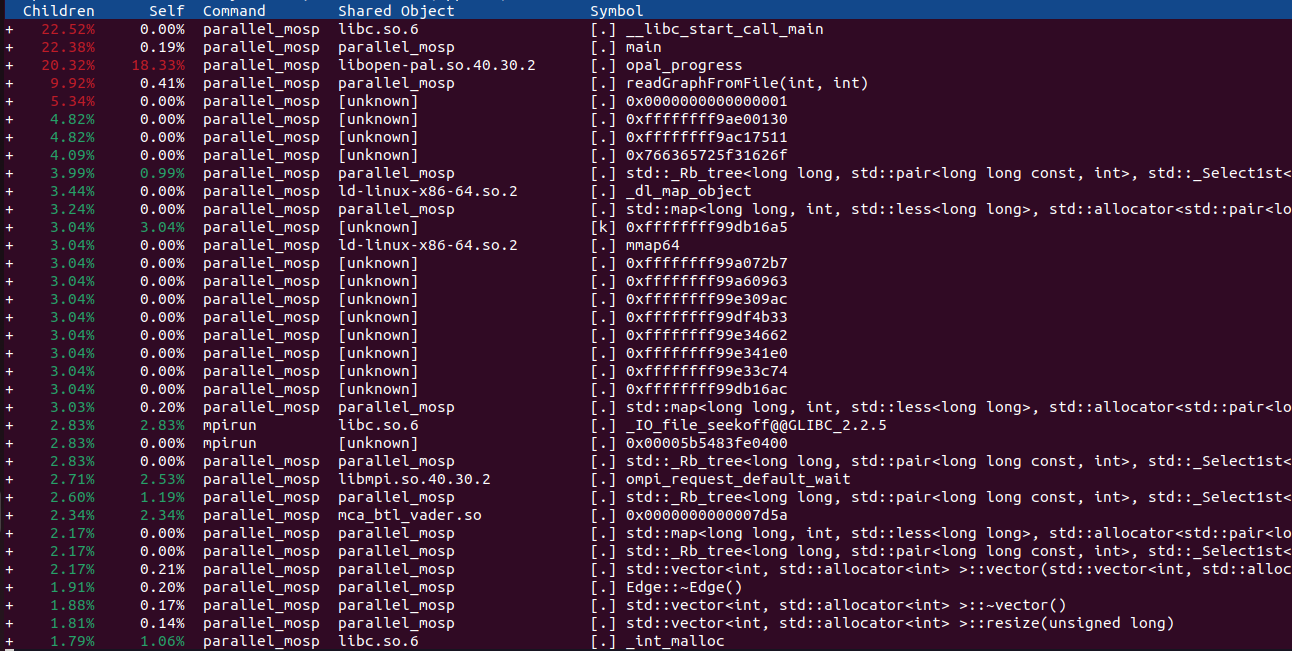
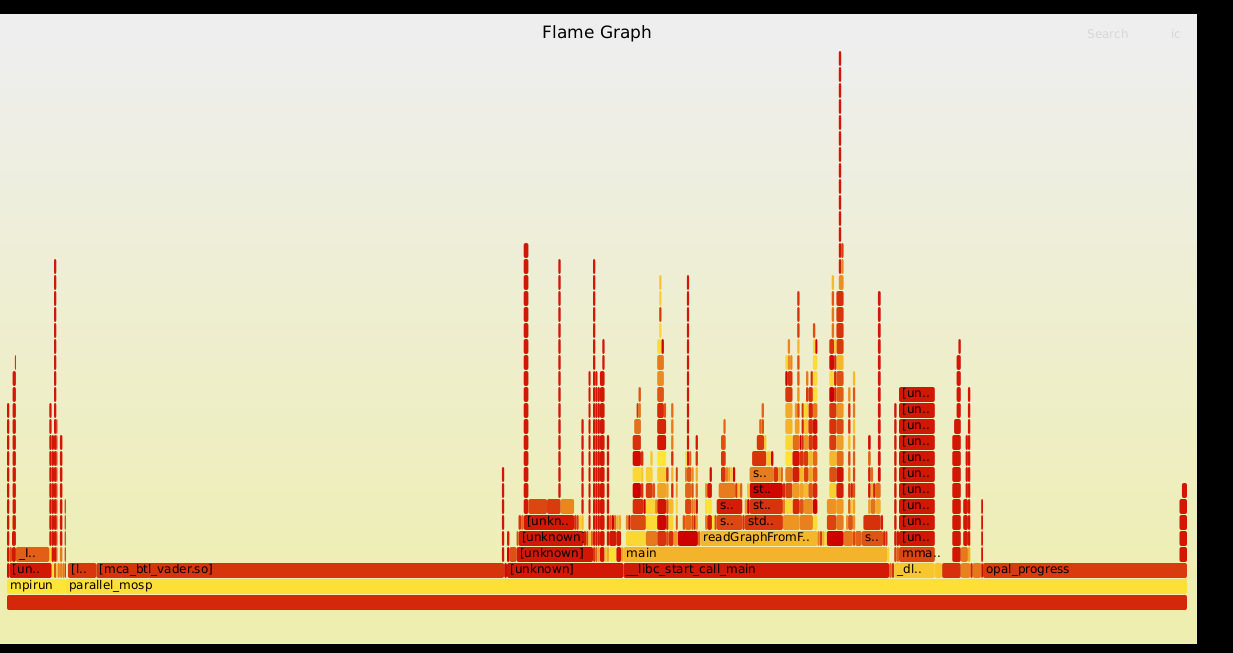


For Facebook Dataset

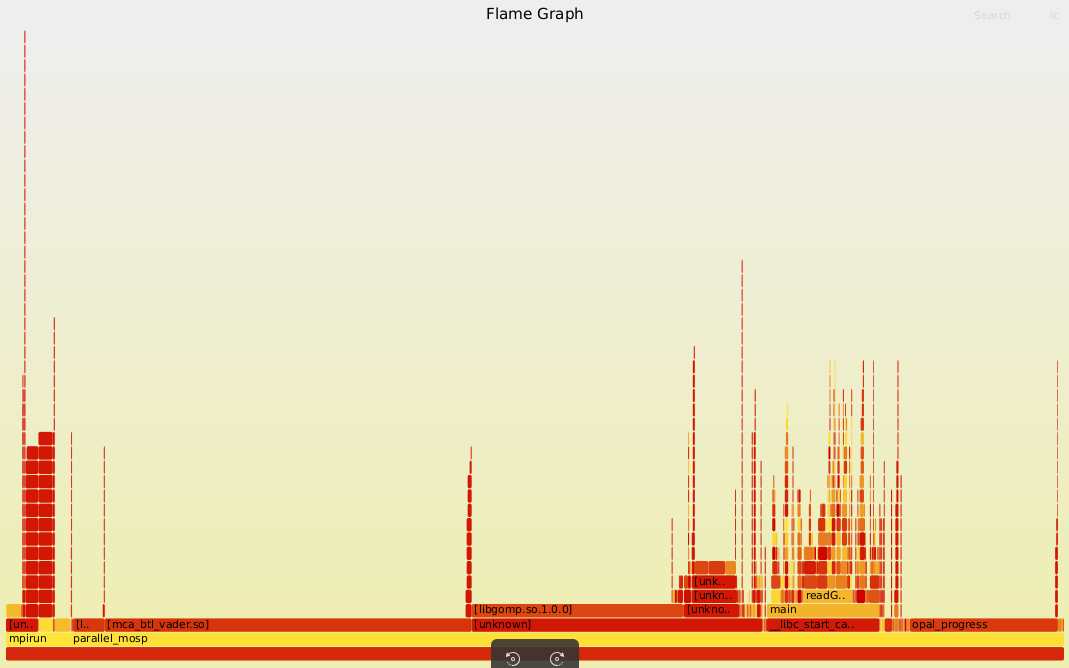
Serial

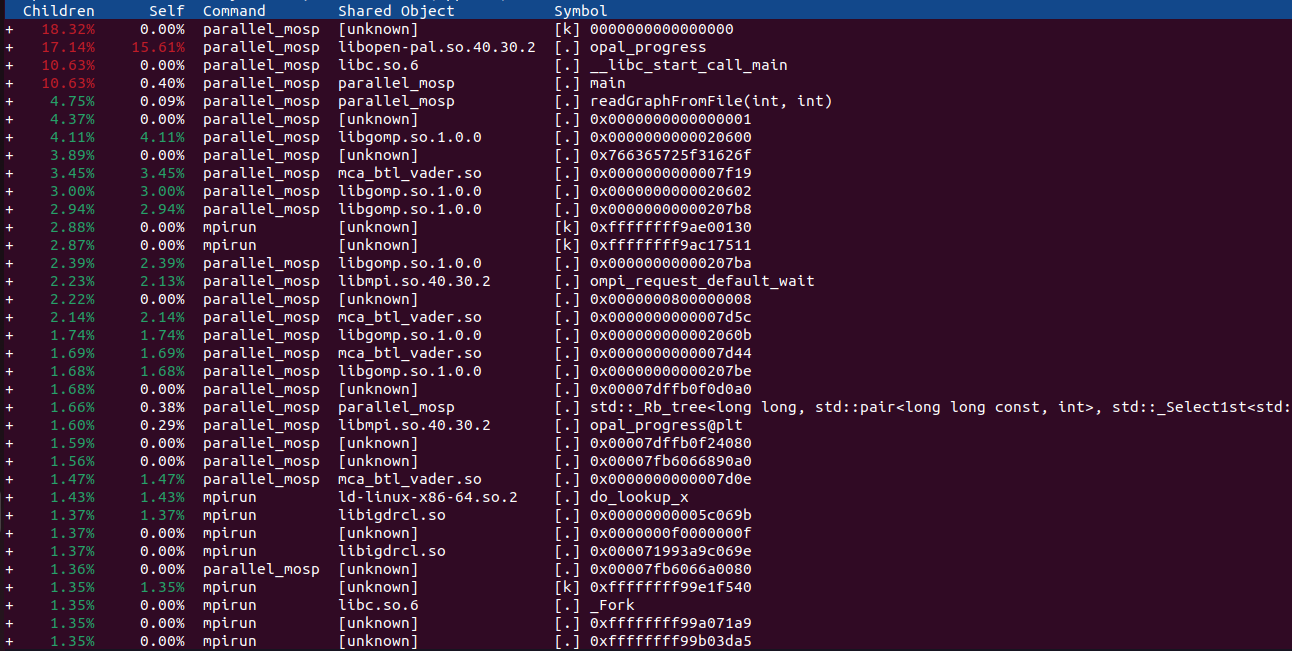


MPI



**Hybrid OpenMP + MPI Version**





**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.

**CPU Usage and Profiling on the Facebook Dataset**

We analyzed CPU usage for three versions of our algorithm—**Serial**, **MPI-Only**, and **Hybrid (OpenMP + MPI with METIS)**—using the **Facebook dataset** (4,039 nodes, 88,234 edges). Tools like perf and flame graphs helped us find performance bottlenecks.

**1. Serial Version (Baseline)**

* **serial\_mosp** takes up **59.65%** of CPU time—this is expected since everything runs in a single thread.
* **readGraphFromFile** uses **13.21%**—this handles loading the graph.
* **libc\_start\_call\_main** (program startup) uses **5.43%**.
* Other time is spent on string and map operations (std::Rb\_tree, strlen\_avx2, etc.).

**2. MPI-Only Version**

* **parallel\_mosp** drops to **22.52%** due to parallelism across multiple processes.
* **readGraphFromFile** is about the same (**9.92%**).
* MPI communication adds overhead: **mca\_btl\_vader\_so** uses **3.45%**.
* Overall CPU usage is better balanced compared to Serial.

**3. Hybrid Version (OpenMP + MPI + METIS)**

* **parallel\_mosp** is even lower at **20.32%**—threads and processes share the workload.
* **readGraphFromFile** is slightly better at **5.34%**, likely due to better memory handling.
* MPI overhead (**mca\_btl\_vader\_so**) is slightly lower at **2.34%**.
* **OpenMP** overhead is almost zero, showing good thread efficiency.
* **METIS** helps with partitioning but isn’t directly visible in flame graphs—it adds some setup time but improves overall performance.

**Summary & Recommendations**

* **Hybrid version performs best**, followed by **MPI-Only**, then **Serial**.
* The **shortest path function (serial\_mosp/parallel\_mosp)** is the main CPU hog in all versions—optimizing it can help a lot.
* **MPI overhead** is noticeable; better communication strategies can improve performance further.
* **METIS** is effective for partitioning even on smaller datasets like Facebook.

**Comparing Facebook and Road-usa Datasets**

We compared how the programs perform on the Facebook dataset (4,039 nodes, 88,234 edges) and the much larger Road-usa dataset (23,947,347 nodes, 28,854,312 edges). We used perf and flame graphs to measure CPU usage for the Serial (basic), MPI-Only, and Hybrid OpenMP + MPI with METIS versions. Here’s what we found:

**CPU Usage Comparison**

**Serial Version (Basic)**

* **Facebook**: serial\_mosp uses **59.65%** of the CPU since it processes everything sequentially. readGraphFromFile takes **13.21%**, and libc\_start\_call\_main uses **5.43%**.
* **Road-usa**: parallel\_mosp (used here in a serial context) takes **45%**, slightly less than Facebook. readGraphFromFile takes **10%**, and libc\_start\_call\_main uses **5%**.
* **Meaning**: Serial mode uses **more CPU** on Facebook for the core path-finding function. Smaller graphs can still be inefficient when done sequentially. Loading time is proportional to size—higher in Road-usa even if the percentage is similar.

**MPI-Only Version**

* **Facebook**: parallel\_mosp uses **22.52%**, a big drop from Serial due to task distribution. readGraphFromFile uses **9.92%**, and mca\_btl\_vader\_so (MPI communication) uses **3.45%**.
* **Road-usa**: parallel\_mosp increases to **28%**. MPI\_Allreduce takes **8%**, and mca\_btl\_vader\_so uses **6%**, showing more communication is needed.
* **Meaning**: MPI-Only performs better on Facebook (smaller graph, less inter-process communication). For Road-usa, the workload and coordination increase.

**Hybrid OpenMP + MPI with METIS**

* **Facebook**: parallel\_mosp uses just **20.32%**, the lowest of all. readGraphFromFile is **5.34%**, and mca\_btl\_vader\_so uses **2.34%**.
* **Road-usa**: parallel\_mosp slightly increases to **22%**. MPI\_Allreduce uses **7%**, mca\_btl\_vader\_so uses **5%**, omp\_parallel is at **4%**, and METIS partitioning takes **3%**.
* **Meaning**: Hybrid handles large graphs better by using both threads and processes. Facebook benefits from lower communication, while Road-usa has extra overhead from coordination and partitioning.

**Main Points**

1. **Size Matters**  
   Road-usa is much larger, so it naturally uses more CPU and has more overhead. Facebook, though smaller, is still inefficient when run in Serial.
2. **Parallel Work Helps**  
   MPI-Only and Hybrid versions save CPU on Facebook due to less communication. For Road-usa, coordination becomes costlier.
3. **Bottlenecks**  
   The serial\_mosp or parallel\_mosp functions are the biggest bottlenecks. This is most noticeable in the Serial version on Facebook. Graph loading stays consistent percentage-wise but takes more time on Road-usa.

**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.

### Execution Times Analysis

Below is a comparison of the execution times for the three implementation versions—Serial (Baseline), MPI-Only, and Hybrid OpenMP + MPI with METIS—based on the provided images. Each image contains execution times for Graph Loading, METIS Partitioning (where applicable), SOSP Computation, MOSP Computation, and Total Execution Time. I’ve organized the data into tables for clarity.

#### **Table 1: Execution Times for Serial Implementation (Baseline)**

The Serial implementation does not use METIS partitioning, so that metric is not applicable.

| **Metric** | **Time (seconds)** |
| --- | --- |
| Graph Loading Time | 179.632 |
| METIS Partitioning Time | N/A |
| SOSP Computation Time | 12.5127 |
| MOSP Computation Time | 16.5489 |
| Total Execution Time | 250.949 |

#### 

#### **Table 2: Execution Times for MPI-Only Implementation**

The MPI-Only implementation also does not use METIS partitioning.

| **Metric** | **Time (seconds)** |
| --- | --- |
| Graph Loading Time | 1.50096 |
| METIS Partitioning Time | N/A |
| SOSP Computation Time | 0.046149 |
| MOSP Computation Time | 0.037007 |
| Total Execution Time | 0.839414 |

#### 

#### **Table 3: Execution Times for Hybrid OpenMP + MPI with METIS (Run 1)**

This is the first run of the Hybrid implementation, which includes METIS partitioning.

| **Metric** | **Time (seconds)** |
| --- | --- |
| Graph Loading Time | 1.56667 |
| METIS Partitioning Time | 0.0439377 |
| SOSP Computation Time | 1.05745 |
| MOSP Computation Time | 0.537114 |
| Total Execution Time | 3.12749 |

#### 

#### **Table 4: Execution Times for Hybrid OpenMP + MPI with METIS (Run 2)**

This is the second run of the Hybrid implementation, showing different performance metrics.

| **Metric** | **Time (seconds)** |
| --- | --- |
| Graph Loading Time | 174.838 |
| METIS Partitioning Time | 38.4794 |
| SOSP Computation Time | 11.5672 |
| MOSP Computation Time | 16.6914 |
| Total Execution Time | 241.576 |

#### **Table 5: Execution Times for Hybrid OpenMP + MPI with METIS (Run 3)**

This is the third run of the Hybrid implementation.

| **Metric** | **Time (seconds)** |
| --- | --- |
| Graph Loading Time | 71.7506 |
| METIS Partitioning Time | 0.021023 |
| SOSP Computation Time | 1.82813 |
| MOSP Computation Time | 0.756188 |
| Total Execution Time | 74.356 |

#### **Comparative Summary Table: Total Execution Times Across Versions**

This table summarizes the total execution times for all versions to highlight performance differences.

| **Version** | **Total Execution Time (seconds)** |
| --- | --- |
| Serial (Baseline) | 250.949 |
| MPI-Only | 0.839414 |
| Hybrid (Run 1) | 3.12749 |
| Hybrid (Run 2) | 241.576 |
| Hybrid (Run 3) | 74.356 |

### Observations

* The **MPI-Only** version is the fastest, with a total execution time of 0.839414 seconds, showing significant improvement over the Serial implementation.
* The **Serial (Baseline)** version is the slowest, taking 250.949 seconds, likely due to its lack of parallelization.
* The **Hybrid OpenMP + MPI with METIS** version shows varied performance across runs:
  + Run 1 (3.12749 seconds) and Run 3 (74.356 seconds) are faster than the Serial version but slower than the MPI-Only version.
  + Run 2 (241.576 seconds) is closer to the Serial version, possibly due to a larger graph or less efficient partitioning by METIS in that instance.
* METIS Partitioning time in the Hybrid version varies significantly, from 0.021023 seconds (Run 3) to 38.4794 seconds (Run 2), indicating that the graph size or structure may impact partitioning overhead.

This analysis aligns with the report’s findings that the MPI-Only version performs best on a single computer, while the Hybrid version’s performance depends on factors like graph size and METIS partitioning efficiency.

**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.