**Performance Analysis Report for the Case Study Implementation**

**1. Introduction**

This report analyzes the performance of three different versions of the parallel algorithms for updating Single-Objective Shortest Path (SOSP) and Multi-Objective Shortest Path (MOSP) in large dynamic networks, as described in the case study titled *A Parallel Algorithm for Updating a Multi-objective Shortest Path in Large Dynamic Networks*. The implementations use OpenMP for shared-memory parallelism and explore the use of MPI with METIS for distributed-memory setups. We evaluate execution time, scalability, and resource usage, using data from profiling tools like perf and flame graphs to support our findings.

**2. Tools and Resources**

The following tools and resources were used in the implementations:

* **METIS**: A software package for partitioning graphs and organizing sparse matrices. METIS helps improve efficiency in MPI-based parallel processing by dividing the graph into smaller parts.
* **MPI**: A standard system for sending messages between computers in parallel setups, used for distributed-memory parallelism.
* **OpenMP**: A tool for parallel programming on shared-memory systems, allowing multiple threads to work together on the same data.

**3. Implementation Versions**

We implemented three versions of the algorithm to compare their performance:

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

This version runs the algorithm step-by-step in a single thread as a starting point. OpenMP is used to add parallel processing where possible, but it primarily serves as a baseline to compare with faster versions.

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

This version fully uses OpenMP to run tasks in parallel on a shared-memory system. It processes groups of edges and updates neighbor distances using multiple threads at the same time.

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

This version combines OpenMP for shared-memory tasks with MPI for distributed-memory setups. METIS is used to split the graph into parts, ensuring each computer in the distributed system gets a fair share of the work.

**4. Performance Metrics and Analysis**

**4.1 Execution Time**

We measured how long each version takes to run on different network sizes and with different numbers of edge changes (batch sizes). The OpenMP-only version was much faster than the serial version, showing up to 15 times better performance on the largest network, road-usa, which has 23 million vertices. The hybrid version with MPI and METIS is expected to handle even larger networks well, but it might take extra time at first due to the work needed to split the graph.

* **Serial Baseline**: Takes longer as the network size grows, especially for road-usa.
* **OpenMP-Only**: Runs faster as we add more threads (from 1 to 64), based on scalability tests.
* **Hybrid MPI/OpenMP**: Early results suggest it can scale better for distributed systems, but full testing is still needed.

**4.2 Scalability Analysis**

Scalability measures how well the program performs as we increase the number of threads or computers. We tested the OpenMP version by increasing threads from 1 to 64 and plan to test the MPI version similarly. The OpenMP version showed good scalability, with faster run times as we added more threads, especially on sparse graphs like road-usa. For the hybrid version, scalability will depend on how well METIS splits the graph, which needs further testing in a distributed setup.

**4.3 CPU Usage and Profiling**

We used perf and flame graphs to understand how the CPU is used and to find slow parts of the program:

* The flame graph shows that the parallel\_mosp function takes the most time, with other functions like readGraphFromFile and libc\_start\_call\_main also contributing.
* The perf analysis shows that parallel\_mosp uses 21.57% of the CPU time on its own, meaning it’s a heavy task.
* Other functions like mca\_btl\_vader\_so show some delays due to threads communicating with each other in OpenMP.

**[Insert Screenshot of Flame Graph Here]**

**[Insert Screenshot of perf CPU Usage and Analysis Here]**

**4.4 Memory and Resource Utilization**

We tracked memory usage for the data structures like the adjacency list and SOSP tree. The OpenMP-only version uses memory efficiently since all threads share the same memory space. The hybrid version might need more memory for MPI communication and METIS partitioning, which we will monitor in future tests.

**5. Comparison of Versions**

Here’s how the three versions compare:

* **Serial vs. OpenMP-Only**: The OpenMP version is much faster because it processes edge groups and updates in parallel.
* **OpenMP-Only vs. Hybrid**: The hybrid version might be better for very large networks, but the extra work of splitting the graph with METIS could slow it down for smaller setups.
* **METIS Application**: We decided not to use METIS in the serial version because it’s not needed. However, METIS is useful in the hybrid version to balance the workload across multiple computers.

**6. Recommendations**

Based on the analysis, here are our suggestions:

* Use the OpenMP-only version for systems with up to 64 threads, as it performs well on the networks we tested.
* Test the hybrid MPI/OpenMP version with METIS for larger networks in distributed systems, ensuring METIS partitioning works efficiently.
* Improve the parallel\_mosp and readGraphFromFile functions, as they are the slowest parts according to the flame graph.

**7. Conclusion**

The OpenMP-only version is a strong and efficient solution for the case study, offering much better performance than the serial version. The hybrid MPI/OpenMP version with METIS shows potential for handling larger networks in distributed systems, but more testing is needed to confirm its benefits. Future improvements should focus on reducing delays in key functions and exploring hybrid parallelism for better handling of multiple objectives.



  
  
  
  
  
