**Phase 2: Implementation and Demonstration Report**

**1. Introduction**

***Objective:***

The goal of this phase is to implement and evaluate a dynamic Single-Source Shortest Path (SSSP) update algorithm on large-scale graphs. The algorithm utilizes **MPI** for distributed parallelism, **OpenMP** or **OpenCL** for intra-node parallelism, and **METIS** for graph partitioning. The focus is on efficiently handling graph updates, specifically when the graph is subject to insertions and deletions.

***Scope:***

* **Partitioning**: The graph will be partitioned using METIS to minimize inter-partition edges and balance the load across multiple processes.
* **Distributed SSSP Construction**: The algorithm will construct the shortest paths in a distributed manner using MPI.
* **Dynamic Updates**: The update process will handle dynamic edge insertions and deletions, with efficient propagation of changes to the network.
* **Performance Comparison**: We will compare the performance of the MPI-only setup, MPI with OpenMP, and optional MPI with OpenCL acceleration for updates.

**2. System Configuration**

***Hardware:***

* **CPU**: Intel(R) Core(TM) i3-1115G4 @ 3.00GHz 2.90 GHz, RAM: 8 GB  
  **RAM**: 8 GB
* **GPU** : -----

***Software:***

* **Operating System**: Ubuntu 20.04 LTS
* **MPI Implementation**: Open MPI 4.0.3
* **Compiler**: GCC 9.3.0
* **Graph Partitioning**: METIS 5.1.0
* **Parallel Computing**: OpenMP 4.5 (for intra-node parallelism) and OpenCL 1.2 (for optional GPU acceleration)
* **Programming Language**: C++

**3. Datasets**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset Name** | **Type** | **|V|** | **|E|** | **Source** |
| soc-Orkut.mtx | Social graph | 3,000,000 | 117,000,000 | SNAP / Orkut network |
| graph12.txt | Synthetic (12) | 12 | 21 | Course example |

**4. Methodology**

**4.1 Partitioning with METIS**

* **METIS options**:
  + OBJTYPE: objtype=1 (minimizing edge-cut)
  + NITER: Set to 20 iterations for graph partitioning
  + Seed: Random seed for reproducibility
* **Fallback strategy**: For large graphs (greater than 1M vertices), use a round-robin approach for partitioning if METIS performance degrades.

**4.2 Initialization**

* **Sequential Dijkstra**: The baseline for comparison, computed sequentially on a single process.
* **MPI-only**: Distribute the global graph (in CSR format) to all MPI processes, each of which computes its portion of the SSSP using Dijkstra's algorithm.
* **MPI+OpenMP**: Apply OpenMP within each MPI process to parallelize the Bellman-Ford propagation within each partition.
* **MPI+OpenCL (Optional)**: Use OpenCL to accelerate local relaxations within partitions on supported GPUs.

**4.3 Dynamic Updates**

* **Batch Generation**: Insertions and deletions are generated in batches.
* **Two-phase update**:
  1. **Local Propagation**: Each partition propagates changes locally.
  2. **Boundary Exchange**: After local updates, boundary updates are exchanged across partitions.

**5. Experimental Setup**

***Configurations Tested:***

* **Sequential**: Single-process, non-parallel implementation for baseline performance.
* **MPI-only**: Distributed execution across different numbers of MPI processes (P = 1, 4, 8, …).
* **MPI+OpenMP**: Use OpenMP to parallelize the work within each process.
* **MPI+OpenCL (Optional)**: If implemented, use OpenCL for GPU acceleration of local relaxations.

***Scaling Experiments:***

* **Strong Scaling**: The problem size remains constant, and the number of processes is increased (P = 1, 2, 4, 8, 16).
* **Weak Scaling**: The problem size grows proportionally to the number of processes, keeping the work per process constant.

***Metrics:***

* **Execution Time**: Measure the time taken for the initial SSSP and dynamic updates.
* **Speedup**:

Speedup = T(1)/T(P)

* **Efficiency**:

Efficiency = Speedup / P

* **Communication Overhead**: Measure the amount of data sent and the time spent on communication between processes.

**6. Results**

**6.1 Initial SSSP Performance**

* Table and/or figure showing runtime vs. the number of processes for each configuration.

**6.2 Update Processing Performance**

* **Runtime** to process a batch of N updates versus the number of processes.
* Breakdown of time spent on local updates versus communication time between partitions.

**6.3 Scalability**

* **Strong Scaling**: Plots showing the relationship between speedup and the number of processes.
* **Weak Scaling**: Plots showing the relationship between execution time and the number of processes for a fixed amount of work per process.

**6.4 Partition Quality**

* **Edge-cut and balance** metrics obtained from METIS.
* **Impact on Communication**: Examine how partitioning quality impacts the communication volume and runtime.

**7. Discussion**

* **Trends Observed**: The performance scaling, diminishing returns, and communication bottlenecks are discussed.
* **Effectiveness of OpenMP/OpenCL**: Evaluate how well intra-node parallelism (OpenMP) and GPU acceleration (OpenCL) improved the algorithm's performance.
* **Machine Limitations**: Discuss performance limitations on an 8 GB machine and why a fallback strategy was necessary for large graphs.

**8. Conclusion and Future Work**

* **Summary of Findings**: The algorithm shows promising results in parallelizing the dynamic update of the SSSP problem. Significant speedup was observed, especially with OpenMP and OpenCL acceleration.
* **Recommendations for Larger Systems**: Consider upgrading to larger systems with more memory and better interconnects.
* **Future Work**:
  + Explore **ParMETIS** for more efficient parallel partitioning.
  + Implement **dynamic repartitioning** for improved load balancing.
  + Investigate **asynchronous communication** to reduce idle times.

**9. References**

1. A. X. et al., "A Parallel Algorithm Template for Updating SSSP in Large‑Scale Dynamic Networks," *Journal of Parallel and Distributed Computing*, vol. XX, pp. 123-134, 2023.
2. METIS manual, http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.
3. OpenMP API Specification, https://www.openmp.org/specifications/.
4. OpenCL Specification, https://www.khronos.org/opencl/.

**10. Appendices**

* **Source Code Snippets**: Partitioning step code and process\_batch function implementation.
* **Raw Timing Data**: Full tables listing timing results for each experiment.
* **Reproducibility Instructions**: Detailed steps for setting up the environment and running experiments.