**Parallel and Distributed Computing**

Ali Aamir Ashraf 22i-1199  
Zaid Ahmed 21i-1504  
Ali Haider 21i-1522

**Performance Evaluation Report**

*SSSP IN LARGE DYNAMIC GRAPHS*

**Project: SSSP in Large-Scale Dynamic Networks**

**1. Introduction**

The Single-Source Shortest Path (SSSP) problem seeks the minimum-distance paths from a designated source vertex to all others in a graph. In dynamic networks—where edges or weights change over time—efficiently updating these shortest paths without a full recompute is critical for real-time applications (e.g. traffic routing, network resilience). This report compares a serial baseline against an OpenMP-parallel update algorithm, evaluating both correctness and performance.

**2. Literature Review**

Khanda *et al.* (2022) propose a generic parallel update template for SSSP in large-scale dynamic networks, combining full-recompute and incremental-update strategies to balance work across threads while minimizing redundant computation.

**3. Implementation Details**

3.1 Serial Implementation

1. Graph building:
   * EdgeList reads ⟨u, v, w⟩ tuples and constructs a CSR (GraphCSR).
2. Initialization:
   * SsspUpdater runs Dijkstra’s algorithm (min-heap) from the chosen source, storing dist[].
3. Processing changes:
   * Each dynamic edit (insert/delete) from Change is applied to the CSR.
4. Update strategy:
   * After every change‐batch, SSSP is recomputed from scratch via Dijkstra.
5. I/O:
   * New distance vector output after each batch.

3.2 OpenMP Parallel Implementation

1. Parallel initialization:
   * Builds CSR serially, then uses
   * #pragma omp parallel for
   * for (auto& v : vertices) dist[v] = …;

to initialize dist[].

1. Hybrid Dijkstra (optional):
   * Leaves priority-queue operations serial, but parallelizes the edge-relaxation loops.
2. Batch update:
   * Groups a batch of changes, then executes
   * #pragma omp parallel for schedule(dynamic)
   * for (auto& e : edges) relax(e);
   * #pragma omp barrier
3. Load balancing:
   * Uses dynamic scheduling to spread work on power-law graphs; minimizes critical sections around heap pops.

**4. Experimental Setup**

4.1 Software Environment

* -fopenmp used for the parallel build
* OpenMP version: 4.5
* CMake: 3.16

4.2 Datasets

* grqc (Physics collaboration graph): 5 242 vertices, 28 980 edges
* roadNet-CA: 2 million vertices, 5 million edges

**5. Performance Evaluation**

5.1 Evaluation Metrics

* Execution time (ms per run)
* Speedup Sp=T1/TpSₚ = T₁/Tₚ
* Efficiency Ep=Sp/pEₚ = Sₚ/p

5.2 Results

Figure 1. grqc: Serial vs OpenMP Execution Time

A graph with orange line

AI-generated content may be incorrect.

Figure 2. roadNet-CA: Serial vs OpenMP Execution Time

A graph of a line

AI-generated content may be incorrect.

5.3 Output Screenshots

A screen shot of a computer

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.

6. Discussion

* Serial baseline: Reliable but limited to single-core performance.
* OpenMP version: Near-linear speedup up to 8 threads on our test hardware; on very large graphs, synchronization and atomic overhead begin to dominate.
* Bottlenecks: Priority-queue operations remain largely serial; load imbalance in high-degree hubs.

7. Conclusion and Future Work

The OpenMP-parallel update template yields substantial speedups—especially on large, sparse graphs—validating Khanda *et al.*’s approach. Future directions include a fully parallel priority queue, NUMA-aware graph partitioning, and exploring GPU offload for relaxation steps.

References

1. Khanda, A., Srinivasan, S., Bhowmick, S., Norris, B., & Das, S. K. (2022). A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks. *IEEE Transactions on Parallel and Distributed Systems*.