PDC Presentation: Updating SSSP in Dynamic Networks

Analysis and Parallelization Strategy

Mohammad Sooban & Umar Nadeem

April 2025

Department of Computer Science

Introduction

- Single Source Shortest Path (SSSP) is crucial in many domains.
- Traditional algorithms assume static graphs.
- Real-world networks are large-scale and dynamic.
- Goal: Update SSSP efficiently without full recomputation.

Motivation

- SSSP needed in:
  - Social networks, transportation, cybersecurity.
- Static algorithms like Dijkstra's are inefficient on frequent updates.
- Recomputing the entire tree wastes computation.

Main Contributions

- A scalable parallel algorithm template for updating SSSP in dynamic networks.
- Platform-agnostic design works on:
  - Shared-memory CPUs (OpenMP)
  - GPUs (CUDA)
- Key features:
  - Rooted-tree structure
  - Edge-change batching
  - Asynchronous updates

Algorithm Overview

1. Identify subgraphs affected by edge insertions or deletions.
2. Update only affected distances and parents.

Data Structures: Distances, Parents, and Affected flags.

Parallelization Strategy

Strategy Overview

- Combine:
  - **MPI** – For inter-node communication
  - **OpenMP/OpenCL** – For intra-node parallelism
  - **METIS** – For graph partitioning
- Enables distributed and shared-memory hybrid parallelism.

MPI – Inter-Node Communication

- Graph is partitioned across nodes.
- Each node processes its local subgraph.
- Border updates (ghost nodes) exchanged using:
  - `MPI_Isend`, `MPI_Irecv`, `MPI_Barrier`
- Benefits:
  - Scalability
  - Distributed load

OpenMP / OpenCL – Intra-Node Parallelism

- OpenMP used for CPU thread-level parallelism.
- Dynamic scheduling avoids workload imbalance.
- OpenCL enables cross-device parallelism (CPU/GPU).
- Converging updates remove need for locks or atomic ops.

METIS – Graph Partitioning

- METIS partitions the graph to:
  - Minimize cross-node edges (edge cut)
  - Balance vertex load
- Input for MPI-distributed algorithm.
- Preprocessing step improves runtime and scalability.

Implementation and Results

GPU Implementation

- Uses **Vertex-Marking Functional Blocks (VMFB)**:
  - Mark $\rightarrow$ Sync $\rightarrow$ Filter steps
- Parallel update of affected vertices.
- Avoids heavy atomic usage.
- Achieves up to **8.5x speedup** vs. Gunrock (recompute).

CPU Shared Memory Implementation

- Implemented using OpenMP and C++.
- Supports:
  - Asynchronous updates
  - Batch processing of edge changes
- Up to **5x faster** than recomputation with Galois.

Experimental Observations

- Works best when:
  - Edge changes are mostly insertions.
  - Fewer than 75–80% nodes affected.
- For large deletions or mass updates, full recomputation may be better.

Conclusion

Conclusion

- A robust, parallel update framework for SSSP in dynamic graphs.
- Efficient across shared and distributed architectures.
- Recommended strategy:
  - MPI + OpenMP/OpenCL + METIS
- Future work:
  - Hybrid switch between update/recompute
  - Predictive scheduling

Thank You!

Questions?