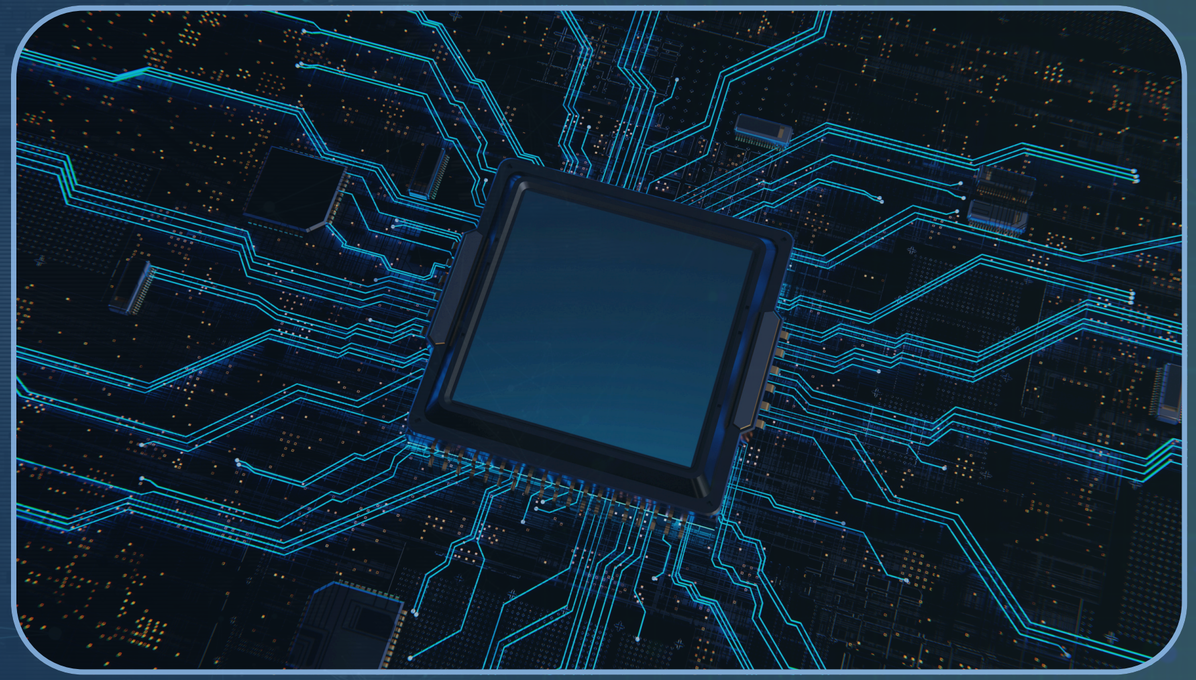
A PARALLEL ALGORITHM TEMPLATE FOR UPDATING SINGLE-SOURCE SHORTEST PATHS IN LARGE-SCALE DYNAMIC NETWORKS

Start

A presentation by:
Abeer Jawad 22i-1041
Mahum Hamid 22i-1009
Maryum Fasih 22i-0756

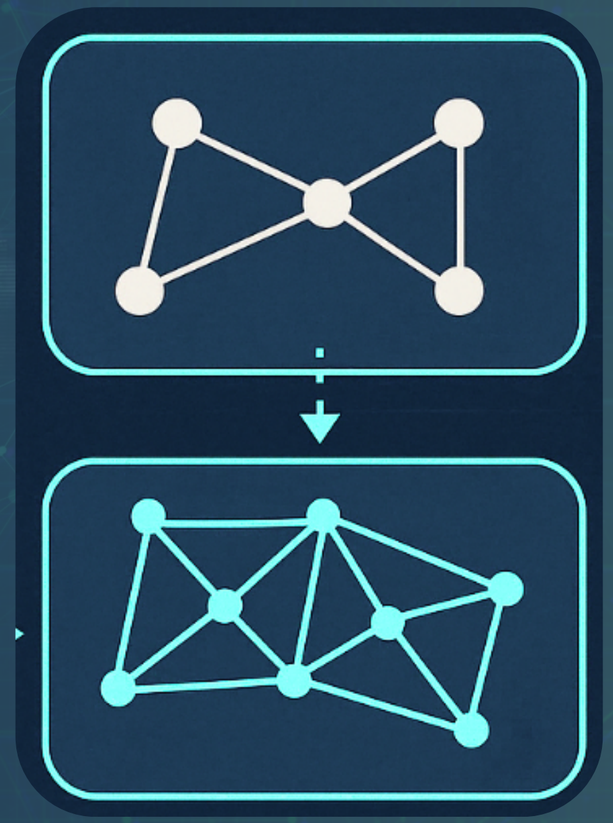
INTRODUCTION

- Real-world networks are large, dynamic
- Static SSSP algorithms are inefficient for dynamic updates
- Recomputing from scratch is slow and not scalable
- Need for a fast, scalable, and parallel update method
- Existing solutions are platform-specific — lack portability
- Aim: Create a unified, efficient SSSP framework for CPU & GPU



KEY CONTRIBUTIONS

- Bridges the gap between static SSSP algorithms and dynamic, large-scale networks
- Proposes a parallel 2-step update framework:
  - Step 1: Identify affected subgraph
  - Step 2: Iteratively update distances
- Uses rooted tree structure to track paths
- No explicit synchronization needed
- Fully implemented on both CPU and GPU
- Handles 100M+ batch edge updates
- Up to 5.6× GPU and 5.0× CPU speedup over traditional recomputation



PROBLEM STATEMENT

- Given a dynamic graph G(V, E) where edges can be inserted or deleted over time
- Need to maintain SSP from a source node efficiently
- Insertions (Ins_k) and Deletions (Del_k) of edges
- Goal: Update the SSSP tree T_k after changes, using the previous tree T_{k-1}
- Must ensure:
  - Correctness
  - Faster performance than recomputation
  - Parallel execution

FRAMEWORK

- Platform Independent.
- Works on both CPU and GPU.

1.

CPU

OpenMP based parallesim

Async updates + batch processing

2.

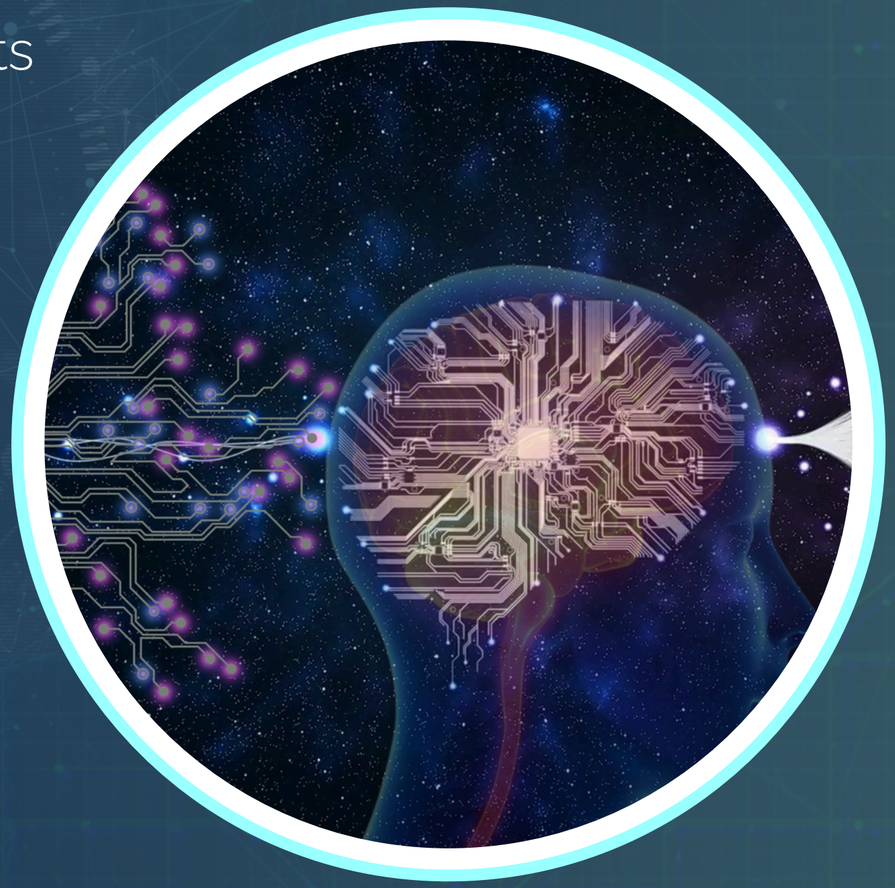
GPU

CUDA with VFMB

Fewer atomics, minimal overhead

PREPROCESSING

- Partition the static graph using METIS
- Generate K balanced partitions with minimal edge cuts
- Assign each partition to a unique MPI rank
- For each partition store:
  - Local adjacency list
  - List of boundary vertices
- Initialize SSSP data structures for all local vertices:
  - distance from sources
  - parent pointer in the SSSP tree
  - affected flag (initially set to false)



PARALLELIZATION STRATEGY

1. Step 1: Affected Subgraph Detection

2. Step 2: Incremental Update

STEP 1: AFFECTED SUBGRAPH DETECTION

- Distribute edge updates across MPI ranks
- Deletions: If edge is in SSSP → set child's distance = ∞, parent = NULL, mark as affected
- Insertions: If new edge shortens path → update distance, parent, mark as affected
- Parallelized using OpenMP/OpenCL — no synchronization needed

STEP 2: INCREMENTAL UPDATE

- Iteratively relax affected vertices in parallel using OpenMP/OpenCL
- For each affected vertex:
  - Relax outgoing edges
  - Update neighbors if shorter path is found
  - Flag updated neighbors for next iteration
- Continue until no affected = true flags remain

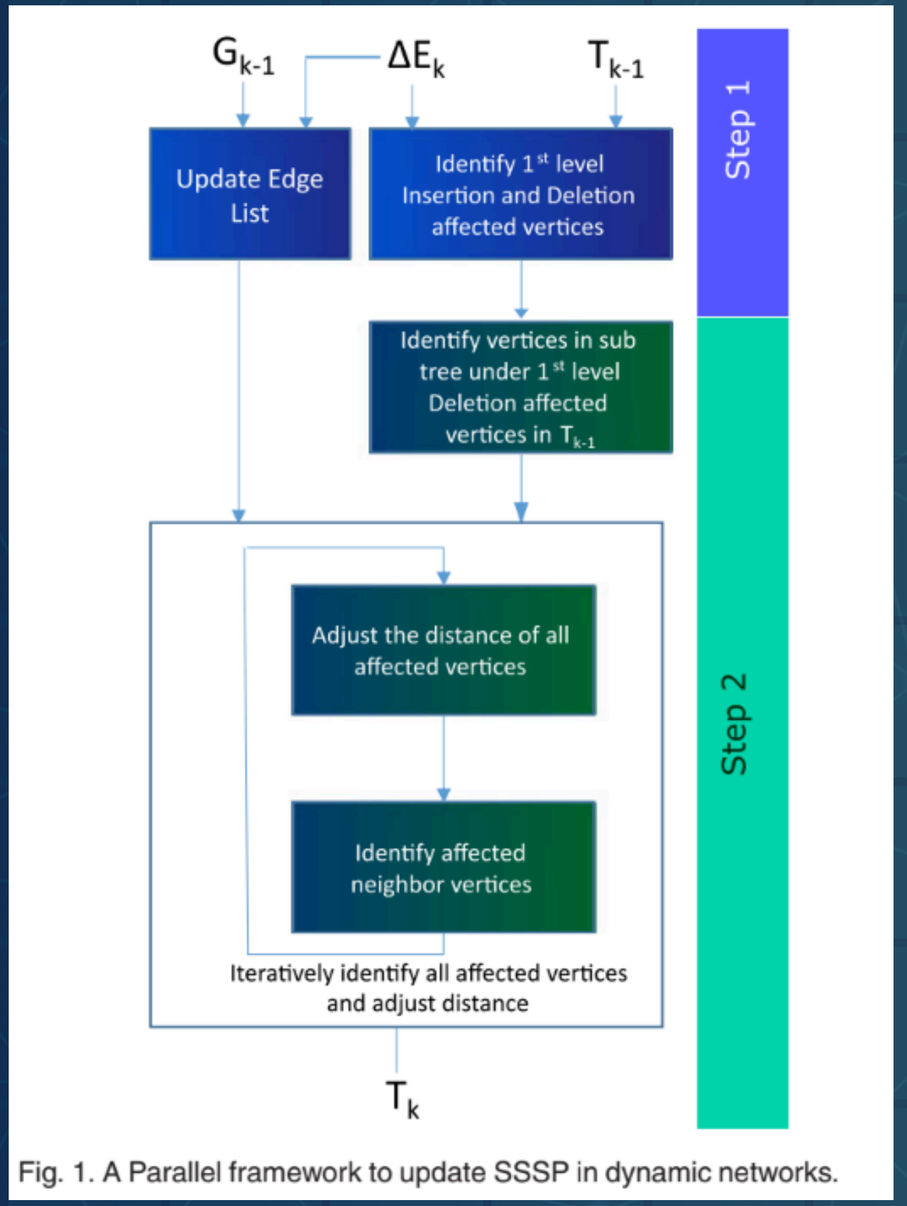


Fig. 1. A Parallel framework to update SSSP in dynamic networks.

CROSS-PARTITION SYNCHRONIZATION

- Resolves dependencies between partitions after local updates
- Boundary vertex exchange:
  - Each MPI rank sends updated distances of boundary vertices to neighboring ranks
  - Uses non-blocking MPI to overlap communication
- Process received updates:
  - if the new distance is shorter, update local data and mark as affected
- Global convergence check:
  - Use MPI_Allreduce to check if any rank has affected = true vertices
  - If yes, repeat local propagation and synchronization

THANK YOU!

QUESTIONS?