PARALLEL ALGORITHM FOR UPDATING SINGLE-SOURCE SHORTEST PATHS IN DYNAMIC NETWORKS

Presented by:

- Haris Ahmed
- Aazib Abdullah
- Amir akbar

INTRODUCTION TO PROBLEM DOMAIN

- Graphs are essential in modelling real-world networks
- Single-Source Shortest Path (SSSP) problems are core in graph analysis.
- Real-world networks (social, traffic) constantly change
- Traditional recomputation is costly and inefficient

PROBLEM STATEMENT

- Dynamic updates to graphs disrupt shortest paths.
- Traditional algorithms like Dijkstra's don't scale well on updates.
- Need for a parallel, platform-independent update framework
- Avoid recomputation by updating affected subgraphs

SSSP PROBLEM & CHALLENGES

- Goal: Find shortest paths from one source to all nodes
- Dynamic graphs: edge insertions and deletions
- Key issues: synchronization, load imbalance, performance

FRAMEWORK OVERVIEW

- Two-step update strategy:
  - Identify affected subgraph
  - Update SSSP tree locally
- Parallel and asynchronous processing
- Works without locks or critical sections

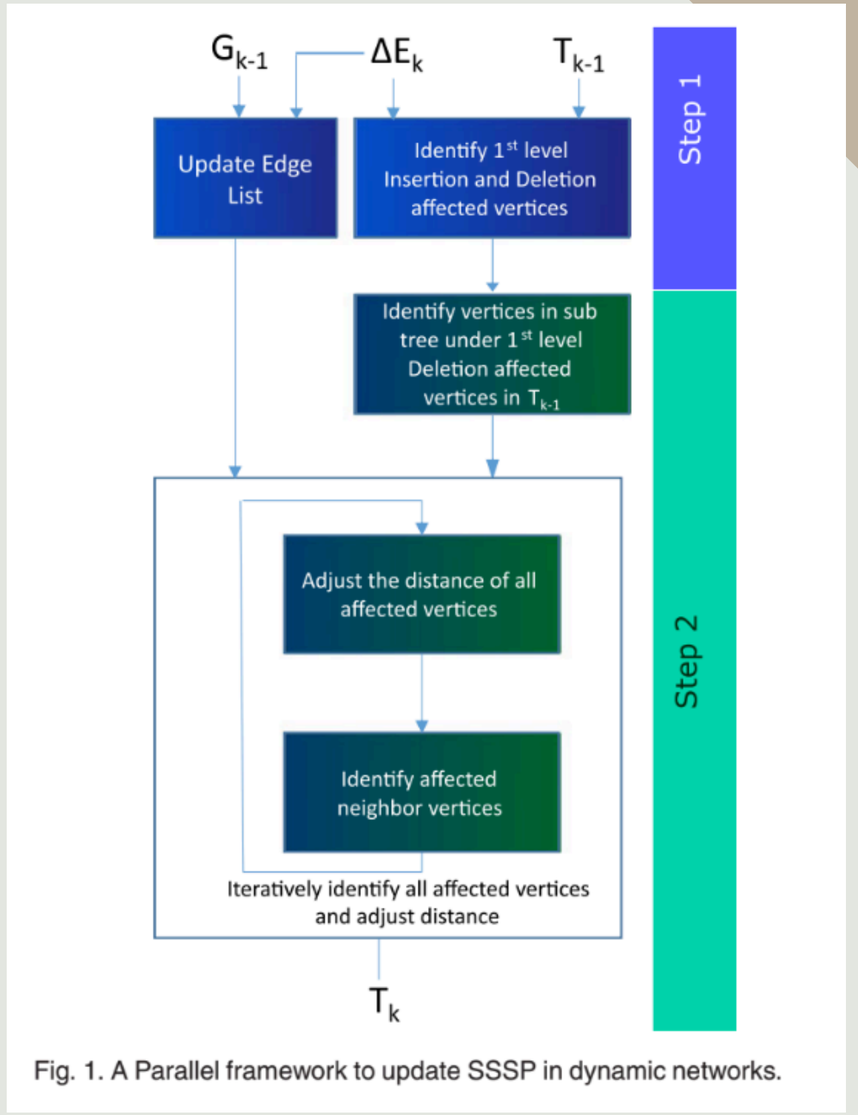

Fig. 1. A Parallel framework to update SSSP in dynamic networks.

STEP 1 – IDENTIFYING AFFECTED SUBGRAPH

- Parallel processing of changed edges
- Deletions: disconnect affected subtrees
- Insertions: update if new shorter path is found
- Affected vertices marked using flags

Algorithm 2. Identify Affected Vertices

1: **Function** ProcessCE ($G, T, \Delta E, Parent, Dist$):
2: Initialize Boolean arrays $Affected\_Del$ and $Affected$ to $false$
3: $G_u \leftarrow G$
4: **for** *each edge* $e(u,v) \in Del_k$ *in parallel* **do**
5: **if** $e(u,v) \in T$ **then**
6: $y \leftarrow argmax_{x \in \{u,v\}}(Dist[x])$
7: Change $Dist[y]$ to infinity
8: $Affected\_Del[y] \leftarrow True$
$Affected[y] \leftarrow True$
9: Mark $e(u,v)$ as deleted
10: **for** *each edge* $e(u,v) \in Ins_k$ *in parallel* **do**
11: **if** $Dist[u] > Dist[v]$ **then**
12: $x \leftarrow v, y \leftarrow u$
13: **else**
14: $x \leftarrow u, y \leftarrow v$
15: **if** $Dist[y] > Dist[x] + W(x,y)$ **then**
16: $Dist[y] \leftarrow Dist[x] + W(x,y)$
17: $Parent[y] \leftarrow x$
18: $Affected[y] \leftarrow True$
19: Add $e(u,v)$ to G_u

STEP 2 – UPDATING SUBGRAPH

- Update affected vertices iteratively
- Traverse neighbors to update distances and parents
- Uses convergence logic instead of synchronization
- Ends when no updates remain

Algorithm 3. Update Affected Vertices

1: **Function** `UpdateAffectedVertices` (G_u, T, $Dist$, $Parent$ $Affected\_Del$ and $Affected$):
2: **while** $Affected\_Del$ has true values **do**
3: **for** each vertex $v \in V$ such that $Affected\_Del[v] = true$ in parallel **do**
4: $Affected\_Del[v] \leftarrow false$
5: **for** all vertex c, where c is child of v in the SSSP tree T **do**
6: Set $Dist[c]$ as infinity
7: $Affected\_Del[c] \leftarrow True$
8: $Affected[c] \leftarrow True$
9: **while** $Affected$ has true values **do**
10: **for** each vertex $v \in V$ such that $Affected[v] = true$ in parallel **do**
11: $Affected[v] \leftarrow False$
12: **for** vertex n, where $n \in V$ and n is neighbor of v **do**
13: **if** $Dist[n] > Dist[v] + W(v, n)$ **then**
14: $Dist[n] \leftarrow Dist[v] + W(v, n)$
15: $Parent[n] \leftarrow v$
16: $Affected[n] \leftarrow True$
17: **else if** $Dist[v] > Dist[n] + W(n, v)$ **then**
18: $Dist[v] \leftarrow Dist[n] + W(n, v)$
19: $Parent[v] \leftarrow n$
20: $Affected[v] \leftarrow True$

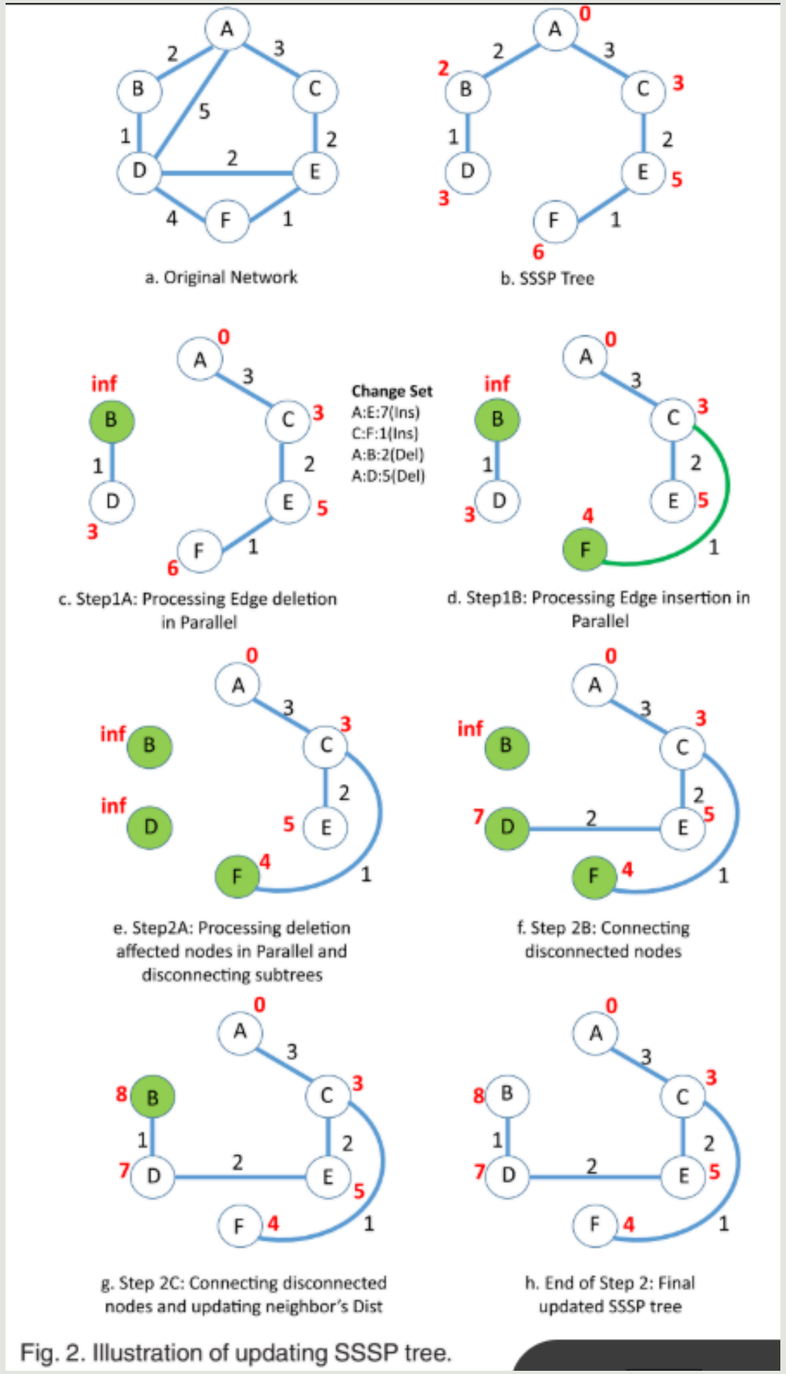

Fig. 2. Illustration of updating SSSP tree.

GPU & CPU IMPLEMENTATION

- OpenMP (CPU): dynamic scheduling, async updates, batches
- CUDA (GPU):
  - Uses VMFB (Vertex-Marking Functional Block)
  - Reduces atomics with a 3-stage marking process
  - Graph stored in CSR format

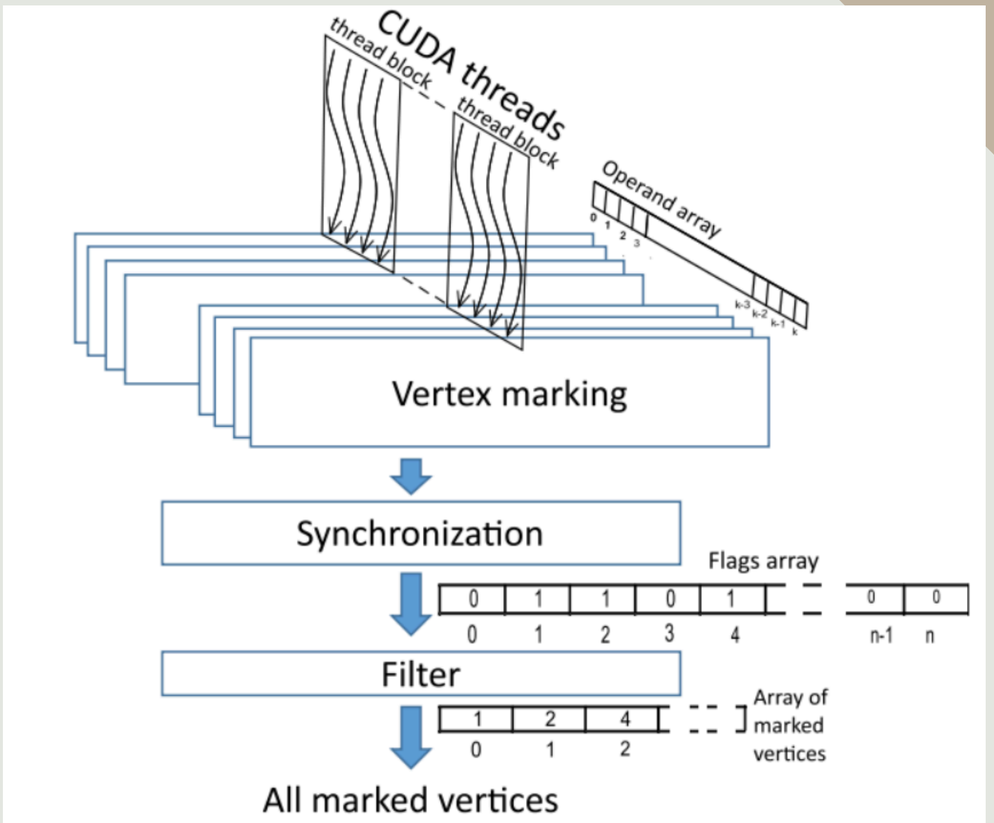

Fig. 4. Vertex-marking functional block (VMFB).

EXPERIMENTAL RESULTS

- Tested on 6 graphs (up to 16M nodes, 258M edges)
- GPU (Tesla V100): up to 8.5x faster than Gunrock
- CPU (Intel Xeon, 72 threads): up to 5x faster than Galois
- Performs best when insertions ≥ 50% of updates

TIME & SPACE COMPLEXITY

- Step 1: O(m/p) — m = changes, p = processors
- Step 2: O(Dxd/p) — D = diameter, x = affected vertices, d = avg. degree
- Memory: O(V) for all data structures (Parent, Dist, Affected)

PARALLELIZATION STRATEGY – OVERVIEW

- Enhance using 3 parallel computing tools:
  - **MPI**: inter-node communication
  - **OpenMP/OpenCL**: intra-node processing
  - **METIS**: graph partitioning

MPI – INTER-NODE PARALLELISM

- Use **MPI** to distribute graph partitions across multiple nodes
- Each node works independently on local subgraphs
- Exchange border vertex updates with neighbors
- Ideal for large graphs on HPC clusters
- Reduces global communication by limiting to boundary vertices
- Integrates well with METIS-partitioned graph structure

OPENMP / OPENCL – INTRA-NODE PARALLELISM

- **OpenMP:**
  - Efficient for shared-memory systems
  - Use threads to process affected vertices/subtrees concurrently
  - Dynamic scheduling handles load imbalance
- **OpenCL:**
  - Platform-independent GPU support (NVIDIA, AMD, Intel)
  - Port CUDA kernels for cross-device compatibility
  - Effective for marking, filtering, and distance updates

METIS – GRAPH PARTITIONING

- METIS partitions graph into subgraphs with minimal edge cuts
- Balances vertex count across partitions
- Ensures locality and load balancing in MPI execution
- Helps reduce inter-node communication
- Required preprocessing step for distributed SSSP update

WHEN TO USE WHICH STRATEGY

Environment	**Recommended Strategy**
Distributed cluster	MPI + METIS
Multicore server	OpenMP
GPU system	CUDA/OpenCL
Mixed updates	Hybrid decision engine

CONCLUSION & FUTURE WORK

- Efficient parallel SSSP update for dynamic graphs
- Reduces overhead vs. recomputation
- Platform-agnostic: runs on CPU and GPU
- Future Work:
  - Hybrid recompute/update logic
  - Prediction models for updates
  - Extend to other metrics (centrality, PageRank)

Thank You

For your attention