A shared-memory algorithm for updating single-source shortest paths in large weighted dynamic networks

Presented by

Sriram Srinivasan (University of Nebraska at Omaha)

sriramsrinivas@unomaha.edu,

github.com/DynamicSSSP/HIPC18

The graph problem

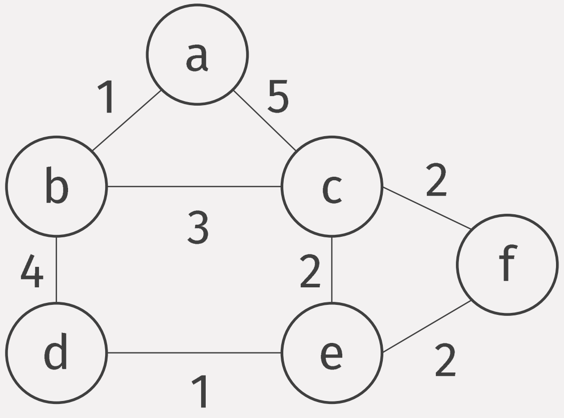
Given a weighted graph, a source vertex, a single-source shortest paths (SSSP) tree, and a batch of graph edge insertions and deletions, update the SSSP tree.

The graph problem

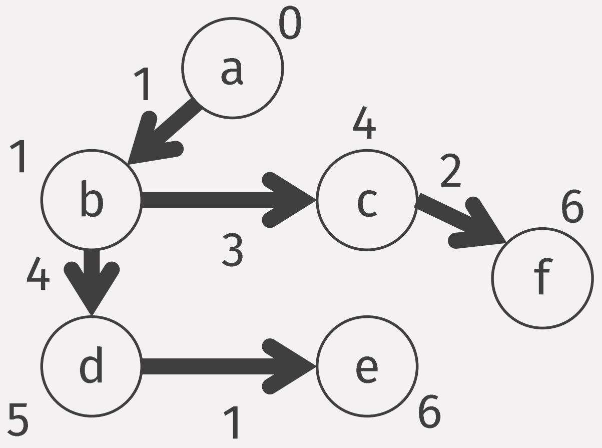
Given a weighted graph, a source vertex, a single-source shortest paths (SSSP) tree, and a batch of graph edge insertions and deletions, update the SSSP tree.

Inputs

Graph



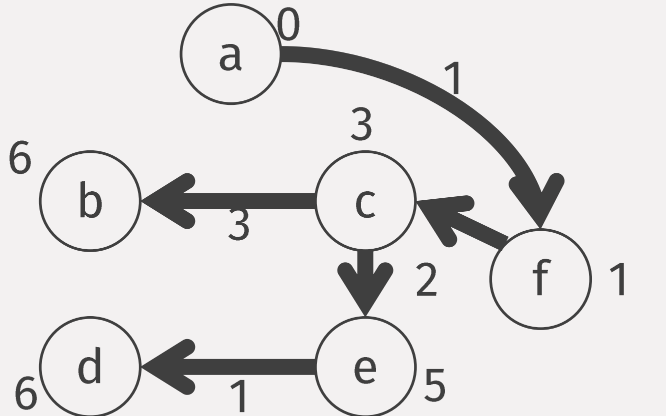
An SSSP tree rooted at vertex a



Graph updates

Insert edge [a, f] of weight 1
Insert edge [c, d] of weight 4
Delete edge [a, c]
Delete edge [a, b]

Output



The naïve approach

Recompute from scratch

Generate new graph after updates, perform SSSP again

Can we do better than recomputing from scratch?

Contributions

A new two-step parallel algorithm for updating the SSSP

- optimizations to improve scalability
- optimizations to reduce redundant/wasteful computation

Correctness proof (see paper)

Empirical evaluation to demonstrate speedup over recomputing SSSP from scratch

S. Srinivasan, S. Riazi, B. Norris, S. K. Das and S. Bhowmick, "A Shared-Memory Parallel Algorithm for Updating Single-Source Shortest Paths in Large Dynamic Networks," *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, Bengaluru, India, 2018, pp. 245-254.

Why dynamic SSSP?

Many applications

- Maps and GPS
- Internet routing
- Path planning for robots
- Discrete event simulations
- Centrality analysis in complex networks

Related Work

Parallel algorithms, implementations for SSSP in static graphs

e.g., Delta-stepping, DSMR

Libraries for dynamic data/graph analysis

e.g., Sandia PHISH, Georgia Tech Stinger

Dynamic graph algorithms

e.g., Ramalingam-Reps, Narvez et al.

Assume batched updates

Consider a sequence of insertions and deletions

Edge operations considered

Vertex insertions and deletions can be modeled by adding and deleting edges

Observations about graph updates

Updates may only affect a subgraph and the complete graph need not be analyzed

Not all updates affect the property (SSSP in this paper)

updates can be processed in parallel

Not all updates affect the same subgraph

affected subgraphs can be processed in parallel

A template for parallel dynamic graph algorithms

Sparsification

Preprocessing step before graph updates

Selection

Identify vertices and edges affected by graph updates

Can be parallelized for each update

Updating

Update set of key edges according to changes

Might require multiple iterations for convergence

Previously-unaffected entities may also be affected

New algorithm for dynamic SSSP

Sparsification: use rooted SSSP tree

Selection: identify affected vertices and edges with a simple distance label check

Updating: propagate changes, iterate until convergence

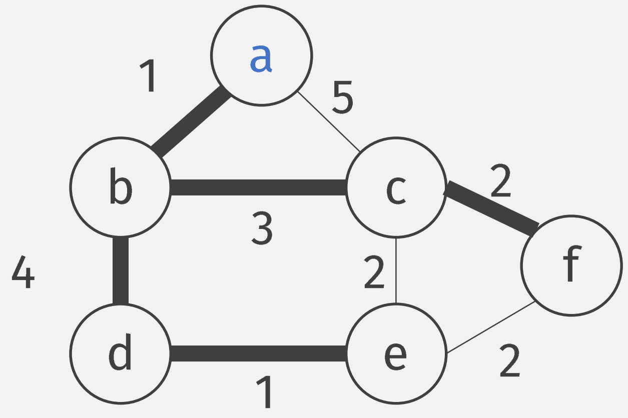
Selection step

Assume dist(s, v) > dist(s, u)

An edge insertion update [u, v, w(u, v)] affects the SSSP tree if dist(s, v) > dist(s, u) + w(u, v). If this condition is met, v is marked as affected.

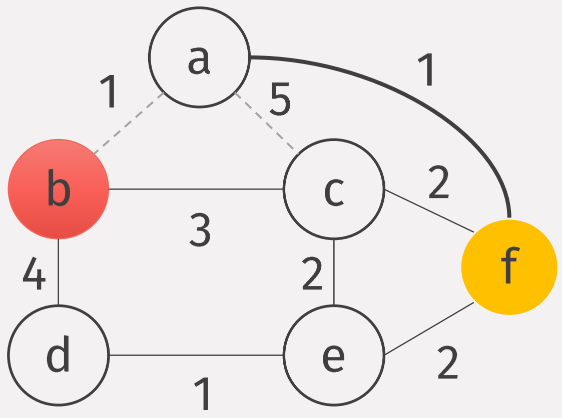
An edge deletion update [u, v, w(u, v)] affects the SSSP tree if the edge is present in the SSSP tree. v's distance label is set to Infinity.

Selection step



Graph updates

Insert edge [a, f] of weight 1
Insert edge [c, d] of weight 4
Delete edge [a, c]
Delete edge [a, b]



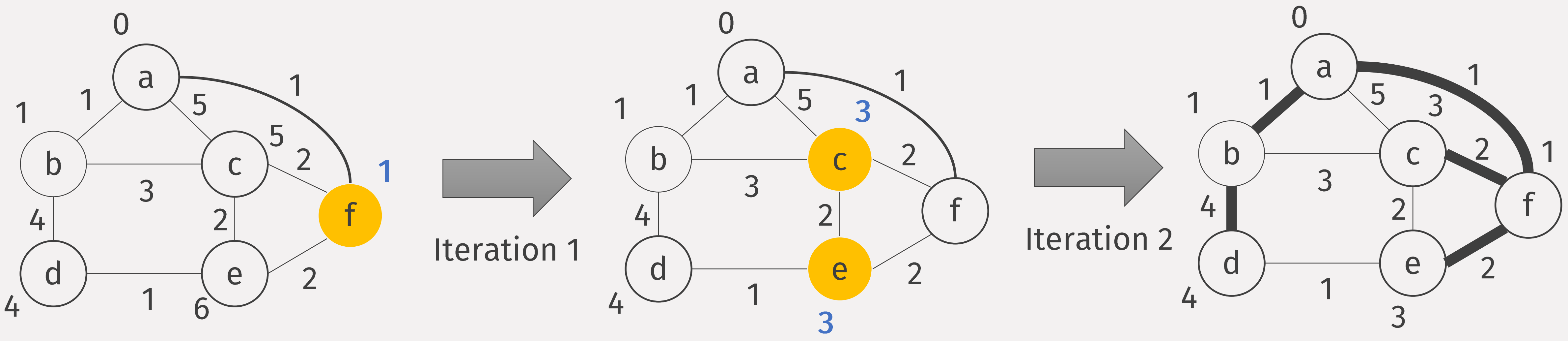
Updating step

The effect of each inserted edge can percolate to a large subgraph, possibly the entire graph

Reduces to edge relaxations from affected vertices

Relaxations can be performed concurrently. Convergence when there are no more affected vertices.

Updating step with single edge insertion



Shared-memory parallelization

The Selection step is easy to implement and shows good load balance

The parallel performance of the Updating step is dependent on the number of affected vertices and the size of the subgraphs they alter. Vertex degree distributions can cause further load imbalance.

Asynchronous updates: can process longer paths instead of just neighbors. Reduce number of synchronization steps.

Empirical results

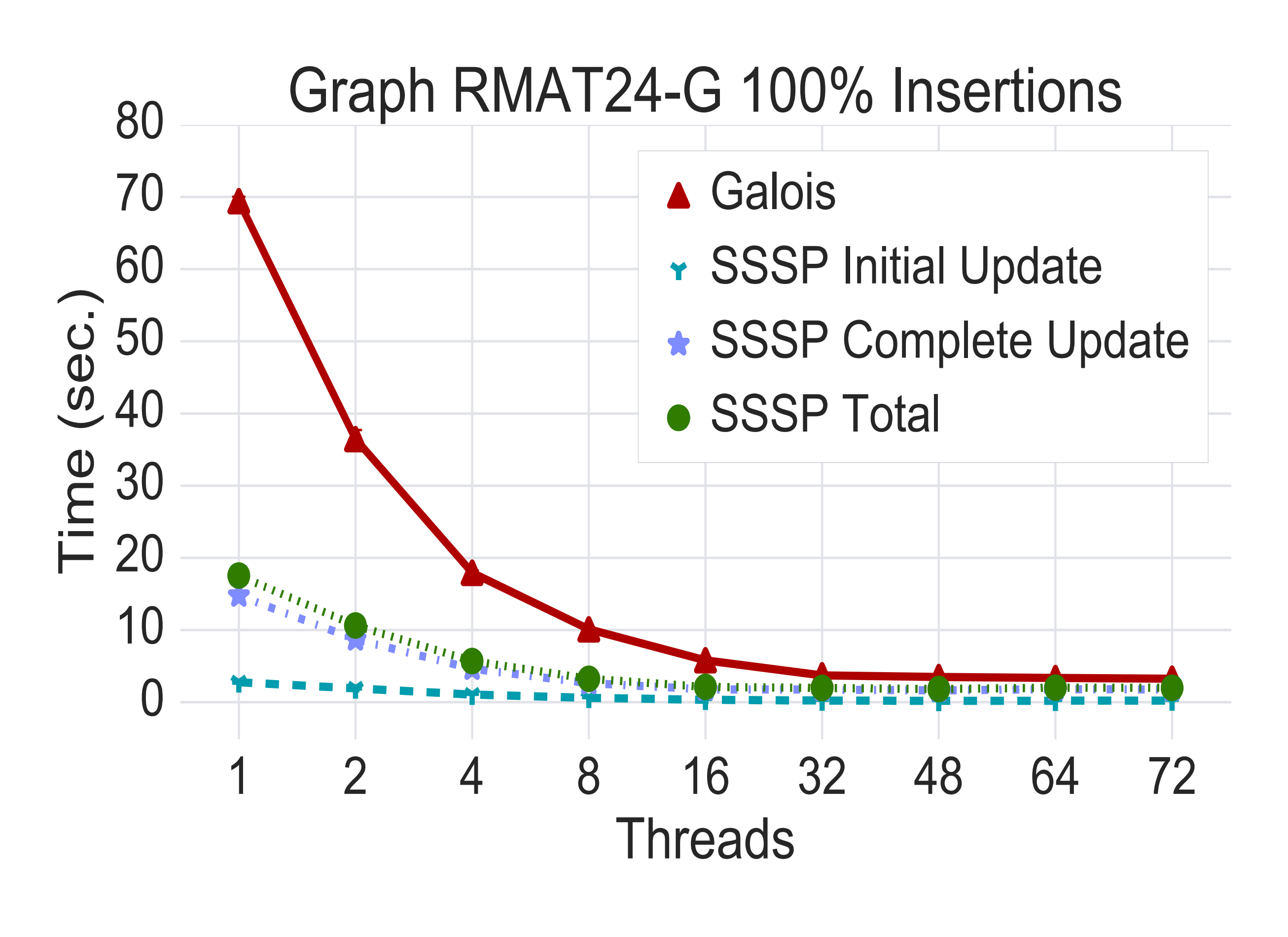
Results on a 36-core Intel Haswell system with 256 GB memory

OpenMP implementation

Comparison to SSSP implementation in Galois v2.2.1

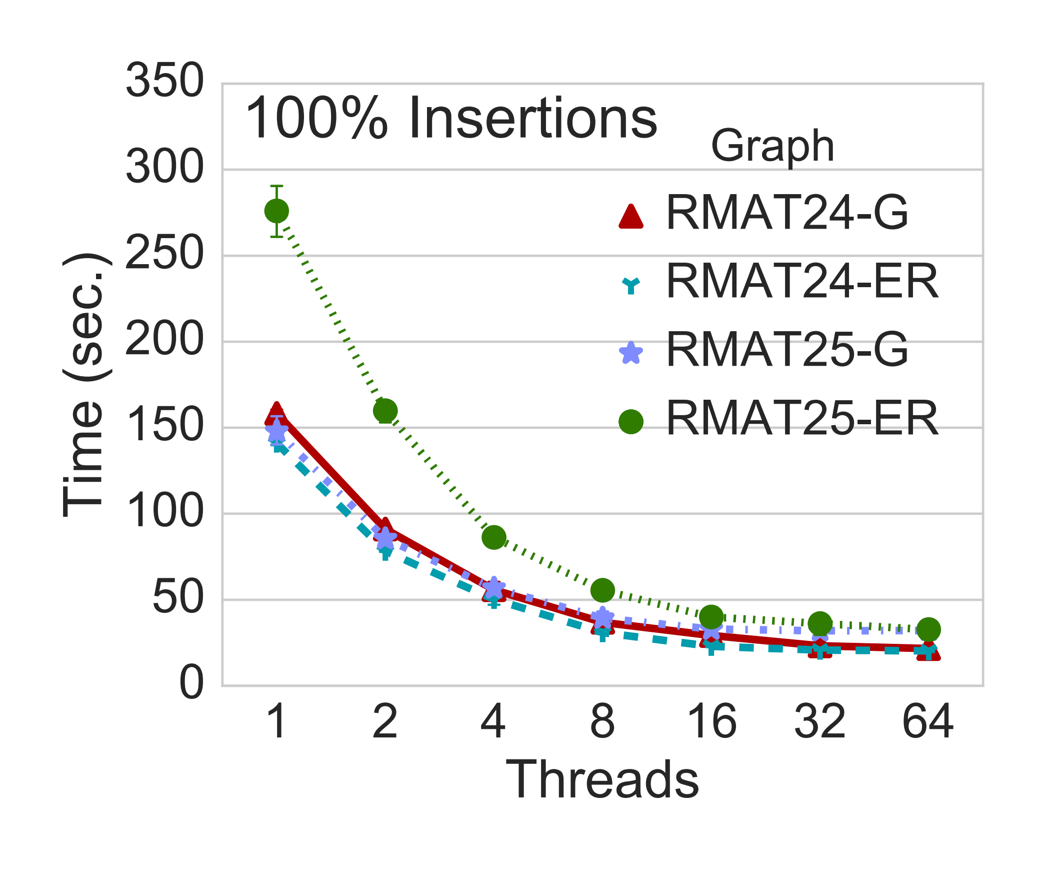
Synthetic RMAT-G (skewed degree distribution) and RMAT-ER (normal degree distribution) graphs, three real-world graphs from SNAP

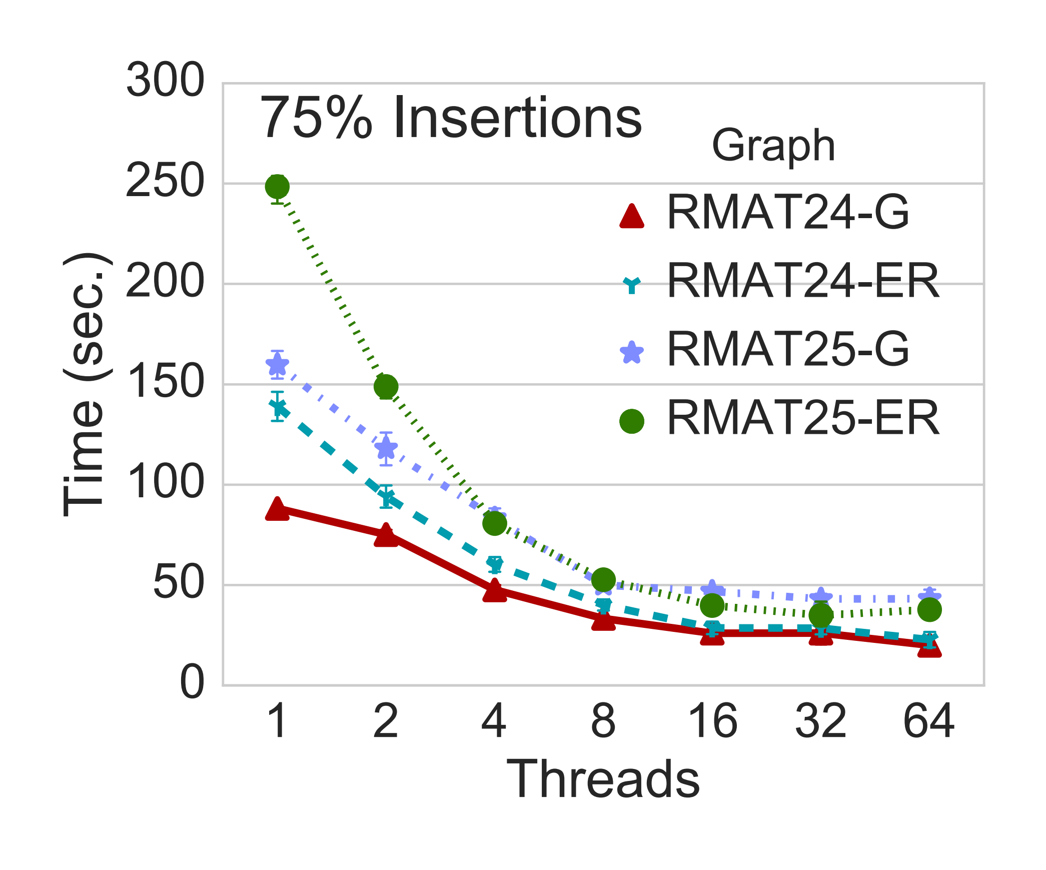
Comparison to recomputation-based approach



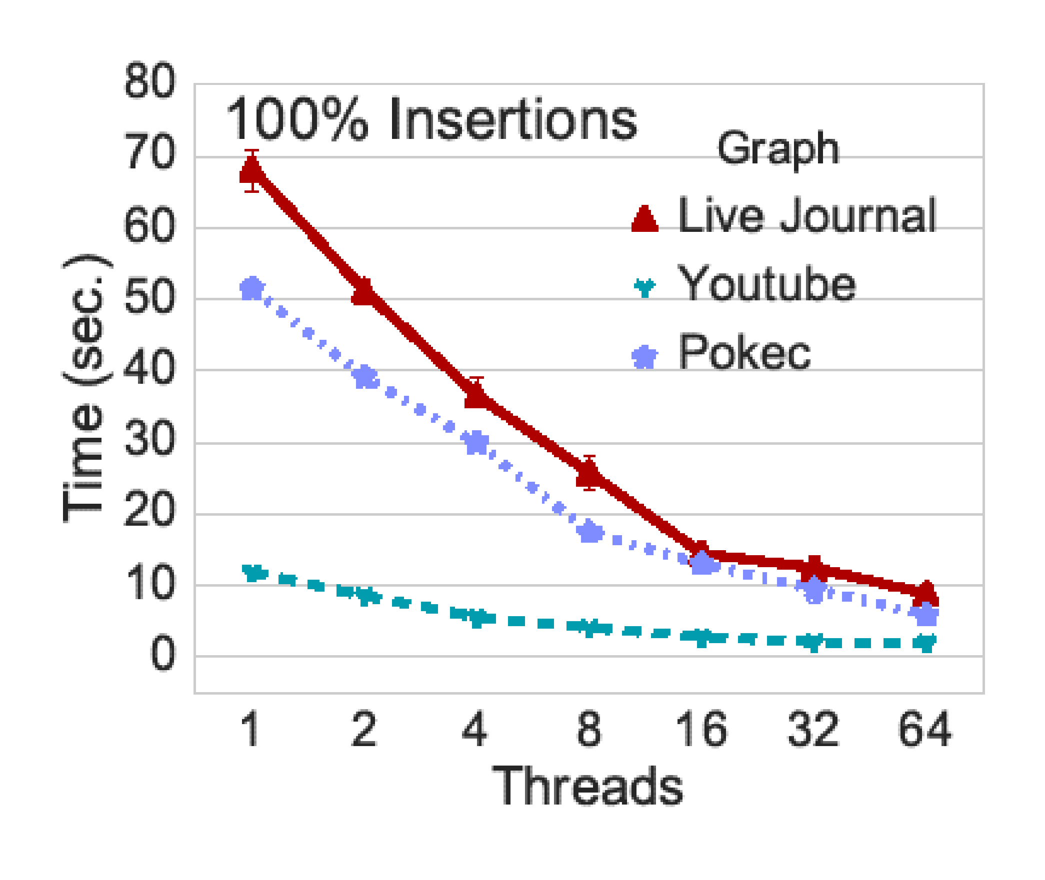
New algorithm is up to 4X faster.

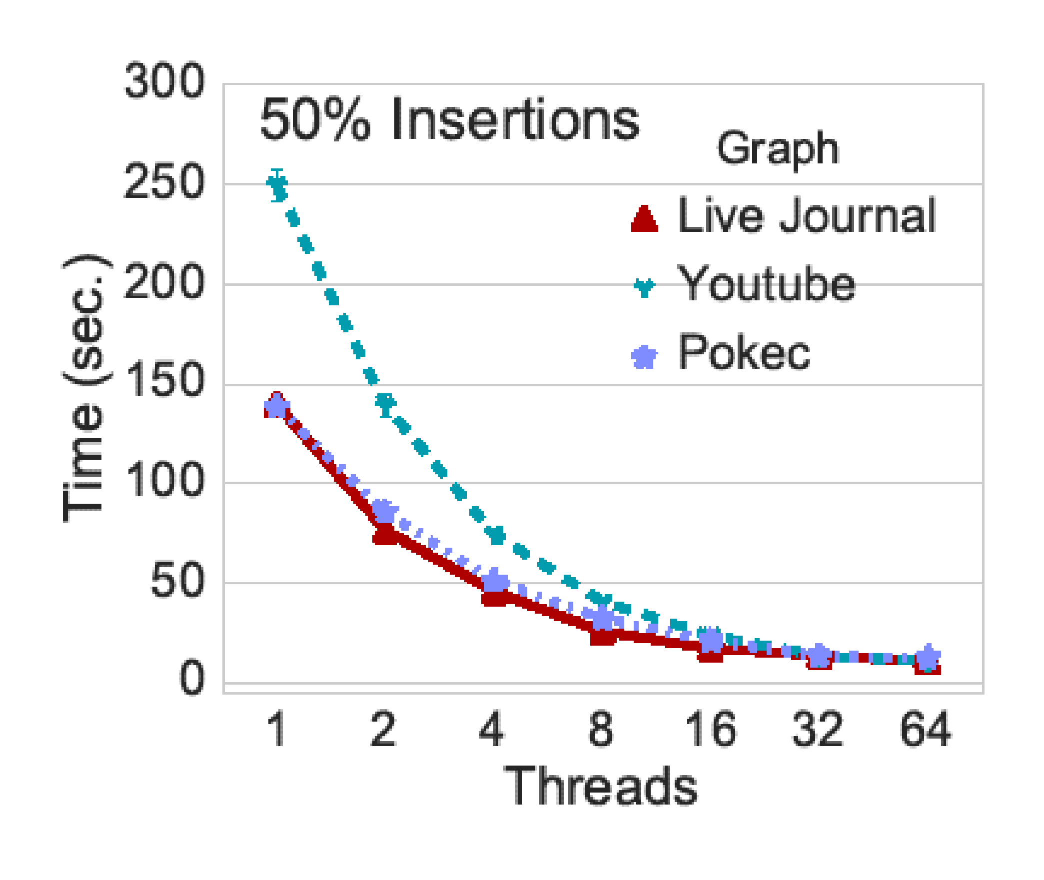
Strong scaling (synthetic graphs)



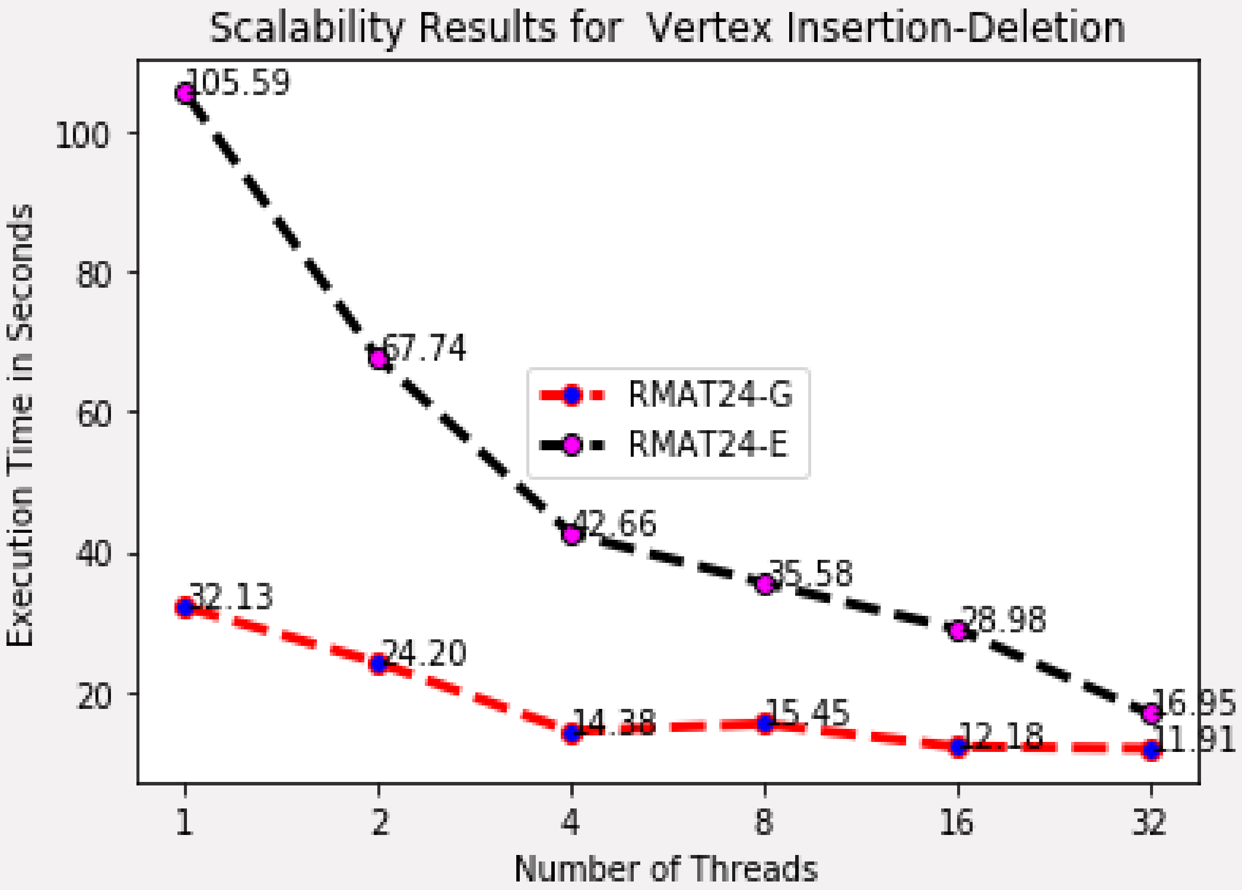


Strong scaling (real-world graphs)





Strong scaling (vertex insertion/deletion)



Empirical evaluation summary

For low thread counts, update algorithm is significantly faster than recomputation. However, recomputation shows better scaling.

Possible parallel scaling bottlenecks

- Set of changed edges not known apriori
- Redundant work in parallel setting

- **August 2015**: Brainstorming on how to analyze dynamic networks
- **June 2016**: Solved Connected Components. Presented in IPDPS 2016
- **Dec 2016**: Brainstorming on how to update MSTand SSSP.
- **May 2017**: Solved MST, Presented in SIAM CS17, and IPDPS (Ph.D.forum) 2017. IEEE Transactions on Big data Journal (Accepted at 06/2018)
- **March 2018**: Solved SSSP, SC18(Accepted @ HIPC 18)

Conclusions and Future Work

New shared-memory algorithm for updating SSSP in dynamic networks

Performance results demonstrate up to a 4X performance improvement over a parallel recomputation-based SSSP code

Plan to extend the general approach to centrality algorithms

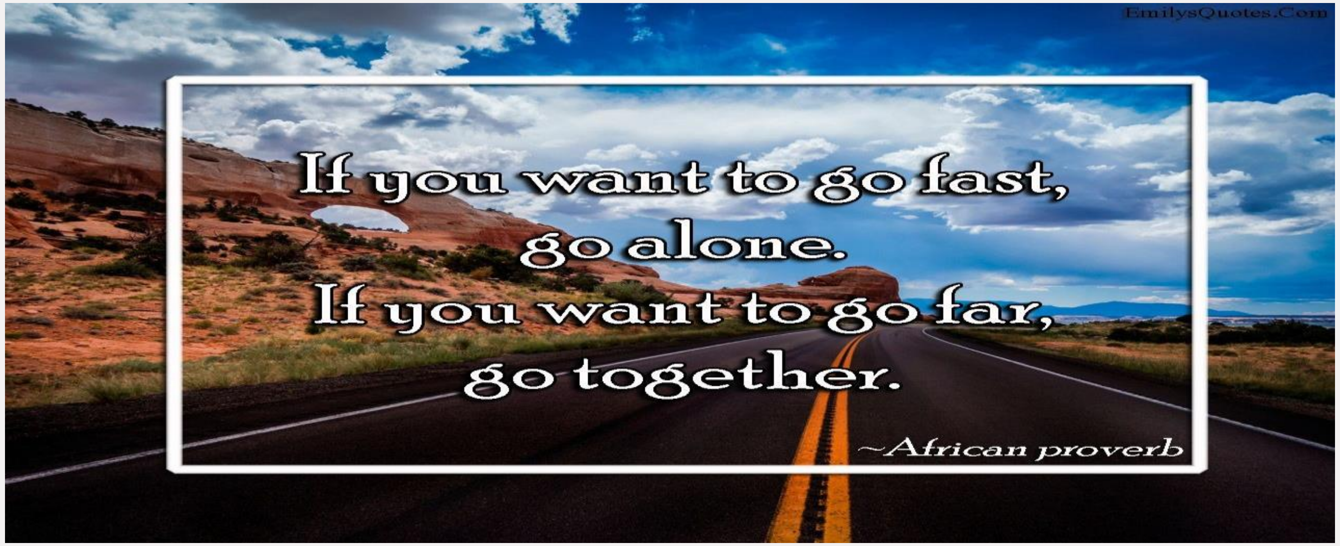
Future GPU and distributed-memory implementations

Acknowledgments & Collaborators

Dr. **Boyana Norris,**
University of Oregon

Dr. **Sajal Das,**
Missouri S&T

Thank you!



Questions?

Corresponding author: Sriram Srinivasan, sriramsrinivas@unomaha.edu

github.com/DynamicSSSP/HIPC18