Rensselaer

Scaling Graph Connectivity to Hundreds of Billions of Edges on Hundreds of GPUs[1]

George M. Slota (RPI) and
Michael Mandulak (RPI)

SIAM ACDA

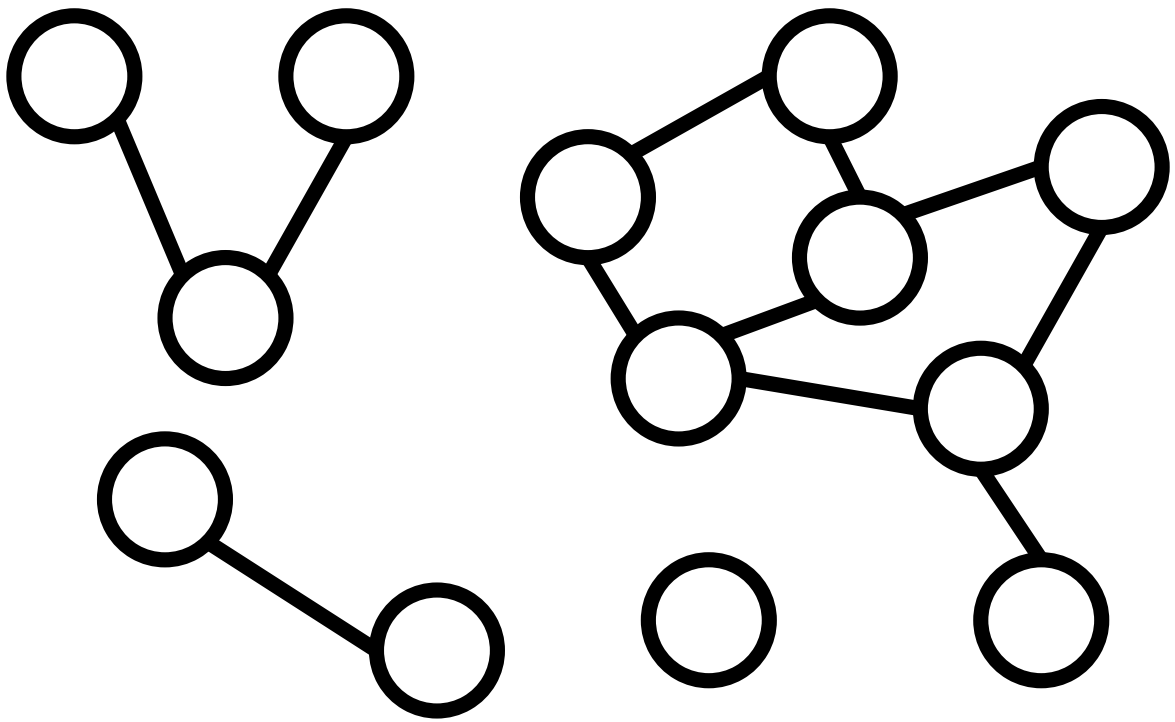
August 1st, 2025

[1]This work is supported by the NSF under Grant No. 2047821

Graph Connectivity

Specifically, 1-connectivity and connected components

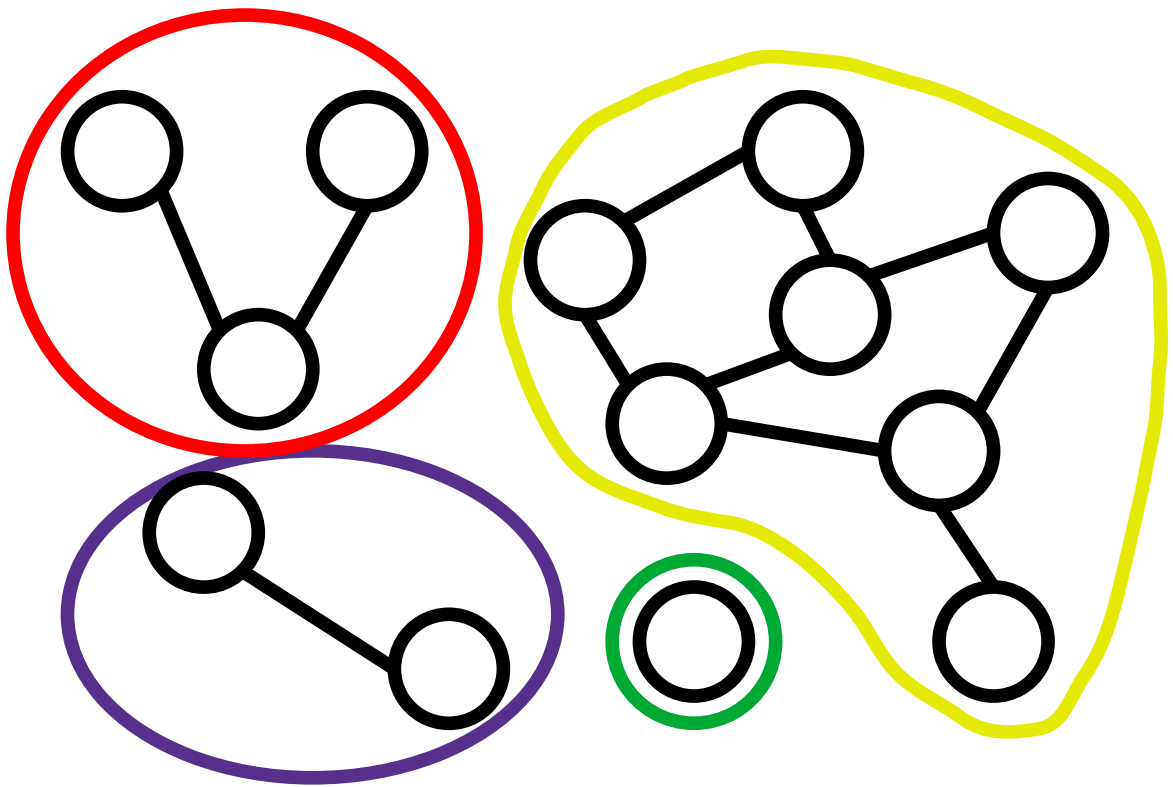
- **Connected Components Decomposition:** Identify all maximal *components* C_i in graph G where all vertex pairs $u, v \in C_i$ are pairwise reachable via some u, v-path.
- This is a basic analytic which is used on its own as well as a subroutine in more complex algorithms.
- One of the most studied problems in the discrete algorithms field.



Graph Connectivity

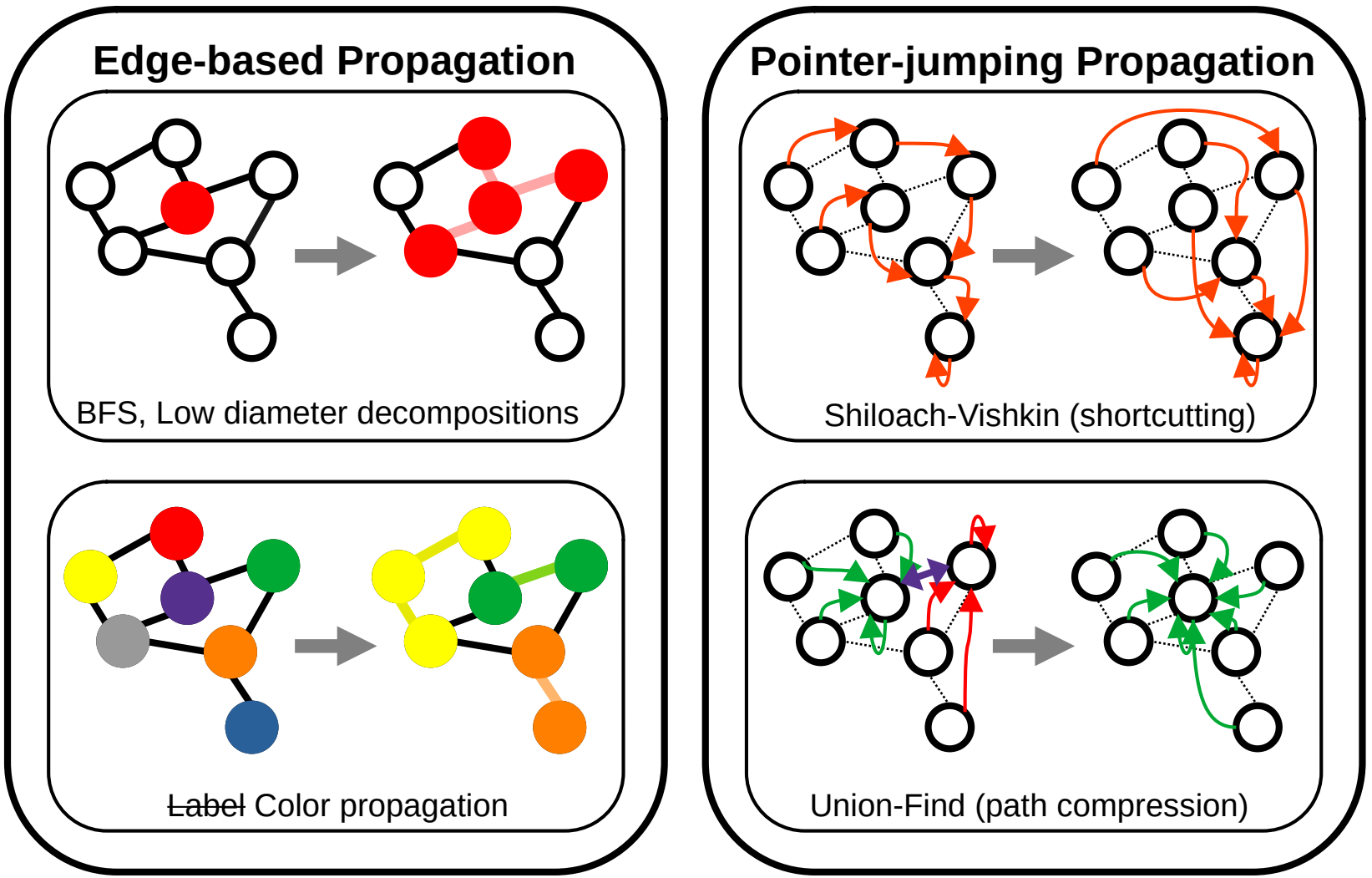
Specifically, 1-connectivity and connected components

- **Connected Components Decomposition:** Identify all maximal *components* C_i in graph G where all vertex pairs $u, v \in C_i$ are pairwise reachable via some u, v-path.
- This is a basic analytic which is used on its own as well as a subroutine in more complex algorithms.
- One of the most studied problems in the discrete algorithms field.



Graph Connectivity Algorithms

We can place them in four semi-related buckets



Graph Connectivity Algorithms

A short overview

Classic Algorithms

- **BFS and Color propagation** – relatively trivial to implement and optimize but limited by parallel depth complexity proportional to graph diameter
- **Shiloach-Vishkin** (SV) – $O(\log n)$ parallel depth PRAM algorithm using pointers and pointer-jumping
- **Union-Find** (UF) – Disjoint trees, similar complexity

Multi-phase Algorithms

- **Multistep**: Trim $\rightarrow$ BFS $\rightarrow$ Color Propagation $\rightarrow$ Serial
- **ConnectIt**: Sampling $\rightarrow$ Finish
  - Sampling: BFS, Low diameter decomposition (LDD), sparsification
  - Finish: Shiloach-Vishkin, Union-Find, any connectivity algorithm

A Large-scale Performance History

Performance on the 2012 Web Data Commons Crawl

Web Data Commons 2012 Crawl

(~3.5 billion vertices and 128 billion edges)

- **2016:** HPCGraph (Multistep: BFS+Color propagation)
  - 90 seconds on 256 CPU nodes
- **2018:** Gluon (Color propagation)
  - 75 seconds on 256 CPU nodes
- **2019:** FastSV (Shiloach-Vishkin)
  - 30 seconds on 4096 CPU nodes
- **2020:** ConnectIt (LDD+Union-Find)
  - 8.2 seconds on 1 CPU node
- **2025:** This Work (BFS+Shiloach-Vishkin)
  - 4.1 seconds on 400 GPUs across 67 nodes

This Work's Novelty

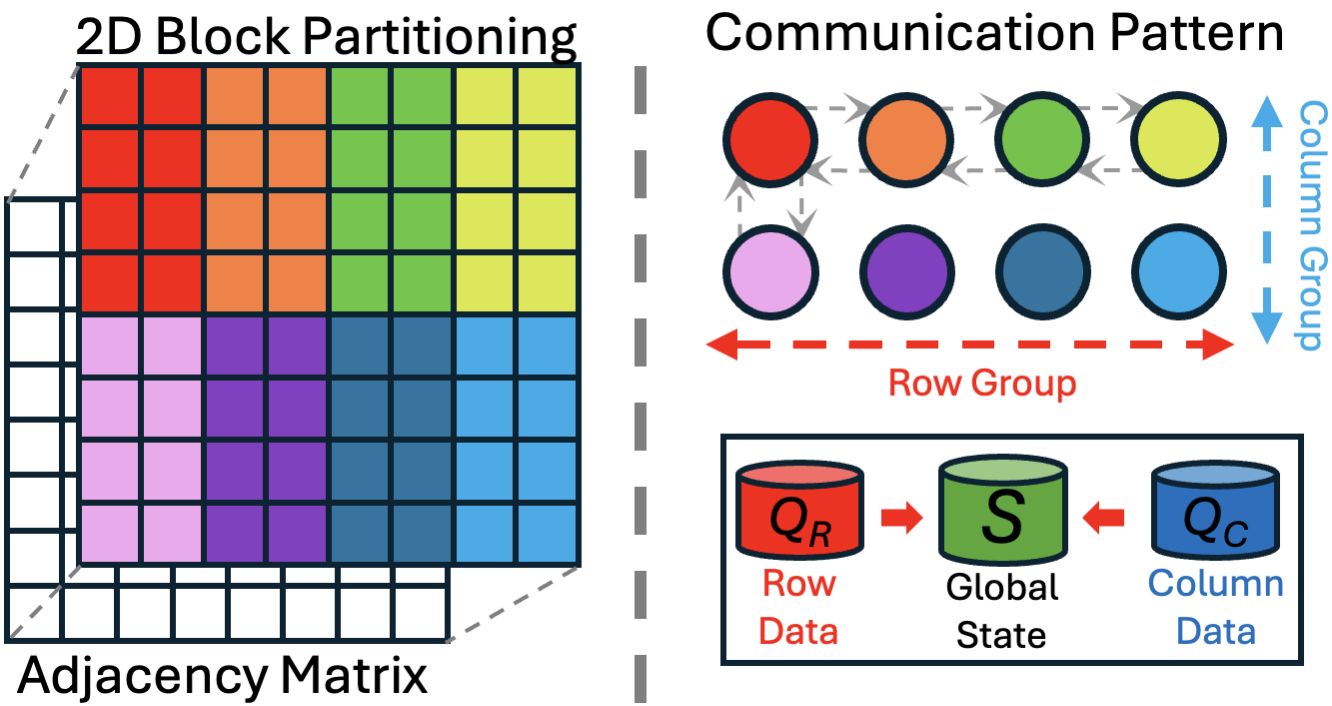
We consider for implementation many of the above proposed algorithms for a distributed multi-GPU environment.

To our knowledge, this is the first work

- ... to optimize a 2D distribution for connectivity.
- ... to run the WDC12 input solely on GPUs.
- ... to break the 5 second performance mark on WDC12.
- ... to run on 1 trillion NNZs with $O(\log n)$ depth algos.

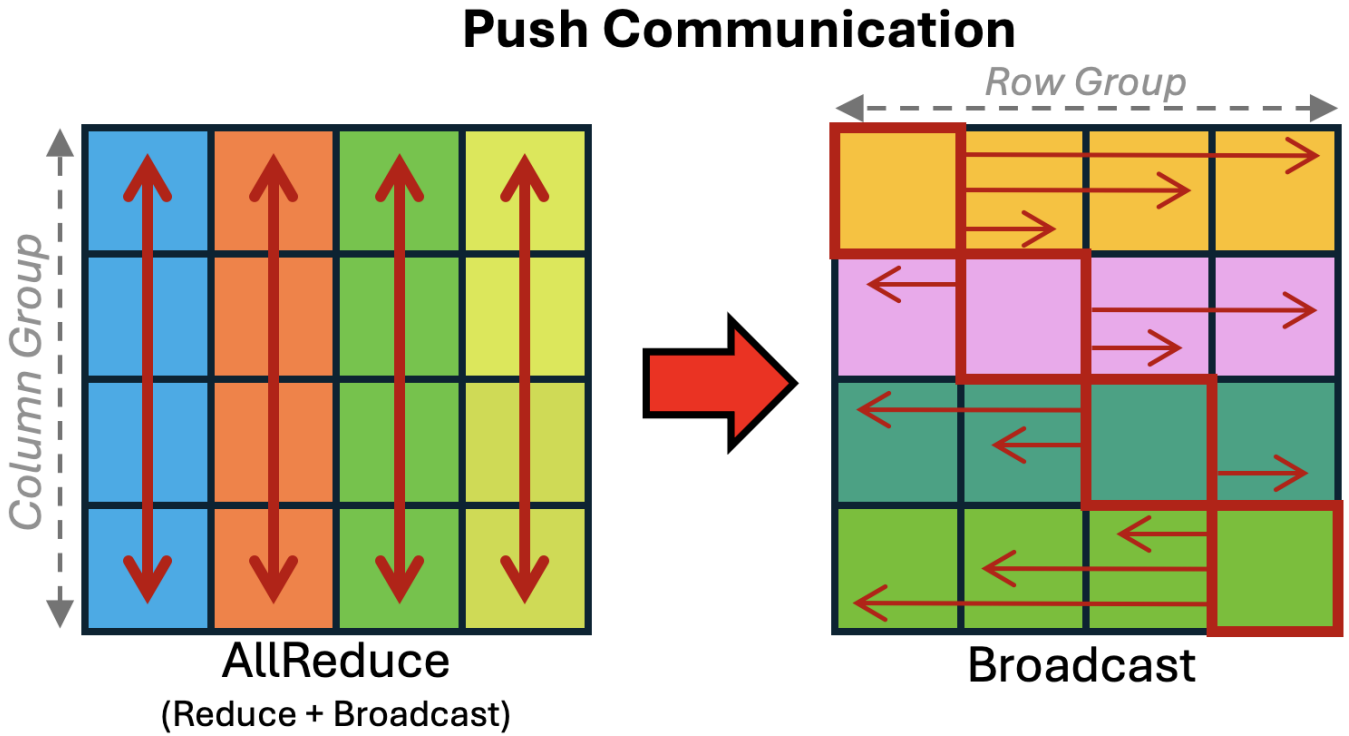
2D Graph Distributions

- Two-phase communications (gather-scatter or reduce-broadcast)
- De-facto default for large-scale performance (e.g., Graph500)
- $O(p)$ messages, but $O(\frac{n}{\sqrt{p}} + \frac{m}{p})$ work and communication volume



Standard 2D Communication Patterns

- Can utilize 'push' or 'pull' style communication
- **Push:** A vertex calculates potential update to a neighbor's state – we communicate such updates across the column group, and then broadcast a reduced new state value across a row group
- **Pull:** Reduce across row groups, broadcast to column groups



Our Implementations

We implement the following algorithms:

- Breadth-first search, low diameter decompositions
- Color propagation
- Shiloach-Vishkin (FastSV)
- Union-Find
- k-out sampling
- Low diameter decompositions

BFS and Color Propagations Implementations

BFS

- We implement a standard 2D direction-optimizing BFS
- Used only within multi-phase framework
- We halt after $\log n$ iterations
  - $O(\log n)$ depth when combined with SV or UF
  - Implicitly relabel source to vertex ID 0 then use min-VID pointer resolution in SV or UF
  - Note: we don't need to communicate pointer values, only found vertex IDs
- Future work: possible Graph500 optimizations

Color Propagation

- Implementations from Slota and Mandulak ICPP'25
- Push-based with variable dense/sparse communications

The Issue with Shiloach-Vishkin and Union-Find

The Big Issue:

- Both algorithms are pointer-based
- Distributed pointer jumping requires messages from any arbitrary vertex u to any arbitrary vertex v
- Standard 2D communications only follow direct adjacencies

We implement a generalized approach for arbitrary $u \rightarrow v$ messages.

Downside: We require $O(p^{1.5})$ messages, but still only use $O(\frac{n}{\sqrt{p}})$ communication volume.

Pointer Jumping Communications

1. For all vertices involved in computation, initialize pointer 'packages' as {head, tail} tuples. (the 'pointers')

Note 1: The tail originally owns this package.

Note 2: Tails are partitioned among a row group such that each rank owns unique packages.

2. Determine row group ownership of heads in packages.

3. Build communication queues and perform all-to-all package exchange along column group, passing from tail to head.

4. Update head values to head[head] in each received package. (the 'jump')

5. Pass updated head values back to tail vertex via column communication. Then broadcast updates across row group.

6. UF: Repeat to (2.) if the head value is actually updated.

Shiloach-Vishkin and Union-Find Implementations

Shiloach-Vishkin

- We compute new pointers via min-VID resolution
- We build packages and pointer jump (the shortcutting)
- Iterate until convergence

Union-Find

- In distributed framework, very similar implementation
- **Union**: determine potential new roots for pointers
  - We use min-VID pointer resolution, as other methods much more communication heavy
  - E.g., tracking tree sizes or tree depth
- **Find**: find root of pointer tree via pointer jumping
  - Update intermediate pointers as we jump (path compression)
  - A single jump is equivalent to Shiloach-Vishkin shortcutting, we jump 3 times at most

Other Methods

That we don't consider in our current results

K-out Sampling

- Compute connectivity on sparsified graph, only k edges per vertex
- Does not help – note that $O(\frac{n}{\sqrt{p}})$ is dominant term in 2D scaling. Reducing edges makes little difference.

Low Diameter Decompositions

- Multi-source BFS, new roots initialized at varying iterations
- Now need to communicate parent labels, which doubles communication volume relative to our baseline BFS
- Requires β tuning parameter, the optimal value of which can be *highly* variable across different graph diameters
- Regardless, still testing if any benefit can be found

Experimental Setup

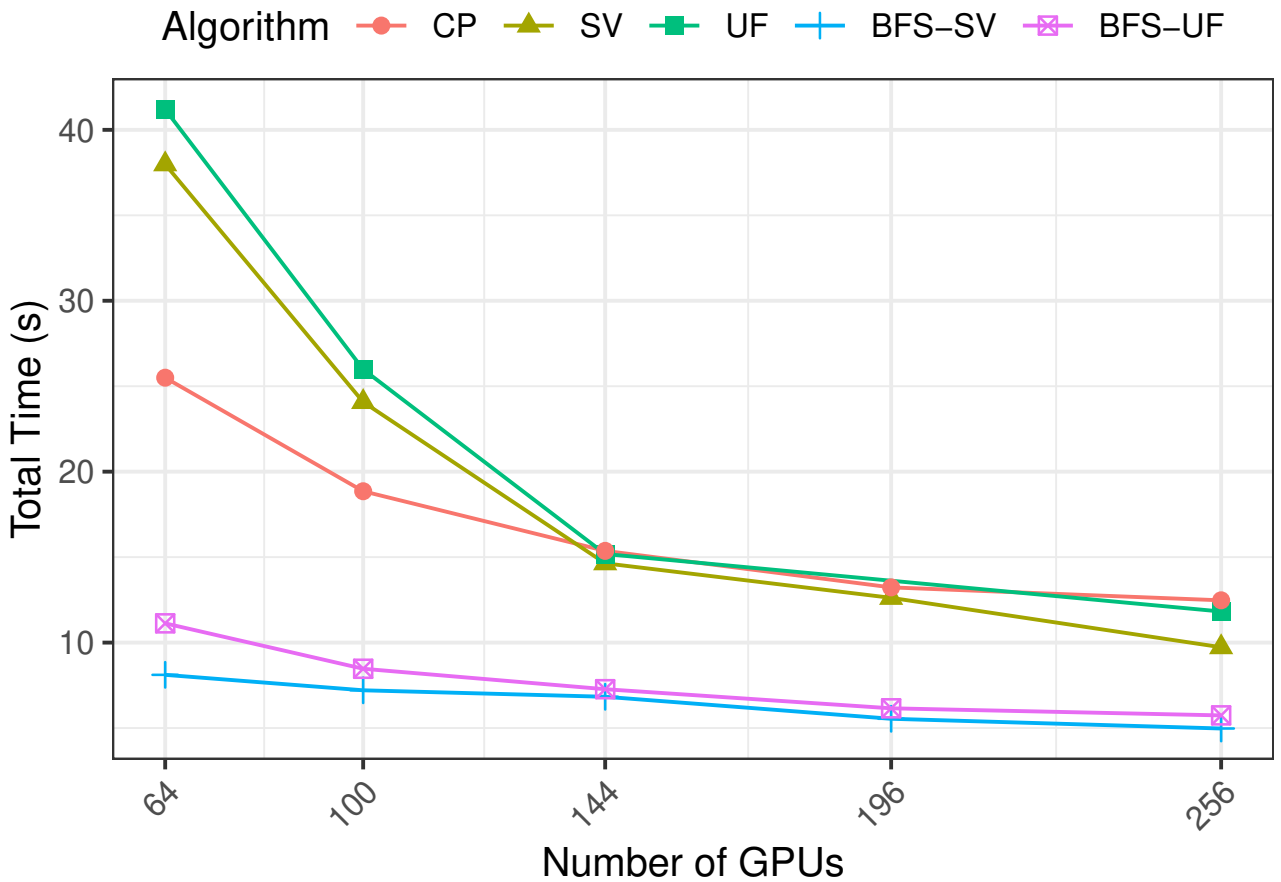
Name	Description	Vertices	NNZs
twitter-2010	Social network	41M	2.8B
USA-road-d	Road network	24M	58M
Queen_4147	3D mesh	4.1M	317M
kmer_A2a	Protein k-mer graph	170M	361M
MOLIERE_2016	Hypothesis network	30M	6.7B
gsh-2015	Web crawl	988M	68B
WDC12	Web crawl	3.5B	256B
RMXX	RMAT graph	2^{24}–2^{32}	2^{28}–2^{40}
ERXX	Erdös-Rényi graph	2^{24}–2^{32}	2^{28}–2^{40}
LINEXX	1D line graph	2^{24}–2^{32}	$\sim 2^{25}$–2^{33}
GRIDXX	2D grid graph	2^{24}–2^{32}	$\sim 2^{29}$–2^{34}

Test Graphs: We use standard benchmark instances for strong scaling and select random graphs for weak scaling.
Test System: AiMOS at RPI, with 6× V100 GPUs per node with EDR Infiniband network, up to 256 GPUs for most tests.
Test Algorithms: We ran various algorithm combinations in both single and multi-phase configurations.

Strong Scaling on WDC12

With select algorithm combinations

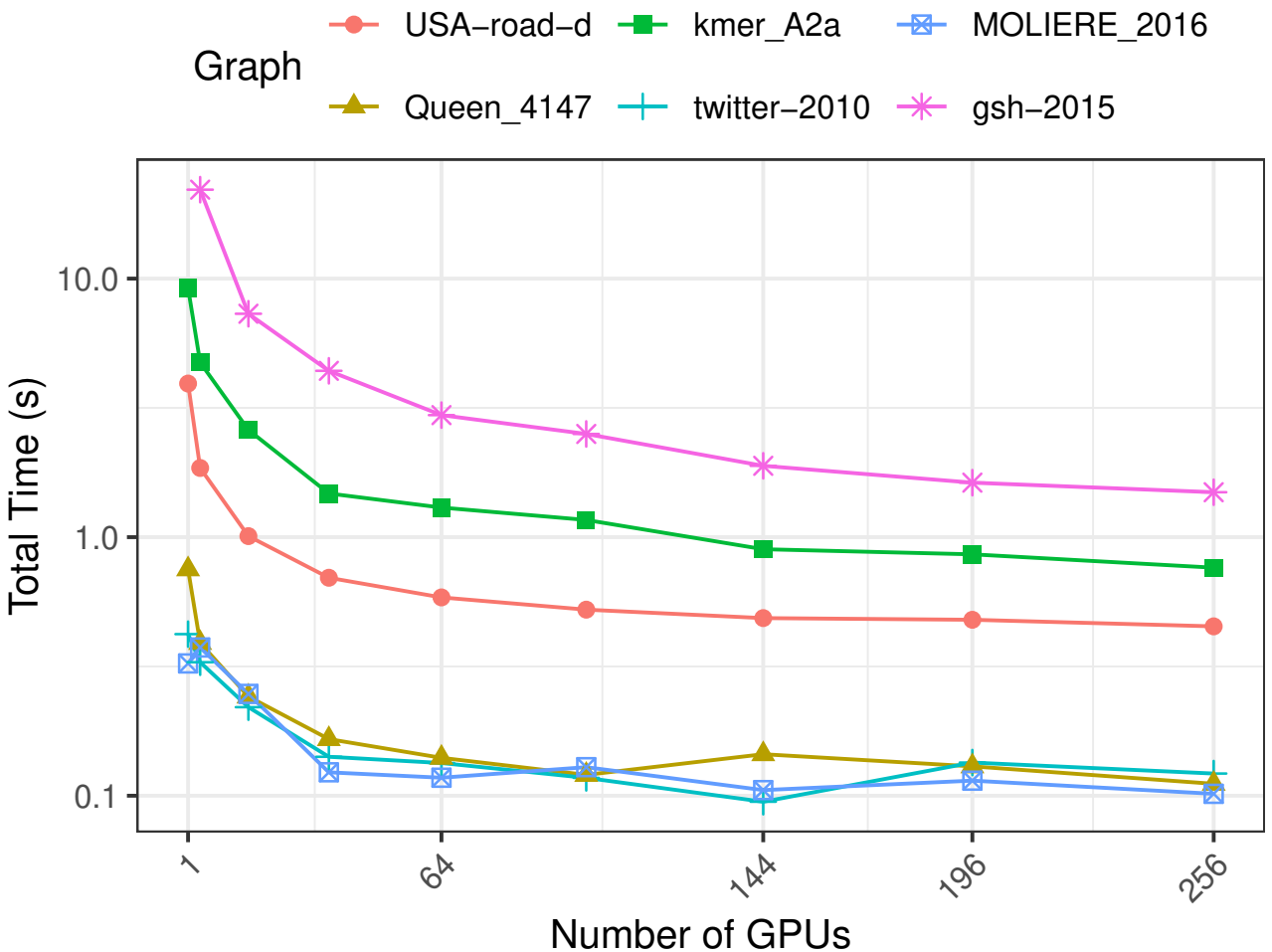
CP: Color propagation, **SV:** Shiloach-Vishkin, **UF:** Union-find
BFS-SV: BFS then Shiloach-Vishkin, **BFS-UF:** BFS then Union-find



Using BFS+Shiloach-Vishkin, we solve WDC12 in 4.9 seconds.

Strong Scaling on smaller inputs

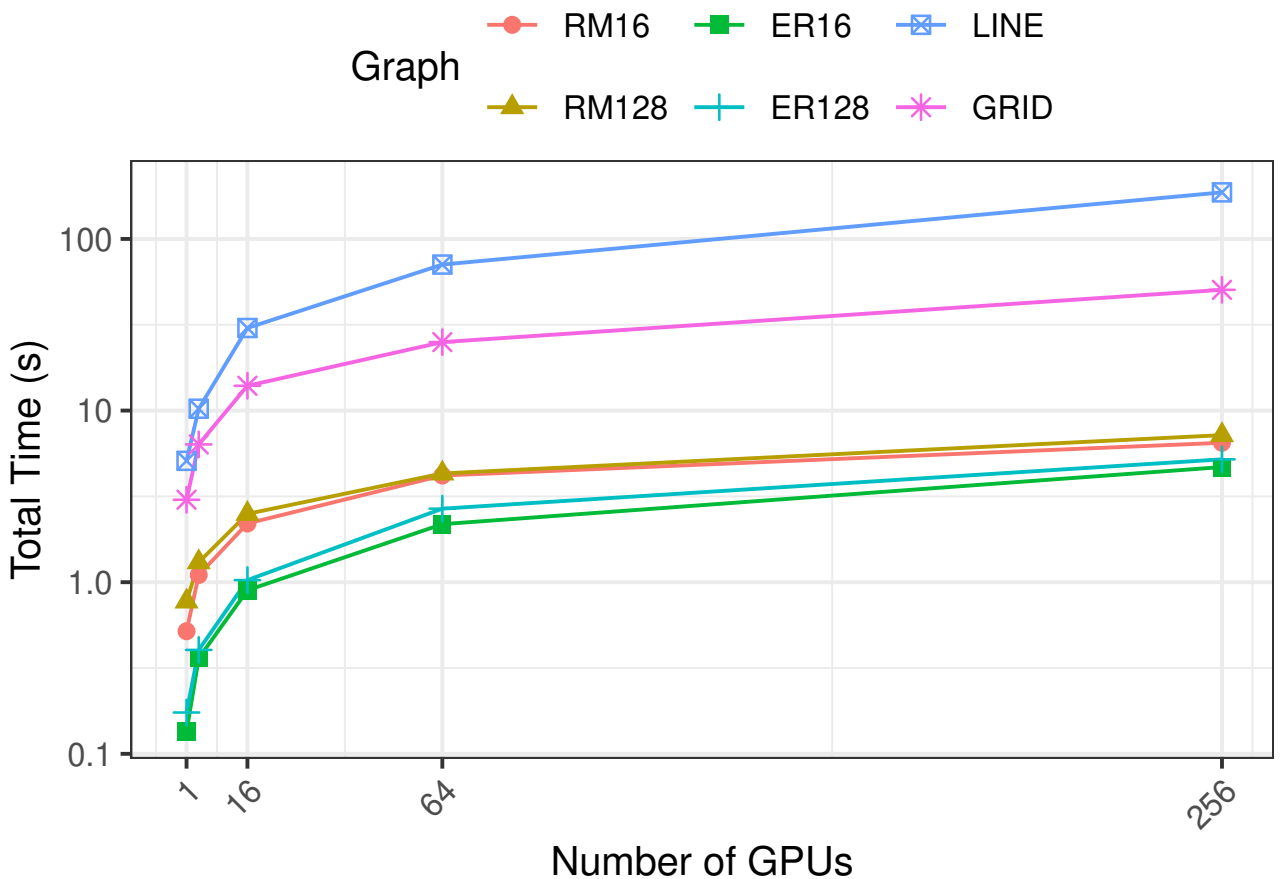
Using BFS then Shiloach Vishkin (BFS-SV)



We observe strong scaling to 256 GPUs on many inputs, including the road network – our smallest input.

Weak Scaling on Random Graphs

Again using BFS then Shiloach Vishkin (BFS-SV)



We solve an Erdös-Rényi graph with 1.1 trillion NNZs in 5.2 seconds. Note $O(\sqrt{p})$ scaling behavior and timing dependence on graph diameter and number of vertices.

Conclusions

and Thanks!

Main takeaway: **High performance distributed connectivity on GPUs at a massive scale is possible.**

- We achieve new performance benchmarks for the graph connectivity problem, both in time-to-solution and problem scale on GPUs.
- It is possible to achieve speedups in a distributed and multi-GPU environment.
- Future work: better overlap computation/communication, improve load balance, explore faster communications?

Contact: `slotag@rpi.edu`, `www.gmslota.com`