Rensselaer

Achieving Speedups for Distributed Graph Biconnectivity

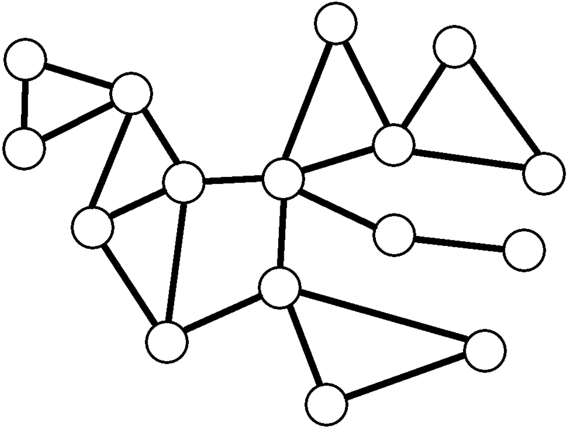
Ian Bogle (AMD) & **George M. Slota** (RPI)

IEEE High Performance Extreme Computing Conference

September 20th, 2022

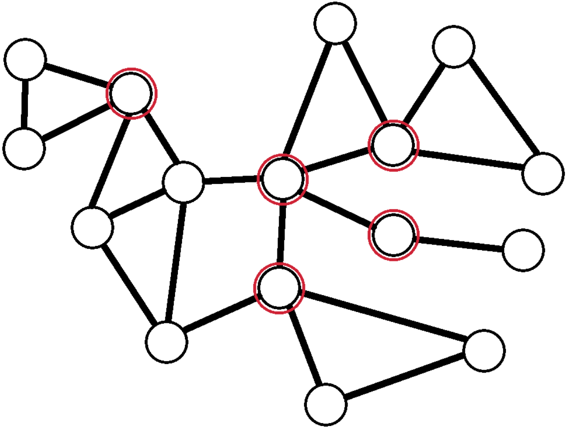
The Biconnectivity Problem

- Given some graph, we seek to ...



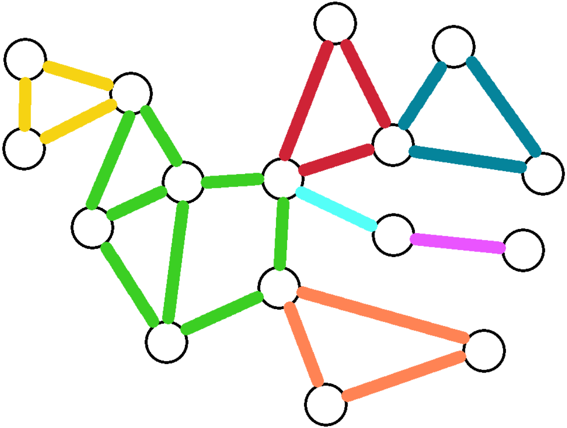
The Biconnectivity Problem

- Given some graph, we seek to ...
- identify all vertices that, when removed, disconnect the graph, and



The Biconnectivity Problem

- Given some graph, we seek to ...
- identify all vertices that, when removed, disconnect the graph, and
- label all *maximal* remaining (biconnected) edge-wise components.



The Biconnectivity Solutions

An Exciting History: Part 1

- **Hopcroft and Tarjan** (1973) - Work optimal serial algorithm using depth-first search
- **Tarjan and Vishkin** (1985) - Shared-memory time optimal (but **not** work optimal) using various subroutines (spanning tree, Euler tour, auxiliary graph construction)
- **Cong and Bader** (2005) - An improvement on Tarjan and Vishkin using **Cheriyan and Thurimella** (1991) edge filtering
  - Only a fraction of edges in most real graphs are necessary for determining separating vertex sets
- **Slota and Madduri** (2014) - Shared-memory breadth-first search and color propagation algorithms with a focus on simplicity (and ease of optimization)

The Biconnectivity Solutions

An Exciting History: Part 2

And now, the distributed algorithms:

- Kazmierczak and Radhakrishnan (2000); Ahmadi and Stone (2006)
  - Ear decomposition-based approaches
  - *Practical issue:* Linear+ time complexities
- Yan et al. (2014) and Feng et al. (2018)
  - Variations of optimization for Tarjan-Vishkin
  - *Practical issue:* No speedup relative to serial (Hopcroft-Tarjan on commodity CPU)

The goal of this work: **Achieve practical speedups for the biconnectivity problem in distributed memory.**

The Goal: Achieve speedups relative to serial

and efficient shared-memory implementations, if we can.

This work overall considers distributed implementations of two algorithms:

1. The **Slota-Madduri** color propagation algorithm
   - *Note:* Uses breadth-first search and label propagation as key subroutines, which are straightforward to implement (**and optimize!**) in distributed memory.
   - However, it is neither time nor work optimal.
2. **Cheriyan-Thurimella** edge filtering
   - *Note:* Can be implemented using breadth-first search and label propagation as well.
   - Edge filtering is applicable to *any* biconnectivity (or even vertex connectivity) algorithm.

Note: We also considered a Tarjan-Vishkin implementation.

Implementation Considerations

We use a standard 1D graph representation

- **Data Structures and Backend: HPCGraph**[1]
  - Utilize modified graph structures, communication routines, and multilevel processing queues
  - Can scale complex routines to trillion+ edge graphs
- **Parallelization Strategy: MPI+OpenMP**
  - Efficient use of "heavyweight" nodes on modern systems
  - Both widely-used, lightweight, and well-optimized
- **Communication Strategy: Synchronous AlltoAll**
  - All routines can effectively utilize this approach
  - Efficient to parallelize communication buffer construction and processing
  - Relatively balanced with block or random partitioning

Note: We consider a true $O(\frac{n}{p})$ per-node memory bound.

[1]Slota et al., IPDPS 2016

Experimental Setup

Test graphs:

Graph Name	Type	$\lvert V\rvert$	$\lvert E\rvert$	D	#BiCCs
soc-LiveJournal1	Social	4.8 M	43 M	46	76 K
com-Friendster	Social	52 M	1.1 B	35	5.5 M
web-Google	Web	855 K	4.3 M	25	60 K
web-ClueWeb09	Web	225 M	1.0 B	40	15 M
dbpedia-link	Info.	18 M	127 M	13	2.8 M
wikipedia_link_en	Info.	14 M	335 M	12	1.9 M
RMAT_25	Random	34 M	537 M	11	174 K

Note: We only consider the largest connected component.

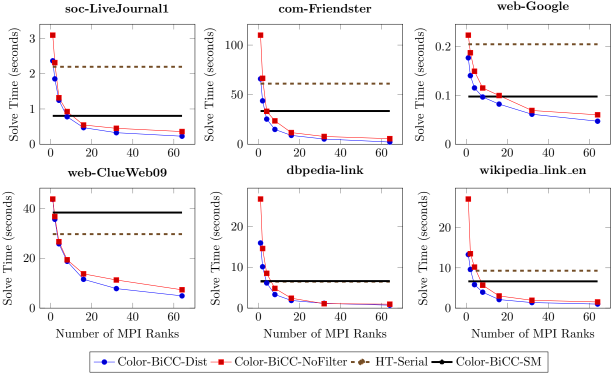
Test system:

AiMOS at RPI – 268 nodes with 2× 20-core 3.15 GHz IBM Power 9 CPUs, 4-6× NVidia V100 GPUs, and 512 GB DDR

Strong scaling

Running on 1-64 ranks of AiMOS.

We test our distributed implementation of the Slota-Madduri algorithm with (Color-BiCC-Dist) and without edge filtering (Color-BiCC-NoFilter) as well as the Hopcroft-Tarjan serial algorithm (HT-Serial) and the Slota-Madduri shared-memory implementation (Color-BiCC-SM).



We consistently achieve speedups vs. serial in 2-4 MPI ranks.

Overall Performance

HT: Serial, SM: Shared-memory, CBD: With filtering, CBNF: Without filtering

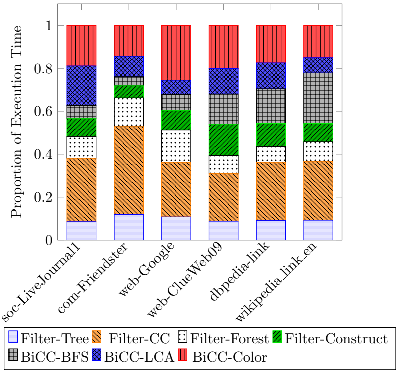
- Times reported are on 64 ranks (20 threads each) for distributed algorithms, 20 threads for the shared-memory algorithm, and a single thread[2] for the serial algorithm. Speedup reported is relative to the serial algorithm.
- We achieve consistent speedups vs. serial and shared-memory, while edge filtering is almost always "worth it".

Graph	HT	SM	CBNF	CBD	Speedup
soc-LiveJournal1	2.2	0.80	0.36	**0.23**	10×
com-Friendster	61	33	5.6	**2.2**	30×
web-Google	0.21	0.098	**0.047**	0.060	3.7×
web-ClueWeb09	30	38	7.3	**4.9**	8.9×
dbpedia-link	6.5	6.6	0.97	**0.72**	22×
wikipedia_link_en	9.3	6.6	1.5	**1.0**	13×

[2] I hope this is obvious.

Performance Breakdown

Using the Color-BiCC-Dist implementation with edge filtering



Note: I am aware the algorithm subroutines were not discussed in detail. I'm including this figure anyways to mainly highlight the edge filtering vs. biconnectivity algorithm relative proportions of execution time.

- Edge filtering takes about half of the total execution time. However, it almost always reduces time more than its cost.
- All routines can be further optimized. E.g., the connectivity decomposition of edge filtering does not use an optimal, or even highly optimized, algorithm.

Conculsions

and Thanks!

Main takeaway: **Distributed biconnectivity speedups are possible.**

- We achieve distributed-memory speedups for the biconnectivity problem relative to serial and an optimized shared-memory implementation in a small number of ranks.
- Cheriyan and Thurimella edge filtering is possible in distributed-memory, and it is often quite worth doing.
- Our future work will look towards better implementations of our constituent subroutines and possible implementations on GPU.

Contact: `gmslota@gmail.com`, `www.gmslota.com`