# Simple Parallel Biconnectivity Algorithms for Multicore Platforms

George M. Slota  Kamesh Madduri

**The Pennsylvania State University**

**HiPC 2014**

**December 17-20, 2014**

**Code, presentation available at graphanalysis.info**

# Acknowledgments


- NSF grants ACI-1253881, OCI-0821527
- DOE Office of Science through the FASTMath SciDAC Institute
  - Sandia National Laboratories is a multi program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

# Contributions

- Two new algorithms for biconnected component decomposition of large sparse graphs.
- Algorithms use (easy-to-parallelize) Breadth-First Search (BFS) and Color propagation as subroutines. Sequential algorithm uses Depth-First Search (DFS).
- Performance results on a 16-core Intel Sandy Bridge server and 10 large-scale graphs
  - Up to 7.1X speedup over DFS-based serial algorithm
  - Up to 4.2X faster than parallel Cong-Bader algorithm

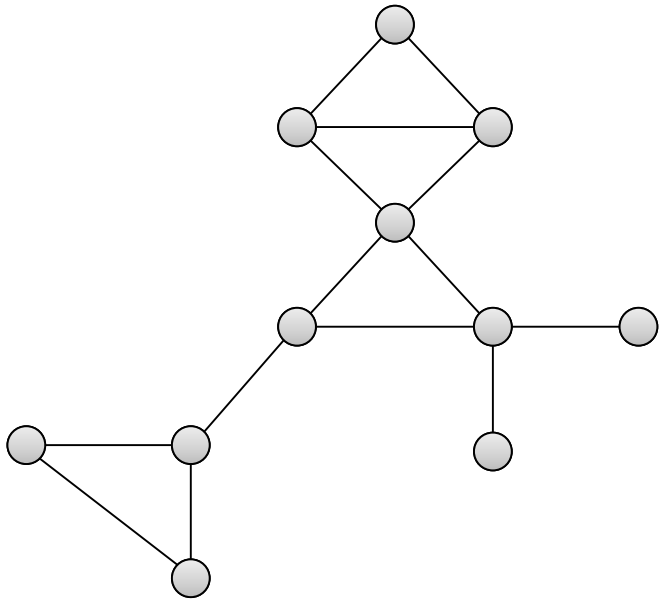
# Talk Outline

- Biconnected components: problem definition
- Prior serial and parallel algorithms
- Our new parallel algorithms
  - BFS-based (this talk)
  - Color propagation-based (see paper)
- Parallel performance results and discussion

# Biconnected components

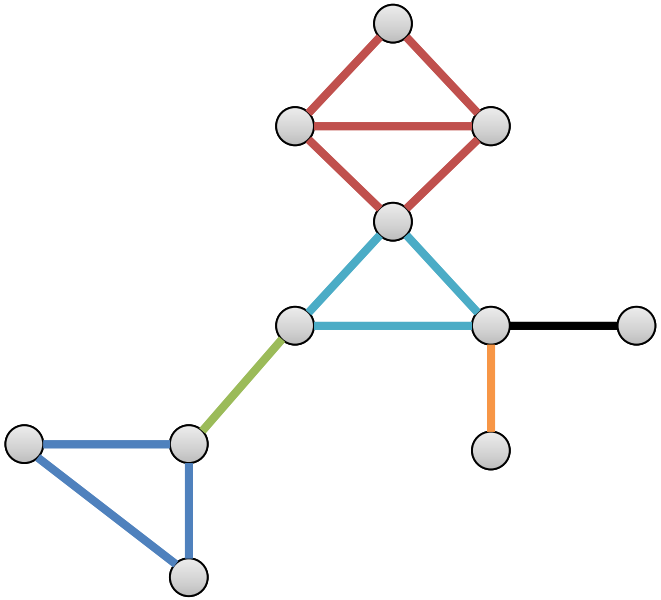
- Consider a connected, undirected graph G
- A biconnected component (BiCC) or block of G is a maximal set of edges such that any two edges in the set lie on a common simple cycle.
- BiCC decomposition: a partitioning of edges into blocks
- An articulation point is a vertex whose removal disconnects G
- A bridge is an edge whose removal disconnects G

# Biconnected components: example



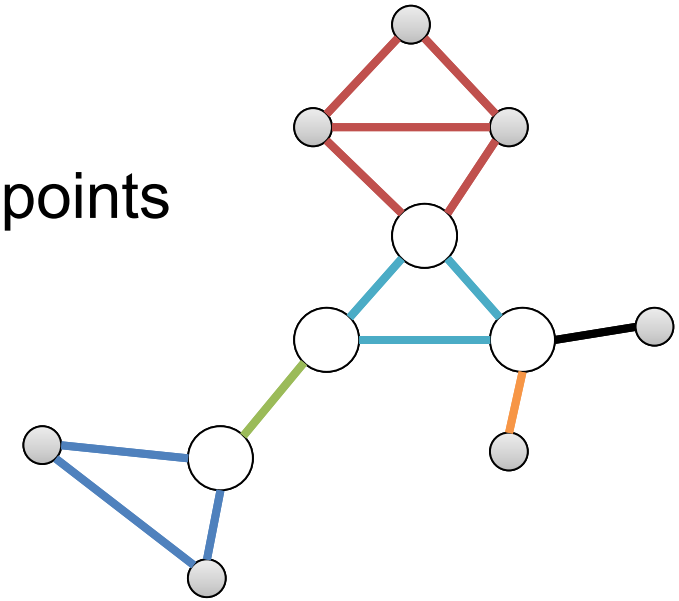
# Biconnected components: example

Six blocks



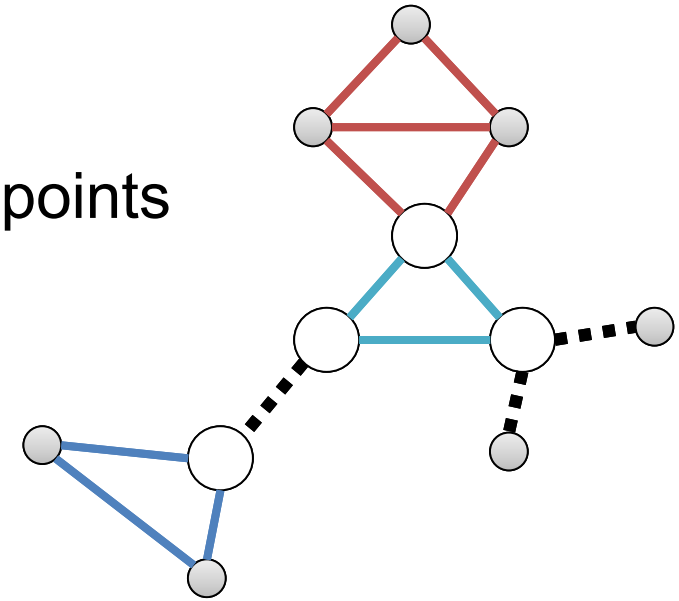
# Biconnected components: example

Six blocks
Four articulation points



# Biconnected components: example

Six blocks
Four articulation points
Three bridges



# Why BiCC decomposition?

- Preprocessing step when analyzing large social and informational networks
- Distribution of BiCC sizes is of interest
- Most real-world networks have one large BiCC, useful to identify that component
- Bridges and articulation points have high centrality values

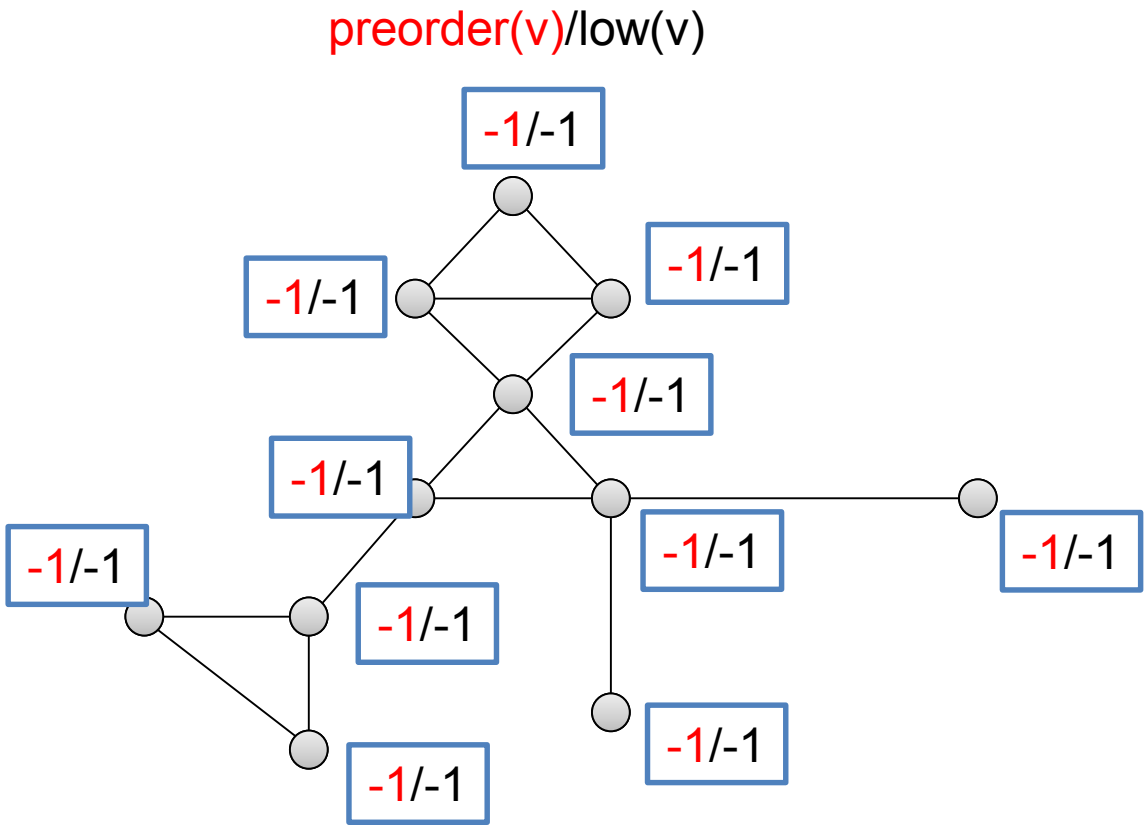
# Serial BiCC algorithm

- Hopcroft-Tarjan (HT) algorithm
- Based on Depth-First Search (DFS)
- O(*m*+*n*) time and space
  - m: # of edges, n: # of vertices

J. Hopcroft, R. Tarjan, Efficient Algorithms for Graph Manipulation, CACM, 16(6), pp 374--378, 1973.

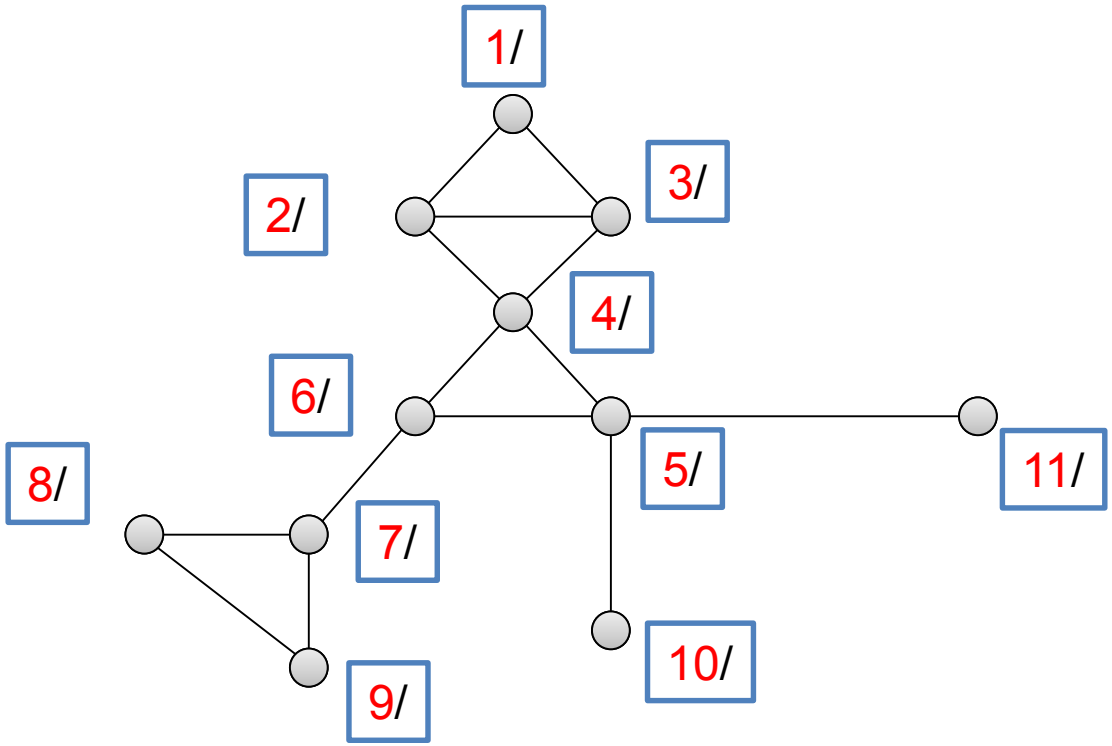
# HT algorithm

- Updates two vertex labels, preorder(v) and low(v)



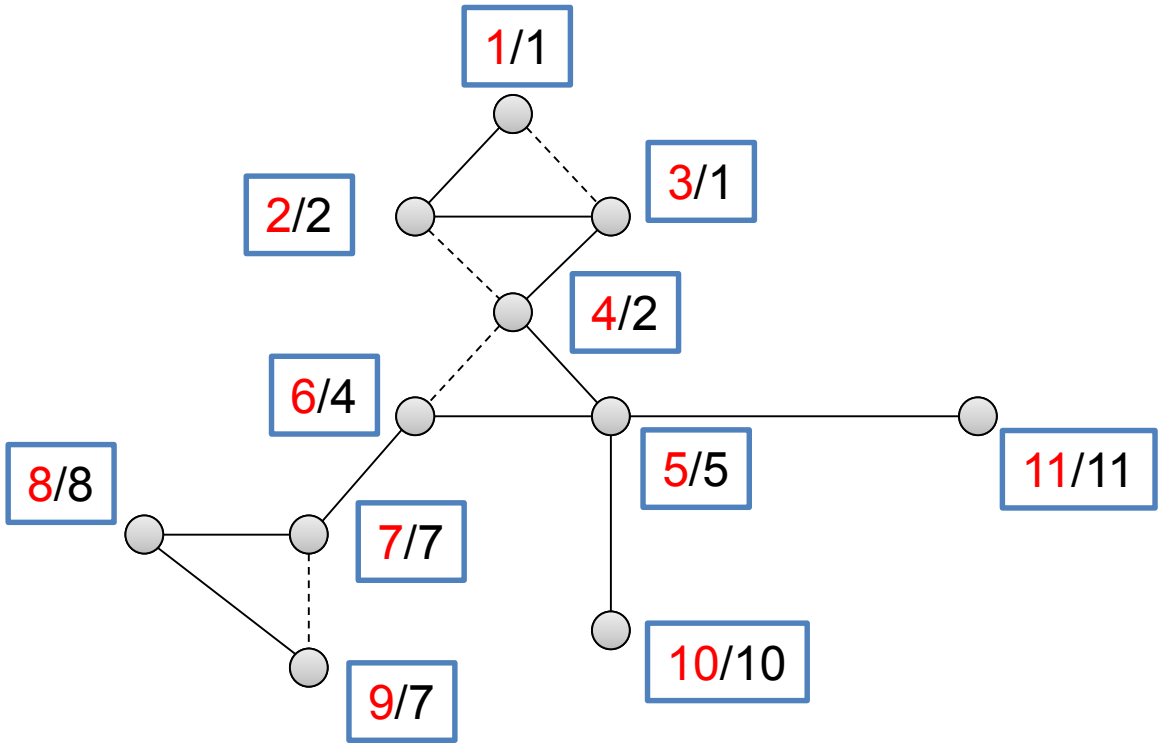
# HT algorithm

- Preorder labels are computed by a DFS using vertex discovery order.



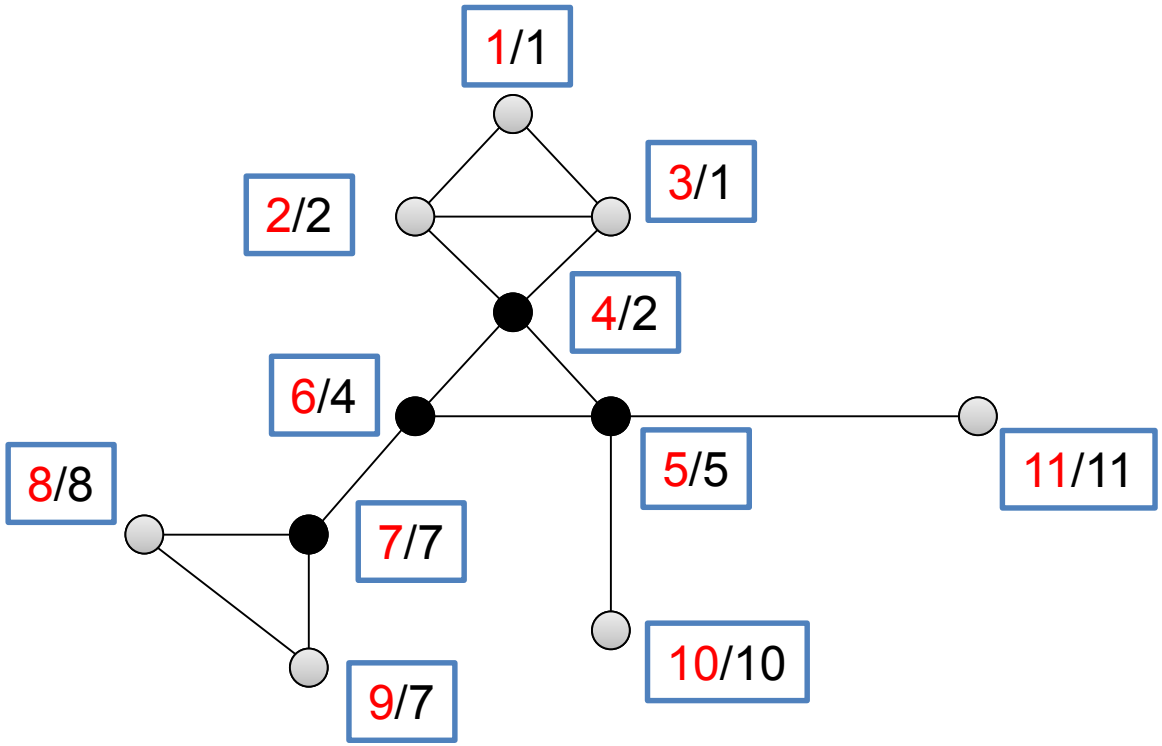
# HT algorithm

- low(v) is smallest value of preorder(w), where w is a vertex that can reached from v following a sequence of (0 or more) tree edges and by at most 1 back edge.



# HT algorithm

- preorder(v) and low(adj(v)) can be used to determine if v is an articulation point or not.



# HT algorithm

- Difficult to parallelize
- No known work-efficient logarithmic time PRAM algorithm for DFS

# Tarjan-Vishkin (TV) algorithm

- O(log n) time, O(m+n) work PRAM algorithm using O(m+n) processors
- Designed to use any spanning tree, not just a depth-first spanning tree
- Spanning tree -> Euler tour -> List ranking to determine an ordering of vertices in V
- Creates an *auxiliary graph G'*, whose vertices are edges in G
- Connected components of G: biconnected components of G'

R. E. Tarjan, U. Vishkin, An Efficient Parallel Biconnectivity Algorithm, SIAM J. Comput., 14(4), pp 862--874, 1985.

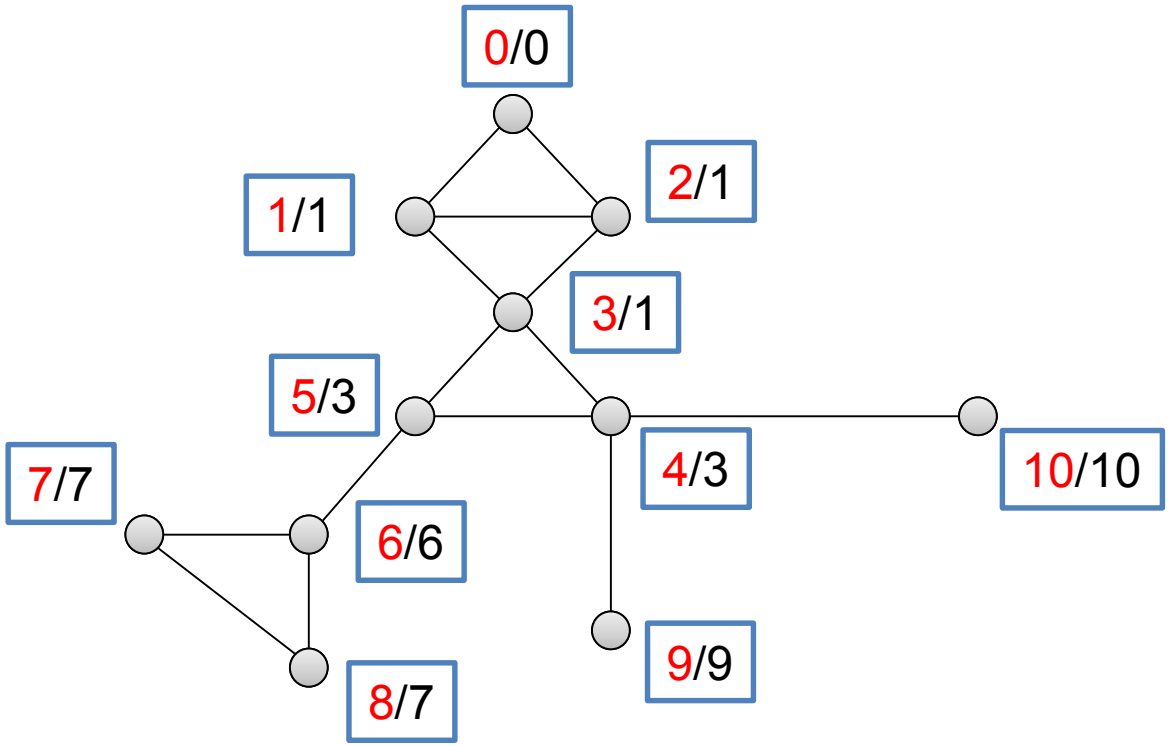
# Cong-Bader algorithm

- Based on the Tarjan-Vishkin algorithm
- Presents a new preprocessing step (filtering out *non-essential* edges) to reduce size of auxiliary graph G'

G. Cong and D.A. Bader, An Experimental Study of Parallel Biconnected Components Algorithms on Symmetric Multiprocessors, Proc. IPDPS 2005.

# HT algorithm

- low(v): lowest vertex that is either a descendant of v, or adjacent to a descendant of v.



# Motivating our new algorithms

- Can we design a simpler parallel algorithm?
  - Preferably an algorithm using a breadth-first spanning tree?
- Simpler than Tarjan-Vishkin?
  - High memory util of TV auxiliary graph is a problem
- Fast on multicore platforms
- Suited for real-world graphs (millions of vertices, billions of edges, one large BiCC)

# BFS-ArtPts algorithm: Main idea

- Back to first principles: A vertex v is an *articulation point* iff there exist two other vertices u and w such that *every path* through u and w passes through v.
- Consider a BFS tree rooted at some vertex s. Compute P(v), the parent of v, and L(v), the depth of v in BFS tree
- Theorem: A non-root vertex v in the BFS tree is an articulation vertex iff it has *at least one child w* that *cannot reach any vertex* of depth at least L(v) when v is removed from G.

# The pseudocode

- Uses a truncated BFS routine that we call BFS-L

**procedure** BFS-ARTPTS($G(V, E)$)
  **for all** $v \in V$ **do**
    $Art(v) \leftarrow false$
    $visited(v) \leftarrow false$
  Select a root vertex $r$
  $P, L \leftarrow$ BFS$(G, r)$
  **for all** $u(\neq r) \in V$ where $P(u) \neq r$ **do**
    $v \leftarrow P(u)$
    **if** $Art(v) = false$ **then**
      $l \leftarrow$BFS-L$(G, L, v, u, visited)$
      **if** $l \geq L(v)$ **then**
        $Art(v) \leftarrow true$
  Check if $r$ is an articulation point

# Truncated BFS pseudocode

```
procedure BFS-L(G(V, E), L, v, u, visited)
    Insert u into Q
    Insert u, v into S
    visited(u) ← true
    visited(v) ← true
    while Q ≠ ∅ do
        for all x ∈ Q do
            Remove x from Q
            for all ⟨w, x⟩ ∈ E where visited(w) = false do
                if L(w) < L(u) then
                    for all s ∈ S do visited(s) ← false
                    return L(w)
                else
                    Insert w into Q
                    visited(w) ← true
    return L(u)
```

# BFS-BiCC

- We extend the BFS-ArtPts algorithm to determine the component that each vertex belongs to
- Maintain two vertex labels
  - Par(v): highest-level articulation point separating v from the root
  - Low(v): biconnected component label
- Once a component is identified, mark vertices as deleted.

# Color propagation-based algorithm

- Alternate approach: use color propagation as inner subroutine instead of truncated BFS
- Better suited for higher-diameter graphs

# Experimental setup, Results, Analysis

- Dual-socket Intel system with Xeon E5-2670 (Sandy Bridge) processors, 64 GB memory
  - 2.6 GHz processors, 20 MB last-level cache per processor
  - 2 processors x 8 cores x 2-way SMT
- C++, OpenMP
- Ten large graphs
- Comparisons to our implementations of Hopcroft-Tarjan (HT) and Cong-Bader (CB) algorithms. CB faster than TV for all test instances.

# Test networks used

| Graph | # vertices | # edges | Approx diameter | # BiCC | Max BiCC size |
|---|---|---|---|---|---|
| Livejournal | 4.8 M | 43 M | 21 | 1.1 M | 3.7 M |
| Orkut | 3.1 M | 117 M | 11 | 68 K | 3 M |
| WikiLinks | 26 M | 543 M | 86 | 3.5 M | 22 M |
| ItWeb | 41 M | 1 B | 46 | 5 M | 33 M |
| Friendster | 63 M | **1.6 B** | 34 | **13 M** | **49 M** |
| Cube | 2.1 M | 59 M | **157** | **1** | 2.1 M |
| Kron_21 | 1.5 M | 91 M | 8 | 238 K | 1.3 M |
| R-MAT_24 | 7.7 M | 133 M | 11 | 2.2 M | 5.4 M |
| GNP_1 | 10 M | 200 M | **7** | 1 | 10 M |
| GNP_10 | 10 M | 200 M | 19 | 19 | **5 M** |

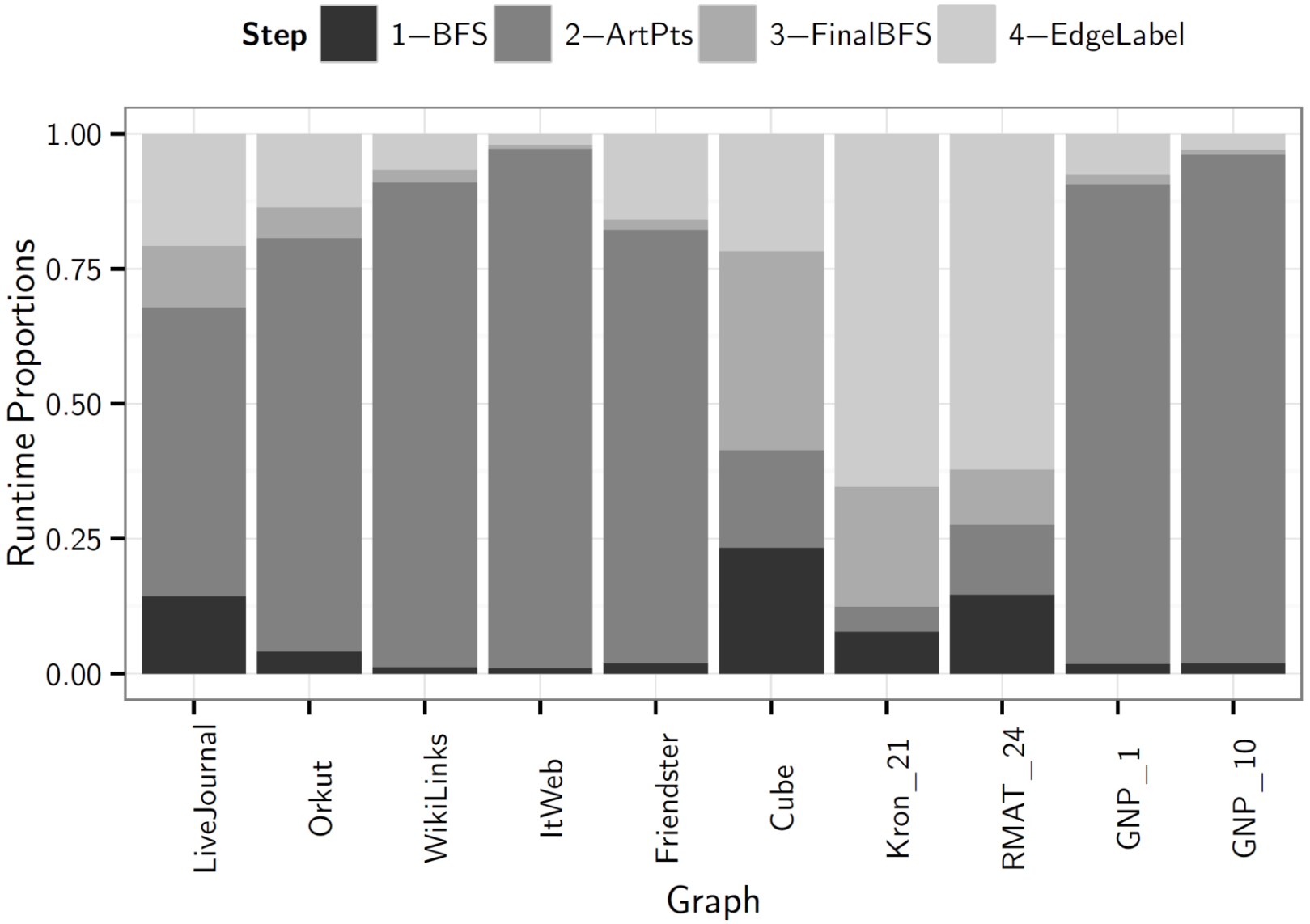
# BFS-BiCC performance

| Graph | HT time (s) | Our alg, seq (s) | Our alg, par (s) | Speedup vs. seq HT | Speedup vs par CB |
|---|---|---|---|---|---|
| Livejournal | 2.1 | 2.5 | 0.38 | 5.5X | **4.2X** |
| Orkut | 3.4 | **3.3** | **0.49** | 6.9X | 3.6X |
| WikiLinks | 25 | 29 | 7 | 3.6X | 3.4X |
| ItWeb | 19 | 460 | 50 | **0.4X** | -- |
| Friendster | **79** | 150 | 20 | **4X** | -- |
| Cube | **1.2** | **0.98** | **0.17** | **7.1X** | 3.8X |
| Kron_21 | 1.8 | 1.1 | 0.60 | 3X | 3.8X |
| R-MAT_24 | 4.7 | 6.2 | 1.5 | 3.1X | 3.9X |
| GNP_1 | 11 | 37 | 5 | 2.2X | 1.2X |
| GNP_10 | 6.5 | 31 | 12 | 0.5X | **0.5X** |

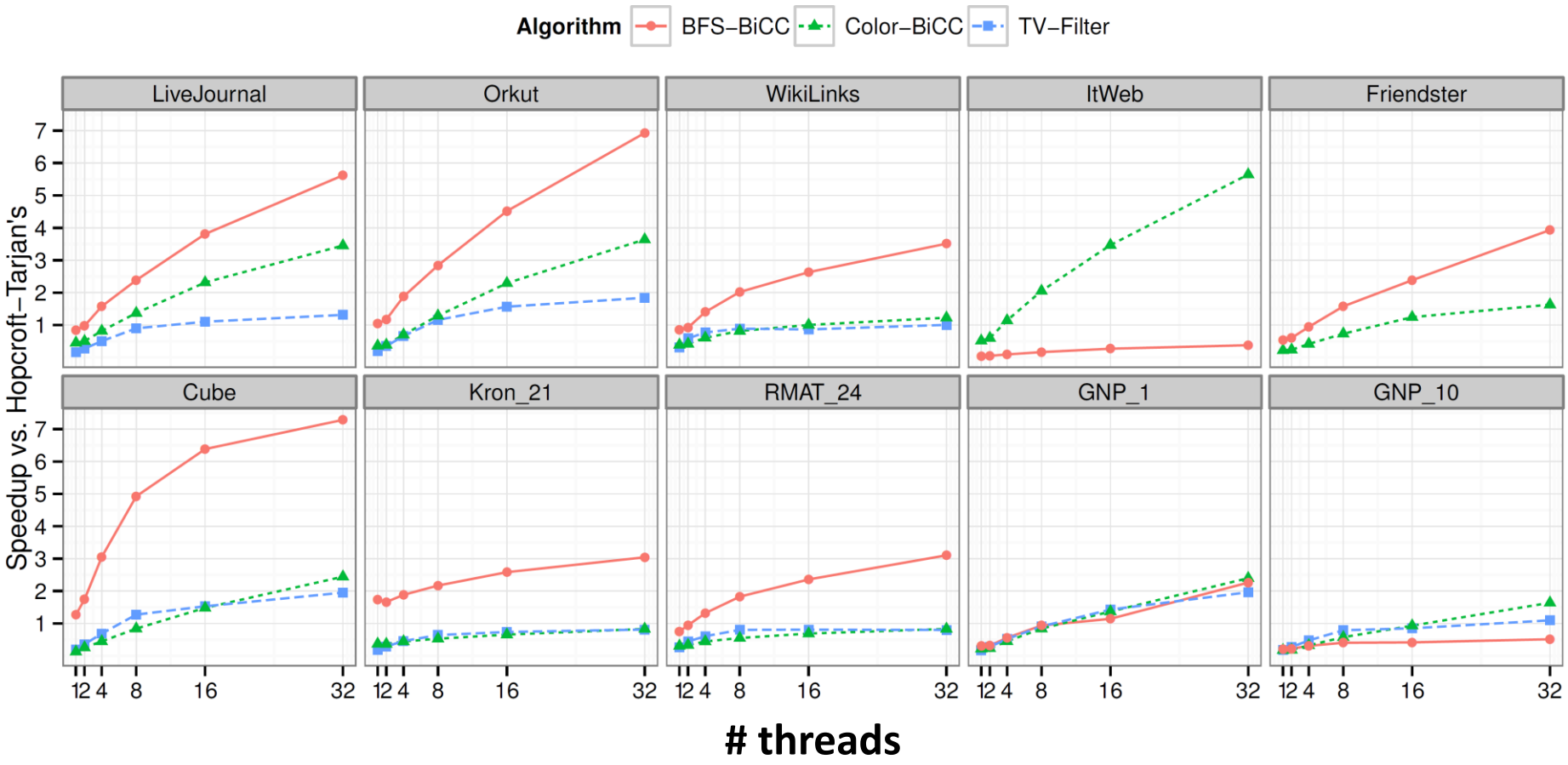
# BFS-BiCC performance: further analysis

| Graph | Par BFS time (s) | Our alg time (s) | Our alg/BFS time ratio | Edge vis ratio |
|---|---|---|---|---|
| Livejournal | 0.032 | 0.38 | 11.9X | 1.0 |
| Orkut | 0.025 | 0.49 | 19.6X | 0.6 |
| WikiLinks | 0.40 | 7 | 17.5X | 1.3 |
| ItWeb | 0.41 | 50 | **122X** | 1.6 |
| Friendster | 0.46 | 20 | 43.5X | 0.95 |
| Cube | 0.04 | 0.17 | **4.25X** | **0.03** |
| Kron_21 | 0.03 | 0.60 | 20X | 0.015 |
| R-MAT_24 | 0.12 | 1.5 | 12.5X | 0.042 |
| GNP_1 | 0.08 | 5 | 62.5X | 3.1 |
| GNP_10 | 0.15 | 12 | 80X | **19** |

# BFS-BiCC breakdown of time



# Parallel scaling



# Conclusions

- We present two simple and high-performance biconnectivity algorithms
  - Can you beat our algorithms?
- Future work
  - Improving scaling, load-balancing of the inner loops
  - Adapting these algorithms for GPUs and Xeon Phis
  - Further theoretical analysis of algorithms: identifying worst-case input instances

# Thank you!

- Questions? Feedback?

- George: gslota@psu.edu, gmslota.com
- Kamesh: madduri@cse.psu.edu, madduri.org