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.

HiPC 2014 talk: code and slides at graphanalysis.info
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

HT algorithm

• Updates two vertex labels, $\text{preorder}(v)$ and $\text{low}(v)$
HT algorithm

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

- \textit{low}(v) is smallest value of preorder(w), where \textit{w} is a vertex that can be reached from \textit{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’

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 $G'$

**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

HiPC 2014 talk: code and slides at graphanalysis.info
```
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
      end for
    end for
  end while
  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

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

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

**Speedup**

HiPC 2014 talk: code and slides at graphanalysis.info
BFS-BiCC performance: further analysis

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

![Graph showing parallel scaling for different algorithms and datasets](image-url)
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, gmslotan.com
• Kamesh: madduri@cse.psu.edu, madduri.org