Aquila: Adaptive Parallel Computation of Graph Connectivity Queries

Yuede Ji
H. Howie Huang

Graph Computing Lab
George Washington University
Part 1: Introduction
Graph Is All Around

```c
void foo()
{
    int x = source();
    if (x < MAX)
    {
        int y = 2 * x;
        sink(y);
    }
}
```
Graph Connectivity Algorithms

- Directed
  - Weakly connected component (WCC)
  - Strongly connected component (SCC)
- Undirected
  - Connected component (CC)
  - Biconnected component (BiCC), articulation point (AP)
  - Bridgeless connected component (BgCC), Bridge
- XCC to represent all

Attributes:
- (W)CC
- SCC
- BiCC
- BgCC
- AP: \{5, 9\}
- Bridge: \{1-5, 9-11, 12-13\}
Applications

- **Graph analytics**
  - Directed acyclic graph (DAG) $\rightarrow$ SCC
  - Betweenness centrality (BC) $\rightarrow$ BiCC [PPoPP'16]

- **Pattern recognition**
  - Connected-component labeling in computer vision $\rightarrow$ CC
  - SBV-Cut node clustering $\rightarrow$ CC, AP, bridge [DKE'12]

- **Cybersecurity**
  - Spamming botnet detection $\rightarrow$ CC [NSDI'09]
  - Suspicious DNS query $\rightarrow$ SCC [ICNP'10]


Motivation

1. Existing parallel graph computation frameworks only support a limited number of connectivity algorithms.

2. Different connected components share similar heterogeneous properties, similar computation techniques, while existing systems did not take advantage of them.

3. Existing methods only provide complete computation, while many queries can actually be answered with partial computation.
Part 2: Aquila Framework
Aquila Overview

Query: Is the graph connected?

Query transformation

Complete computation
- Partial computation
- Largest XCC
- Small XCC
- AP/Bridge

Workload reduction
- Trim
- Single parent only (SPO)

Adaptive parallel computation
- Adaptive parallel
- Enhanced BFS

Result
Technique #1: Query Transformation

- Complete computation
- Partial computation
  - Largest XCC
  - Small XCC
  - AP and Bridge
Benefit of Small XCC Query

- Over complete computation
  - 7x, 8x, 132x, 2,137x for (W)CC, SCC, BiCC, and BgCC, respectively

- Over selecting an arbitrary pivot
  - 6x, 85x, 62x, 209x for (W)CC, SCC, BiCC, and BgCC, respectively
Benefit of Largest XCC Query

- 1.2x, 1.1x, 1.03x, and 1.1x for (W)CC, SCC, BiCC, and BgCC, respectively
Benefit of AP and Bridge Query

- Articulation point (AP)
  - 423x, 43x, 50x, 2.3x, 2.5x speedup over Boost, DFS, Slota_BFS, Slota_LP, Aquila’s complete computation, respectively

- Bridge
  - 12x and 1.3x speedup over DFS and Aquila’s complete computation
Technique #2: Workload Reduction

- State-of-the-art BiCC computation algorithm [HiPC’14]
  - Lemma 1: On the constructed BFS tree, for any non-root vertex $p$, after removing it, if any of its children cannot reach a vertex at the same level of $p$, then $p$ is an AP. Similarly for the bridge.

- Observation
  - It needs to run up to $|V|$ BFSes, but only less than 1% of them will find a BiCC.

```
Algorithm 1: bfsBiCC(G, *level, *parent)
1 pivot = selectPivot(G);
2 bfs(G, level, parent);
3 foreach $v \in reverseBfsOrder(G)$ do
4     $p = parent[v]$;
5     $l = bfsConstrained(G, v, p, level)$;
6     if $l < level[p]$ then
7         Mark the visited edges as one BiCC;
```
Technique #2: Workload Reduction

- Single parent only (SPO) technique

- Lemma 2: For any non-root vertex \( p \), after removing it, if one of its children \( v \) can reach a vertex at the same level of \( p \), then \( p \) is not an AP from the view of \( v \). Not checking vertex \( v \) will not affect the correctness. Similarly for the bridge.

- Two types of second parent
  - Direct & Sibling induced
    
    ![Diagram](image)

    (a) Direct second parent
    
    (b) Sibling induced second parent
Technique #2: Workload Reduction

- **Trim**
  - Quickly remove trivial XCCs as the real-world graphs have a large number of them
  - Leverage the existing size-1 and size-2 trim for CC and SCC
  - Design new trim patterns for BiCC, BgCC, and related algorithms

![Diagram showing different components of a graph: XCC, CC/WCC, SCC, BiCC/BgCC]
Benefits of Workload Reduction

- Trim
  - 21% for both BiCC and BgCC
- SPO
  - 74% for BiCC, 77% for BgCC
- Together
  - 95% for BiCC (upperbound 99.6%)
  - 98% for BgCC (upperbound 99.7%)
Technique #3: Adaptive Parallel Computation

- Observation: Irregular task property
  - A few large XCCs take majority of the graph and their sizes are close to the order of graph size
  - The rest is a large number of trivial XCCs whose count is close to the order of graph size as well
Technique #3: Adaptive Parallel Computation

- Data parallel vs. task parallel
  - Data parallel → all the “workers” work together for one task
    - Fit for the large tasks in XCC
  - Task parallel → each “worker” works for a different task
    - Fit for the rest tasks in XCC
BFS for CC Computation

- **Pros**: Efficient for finding one specific CC
- **Cons**: Low efficient for many CCs

(a) An example graph  
(b) BFS tree, the identified CC  
(c) Start the 2\textsuperscript{nd} BFS on the remaining graph  
(d) Repeat with the 3\textsuperscript{rd} BFS  
(e) End of CC computation

• Pros: Efficient for finding one specific CC
• Cons: Low efficient for many CCs
Label Propagation for CC Computation

- **Pros:** can find all the CCs by running once (although several iterations)
- **Cons:** inflated workload

![Frontier queue](image)

- (f) Label initialization
- (g) 1\(^{st}\) iteration of label propagation
- (h) 2\(^{nd}\) iteration
- (i) 3\(^{rd}\) iteration
Technique #3: Adaptive Parallel Computation

- Large task:
  - Enhanced parallel BFS
  - Multi-pivot
  - Adaptive synchronization [SC’18]

- Small tasks
  - Label propagation $\rightarrow$ CC, SCC
  - Concurrent BFS $\rightarrow$ BiCC, BgCC

Part 3: Experiment
### Graph Benchmarks

<table>
<thead>
<tr>
<th>Graph</th>
<th>Abbrev.</th>
<th>Description</th>
<th># Nodes</th>
<th># Directed Edges</th>
<th># Undirected Edges</th>
<th># CCs</th>
<th>Largest CC Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baidu</td>
<td>BD</td>
<td>Baidu Baike hyperlink network</td>
<td>2.1M</td>
<td>17.8M</td>
<td>34M</td>
<td>15,561</td>
<td>98.4%</td>
</tr>
<tr>
<td>Pokec</td>
<td>PK</td>
<td>Pokec online social network</td>
<td>1.6M</td>
<td>30.6M</td>
<td>44.6M</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>Livejournal</td>
<td>LJ</td>
<td>Livejournal online social network</td>
<td>4.8M</td>
<td>68.5M</td>
<td>85.7M</td>
<td>1,876</td>
<td>99.9%</td>
</tr>
<tr>
<td>WikiEn</td>
<td>WE</td>
<td>Wikipedia hyperlink network</td>
<td>18.3M</td>
<td>172.2M</td>
<td>253.8M</td>
<td>1,366</td>
<td>99.9%</td>
</tr>
<tr>
<td>WikiLinkEn</td>
<td>WL</td>
<td>Wikipedia hyperlink network</td>
<td>11.2M</td>
<td>340.3M</td>
<td>516.9M</td>
<td>3,061</td>
<td>99.9%</td>
</tr>
<tr>
<td>Facebook</td>
<td>FB</td>
<td>Facebook connection</td>
<td>96.1M</td>
<td>679.7M</td>
<td>1.2B</td>
<td>5</td>
<td>99.9%</td>
</tr>
<tr>
<td>TwitterWww</td>
<td>TW</td>
<td>Twitter user following network</td>
<td>41.7M</td>
<td>1.5B</td>
<td>2.4B</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>TwitterMpi</td>
<td>TM</td>
<td>Twitter user following network</td>
<td>52.6M</td>
<td>2B</td>
<td>3.2B</td>
<td>29,533</td>
<td>99.9%</td>
</tr>
<tr>
<td>Friendster</td>
<td>FR</td>
<td>Friendster friendship network</td>
<td>68.3M</td>
<td>2.6B</td>
<td>3.6B</td>
<td>323,276</td>
<td>98.7%</td>
</tr>
<tr>
<td>RMAT</td>
<td>RM</td>
<td>R-MAT generator</td>
<td>4M</td>
<td>256M</td>
<td>506.2M</td>
<td>1.9M</td>
<td>52.1%</td>
</tr>
<tr>
<td>Random</td>
<td>RD</td>
<td>Gtgraph generator</td>
<td>4M</td>
<td>256M</td>
<td>512M</td>
<td>1</td>
<td>100%</td>
</tr>
</tbody>
</table>
Experiment Setting

• Hardware
  • A server with two Intel Xeon Gold 6126 CPUs
  • Each CPU has 12 cores
  • Hyper-threading enabled

• Software
  • GCC 4.8.5
  • OpenMP 3.1
  • Optimization level: -O3
Compared Works

- State-of-the-art method for each XCC
  - Multistep for CC [IPDPS’14], iSpan for SCC [SC’18], Slota_BFS for BiCC [HiPC’14]
- Popular graph computation systems
  - Galois [SOSP’13], X-Stream [SOSP’13], GraphChi [OSDI’12], Ligra [PPoPP’13]
- Other works
  - Hong’s FW-BW method for SCC [SC’13], Slota_LP for BiCC [HiPC’14]
  - Serial implementations: DFS-based, boost graph library
Complete Computation Performance

- (Weakly) Connected Component, (W)CC
  - State-of-the-art: Multistep (5.9x)
  - Four graph systems: Galois (53x), X-Stream (1,548), Ligra (264x), GraphChi (31x)
  - Serial implementations: DFS (67x), Boost (359x)
Complete Computation Performance

- Strongly Connected Component (SCC)
  - State-of-the-art: iSpan (1.1x)
  - Two parallel methods: Multistep (3.6x), Hong (3.7x)
  - Two graph systems: X-Stream (1,191x), GraphChi (918x)
  - Serial implementations: Boost (183x), DFS (83x)

![Speedup (log2 scale)](image)
Complete Computation Performance

- BiConnected Component (BiCC)
- State-of-the-art: Slota_BFS (21x)
- Slota’s label propagation method, Slota_LP (23x)
- Serial implementations: Boost (223x), DFS (22x)
Complete Computation Performance

• Bridgeless Connected Component (BgCC)
• Serial implementations: DFS (23x)
Technique Benefit

(a) CC

(b) SCC

(c) BiCC

(d) BgCC
Part 4: Conclusion
Conclusion

- New framework specialized for graph connectivity algorithms
- New computation strategies towards partial computation queries
- Adaptive parallel computation strategies with new techniques, e.g., single parent only
- Evaluation: outperform previous works for (W)CC, SCC, BiCC, and BgCC
Thank You & Questions

Source code will be available at https://github.com/iHeartGraph/Aquila

Contact us: yuedeji@gwu.edu, howie@gwu.edu