Ligra: A Lightweight Graph Processing Framework for Shared Memory

Julian Shun and Guy Blelloch 2013
Motivation

1. Shared Memory vs Distributed Memory
   - Many graph-processing frameworks designed for distributed memory systems (e.g. Pregel)
   - Advancements in technology -> enough storage in shared-memory machines (can handle graphs 100 billion edges in main memory)
   - Data locality, cheaper communication costs

2. Beamer et al, 2011 and 2012: hybrid approach to BFS exploiting variation in number of vertices and edges (i.e. frontier size) computed in each iteration of a parallel process
   - Can we generalize to other algorithms?
What is Ligra?

Simple shared-memory parallel graph-processing framework:

2 main data types and 2 main functions!

Graph
\( G = (V,E) \) or \( G = (V,E,w) \)

Vertex Subset
\( U \subseteq V \)

EDGE MAP
\( (G, U, F, C) \)

VERTEX MAP
\( (U, F) \)
Figure 1. Pseudocode for Breadth-First Search in our framework. The compare-and-swap function CAS(loc,oldV,newV) atomically checks if the value at location loc is equal to oldV and if so it updates loc with newV and returns true. Otherwise it leaves loc unmodified and returns false.
Application: BFS

1: Parents = \{-1, \ldots, -1\} \quad \triangleright \text{initialized to all -1's}
2: 
3: procedure UPDATE(s, d)
4: \quad return (CAS(&Parents[d], -1, s))
5: 
6: procedure COND(i)
7: \quad return (Parents[i] == -1)
8: 
9: procedure BFS(G, r) \quad \triangleright r is the root
10: \quad Parents[r] = r
11: \quad Frontier = \{r\} \quad \triangleright \text{vertex subset initialized to contain only } r
12: \quad while (SIZE(Frontier) \neq 0) do
13: \quad \quad Frontier = EDGEMAP(G, Frontier, UPDATE, COND)

Figure 1. Pseudocode for Breadth-First Search in our framework. The compare-and-swap function \text{CAS(loc,oldV,newV)} atomically checks if the value at location \text{loc} is equal to \text{oldV} and if so it updates \text{loc} with \text{newV} and returns \text{true}. Otherwise it leaves \text{loc} unmodified and returns \text{false}. 
### Application: BFS

1: Parents = \{-1, \ldots, -1\} \quad \triangleright \text{initialized to all -1’s}

2:

3: procedure UPDATE(s, d)
4: \quad return (CAS(&Parents[d], -1, s))

5:

6: procedure COND(i)
7: \quad return (Parents[i] == -1)

8:

9: procedure BFS(G, r) \quad \triangleright r \text{ is the root}
10: \quad Parents[r] = r
11: \quad Frontier = \{r\} \quad \triangleright \text{vertex subset initialized to contain only } r
12: \quad \textbf{while} \ (\text{SIZE(Frontier)} \neq 0) \ \textbf{do}
13: \quad \quad Frontier = EDGEMAP(G, Frontier, UPDATE, COND)

---

**Figure 1.** Pseudocode for Breadth-First Search in our framework. The compare-and-swap function \(\text{CAS}(loc,oldV,newV)\) atomically checks if the value at location \(loc\) is equal to \(oldV\) and if so it updates \(loc\) with \(newV\) and returns \(true\). Otherwise it leaves \(loc\) unmodified and returns \(false\).
Application: BFS

```plaintext
1: Parents = \{-1, \ldots, -1\}  \quad \triangleright \text{initialized to all -1's}
2: 
3: \textbf{procedure} UPDATE(s, d)
4: \quad \textbf{return} (CAS(&Parents[d], -1, s))
5: 
6: \textbf{procedure} COND(i)
7: \quad \textbf{return} (Parents[i] == -1)
8: 
9: \textbf{procedure} BFS(G, r)  \quad \triangleright r \text{ is the root}
10: \quad \text{Parents}[r] = r
11: \quad \text{Frontier} = \{r\}  \quad \triangleright \text{vertexSubset initialized to contain only } r
12: \quad \textbf{while} (\text{SIZE(Frontier)} \neq 0) \textbf{do}
13: \quad \quad \text{Frontier} = \text{EDGEMAP}(G, \text{Frontier}, \text{UPDATE}, \text{COND})
```

Figure 1. Pseudocode for Breadth-First Search in our framework. The compare-and-swap function \text{CAS(loc,oldV,newV)} atomically checks if the value at location \text{loc} is equal to \text{oldV} and if so it updates \text{loc} with \text{newV} and returns \text{true}. Otherwise it leaves \text{loc} unmodified and returns \text{false}. 
Hybrid Model for Varying Frontier-Size

Check out-neighbors for each vertex in the frontier

Check in-neighbors for each target vertex
Application: BFS

1: Parents = \{-1, \ldots, -1\} \quad \triangleright \text{initialized to all -1's}
2: 
3: procedure UPDATE(s, d)
4: \quad \text{return (CAS(&Parents[d], -1, s))}
5: 
6: procedure COND(i)
7: \quad \text{return (Parents[i] \equiv -1)}
8: 
9: procedure BFS(G, r) \quad \triangleright r \text{ is the root}
10: \quad Parents[r] = r
11: \quad Frontier = \{r\} \quad \triangleright \text{vertex subset initialized to contain only } r
12: \quad \text{while (SIZE(Frontier) \neq 0) do}
13: \quad \quad Frontier = EDGE MAP(G, Frontier, UPDATE, COND)

Figure 1. Pseudocode for Breadth-First Search in our framework.
The compare-and-swap function \text{CAS(loc,oldV,newV)} atomically checks if the value at location loc is equal to oldV and if so it updates loc with newV and returns true. Otherwise it leaves loc unmodified and returns false.
**Application: BFS**

```
1: Parents = \{-1, \ldots, -1\}  \quad \triangleright \text{initialized to all -1's}
2:
3: procedure UPDATE(s, d)
4:    return (CAS(&Parents[d], -1, s))
5:
6: procedure COND(i)
7:    return (Parents[i] == -1)
8:
9: procedure BFS(G, r)  \quad \triangleright r is the root
10:   Parents[r] = r
11:   Frontier = \{r\}  \quad \triangleright \text{vertex subset initialized to contain only } r
12:   while (SIZE(Frontier) \neq 0) do
13:      Frontier = EDGE_MAP(G, Frontier, UPDATE, COND)
```

*Figure 1.* Pseudocode for Breadth-First Search in our framework. The compare-and-swap function CAS(loc,oldV,newV) atomically checks if the value at location loc is equal to oldV and if so it updates loc with newV and returns true. Otherwise it leaves loc unmodified and returns false.
BFS Evaluation

• Implemented on a 40-core Intel machine with 256GB of RAM (and with multithreading)

https://www.youtube.com/watch?v=W5mDx_G45RQ&ab_channel=MMDSFoundation

• Comparing against direction-optimizing code by Beamer et al.
Other Applications and Results

<table>
<thead>
<tr>
<th>Input</th>
<th>Num. Vertices</th>
<th>Num. Directed Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D-grid</td>
<td>$10^7$</td>
<td>$6 \times 10^7$</td>
</tr>
<tr>
<td>random-local</td>
<td>$10^7$</td>
<td>$9.8 \times 10^7$</td>
</tr>
<tr>
<td>rMat24</td>
<td>$1.68 \times 10^7$</td>
<td>$9.9 \times 10^7$</td>
</tr>
<tr>
<td>rMat27</td>
<td>$1.34 \times 10^8$</td>
<td>$2.12 \times 10^8$</td>
</tr>
<tr>
<td>Twitter</td>
<td>$4.17 \times 10^7$</td>
<td>$1.47 \times 10^8$</td>
</tr>
<tr>
<td>Yahoo*</td>
<td>$1.4 \times 10^9$</td>
<td>$12.9 \times 10^9$</td>
</tr>
</tbody>
</table>

*The original asymmetric graph has $6.6 \times 10^9$ edges.

Table 1. Graph inputs.

<table>
<thead>
<tr>
<th>Application</th>
<th>3D-grid</th>
<th>random-local</th>
<th>rMat24</th>
<th>rMat27</th>
<th>Twitter</th>
<th>Yahoo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(SU)</td>
<td>(SU)</td>
<td>(SU)</td>
<td>(SU)</td>
<td>(SU)</td>
<td>(SU)</td>
</tr>
<tr>
<td>Breadth-First Search</td>
<td>2.9</td>
<td>0.28</td>
<td>10.4</td>
<td>2.11</td>
<td>0.073</td>
<td>28.9</td>
</tr>
<tr>
<td>Betweenness Centrality</td>
<td>9.15</td>
<td>0.765</td>
<td>12.0</td>
<td>8.53</td>
<td>0.265</td>
<td>32.2</td>
</tr>
<tr>
<td>Graph Radii</td>
<td>351</td>
<td>10.0</td>
<td>35.1</td>
<td>25.6</td>
<td>0.734</td>
<td>34.9</td>
</tr>
<tr>
<td>Connected Components</td>
<td>51.5</td>
<td>1.71</td>
<td>30.1</td>
<td>14.8</td>
<td>0.399</td>
<td>37.1</td>
</tr>
<tr>
<td>PageRank (1 iteration)</td>
<td>4.29</td>
<td>0.145</td>
<td>29.6</td>
<td>6.55</td>
<td>0.224</td>
<td>29.2</td>
</tr>
<tr>
<td>Bellman-Ford</td>
<td>63.4</td>
<td>2.39</td>
<td>26.5</td>
<td>18.8</td>
<td>0.677</td>
<td>27.8</td>
</tr>
</tbody>
</table>

*Running times (in seconds) of algorithms over various inputs on a 40-core machine (with hyper-threading). (SU) indicates the speedup of the application (single-thread time divided by 40-core time).

Table 2.
Experimental Results

Connected Components

https://www.youtube.com/watch?v=W5mDx_G45RQ&ab_channel=MMDSFoundation
Experimental Results

PageRank

https://www.youtube.com/watch?v=W5mDx_G45RQ&ab_channel=MMDSFoundation
Summary

- Lightweight parallel graph processing framework
- Dependence on shared-memory systems (no communication overhead as compared to distributed systems)
- Designed for frontier-based algorithms
- Comparable to or outperformed then graph-processing frameworks
Discussion

• Incomplete evaluation

• Performance to be hardware dependent
  • Worse performance with a different set-up (64-core AMD Opteron machine)

• Scalability and speedup limited by tech
  • Expansion to GPUs? [Shun et al. 2013]

• Exploring applications in other graph algorithms (e.g. max flow, biconnected components, belief propagation, Markov clustering) [Shun et al. 2013]

• Dynamic graph data, graph data stream processing?
Extensions

- Ligra+ (2015) - compressed graphs -> require less memory

- Julienne (2017) - bucketing-based algorithms (generalization of frontier-based algorithms)

- Hygra (2020) - support for hyper graphs

Code at https://github.com/jshun/ligra
Thank you!
Questions?
References


