Ligra: A Lightweight Graph Processing Framework for Shared Memory

Paper authors: Julian Shun, Guy Blelloch (Carnegie Mellon University)
Presenter: Mihai-Ionut Enache

25 October 2021
Motivation - Why Study Graphs?

- Many applications: social networks, Web graph, medicine
- Types of problems:
  - Shortest path
  - Clustering (e.g. community recovery)
  - Recommendation engines
  - Scientific computations
  - others
Shared Memory vs. Distributed Systems

- **In the past:**
  - Memory scarce, few cores available; hard to handle large graphs
  - Most of the frameworks designed to run on distributed systems

- **Today:**
  - Single multicore commodity computers can have TBs of memory
  - Can accommodate graphs of billions of edges

- **Why shared memory?**
  - More efficient (per dollar / core / joule)
  - Low communication costs $\Rightarrow$ better performance
  - Simplicity: easier to write algorithms for shared memory
  - More reliable: shared memories can run months / years without failure
Ligra - Preview

- Lightweight
  - Interface: only a few functions
  - Implementation: simple and fast
- 2 datatypes: one for graph $G = (V, E)$ and one for subsets of $V$ ($\text{VertexSubset}$)
- 2 essential functions:
  - $\text{VertexMap}$ (maps over $V$ or subsets of $V$)
  - $\text{EdgeMap}$
    - Useful in graph traversal algorithms
- Compare-and-swap (CAS): atomic instruction for conditional swapping
Application: Breadth-first Search
Application: Breadth-first Search
Application: Breadth-first Search
Application: Breadth-first Search
BFS in Ligra

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

- Interface allows processing edges in different orders
  - Ligra is edge-oriented
  - Previous systems mostly vertex-oriented
- 3 ways to process edges:
  - **Sparse representation**: iterate over the active source vertices and check (target of) out-edges
  - **Dense representation**: iterate over the destination vertices and check (source of) in-edges
  - Flat map: check all edges
Application: BFS (sparse representation)
Application: BFS (dense representation)
Sparse or Dense Representation?

- Idea: use a hybrid approach
  - Choose based on the size of the frontier and the number of out-edges
  - If larger than a fixed threshold, use dense, otherwise sparse
- Inspired from previous work of an efficient BFS implementation
  - Ligra generalizes the idea
Interface

- Apply $F$ on all edges $(s, t)$ s.t. $s \in U$ and $C(t)$ hold
- $F$ can run in parallel
- User’s responsibility for parallel correctness
- $F$ can have side effects
- For weighted graphs $F$ takes an additional argument
- $C$ is optional - useful for algorithms when data needs to be updated only once (BFS)

```
EDGEMAP(G : graph,
  U : vertexSubset,
  F : (vertex x vertex) -> bool,
  C : vertex -> bool) : vertexSubset.
```

```
VERTEXMAP(U : vertexSubset,
  F : vertex -> bool) : vertexSubset.
```
EdgeMap - Implementation

Algorithm 1 \texttt{EDGE\textsc{Map}}

1: procedure \texttt{EDGE\textsc{Map}}(G, U, E, C)
2: \hspace{1em} if ([U] + sum of out-degrees of U > threshold) then
3: \hspace{2em} return \texttt{EDGE\textsc{MapDense}}(G, U, P, C)
4: \hspace{1em} else return \texttt{EDGE\textsc{MapSparse}}(G, U, F, C)
### Algorithm 1 \textsc{EdgeMap}

1: procedure \textsc{EdgeMap}(G, U, F, C) \\
2: \hspace{1em} if (|U| + sum of out-degrees of \(U > \) threshold) then \\
3: \hspace{2em} return \textsc{EdgeMapDense}(G, U, F, C) \\
4: \hspace{1em} else return \textsc{EdgeMapSparse}(G, U, F, C)

### Algorithm 3 \textsc{EdgeMapDense}

1: procedure \textsc{EdgeMapDense}(G, U, F, C) \\
2: \hspace{1em} Out = {} \\
3: \hspace{2em} parfor \(i \in \{0, \ldots, |V| - 1\} \) do \\
4: \hspace{3em} if (\(C(i) == 1\)) then \\
5: \hspace{4em} for \(ngh \in N^-(i) \) do \\
6: \hspace{5em} if (\(ngh \in U \) and \(F(ngh, i) == 1\)) then \\
7: \hspace{6em} Add i to Out \\
8: \hspace{5em} if (\(C(i) == 0\)) then break \\
9: \hspace{1em} return Out
Implementation

Algorithm 1: EDGEMAP
1: procedure EDGEMAP(G, U, F, C)
2: if (|U| + sum of out-degrees of U > threshold) then
3: return EDGEMAPDENSE(G, U, F, C)
4: else return EDGEMAPSPARSE(G, U, F, C)

In parallel

Algorithm 3: EDGEMAPDENSE
1: procedure EDGEMAPDENSE(G, U, F, C)
2: Out = {}
3: parfor i ∈ {0, ..., |V| - 1} do
4: if (C(i) == 1) then
5: for ngh ∈ N−(i) do
6: if (ngh ∈ U and F(ngh, i) == 1) then
7: Add i to Out
8: if (C(i) == 0) then break
9: return Out
Implementation

Algorithm 1 EDGEMAP

1: procedure EDGEMAP(G, U, F, C)
2: if (|U| + sum of out-degrees of U > threshold) then
3: \[\text{return EDGEMAPDENSE}(G, U, F, C)\]
4: else return EDGEMAPSPARSE(G, U, F, C)

Algorithm 3 EDGEMAPDENSE

1: procedure EDGEMAPDENSE(G, U, F, C)
2: Out = \{
3: \parfor i \in \{0, \ldots, |V| - 1\} do
4: \text{if } (C(i) == 1) \text{ then}
5: \text{for ngh \in N^-(i) do}
6: \text{if } (\text{ngh} \in U \text{ and } F(\text{ngh}, i) == 1) \text{ then}
7: \text{Add i to Out}
8: \text{if } (C(i) == 0) \text{ then break}
9: return Out

In parallel
Sequentially
Implementation

Algorithm 1 EDGEMAP

1: procedure EDGEMAP\((G, U, F, C)\)
2: 
3: if \((|U| + \text{sum of out-degrees of } U > \text{threshold})\) then
4: 
5: return EDGEMAPDENSE\((G, U, F, C)\)
6: 
7: else return EDGEMAPSPARSE\((G, U, F, C)\)

Algorithm 2 EDGEMAPSPARSE

1: procedure EDGEMAPSPARSE\((G, U, F, C)\)
2: 
3: Out = \(\{\}\)
4: 
5: parfor each \(v \in U\) do
6: 
7: parfor ngh \(\in N^+(v)\) do
8: 
9: if \((C(\text{ngh}) == 1 \text{ and } F(v, \text{ngh}) == 1)\) then
10: 
11: Add ngh to Out
12: 
13: Remove duplicates from Out
14: 
15: return Out

Algorithm 3 EDGEMAPDENSE

1: procedure EDGEMAPDENSE\((G, U, F, C)\)
2: 
3: Out = \(\{\}\)
4: 
5: parfor \(i \in \{0, \ldots, |V| - 1\}\) do
6: 
7: if \((C(i) == 1)\) then
8: 
9: for ngh \(\in N^-(i)\) do
10: 
11: if \((\text{ngh} \in U \text{ and } F(\text{ngh}, i) == 1)\) then
12: 
13: Add i to Out
14: 
15: if \((C(i) == 0)\) then break
16: 
17: return Out
Implementation

Algorithm 1 EDGEMAP

1: procedure EDGEMAP\((G, U, F, C)\)
2: \[\text{if } (|U| + \text{sum of out-degrees of } U > \text{threshold}) \text{ then}\]
3: \[\text{return EDGEMAPDENSE}(G, U, F, C)\]
4: \[\text{else return EDGEMAPSPARSE}(G, U, F, C)\]

Algorithm 2 EDGEMAPSPARSE

1: procedure EDGEMAPSPARSE\((G, U, F, C)\)
2: \[\text{Out } = \{\}\]
3: \[\text{parfor each } v \in U \text{ do}\]
4: \[\text{parfor ngh } \in N^+(v) \text{ do}\]
5: \[\text{if } (C(\text{ngh}) == 1 \text{ and } F(v, \text{ngh}) == 1) \text{ then}\]
6: \[\text{Add ngh to Out}\]
7: \[\text{Remove duplicates from Out}\]
8: \[\text{return Out}\]

Algorithm 3 EDGEMAPDENSE

1: procedure EDGEMAPDENSE\((G, U, F, C)\)
2: \[\text{Out } = \{\}\]
3: \[\text{parfor } i \in \{0, \ldots, |V| - 1\} \text{ do}\]
4: \[\text{if } (C(i) == 1) \text{ then}\]
5: \[\text{for ngh } \in N^-(i) \text{ do}\]
6: \[\text{if } (\text{ngh } \in U \text{ and } F(\text{ngh}, i) == 1) \text{ then}\]
7: \[\text{Add i to Out}\]
8: \[\text{if } (C(i) == 0) \text{ then break}\]
9: \[\text{return Out}\]
**VertexMap - Implementation**

```
Algorithm 4 VERTEXMAP

1: procedure VERTEXMAP(U, F)
2:    Out = {}
3:    parfor u ∈ U do
4:        if (F(u) == 1) then Add u to Out
5:    return Out
```
Optimizations

- F in EdgeMapDense is applied sequentially $\Rightarrow$ doesn’t need atomicity
  - Optimized version of EdgeMap: accepts 2 versions of F
  - Authors found this to be slightly faster for some applications
- Users can set a different threshold for EdgeMapSparse vs EdgeMapDense
  - Default is $|E| / 20$
- Inner-loop of EdgeMapDense can also run in parallel
  - User needs to give up the “break” option to enable this

```latex
\begin{algorithm}
\caption{EDGEMAPDENSE}
\begin{algorithmic}[1]
\Procedure{edgemapdense}{$G, U, F, C$}
\State Out $\leftarrow \emptyset$
\ParFor {$i \in \{0, \ldots, |V| - 1\}$}
\If {$C(i) == 1$}
\For {$\text{ngh} \in N^-(i)$}
\If {$\text{ngh} \in U$ and $F(\text{ngh}, i) == 1$}
\State Add $i$ to Out
\EndIf
\EndFor
\EndIf
\If {$C(i) == 0$} \textbf{break} \EndIf
\EndFor
\State \Return Out
\EndProcedure
\end{algorithmic}
\end{algorithm}
```
Applications

1. BFS
2. Betweenness Centrality
3. Graph Radii Estimation and Multiple BFS
4. Connected Components
5. PageRank + PageRank-Delta
6. Bellman-Ford Shortest Paths
Experiments

<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>$2.12 \times 10^9$</td>
</tr>
<tr>
<td>Twitter</td>
<td>$4.17 \times 10^7$</td>
<td>$1.47 \times 10^9$</td>
</tr>
<tr>
<td>Yahoo*</td>
<td>$1.4 \times 10^9$</td>
<td>$12.9 \times 10^9$</td>
</tr>
</tbody>
</table>

Table 1. Graph inputs. *The original asymmetric graph has $6.6 \times 10^9$ edges.
# Experiments (continued)

<table>
<thead>
<tr>
<th>Application</th>
<th>3D-grid (1)</th>
<th>3D-grid (40h)</th>
<th>random-local (SU)</th>
<th>random-local (40h)</th>
<th>rMat24 (1)</th>
<th>rMat24 (40h)</th>
<th>rMat27 (1)</th>
<th>rMat27 (40h)</th>
<th>Twitter (1)</th>
<th>Twitter (40h)</th>
<th>Yahoo (1)</th>
<th>Yahoo (40h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-First Search</td>
<td>2.9</td>
<td>0.28</td>
<td>10.4</td>
<td></td>
<td>2.11</td>
<td>0.073</td>
<td>28.9</td>
<td>2.83</td>
<td>0.104</td>
<td>27.2</td>
<td>11.8</td>
<td>6.92</td>
</tr>
<tr>
<td>Betweenness Centrality</td>
<td>9.15</td>
<td>0.765</td>
<td>12.0</td>
<td></td>
<td>8.53</td>
<td>0.265</td>
<td>32.2</td>
<td>11.3</td>
<td>0.37</td>
<td>30.5</td>
<td>113</td>
<td>4.07</td>
</tr>
<tr>
<td>Graph Radii</td>
<td>351</td>
<td>10.0</td>
<td>35.1</td>
<td></td>
<td>25.6</td>
<td>0.734</td>
<td>34.9</td>
<td>39.7</td>
<td>1.21</td>
<td>32.8</td>
<td>337</td>
<td>12.0</td>
</tr>
<tr>
<td>Connected Components</td>
<td>51.5</td>
<td>1.71</td>
<td>30.1</td>
<td></td>
<td>14.8</td>
<td>0.399</td>
<td>37.1</td>
<td>14.1</td>
<td>0.527</td>
<td>26.8</td>
<td>204</td>
<td>10.2</td>
</tr>
<tr>
<td>PageRank (1 iteration)</td>
<td>4.29</td>
<td>0.145</td>
<td>29.6</td>
<td></td>
<td>6.55</td>
<td>0.224</td>
<td>29.2</td>
<td>8.93</td>
<td>0.25</td>
<td>35.7</td>
<td>243</td>
<td>6.13</td>
</tr>
<tr>
<td>Bellman-Ford</td>
<td>63.4</td>
<td>2.39</td>
<td>26.5</td>
<td></td>
<td>18.8</td>
<td>0.677</td>
<td>27.8</td>
<td>17.8</td>
<td>0.694</td>
<td>25.6</td>
<td>116</td>
<td>4.03</td>
</tr>
</tbody>
</table>

Table 2. 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).
Comparison to other frameworks

- Related frameworks: Pregel, KDT, Pegasus, PowerGraph
- BFS: **10-28 speedup** + almost as efficient as Beam’s highly optimized BFS
- Betweenness centrality: **12-32 speedup**
- Graph radii estimation: **23-35 speedup**
- Connected components: **20-37 speedup**
- PageRank: **29-39 speedup** for a single iteration
- Bellman-Ford: **18-28 speedup**
Subsequent work

- Ligra+ - framework for processing compressed graphs (half the space of uncompressed graphs)
- Hygra - framework for hypergraphs (hyperedge = edge with arbitrary number of vertices)

(source: https://github.com/jshun/ligra/graphs/contributors)
Summary

- Ligra is a graph processing framework targeting a class of parallel algorithms
- It comes with a lightweight interface and implementation
- Experimental evaluation shows it performs better than existing work and almost as good as highly optimized code
- Limitation: no support for algorithms that need to modify the input graph