Smaller and Faster: Parallel Processing of Compressed Graphs with Ligra+

Julian Shun

Based on joint work with Guy Blelloch and Laxman Dhulipala
Ligra Graph Processing Framework

EdgeMap
- Breadth-first search
- Betweenness centrality
- Connected components
- Triangle counting
- K-core decomposition
- Maximal independent set
- Set cover

VertexMap
- Single-source shortest paths
- Eccentricity estimation
- (Personalized) PageRank
- Local graph clustering
- Biconnected components
- Collaborative filtering

Simplicity, Performance, Scalability
Steps for Graph Traversal

• Operate on a subset of vertices
• Map computation over subset of edges in parallel
• Return new subset of vertices
• Map computation over subset of vertices in parallel
Large Graphs

Amazon EC2

<table>
<thead>
<tr>
<th>vCPU</th>
<th>ECU</th>
<th>Memory (GiB)</th>
<th>Instance Storage (GB)</th>
<th>Linux/UNIX Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1e.xlarge</td>
<td>4</td>
<td>122</td>
<td>1 x 120 SSD</td>
<td>$0.834 per Hour</td>
</tr>
<tr>
<td>x1e.2xlarge</td>
<td>8</td>
<td>244</td>
<td>1 x 240 SSD</td>
<td>$1.668 per Hour</td>
</tr>
<tr>
<td>x1e.4xlarge</td>
<td>16</td>
<td>488</td>
<td>1 x 480 SSD</td>
<td>$3.336 per Hour</td>
</tr>
<tr>
<td>x1e.8xlarge</td>
<td>32</td>
<td>976</td>
<td>1 x 960</td>
<td>$6.672 per Hour</td>
</tr>
<tr>
<td>x1e.16xlarge</td>
<td>64</td>
<td>1952</td>
<td>1 x 1920 SSD</td>
<td>$13.344 per Hour</td>
</tr>
<tr>
<td>x1e.32xlarge</td>
<td>128</td>
<td>3904</td>
<td>2 x 1920 SSD</td>
<td>$26.688 per Hour</td>
</tr>
</tbody>
</table>

- Most can fit on commodity shared memory machine

Example

Dell PowerEdge R930:
Up to 96 cores and 6 TB of RAM
What if you don’t have or can’t afford that much memory?

Graph Compression
Ligra+: Adding Graph Compression to Ligra
Ligra+: Adding Graph Compression to Ligra

- Same interface as Ligra
- All changes hidden from the user!
Graph representation

Vertex IDs
0  1  2  3

Offsets
0  4  5  11 ...

Edges
2  7  9  16  0  1  6  9  12 ...

Compressed Edges
2  5  2  7 -1 -1 5  3  3 ...

Sort edges and encode differences

2 - 0 = 2  7 - 2 = 5  1 - 2 = -1
Variable-length codes

• k-bit codes
  • Encode value in chunks of k bits
  • Use k-1 bits for data, and 1 bit as the “continue” bit
• Example: encode “401” using 8-bit (byte) code
• In binary: 

```
1 1 0 0 1 0 0 0 1
```

7 bits for data

```
1 0 0 1 0 0 0 1
```

“continue” bit

```
0 0 0 0 0 0 0 1 1
```
Encoding optimization

- Another idea: get rid of “continue” bits

<table>
<thead>
<tr>
<th>x_1</th>
<th>x_2</th>
<th>x_3</th>
<th>x_4</th>
<th>x_5</th>
<th>x_6</th>
<th>x_7</th>
<th>x_8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Use run-length encoding

Number of bytes required to encode each integer

Number of bytes per integer

Size of group (max 64)

Integers in group encoded in byte chunks

- Increases space, but makes decoding cheaper (no branch misprediction from checking “continue” bit)
Ligra+: Adding Graph Compression to Ligra

- Same interface as Ligra
- All changes hidden from the user!

- Use compressed representation
- Same as before
- Decode edges on-the-fly
- Same as before
Modifying EdgeMap

- Processes outgoing edges of a subset of vertices

![Diagram of VertexSubset with vertices 0, 7, 16, 25, 44]

```
<table>
<thead>
<tr>
<th>2</th>
<th>5</th>
<th>2</th>
<th>7</th>
<th>9</th>
<th>2</th>
<th>1</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-16</td>
<td>2</td>
<td>19</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
```

All vertices processed in parallel

What about high-degree vertices?
Handling high-degree vertices

<table>
<thead>
<tr>
<th>-1</th>
<th>2</th>
<th>4</th>
<th>3</th>
<th>16</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>8</th>
<th>19</th>
<th>4</th>
<th>1</th>
<th>23</th>
<th>14</th>
<th>12</th>
<th>1</th>
<th>9</th>
<th>10</th>
<th>3</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
</table>

Chunks of size T

-1 2 4 3 16 2

27 5 8 19 4 1

87 14 12 1 9 10

Encode first entry relative to source vertex

All chunks can be decoded in parallel!

- We chose T=1000
- Similar performance and space usage for a wide range of T
Ligra+ Space Savings

- Space savings of about 1.3—3x
- Could use more sophisticated schemes to further reduce space, but more expensive to decode
- Cost of decoding on-the-fly?
• Cost of decoding on-the-fly?
• Memory subsystem is a scalability bottleneck in parallel as these graph algorithms are memory-bound
• Ligra+ decoding gets better parallel speed up
Ligra Summary

VertexSubset  VertexMap  EdgeMap

Optimizations: Hybrid traversal and graph compression

Breadth-first search
Betweenness centrality
Connected components
Triangle counting
K-core decomposition
Maximal independent set
...

Single-source shortest paths
Eccentricity estimation
(Personalized) PageRank
Local graph clustering
Biconnected components
Collaborative filtering
...

Simplicity, Performance, Scalability
Thank you!


Code: https://github.com/jshun/ligra/