Lightweight Processing on Compressed Graphs

Guy Blelloch
Carnegie Mellon University

Joint work with Julian Shun and Laxman Dhulipala

http://www.istc-cc.cmu.edu/
Large Graphs

Need to efficiently analyze these graphs

- Computational efficiency
- Space efficiency
- Programming efficiency
Breadth-first Search (BFS)

- Compute a BFS tree rooted at source $r$ containing all vertices reachable from $r$

  - Can process each frontier in parallel
    - Race conditions, load balancing
BFS Abstractly: Frontier Based

1. Operate on a subset of vertices
2. Map computation over subset of edges in parallel
3. Return new subset of vertices
4. (Map computation over subset of vertices in parallel)

BFS visits every vertex once, but in general can visit many times. Synchronous.

Can we build an abstraction for these types of algorithms?

- Breadth-first search
- Betweenness centrality
- Connected components
- Delta stepping
- Bellman-Ford shortest paths
- Graph eccentricity estimation
- PageRank
- Diameter estimation

ISTC 2015
Graph Processing Systems

• Existing: Pregel/Giraph, GraphLab, Pegasus, Knowledge Discovery Toolbox, GraphChi, Parallel BGL, and many others...

• Our system: Ligra - Lightweight graph processing system for shared memory
  ▫ Efficient for “frontier-based” algorithms

• Probably no one-size fits all
Why the Cloud

- Costs
- Cloud will enable wide use

<table>
<thead>
<tr>
<th>AWS</th>
<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>r3.large</td>
<td>2</td>
<td>6.5</td>
<td>15</td>
<td>1 x 32 SSD</td>
<td>$0.175 per Hour</td>
</tr>
<tr>
<td>r3.xlarge</td>
<td>4</td>
<td>13</td>
<td>30.5</td>
<td>1 x 80 SSD</td>
<td>$0.350 per Hour</td>
</tr>
<tr>
<td>r3.2xlarge</td>
<td>8</td>
<td>26</td>
<td>61</td>
<td>1 x 160 SSD</td>
<td>$0.700 per Hour</td>
</tr>
<tr>
<td>r3.4xlarge</td>
<td>16</td>
<td>52</td>
<td>122</td>
<td>1 x 320 SSD</td>
<td>$1.400 per Hour</td>
</tr>
<tr>
<td>r3.8xlarge</td>
<td>32</td>
<td>104</td>
<td>244</td>
<td>2 x 320 SSD</td>
<td>$2.800 per Hour</td>
</tr>
</tbody>
</table>
Ligra

- Operate on a subset of vertices
- Map computation over subset of edges in parallel and return new subset of vertices
- (Map computation over subset of vertices in parallel)
bool f(v) {
    data[v] = data[v] + 1;
    return (data[v] == 1);
}
Ligra Framework

Why edge based?
- Parallel over the edges
- Sparse/dense (discussed later)
Breadth-first Search in Ligra

parents = {-1, ..., -1};  // -1 indicates "unvisited"

procedure UPDATE(s, d):
    return compare_and_swap(parents[d], -1, s);

procedure COND(i):
    return parents[i] == -1;  // checks if "unvisited"

procedure BFS(G, r):
    parents[r] = r;
    frontier = {r};  // VertexSubset
    while (size(frontier) > 0):
        frontier = EDGEMAP(G, frontier, UPDATE, COND);
#include "ligra.h"

struct BFS_F {
    intT* Parents;
    
    BFS_F(intT* _Parents) : Parents(_Parents) {} 

    inline bool update (intT s, intT d) { //Update 
        if(Parents[d] == -1) { Parents[d] = s; return 1; } 
        else return 0; 
    } 

    inline bool updateAtomic (intT s, intT d){ //atomic version of Update 
        return (CAS(&Parents[d],(intT)-1,s)); 
    } 

    //cond function checks if vertex has been visited yet 
    inline bool cond (intT d) { return (Parents[d] == -1); } 
}; 

template <class vertex>
void Compute(graph<vertex> GA, intT start) {
    intT n = GA.n; 
    //creates Parents array, initialized to all -1, except for start 
    intT* Parents = newA(intT,GA.n); 
    parallel_for(intT i=0;i<GA.n;i++) Parents[i] = -1; 
    Parents[start] = start; 

    vertexSubset Frontier(n,start); //creates initial frontier 

    while(!Frontier.isEmpty()) { //loop until frontier is empty 
        vertexSubset output = edgeMap(GA, Frontier, BFS_F(Parents)); 
        Frontier.del(); 
        Frontier = output; //set new frontier 
    }
    Frontier.del(); 
    free(Parents);
procedure **EDGEMAP**\( (G, \text{frontier}, \text{Update}, \text{Cond}) \):

\[
\text{if (} |\text{frontier}| + \text{sum of out-degrees} > \text{threshold} \text{)} \text{ then:} \\
\text{return } \text{EDGEMAP\_DENSE}(G, \text{frontier}, \text{Update}, \text{Cond});
\]

\[
\text{else:} \\
\text{return } \text{EDGEMAP\_SPARSE}(G, \text{frontier}, \text{Update}, \text{Cond});
\]

Loop through outgoing edges of frontier vertices in parallel

Loop through incoming edges of “unexplored” vertices (in parallel), breaking early if possible

- First used by Beemer for BFS, but Ligra shows that useful for a wide variety of algorithms
Frontier Plots

(a) BFS

(b) Betweenness Centrality

(c) Radii Estimation

(d) Connected Components

(e) PageRank-Delta

(f) Bellman-Ford
Benefit of Sparse/Dense Traversal

Twitter graph (41M vertices, 1.5B edges)

- BFS
- Betweenness Centrality
- Connected Components
- Eccentricity Estimation

40-core running time (seconds)

Sparse

Sparse/Dense

ISTC 2015
Ligra Performance

- Ligra performance close to hand-written code

Twitter graph (41M vertices, 1.5B edges)

- Running time (seconds)
- Ligra (40-core machine)
- Hand-written Cilk/OpenMP (40-core machine)

- Page Rank iteration
- BFS
- Connected Components
Ligra Performance

- Ligra performance close to hand-written code
- Faster than distributed-memory on per-core basis
- Several shared-memory graph processing systems subsequently developed: Galois [SOSP ‘13], X-stream [SOSP ‘13], PRISM [SPAA ‘14], Polymer [PPoPP ‘15], Ringo [SIGMOD ‘15]
Large Graphs

- All fit in a Terabyte of memory; can fit on commodity shared memory machine
- *What if you don’t have that much RAM, or don’t want to “rent” that much RAM?*
• Difference encoding (using variable-length codes) for sorted edges per vertex
• Modify EdgeMap: parallel edge decoding on-the-fly
• All hidden from the user!
Graph Compression

- Compress the graph so that it uses less memory

Vertex IDs

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

Offsets

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>5</th>
<th>11</th>
<th>...</th>
</tr>
</thead>
</table>

Edges

<table>
<thead>
<tr>
<th>2</th>
<th>7</th>
<th>9</th>
<th>16</th>
<th>0</th>
<th>1</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>...</th>
</tr>
</thead>
</table>

Compressed Edges

<table>
<thead>
<tr>
<th>2</th>
<th>5</th>
<th>2</th>
<th>7</th>
<th>-1</th>
<th>-1</th>
<th>5</th>
<th>3</th>
<th>3</th>
<th>...</th>
</tr>
</thead>
</table>

- Sort edges
- First edge: Store difference between source and target vertex IDs
- Subsequent edges: store difference with previous edge

- Graph reordering to improve locality
- Goal: give neighbors IDs close to vertex ID
- BFS, DFS, METIS, our own separator-based algorithm
Cost of decoding on-the-fly?
Memory bottleneck a bigger issue as graph algorithms are memory-bound
Conclusion

• Ligra: lightweight graph processing framework for shared-memory
  ▫ “frontier-based” algorithms
  ▫ Switches computation based on frontier size

• Ligra+: extension which incorporates graph compression
  ▫ Reduces space usage and improves parallel performance

• Code: http://github.com/jshun/ligra