Green-Marl: A DSL for Easy and Efficient Graph Analysis

Sungpack Hong, Hassan Chafi, Eric Sedlar, Kunle Olukotun

Presented by: Nnaemeka Obodoekwe
Problem

- Difficult to execute graph analysis on large dataset
- Current graph processing frameworks force user to rewrite their program

The paper offers a Domain Specific Language (DSL)

- Easy to express graph algorithms
- Expose data-level parallelism
- Can compile to various backends
Betweenness Centrality algorithm in Green-Marl

```plaintext
Procedure Compute_BC(
    G: Graph, BC: Node_PROP<Float>(G)) {
  G.BC = 0;    // initialize BC
  Foreach(s: G.Nodes) {
    // define temporary properties
    Node_PROP<Float>(G) Sigma;
    Node_PROP<Float>(G) Delta;
    s.Sigma = 1; // Initialize Sigma for root
    // Traverse graph in BFS-order from s
    InBFS(v: G.Nodes From s)(v!=s) {
      // sum over BFS-parents
      v.Sigma = Sum(w: v.UpNbrs) {w.Sigma};
    }
    // Traverse graph in reverse BFS-order
    InRBFS(v!=s) {
      // sum over BFS-children
      v.Delta = Sum (w:v.DownNbrs) {
        v.Sigma / w.Sigma * (1+ w.Delta)
      };
      v.BC += v.Delta @s; // accumulate BC
    }
  }
}
```

Scope of the language

- Graph is an ordered pair of nodes and edges – $G = (N, E)$
- Each node/edge has some properties
- Given a graph and a set of properties Green Marl can:
  - Compute scalar value
  - Compute a new property
  - Select subgraph from original graph
Language Construct

- Data types – 5 primitive types
  - Int, bool, string, double and float
- Collections
  - Set, Order, Sequence

### Semantics on collections

<table>
<thead>
<tr>
<th>Group</th>
<th>Op-Name</th>
<th>sequential</th>
<th>parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$S$</td>
<td>$O$</td>
</tr>
<tr>
<td>Grow</td>
<td>Add</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>Push(Front/Back)</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>Shrink</td>
<td>Remove</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>Pop(Front/Back)</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>Clear</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>Lookup</td>
<td>Has</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>Front(Back)</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>Size</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>Copy</td>
<td>$=$</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>Iteration</td>
<td>Items</td>
<td>v</td>
<td>v</td>
</tr>
</tbody>
</table>

Modification under iteration → Shrink, Grow, or Copy: X

Conflicts under parallel execution →
  - Grow-Shrink: X
  - Lookup-Shrink: ?
  - Lookup-Grow: ?
Iterations and Traversals

- For
- Foreach
- Breadth-First Search
- Depth-First Search
Deferred Assignment

52    Foreach (s:G.Nodes) { 
53        // no conflict. t.X gives 'old' value 
54        s.X <= Sum(t:s.Nbrs) {t.X} @ s 
55    }
56    // All the writes to X becomes visible simultaneously
57    // at the end of the s iteration.

Reductions

58    Int x,y;
59    x = Sum(t:G.Nodes){t.A};
60    y = 0;
61    Foreach (t:G.Nodes)
62    y+= t.A;

<table>
<thead>
<tr>
<th>In-place</th>
<th>Assignment</th>
<th>In-place</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>&amp;&amp;=</td>
<td>Sum</td>
<td>+=</td>
</tr>
<tr>
<td>Any</td>
<td></td>
<td></td>
<td>=</td>
</tr>
<tr>
<td>Min</td>
<td>min=</td>
<td>Count</td>
<td>++</td>
</tr>
<tr>
<td>Max</td>
<td>max=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Overview of Green-Marl DSL-compiler Usage

User Application
- Graph Analysis

Green-Marl Code
- Target Code

Graph Data Structure (LIB)

Green-Marl Compiler
- Parsing & Checking
- Front-end Transform
- Back-end Transform
- Code Gen
Some optimizations

Set graph loop fusion

```
103 Foreach(s: G.Nodes)(f(s))
104   s.A = X(s.B);
105 Foreach(t: G.Nodes)(g(t))
106   t.B = Y(t.A)
```

becomes

```
107 Foreach(s: G.Nodes)(
108   if (f(s)) s.A = X(s.B);
109   if (g(s)) s.B = Y(s.A);
110 )
```

Loop fusion

```
139 Node_Set S(G); // ...
140 Foreach(s: S.Items)
141   s.A = x(s.B);
142 Foreach(t: G.Nodes)(g(t))
143   t.B = y(t.A)
```

becomes

```
144 Foreach(s: G.Nodes)(
145   if (S.Has(s)) s.A = x(s.B);
146   if (g(s)) s.B = y(s.A);
147 )
```
<table>
<thead>
<tr>
<th>Name</th>
<th>LOC Original</th>
<th>LOC Green-Marl</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td>350</td>
<td>24</td>
<td>[9] (C OpenMp)</td>
</tr>
<tr>
<td>Conductance</td>
<td>42</td>
<td>10</td>
<td>[9] (C OpenMp)</td>
</tr>
<tr>
<td>Vertex Cover</td>
<td>71</td>
<td>25</td>
<td>[9] (C OpenMp)</td>
</tr>
<tr>
<td>PageRank</td>
<td>58</td>
<td>15</td>
<td>[2] (C++, sequential)</td>
</tr>
<tr>
<td>SCC(Kosarakju)</td>
<td>80</td>
<td>15</td>
<td>[3] (Java, sequential)</td>
</tr>
</tbody>
</table>
Betweenness Centrality

(a) RMAT

(b) Uniform
Conductance

(a) RMAT

(b) Uniform
Vertex Cover

(a) RMAT

(b) Uniform
Page Rank

(a) RMAT

(b) Uniform
222  Foreach(s: G.Nodes)
223      For(t: s.Nbrs)
224      s.A = s.A + t.B;

becomes
225      OMP(parallel for)
226      for(index_t s = 0; s < G.numNodes(); s++) {
227         // iterate over node’s edges
228         for(index_t t = G.edge_idx[s]; t < G.edge_idx[s+1]; t++){
229            // get node from the edge
230            index_t t = G.node_idx[t];
232         }
233      }
Critique of Paper

• Like
  • Easy way to process graph
  • Increased programmer programmability
  • Architecture portability

• Dislike
  • Graph is immutable during the analysis
  • Syntax can take a bit of getting used to.
  • Only compares to SNAP