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

Sungpack Hong, Hassan Chafi, Eric Sedlar, Kunle Olukotun

Presentation by Anna Talas for R244
Motivation

- Increasing need for large-scale graph analysis
- Efficient execution remains *difficult*
  - Capacity
    - Limited physical memory
  - Performance
    - Dependant on *size*
  - Implementation
    - Requires knowledge of algorithm and hardware
Related works

- Open Multi-Processing
- SNAP library
- GraphLab
  - Future backend
Problem to solve

- Concurrent programming and optimisation is difficult
- Software developers don’t want to use new languages
  - (Rewriting code is tedious)
- Contribution from authors:
  - Green-Marl
  - Green-Marl compiler
  - Interdisciplinary DSL approach
Green-Marl

- DSL - Domain-Specific Language
- Intuitive
- Exposes inherent parallelism
  - Fork-join style
- Built-in constructs for graph analysis
- Can calculate:
  - New property
  - Subgraph selection
  - Scalar value e.g. conductance

```cpp
// Obtaining vertex cover
Procedure vertex_cover(G: Graph, VC:Edge_PROP<Bool>(G)): Int {
  Node_PROP<Int>(G) Deg;
  Node_PROP<Bool>(G) covered;
  G.covered = False;                       // Initial state
  G.Deg = G.InDegree() + G.OutDegree();   // Degree of each node
  G.VC = False;
  Int remain = G.NumEdges() * 2;

  Do {
    Int maxVal = 0;
    Node(G) from, to;
    Edge(G) e;
    Foreach(s: G.Nodes) (!G.covered) {
      Foreach(t: s.OutNbrs) {
        maxVal < from, to, e > max{ (s.Deg + t.Deg) < s, t, t.GetEdge() > }
      }  
      e.VC = True;
      from.covered = to.covered = True;
      from.Deg = to.Deg = 0;
      remain = remain - maxVal;
    }  While (remain > 0)

    Int C = Count(t: G.Nodes) (t.covered);
    Return C;
  }
}  
```
Green-Marl compiler

- Automatic optimisation and parallelisation
- Uses OpenMP
- Templates for DFS and BFS
- Can detect some conflicts
  - E.g. read-write
- Different optimisations:
  - E.g. Loop Fusion, Reduction Bound Relaxation
  - Architecture dependent optimizations

Figure 3. Overview of Green-Marl DSL-compiler Usage
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   )
Set-Graph 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  )
Data types

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

Modification under iteration → Shrink, Grow, or Copy: X
Conflicts under parallel execution →

Table 1. Operations on Collections: $S,O,$ and $Q$ denotes set, order and sequence, respectively. In the table, ‘v’, ‘X’, ‘?’ stands for the operation being valid, invalid, and undefined for the selected collection type under the selected execution context.
Evaluation

- Works as well (or better) than highly-tuned hand-coded implementations
- Parallel graph library SNAP
  - 3 different algorithms
  - 32M nodes, 256M edges
  - Small instance: 100k nodes, 800k edges

**Figure 4.** Speed-up of Betweenness Centrality. Speed-up is over the SNAP library [9] version running on a single-thread. NoFlipBE and NoSaveCh means disabling the *Flipping Edges* (Section 3.3) and *Saving BFS Children* (Section 3.5) optimizations respectively.
## Fewer lines of code

<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(Kosaraju)</td>
<td>80</td>
<td>15</td>
<td>[3] (Java, sequential)</td>
</tr>
</tbody>
</table>

### Table 3. Graph algorithms used in the experiments and their Lines-of-Code (LOC) when implemented in Green-Marl and in a general purpose language.
Limitations

- Only graphs which fit into physical memory
- The graph **must be immutable**
- **No aliases** between graph properties
- Type conversion may be needed
- So far **only C++ supported**
- Plans for future works:
  - Support alternative architectures, e.g. clusters, GPUs
  - plan to preserve comment blocks in future versions of our compiler.
  - interpreter for Green-Marl applications that will feature step-wise code execution and a visual graph representation.
Impact

**Green-Marl: a DSL for easy and efficient graph analysis**

*S Hong, H Chafi, E Sedlar, K Olukotun* - Proceedings of the seventeenth ..., 2012 - dl.acm.org

The increasing importance of graph-data based applications is fueling the need for highly efficient and parallel implementations of graph analysis software. In this paper we describe ...
Green-Marl: a DSL for easy and efficient graph analysis
S Hong, H Chafi, E Sedlar, K Olukotun - Proceedings of the seventeenth ..., 2012 - dl.acm.org
The increasing importance of graph-data based applications is fueling the need for highly efficient and parallel implementations of graph analysis software. In this paper we describe ...
Green-Marl

A DSL for efficient Graph Analysis

C++    95    35
Green-Marl

A DSL for efficient Graph Analysis

C++ 95 35
**Current situation**

- Not the fastest anymore (except Betweenness Centrality)
- Still the fewest lines of code
- GraphIt

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PR</th>
<th>BFS</th>
<th>CC</th>
<th>CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphIt</td>
<td>0.342 8.707 16.393 0.909 32.571</td>
<td>0.035 0.298 0.645 0.216 0.490</td>
<td>0.068 0.890 1.960 17.100 2.630</td>
<td>1.286 4.588</td>
</tr>
<tr>
<td>Ligra</td>
<td>1.190 49.000 68.100 1.990 201.000</td>
<td>0.027 0.336 0.915 1.041 0.677</td>
<td>0.061 2.780 5.810 25.990 13.000</td>
<td>5.350 25.500</td>
</tr>
<tr>
<td>GraphMat</td>
<td>0.560 20.400 35.000 1.190</td>
<td>0.100 2.800 4.800 1.960</td>
<td>0.365 9.8 17.9 84.5</td>
<td>5.010 21.600</td>
</tr>
<tr>
<td>Green-Marl</td>
<td>0.516 21.039 42.482 0.931</td>
<td>0.049 1.798 1.830 0.529</td>
<td>0.187 5.142 11.676 107.933</td>
<td>0.124 5.055 15.823 12.658 18.541</td>
</tr>
<tr>
<td>Galois</td>
<td>2.788 30.751 46.270 9.607 117.468</td>
<td>0.038 1.339 1.183 0.220 3.440</td>
<td>0.150 3.850 9.660 85.000 13.772</td>
<td>0.084 1.730 3.208 12.200 5.880</td>
</tr>
<tr>
<td>Gemini</td>
<td>0.430 10.980 16.440 1.100 44.600</td>
<td>0.060 0.490 0.980 10.550 0.730</td>
<td>0.102 1.550 2.500 0.650 3.750</td>
<td>0.087 1.931 3.619 2.530 6.180</td>
</tr>
<tr>
<td>Grazelle</td>
<td>0.368 15.700 20.650 0.740 54.360</td>
<td>0.052 0.348 0.828 1.788 0.512</td>
<td>0.082 3.600 6.400 29.050</td>
<td>0.149 1.358 2.299 31.055 3.849</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>SSSP</th>
<th>PRDelta</th>
<th>BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphIt</td>
<td>0.055 1.349 1.680 0.285 4.302</td>
<td>0.183 4.720 7.143 0.494 12.576</td>
<td>0.102 1.550 2.500 0.650 3.750</td>
</tr>
<tr>
<td>Ligra</td>
<td>0.051 1.554 1.895 1.301 11.933</td>
<td>0.239 9.190 19.300 0.691 40.800</td>
<td>0.087 1.931 3.619 2.530 6.180</td>
</tr>
<tr>
<td>GraphMat</td>
<td>0.095 2.200 5.000 43.000</td>
<td>0.239 9.190 19.300 0.691 40.800</td>
<td>0.087 1.931 3.619 2.530 6.180</td>
</tr>
<tr>
<td>Green-Marl</td>
<td>0.093 1.922 4.265 93.495</td>
<td>0.239 9.190 19.300 0.691 40.800</td>
<td>0.087 1.931 3.619 2.530 6.180</td>
</tr>
<tr>
<td>Galois</td>
<td>0.091 1.941 2.290 0.926 4.643</td>
<td>0.239 9.190 19.300 0.691 40.800</td>
<td>0.087 1.931 3.619 2.530 6.180</td>
</tr>
<tr>
<td>Gemini</td>
<td>0.080 1.360 2.800 7.420 6.147</td>
<td>0.239 9.190 19.300 0.691 40.800</td>
<td>0.087 1.931 3.619 2.530 6.180</td>
</tr>
</tbody>
</table>
Opinion

● The good:
  ○ Low-effort parallelisation
  ○ Intuitive to use
  ○ Fast and minimal code needed
  ○ No need to re-write whole application
  ○ Optimisations aren’t hardware specific

● The bad:
  ○ Scalability is limited
    ■ Only C++
    ■ Immutable graphs only
  ○ The optimisation isn’t novel
  ○ Still needs time to get used to
  ○ Testing is done only against SNAP
Thank you!
Bibliography
