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

```
1   Procedure Compute_BC(
2    G: Graph, BC: Node_Prop<Float>(G)) {
3     G.BC = 0;        // initialize BC
4     Foreach(s: G.Nodes) {
5      // define temporary properties
6       Node_Prop<Float>(G) Sigma;
7       Node_Prop<Float>(G) Delta;
8       s.Sigma = 1; // Initialize Sigma for root
9      // Traverse graph in BFS-order from s
10      InBFS(v: G.Nodes From s)(v!=s) {
11        // sum over BFS-parents
12        v.Sigma = Sum(w: v.UpNbrs) {w.Sigma};
13      }
14     // Traverse graph in reverse BFS-order
15      InRBFS(v!=s) {
16        // sum over BFS-children
17        v.Delta = Sum (w:v.DownNbrs) {
18           v.Sigma / w.Sigma * (1+ w.Delta)
19        };
20        v.BC += v.Delta @s; //accumulate BC
21  } } }
```

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

		sequential			parallel		
Group	Op-Name	S	O	Q	S	O	Q
Grow	Add	v			v		
	Push(Front/Back)		v	v		v	v
Shrink	Remove	v			v		
	Pop(Front/Back)		v	v		?	?
	Clear	v	v	v	v	v	v
Lookup	Has	v	v	v	v	v	v
	Front(Back)		v	v		v	v
	Size	v	v	v	v	v	v
Copy	=	v	v	v	X	X	X
Iteration	Items	v	v	v	v	v	v

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

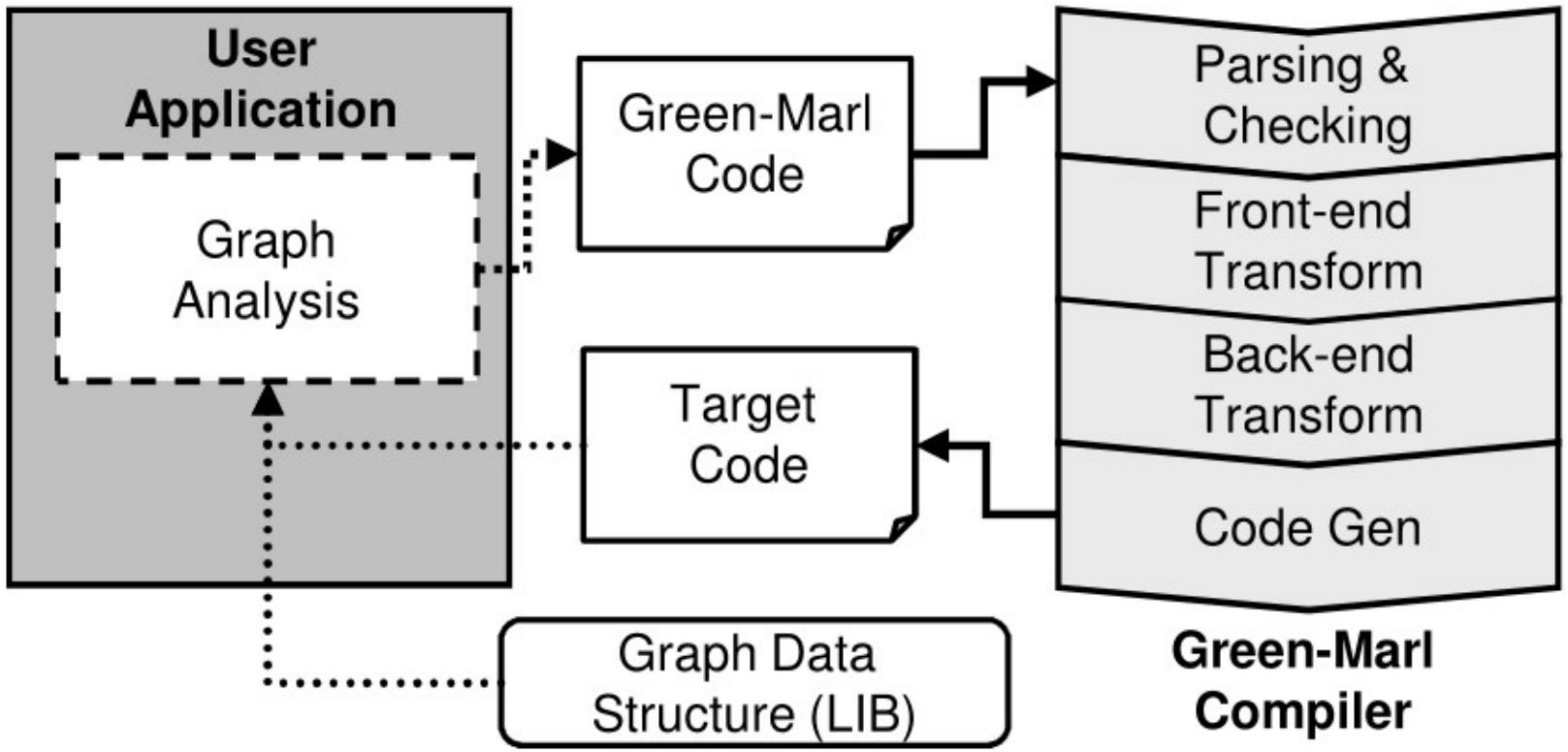
```
52  Foreach(s:G.Nodes) {
53      // no conflict. t.X gives 'old' value
54      s.X <= Sum(t:s.Nbrs) {t.X} @ s
55  }
56  // All the writes to X becomes visible simultaneously
57  // at the end of the s iteration.
```

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;
```

In-place	Assignment	In-place	Assignment
All	&&=	Sum	+=
Any	||=	Product	*=
Min	min=	Count	++
Max	max=		

Overview of Green-Marl DSL-compiler Usage



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)
```

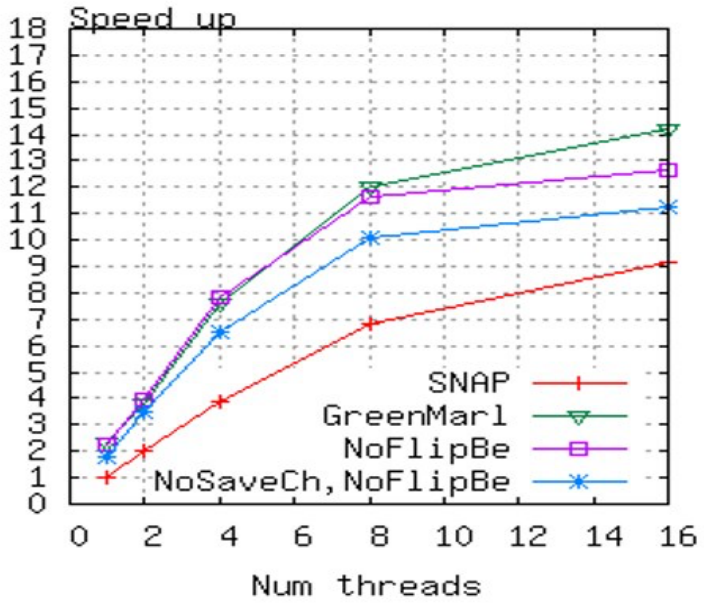
becomes

```
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   }
```

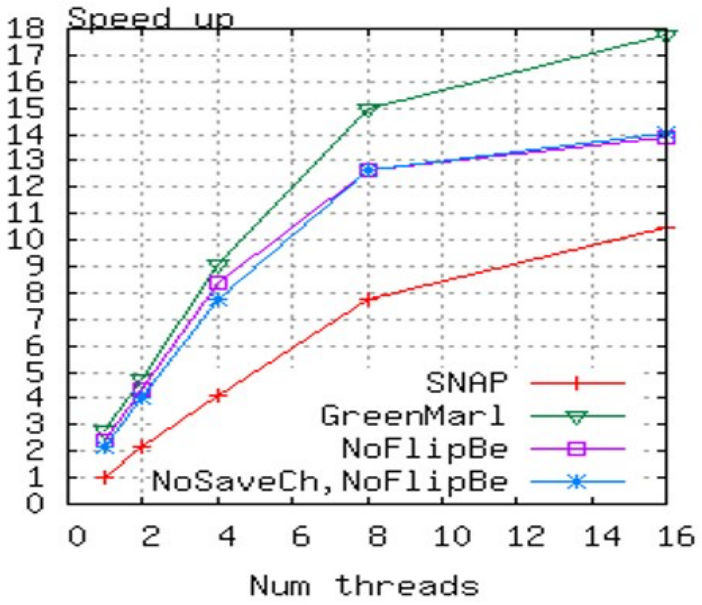
Name	LOC Original	LOC Green-Marl	Source
BC	350	24	[9] (C OpenMp)
Conductance	42	10	[9] (C OpenMp)
Vetex Cover	71	25	[9] (C OpenMp)
PageRank	58	15	[2] (C++, sequential)
SCC(Kosaraju)	80	15	[3] (Java, sequential)

Lines of code

Betweenness Centrality

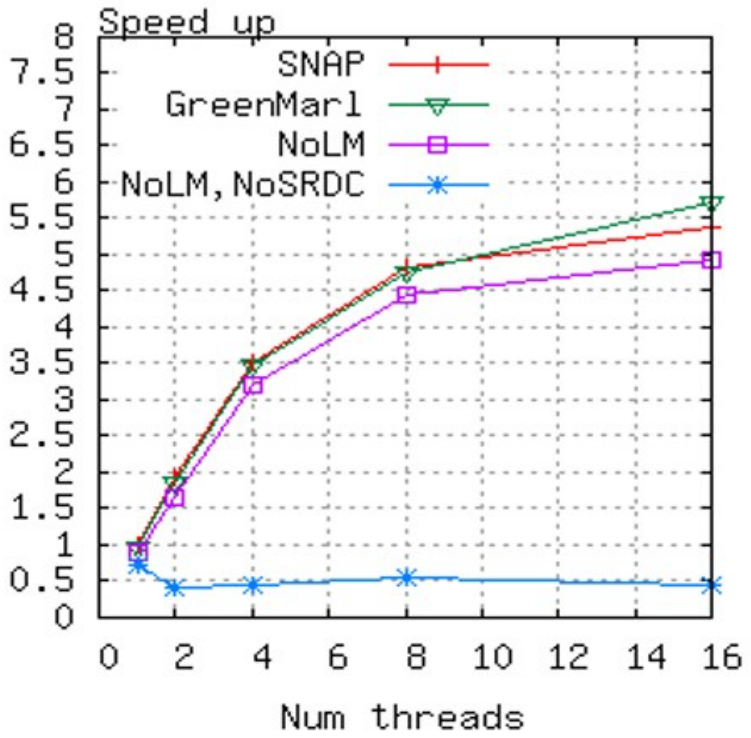


(a) RMAT

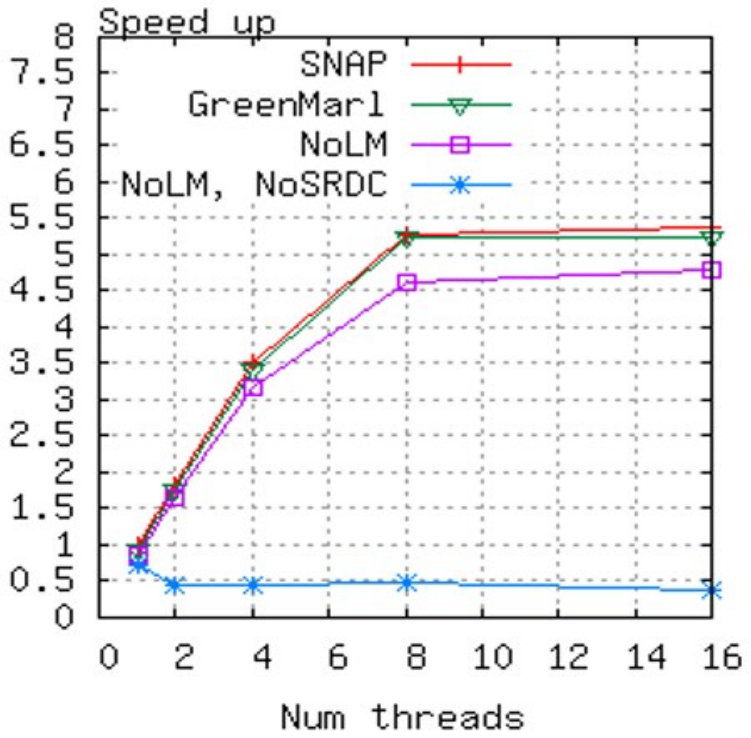


(b) Uniform

Conductance

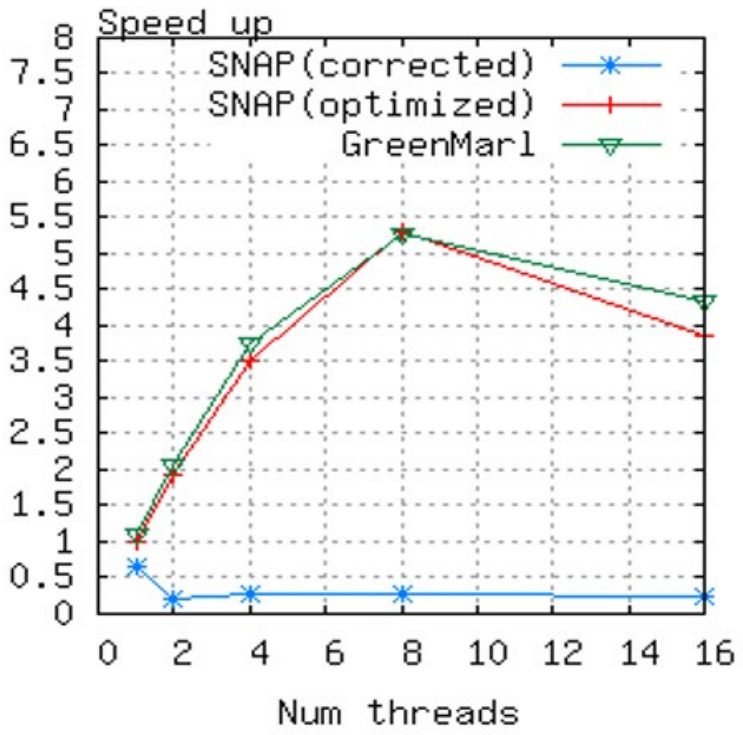


(a) RMAT

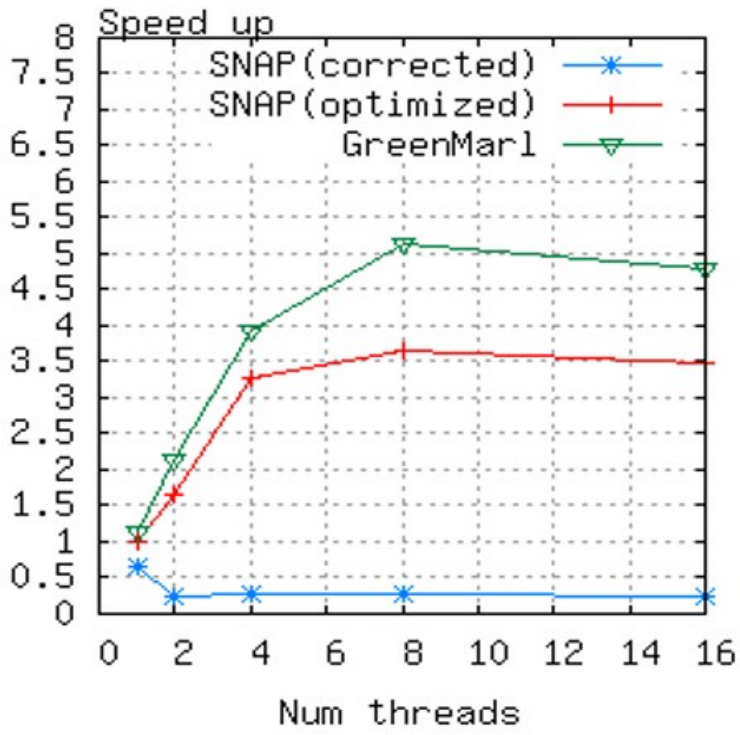


(b) Uniform

Vertex Cover

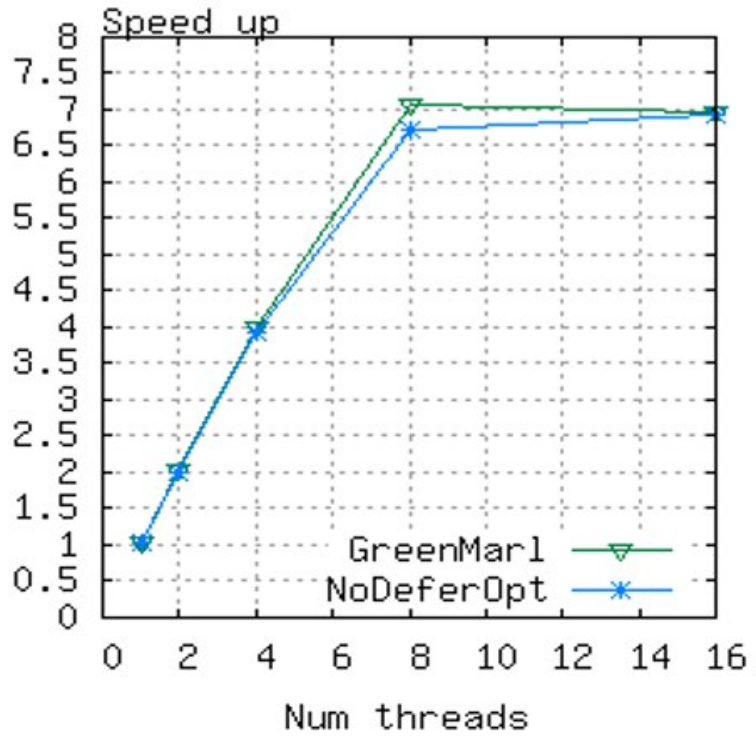


(a) RMAT

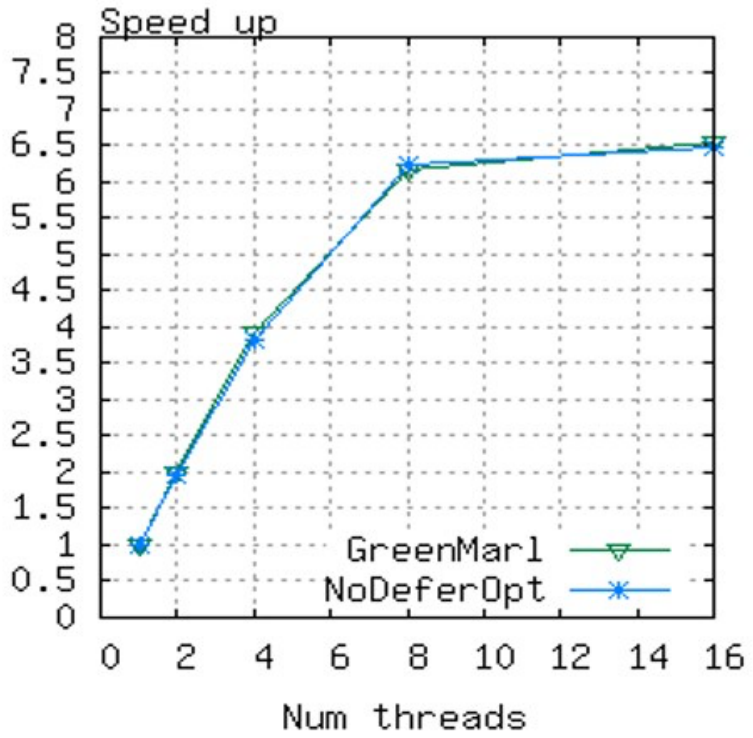


(b) Uniform

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];
231       A[s] = A[s] + B[t];
232   } }
```

Code Generation

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