GREEN-MARL: A DSL FOR EASY AND EFFICIENT GRAPH ANALYSIS

Sungpack Hong, Hassan Chafi, Eric Sedlar, Kunle Olukotun

K.M.D.M Karunarathna

University Of Cambridge - 17th Nov 2015

Current Issues

Issues with large-scale graph analysis

- Performance
- Implementation
- Capacity

Performance Issues

- RAM latency dominates running time for large graphs

Solution: Solved by exploiting data parallelism

Implementation Issues

- Writing concurrent code is *hard*
- Race-conditions
- Deadlock
- Efficiency requires deep hardware knowledge
- Couples code to underlying architecture

Solution: A DSL Green-Marl and its compiler

- High level graph analysis language
- Hides underlying complexity
- Exposes algorithmic concurrency
- Exploits high level domain information for optimisations

Example

```
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
} } }
```

Green-Marl Language Design

- Scope of the Language

  Based on processing graph properties, mappings from a node/edge to a value

  - e.g. the average number of phone calls between two people

- Green-Marl is designed to compute,
  - *scalar values from a graph and its properties*
  - *new properties for nodes/edges*
  - *selecting subgraphs (instance of above)*

Green-Marl Language Design

- Parallelism in Green-marl

  Support for parallelism (fork-join style)

  - *Implicit*

    ```
    G.BC = 0;
    ```

  - *Explicit*

    ```
    Foreach(s: G.Nodes) (s!=t)
    ```

  - *Nested*

    ```
    p_sum *= t.B;
    ```

Language Constructs

- Data Types and Collections - *DATA*

  a) Five primitive types (**`Bool, Int, Long, Float, Double`**)

  b) Defines two graph types (**`DGraph`** and **`UGRaph`**)

  c) Second, there is a node type and an edge type both of which are always bound to a graph instance

  d) e node properties and edge properties which are bound to a graph but have base-types as well

Language Constructs

- Data Types and Collections - ***COLLECTION***

: Set, Order, and Sequence.

a) Elements in a Set are unique while a Set is unordered.

b) Elements in an Order are unique while an Order is ordered.

c) Elements in a Sequence are not unique while a Sequence is ordered

Language Constructs

- Iterations and Traversals

```
Foreach (iterator:source(-).range)(filter)
body_statement
```

Language Constructs

- Deferred Assignment
  a) Supports bulk synchronous consistency via deferred assignments.
  b) Deferred assignments are denoted by **<=** and followed by a binding symbol

Language Constructs

Reductions

- an expression form (or in-place from)
- an assignment form

```
y+= t.A;
```

Compiler

- Compiler Overview

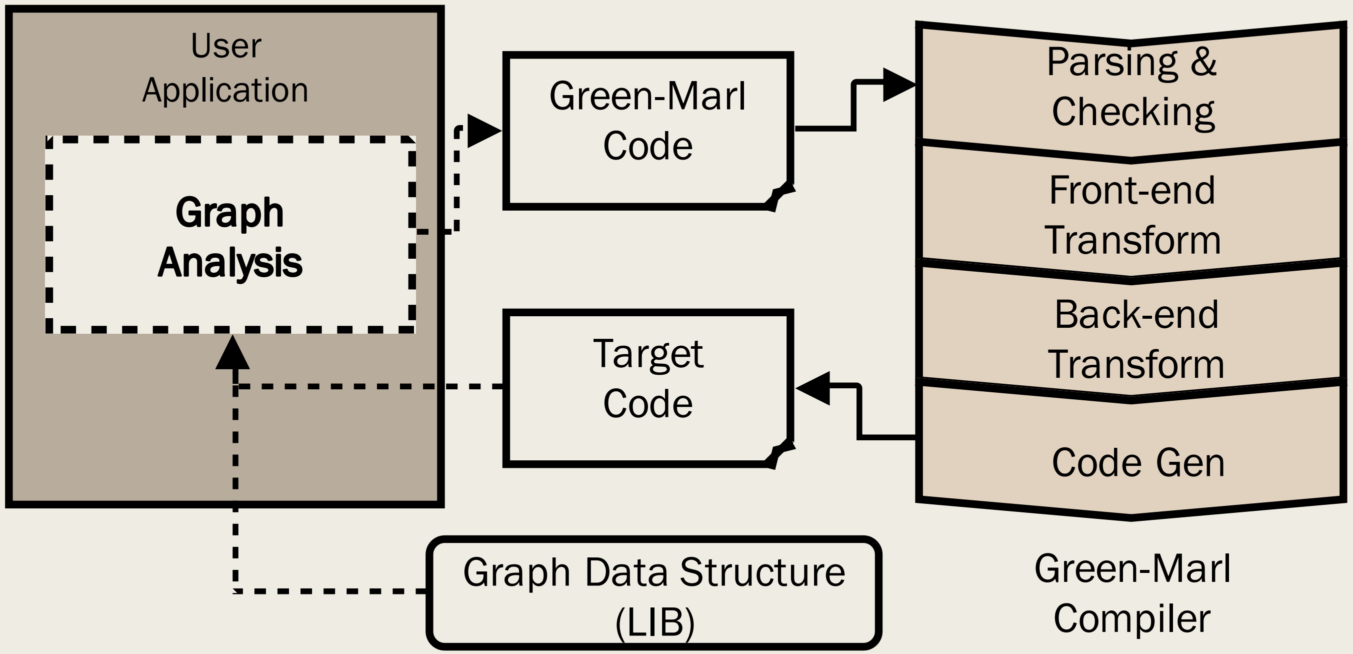

Figure. *Overview of Green-Marl DSK-compiler Usage*

Compiler

- Architecture Independent Optimizations
  - *Group Assignment*
  - *In-place Reduction*
  - *Loop Fusion*
  - *Hoisting Definitions*
  - *Reduction Bound Relaxation*
  - *Flipping Edges*

```
Foreach(t:G.Nodes)(f(t))
        Foreach(s:t.InNbrs)(g(s))
            t.A += s.B;
```

Becomes

```
Foreach(s:G.Nodes)(g(s))
        Foreach(t:s.OutNbrs)(f(t))
            t.A += s.B;
```

Compiler

- Architecture Dependent Optimizations
  - *Set-Graph Loop Fusion*
  - *Selection of Parallel Regions*
  - *Deferred Assignment*
  - *Saving BFS Children*

```
InBFS(v:G.Nodes; s) { ... //forward }
  InRBFS { // reverse-order traverse
   Foreach(t: v.DownNbrs) {
      DO_THING(t);
} }
```

Becomes

```
_prepare_edge_marker(); // O(E) array
    for (e = edges ... ) {
      index_t t = ...node(e);
        if (isNextLevel(t)) {
           edge_marker[e] = 1;
} }
```

→

```
for (e = edges ..) {
    if (edge_marker[e] ==1) {
       index_t  t = ...node(e);
       DO_THING(t);
} }}
```

Compiler

- Code Generation
  - *Graph and Neighborhood Iteration*
  - *Efficient DFS and BFS traversals*
  - *Small BFS Instance Optimization*
  - *Reduction on Properties*
  - *Reduction on Scalars*

Experiments

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)

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

Experiments

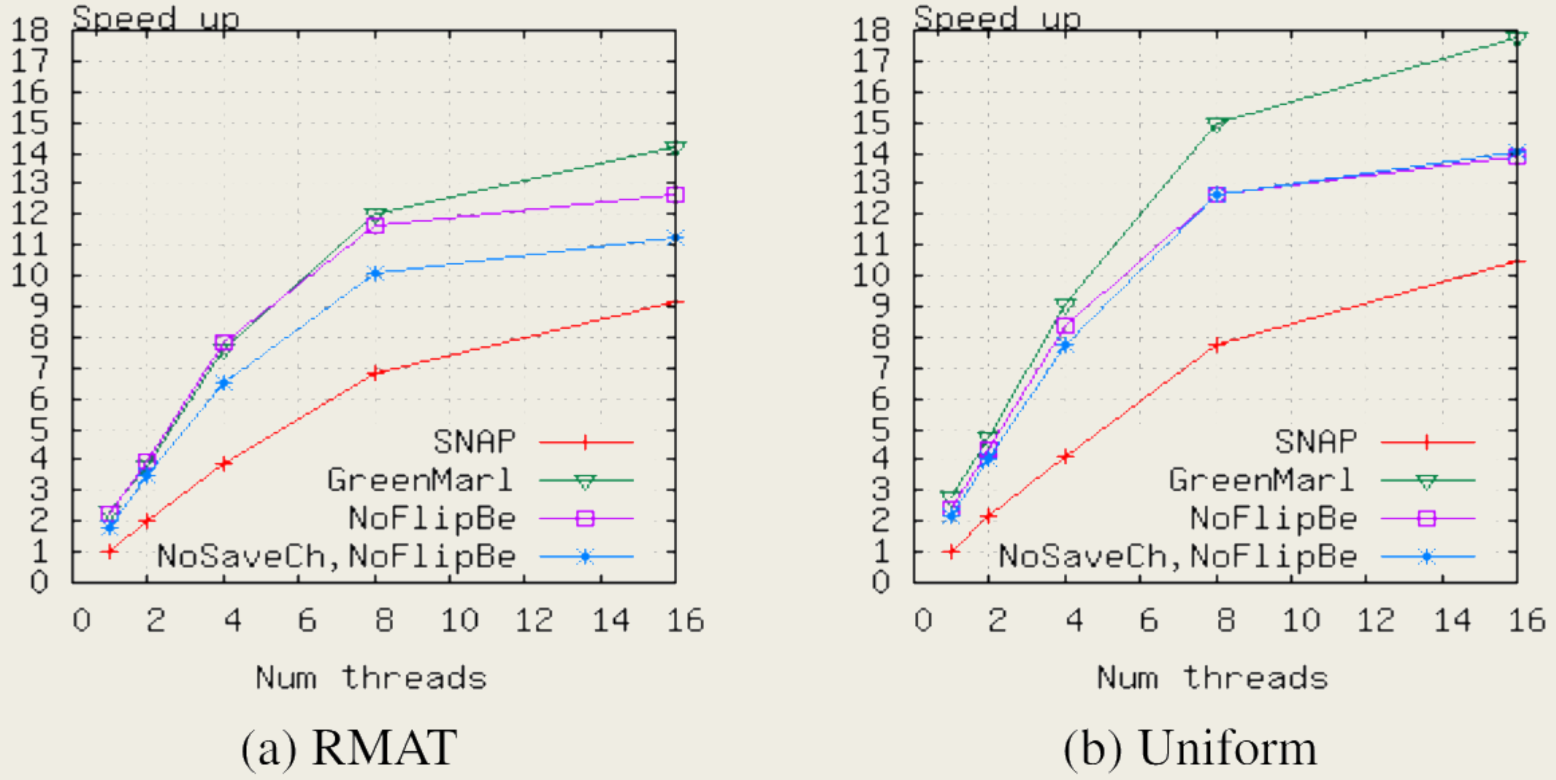

Figure. Speed-up of Betweenness Centrality. Speed-up is over the SNAP library [9] version running on a single-thread. NoFlipBE and NoSaveCh means disabling the Flipping Edges (Section 3.3 Architecture Independent Optimizations) and Saving BFS Children (Section 3.5 Code Generation) optimizations respectively.

Experiments

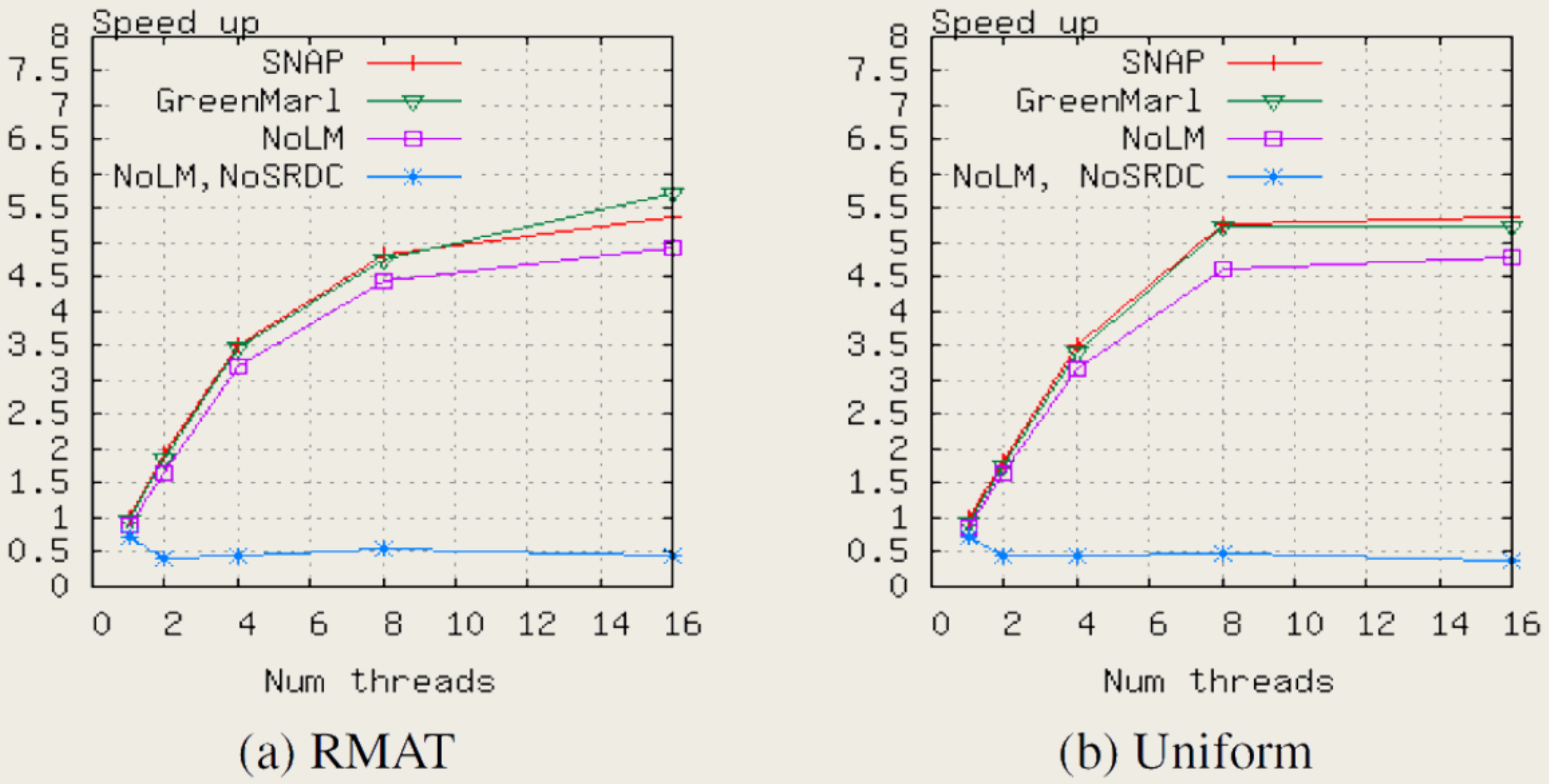

Figure. Speed-up of Conductance. Speed-up is over the SNAPlibrary [9] version running on a single-thread. NoLM and NoSRDCmeans disabling theLoop Fusion(Section 3.3 *Architecture Independent Optimizations*) andReduction onScalars(Section 3.5 *Code Generation*) optimizations, respectively.

Future Works

- Solutions for Capacity Issue
- Comments block to green Marl
- Combining with Graph Lab as back end.(machine learning type)
- generate code for alternative architectures(Clusters, GPU).
- Green Marl as internal DSL.

Pros

- *Easier to write graph algorithms*
- *Algorithms perform better*
- *Don’t need to rewrite entire application*
- *Code is portable across platforms*

Critical Evaluation

- *Assumes graph is immutable during the analysis*

Thank you…