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

Hong, Chafi, Sedlar and Olukotun

Reviewed by Neil Satra (ns532)

OpenMP implementation

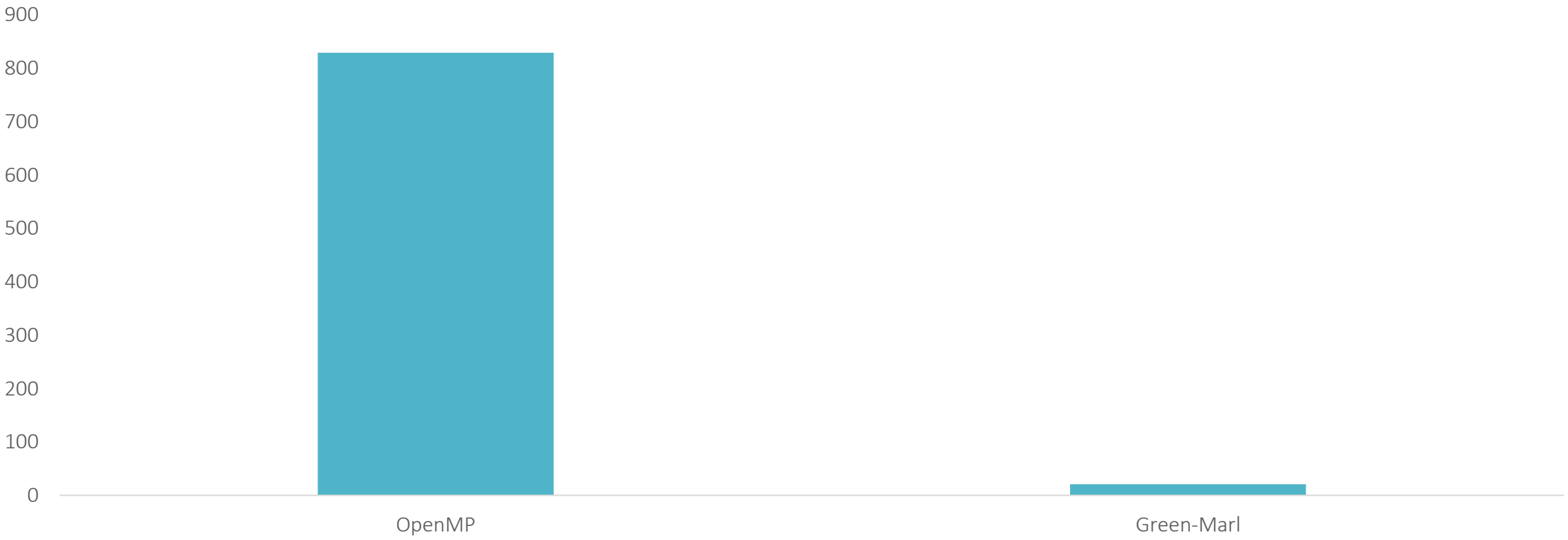
vertex_betweenness_centrality.c 829 lines (701 with data), 21.5 kB

```
1   #include "graph_defs.h"
2   #include "graph_metrics.h"
3   #include "utils.h"
4   #include "sprng.h"
5
6   void vertex_betweenness_centrality_parBFS(graph_t* G, double* BC, long numSrcs) {
7
8       attr_id_t *S;      /* stack of vertices in the order of non-decreasing
9                             distance from s. Also used to implicitly
10                            represent the BFS queue */
11      plist_t* P;        /* predecessors of a vertex v on shortest paths from s */
12      double* sig;       /* No. of shortest paths */
13      attr_id_t* d;      /* Length of the shortest path between every pair */
14      double* del;       /* dependency of vertices */
15      attr_id_t *in_degree, *numEdges, *pSums;
16      attr_id_t* pListMem;
17  #if RANDSRCS
18      attr_id_t* Srcs;
19  #endif
20      attr_id_t *start, *end;
21      long MAX_NUM_PHASES;
```

Green-Marl implementation

```
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 } } }
```

Green-Marl needs way fewer Lines of Code



Green-Marl is a Domain Specific Language

For	Graph analysis algorithms
With	Intuitive high-level constructs
Which	Expose data-level parallelism inherent in the algorithm

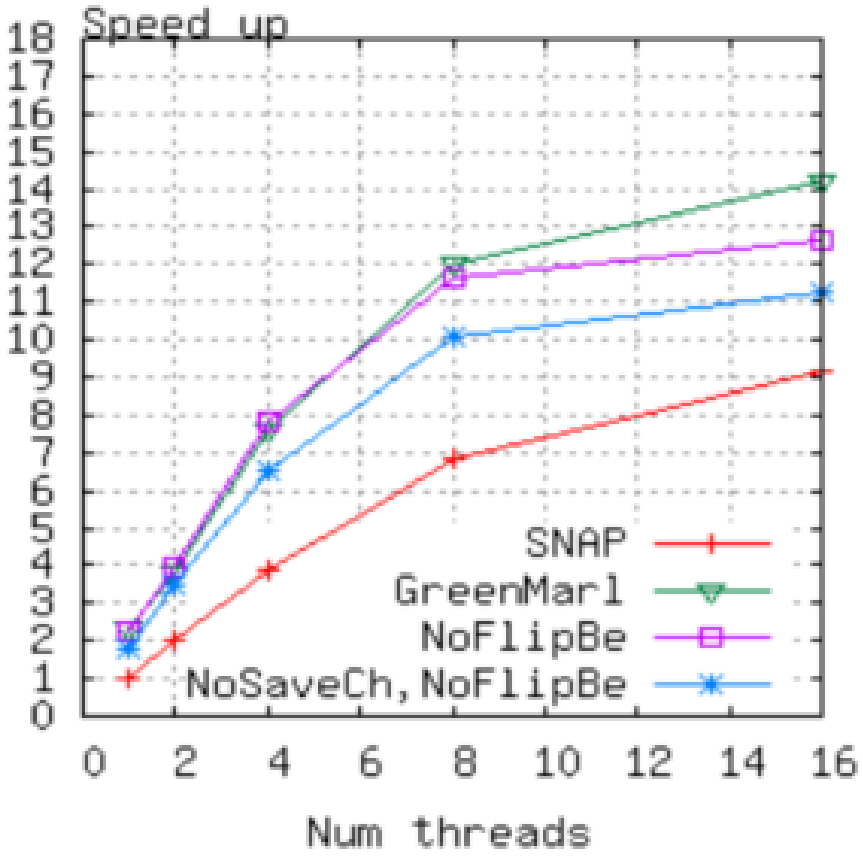
High level constructs

- Graphs, nodes, edges
- Neighbours (in, out, up and down)
- Breadth-First and Depth-first search

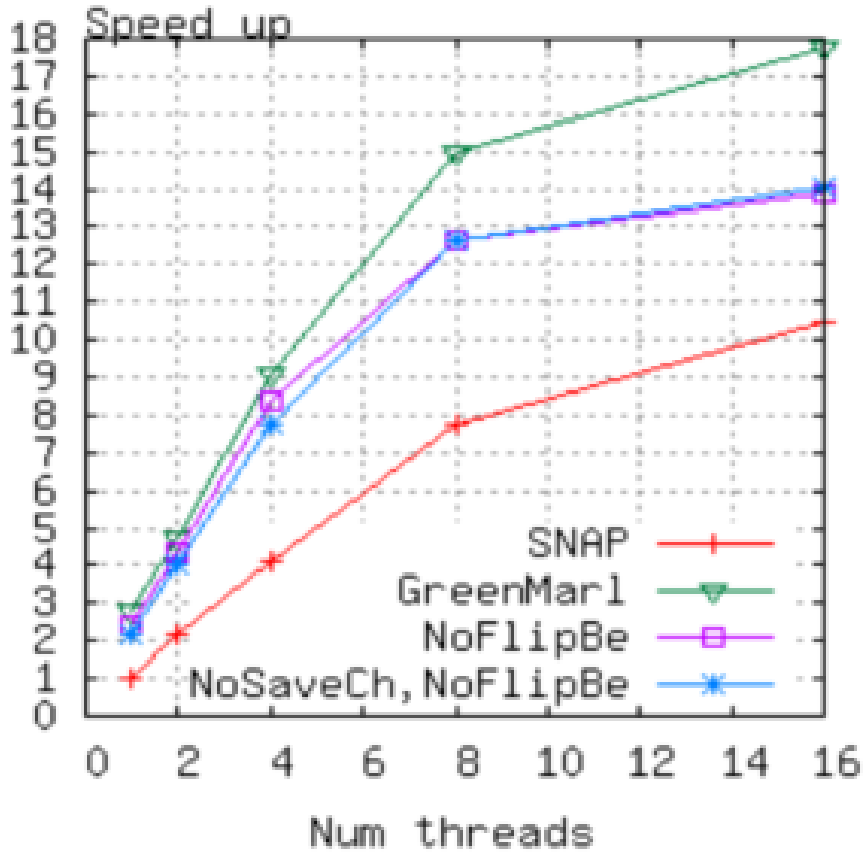
In goes Green-Marl code

Out comes C++/OpenMP code

Objection: Performance



(a) RMAT



(b) Uniform

Objection: New Language

- Can interleave with C++ code
- Tutorial on Github
- Detailed language specs available online

Objection: Adoption

- Production ready – actively maintained on Github
- Built-in support for Giraph (in sequel to this paper)

In goes Green-Marl code

Out comes C++/OpenMP code

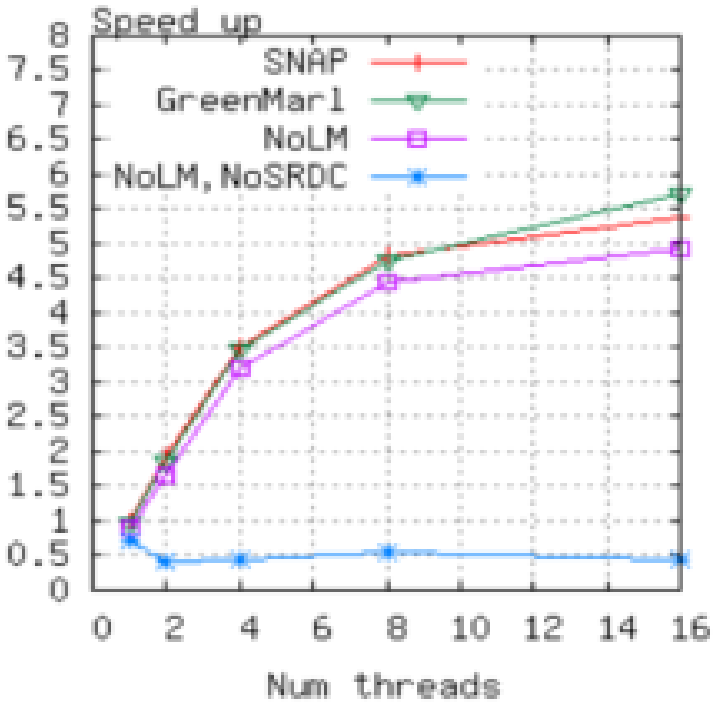
Objection: Adoption

- Production ready – actively maintained on Github
- Built-in support for Giraph (in sequel to this paper)
- Oracle adoption in their graph analytic framework, Oracle PGX
- No lock in

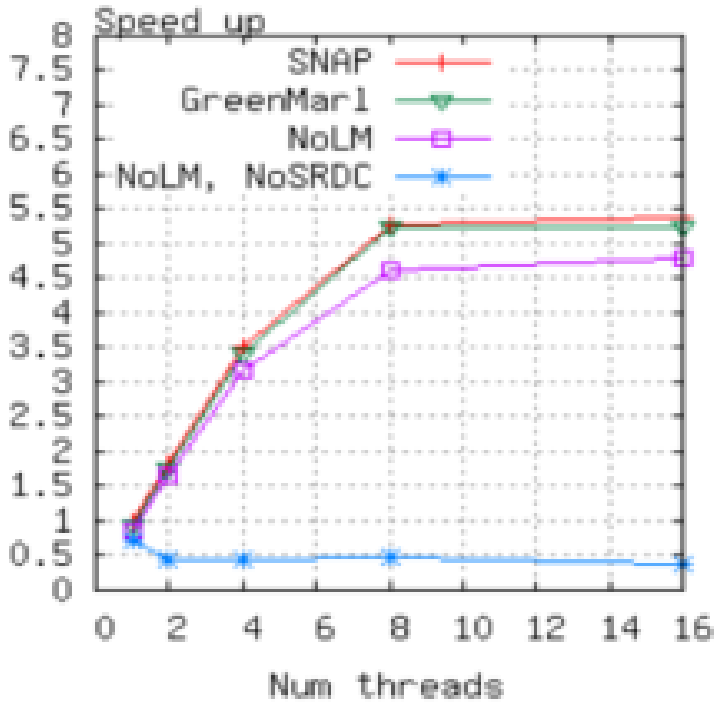
Advantages

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

Well Evaluated



(a) RMAT



(b) Uniform

- Tested on Random and Power-law graphs
- Individual optimisations tested

Weakness

Graph is immutable during the analysis

Summary

- Write graph analysis portion of software in Green-Marl
- Get human-readable output in target language
- With automatic optimisations

Reviewed by Neil Satra
ns532@cam.ac.uk