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

```
//------------------------------------------------------------------
// 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;                                            // In
  G.Deg = G.InDegree() + G.OutDegree();                         // De
  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) {                           // Cl
      Foreach(t: s.OutNbrs)                                     //
        maxVal <from, to, e> max= (s.Deg + t.Deg) <s, t, t.GetEdge()>
    }
    e.VC = True;                                                // Se
    from.covered = to.covered = True;                           // No
    from.Deg = to.Deg = 0;
    remain = remain - maxVal;
  } While (remain > 0)                                          // Fi

  Int C = Count(t:G.Nodes)(t.covered);                          // Co
  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

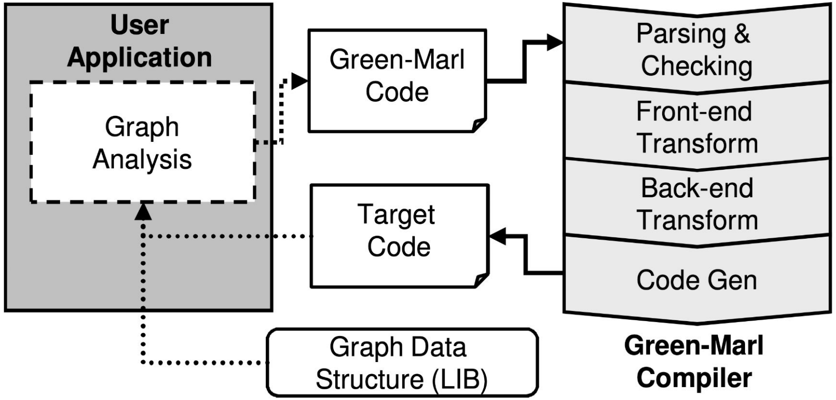


Figure 3. Overview of Green-Marl DSL-compiler Usage

Loop Fusion

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

becomes

```
Foreach(s: G.Nodes)(
    if (f(s)) s.A = X(s.B);
    if (g(s)) s.B = Y(s.A);
}
```

Set-Graph Loop Fusion

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

becomes

```
Foreach(s: G.Nodes)(
    if (S.Has(s)) s.A = x(s.B);
    if (g(s)) s.B = y(s.A);
}
```

Data types

Group	Op-Name	sequential S	sequential O	sequential Q	parallel S	parallel O	parallel 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 $\rightarrow$ Shrink, Grow, or Copy: X

Conflicts under parallel execution $\rightarrow$

Grow-Shrink: X Lookup-Shrink: ? Lookup-Grow: ?

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

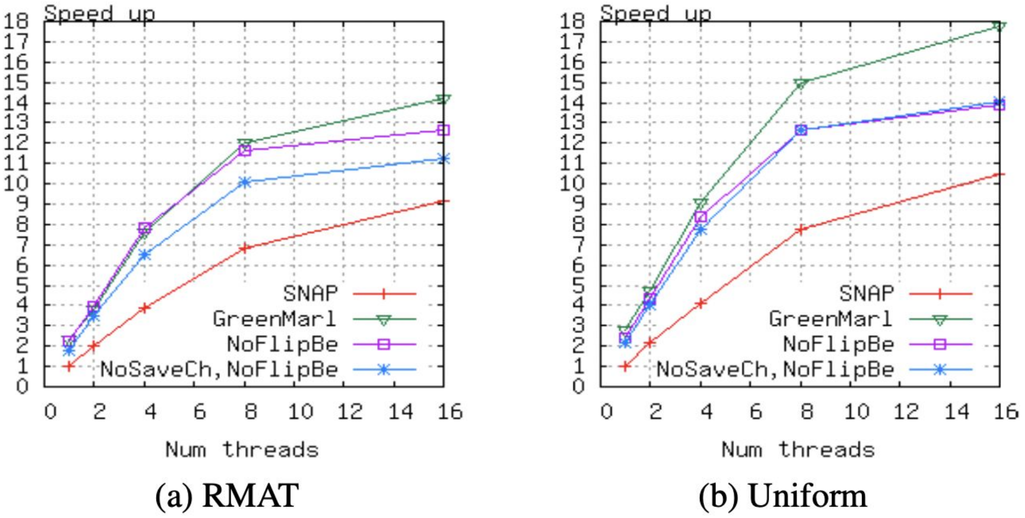


Figure 4. Speed-up of Betweenness Centrality. Speed-up is over the SNAP library [9] version running on a single-thread. NoFlipBE and NoSaveCh means disabling the *Flipping Edges* (Section 3.3) and *Saving BFS Children* (Section 3.5) optimizations respectively.

Fewer lines of code

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 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 …

☆ Save 99 Cite Cited by 385 Related articles Web of Science: 5

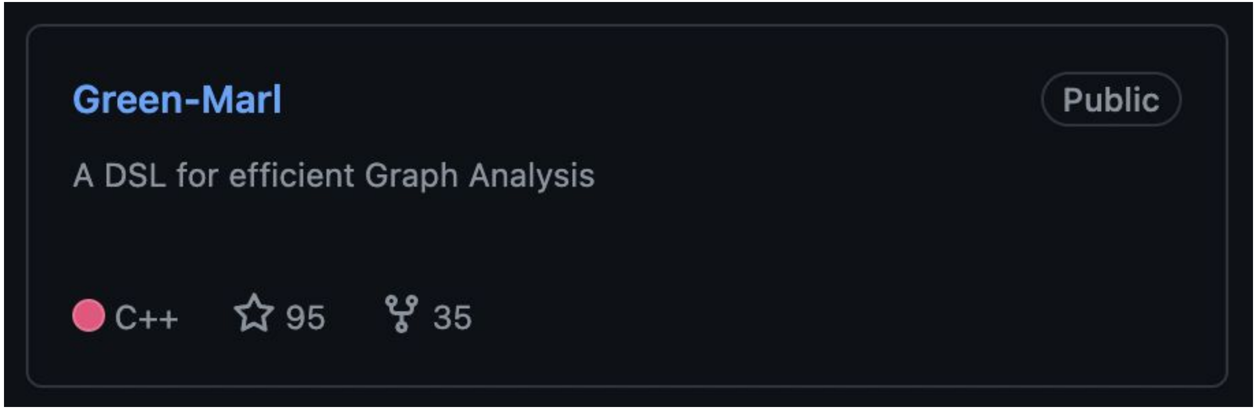
Impact

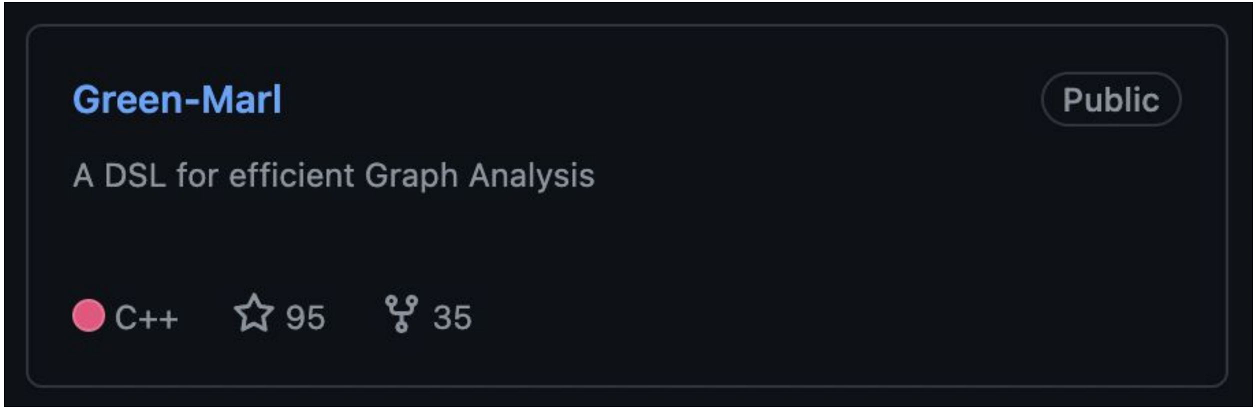
Green-Marl: a DSL for easy and efficient graph analysis

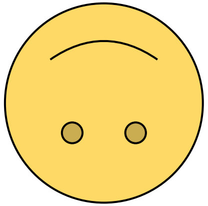
S Hong, H Chafi, E Sedlar, K Olukotun - Proceedings of the seventeenth …, 2012 - dl.acm.org

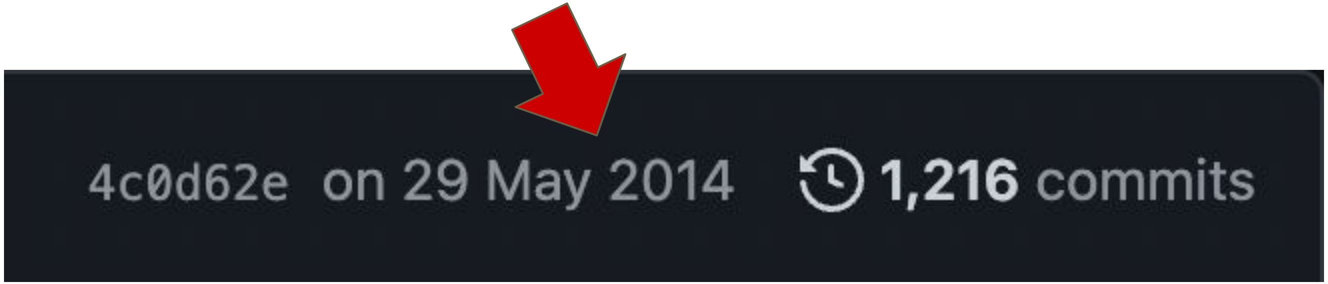
The increasing importance of graph-data based applications is fueling the need for highly efficient and parallel implementations of graph analysis software. In this paper we describe …

☆ Save 99 Cite Cited by 385 Related articles Web of Science: 5


Green-Marl
Public
A DSL for efficient Graph Analysis
C++
95
35


Green-Marl
Public
A DSL for efficient Graph Analysis
C++
95
35




4c0d62e on 29 May 2014
1,216 commits

Current situation

	GraphIt	Ligra	GraphMat	Green-Marl	Galois	Gemini	Grazelle
PR	34	74	140	20	114	127	388
BFS	22	30	137	1	58	110	471
CC	22	44	90	25	94	109	659
SSSP	25	60	124	30	88	104	

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

Algorithm	PR					BFS					CC					CF	
Graph	LJ	TW	WB	RD	FT	LJ	TW	WB	RD	FT	LJ	TW	WB	RD	FT	NX	NX2
GraphIt	**0.342**	**8.707**	**16.393**	0.909	**32.571**	0.035	**0.298**	**0.645**	**0.216**	**0.490**	0.068	**0.890**	**1.960**	17.100	**2.630**	**1.286**	**4.588**
Ligra	1.190	49.000	68.100	1.990	201.000	**0.027**	0.336	0.915	1.041	0.677	**0.061**	2.780	5.810	25.900	13.000	5.350	25.500
GraphMat	0.560	20.400	35.000	1.190		0.100	2.800	4.800	1.960		0.365	9.8	17.9	84.5		5.010	21.600
Green-Marl	0.516	21.039	42.482	0.931		0.049	1.798	1.830	0.529		0.187	5.142	11.676	107.933			
Galois	2.788	30.751	46.270	9.607	117.468	0.038	1.339	1.183	0.220	3.440	0.125	5.055	15.823	12.658	18.541		
Gemini	0.430	10.980	16.440	1.100	44.600	0.060	0.490	0.980	10.550	0.730	0.150	3.850	9.660	85.000	13.772		
Grazelle	0.368	15.700	20.650	**0.740**	54.360	0.052	0.348	0.828	1.788	0.512	0.084	1.730	3.208	**12.200**	5.880		

Algorithm	SSSP					PRDelta					BC				
Graph	LJ	TW	WB	RD	FT	LJ	TW	WB	RD	FT	LJ	TW	WB	RD	FT
GraphIt	0.055	**1.349**	**1.680**	**0.285**	**4.302**	**0.183**	**4.720**	**7.143**	**0.494**	**12.576**	0.102	1.550	2.500	**0.650**	**3.750**
Ligra	**0.051**	1.554	1.895	1.301	11.933	0.239	9.190	19.300	0.691	40.800	0.087	1.931	3.619	2.530	6.160
GraphMat	0.095	2.200	5.000	43.000											
Green-Marl	0.093	1.922	4.265	93.495							**0.082**	3.600	6.400	29.050	
Galois	0.091	1.941	2.290	0.926	4.643						0.237	3.398	4.289	0.806	9.897
Gemini	0.080	1.360	2.800	7.420	6.147						0.149	**1.358**	**2.299**	31.055	3.849

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

- S. Hong, H. Chafi, E. Sedlar, K.Olukotun: Green-Marl: A DSL for Easy and Efficient Graph Analysis, ASPLOS, 2012

.

- Yunming Zhang, Mengjiao Yang, Riyadh Baghdadi, Shoaib Kamil, Julian Shun, and Saman Amarasinghe. 2018. GraphIt: a high-performance graph DSL. Proc. ACM Program. Lang. 2, OOPSLA, Article 121 (November 2018), 30 pages.