Gunrock: A High-Performance Graph Processing Library on the GPU

Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, John D. Owens

University of California, Davis

Overview

Gunrock is a stable, powerful, forward-looking, open-source substrate for GPU-based graph-centric research and development. Gunrock offers:

- the best performance on GPU graph analytics;
- a high-level abstraction for graph algorithms on the GPU; and
- the widest range of primitives.

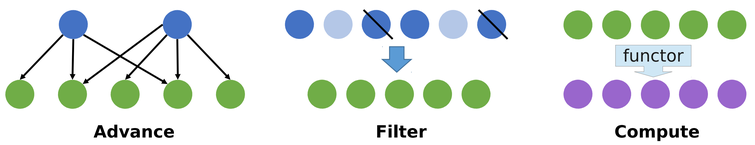
What is Gunrock's Data-centric Programming Model?

A **frontier** is a compact queue of nodes or edges. Gunrock's three operators (below) manipulate frontiers.

advance generate a new frontier from the edges or vertices of the current frontier

filter generate a new frontier from a current frontier using a user-specified predicate

compute run a user-specified computation in parallel on each element in the current frontier



How does Gunrock express graph algorithms?

```
### import libraries
from ctypes import *
gunrock = cdll.LoadLibrary('../../build/lib/libgunrock.so')

### read in input CSR arrays from files
row_list = [int(x.strip())
for x in open('path/to/rowoffsets/r_file')]
col_list = [int(x.strip())
for x in open('path/to/columnindices/c_file')]

### convert CSR graph inputs for gunrock input
row = pointer((c_int * len(row_list))(*row_list))
col = pointer((c_int * len(col_list))(*col_list))
nodes = len(row_list) - 1
edges = len(col_list)

### output array
scores = pointer((c_float * nodes)())

### call gunrock function on device
gunrock.bc(scores, nodes, edges, row, col, 0)

### sample results
print 'node_bc_scores:',
for idx in range(nodes): print scores[0][idx],
```

(a) Compute BC in Python.

```
BCProblem::Init(); // Initialization
// Accumulate sigma values
while (frontier_queue_length > 0) {
forward_queue_offsets.push(new_offsets);
  // Get neighbors and update scores
  BCEnactor::gunrock::oprtr::
  advance<BCProblem, ForwardEnactor>();
  // Cenerate new vertex frontier
  BCEnactor::gunrock::oprtr::
  filter<BCProblem, ForwardEnactor>();
}

// Compute delta values
while (!forward_queue_offsets.empty()) {
  // Compute delta values
  BCEnactor::gunrock::oprtr::
  advance<BCProblem, BackwardEnactor>();
}

BFSProblem::Extract(); // Get result
```

(b) Develop BC using Gunrock.

Figure: Code snapshot of working with Gunrock and using Gunrock.

Primitives in Gunrock:

traversal-based: breadth-first search, single-source shortest path;

node-ranking: HITS, SALSA, PageRank, betweenness centrality;

global: connected component, minimum spanning tree.

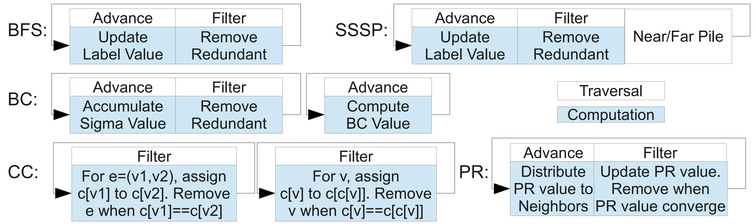
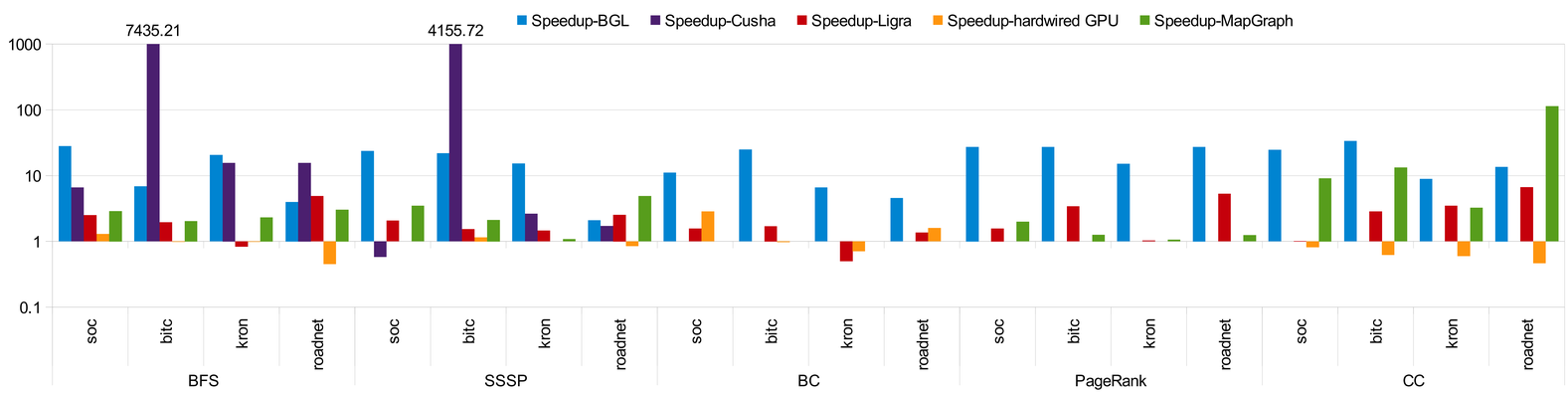
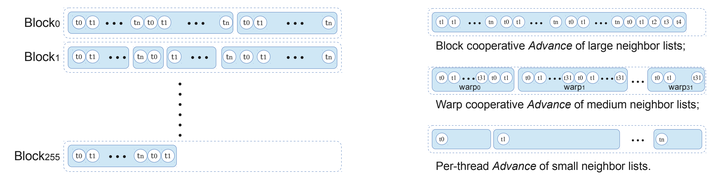


Figure: Several graph primitives in Gunrock.



Why is Gunrock fast?

Powerful load-balancing capabilities that effectively address the inherent irregularity in graphs:



(a) Load-balanced traversal. (b) Dynamic-grouped traversal.

Figure: Two core load-balancing strategies in Gunrock.

Future Work

- Scale to multiple GPUs/nodes;
- Asynchronous model;
- Out-of-core and streaming support;
- Expand core operators and new primitives;
- In-depth performance characterization.

Funding Agencies

DARPA XDATA W911QX-12-C-0059, STTR D14PC00023; NSF OCI-1032859, CCF-1017399.

Contact Information

- Gunrock Website: http://gunrock.github.io/
- Author's Email: yzhwang@ucdavis.edu

UC DAVIS COLLEGE OF ENGINEERING