Gunrock: GPU Graph Analytics

Yangzihao Wang, Yuechao Pan, Andrew Davidson, Yuduo Wu, Carl Yang, Leyuan Wang, Muhammad Osama, Chenshan Yuan, Weitang Liu, Andy Riffel, John D. Owens (PI)

University of California, Davis

Overview

Gunrock is a CUDA library for graph-processing designed specifically for the GPU. Gunrock offers:

- a high-level, bulk-synchronous, data-centric programming model with
- best-in-class and scalable performance on single-GPU and multi-GPU-single-node configurations.

What is Gunrock's Data-centric Programming Model?

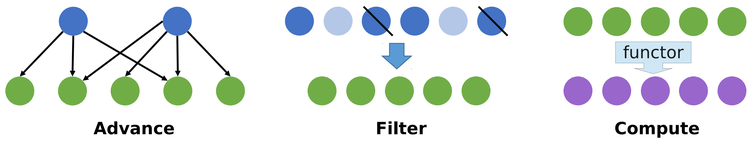
A **frontier** is a compact set of vertices or edges. Gunrock is built around **operators** that manipulate frontiers, including:

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

segmented-intersection: input two input frontiers; output each element that is in both input frontiers



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 '\nnode\tbc\tscores:',
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) Call Gunrock's BC in Python.

Figure: Code snapshots of programming a Gunrock primitive and using Gunrock.

Current primitives in Gunrock:

traversal-based: (direction-optimized) breadth-first search, single-source shortest path;

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

global: connected component, minimum spanning tree, triangle counting, subgraph matching.

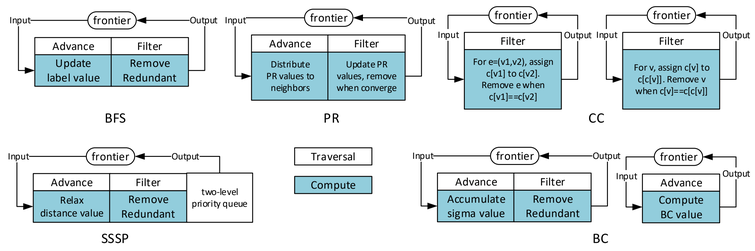


Figure: Operation flow chart for selected primitives in Gunrock (a black line with an arrow at one end indicates a while loop that runs until the frontier is empty).

Multi-GPU Framework

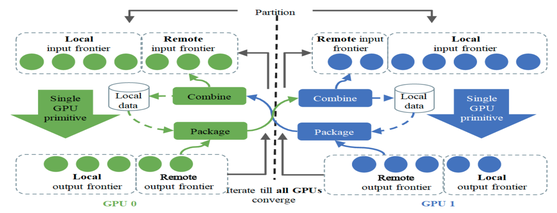


Figure: Single-node multiple-GPU communication flow.

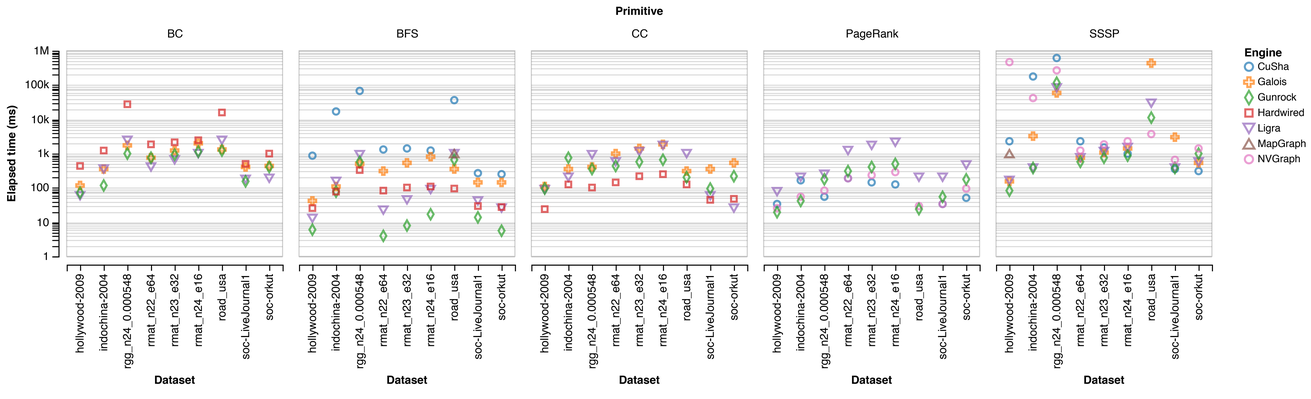
Optimization Strategies

- Static/dynamic workload mapping and load-balancing strategy;
- Enable idempotent operations;
- Pull vs. push traversal;
- Priority Queue, reorganizing frontier items into "near" and "far" slices;
- Kernel fusion;
- Efficient and effective coalesced-memory access.

Future Work

- Enhanced scalability (scale-up, scale-out);
- Support dynamic/streaming graphs;
- Expand core operators and add new primitives;
- Support higher-level programming models.

Gunrock Performance (lower is better)



Funding Agencies

DARPA XDATA W911QX-12-C-0059, STTR D14PC00023; NSF OCI-1032859, CCF-1017399, CCF-1629657.

Contact Information

- Gunrock Website: http://gunrock.github.io/
- Contact: John Owens, jowens@ece.ucdavis.edu

UCDAVIS COLLEGE OF ENGINEERING