Gunrock: A Fast and Programmable Multi-GPU Graph Processing Library

Yangzihao Wang and Yuechao Pan with Andrew Davidson, Yuduo Wu, Carl Yang, Leyuan Wang, Andy Riffel and John D. Owens
University of California, Davis
{yzhwang, ychpan}@ucdavis.edu
Why use GPUs for Graph Processing?

**Graphs**
- Found everywhere
  - Road & social networks, web, etc.
- Require fast processing
  - Memory bandwidth, computing power and GOOD software
- Becoming very large
  - Billions of edges
- Irregular data access pattern and control flow
  - Limits performance and scalability

**GPUs**
- Found everywhere
  - Data center, desktops, mobiles, etc.
- Very powerful
  - High memory bandwidth (288 GBps) and computing power (4.3 Tflops)
- Limited memory size
  - 12 GB per NVIDIA K40
- Hard to program
  - Harder to optimize

**Scalability**

**Performance**

**Programmability**
Current Graph Processing Systems

Single-node CPU-based systems: Boost Graph Library

Multi-CPU systems: Ligra, Galois

Distributed CPU-based systems: PowerGraph

Specialized GPU algorithms

GPU-based systems: CuSha, Medusa, Gunrock...
Why Gunrock?

- Data-centric abstraction is designed for GPU
- Our APIs are simple and flexible
- Our optimizations achieve high performance
- Our framework enables multi-GPU integration
What we want to achieve with Gunrock?

Performance

- High performance GPU computing primitives
- High performance framework
- Optimizations
- Multi-GPU capability

Programmability

- A data-centric abstraction designed specifically for the GPU
- Simple and flexible interface to allow user-defined operations
- Framework and optimization details hidden from users, but automatically applied when suitable
**Idea: Data-Centric Abstraction & Bulk-Synchronous Programming**

**Data-centric abstraction**
- Operations are defined on a group of vertices or edges ≜ a frontier
  => Operations = manipulations of frontiers

**Bulk-synchronous programming**
- Operations are done one by one, in order
- Within a single operation, computing on multiple elements can be done in parallel, without order

A generic graph algorithm:

Loop until convergence

- A group of V or E
  - Do something
- Resulting group of V or E
  - Do something
- Another resulting group of V or E
Gunrock’s Operations on Frontiers

**Generation**

Advances: visit neighbor lists

**Filter**

Select and reorganize

**Computation**

Compute: per-element computation, in parallel
can be combined with advance or filter
Optimizations: Workload mapping and load-balancing

P: uneven neighbor list lengths
S: trade-off between extra processing and load balancing
First appeared in various BFS implementations, now available for all advance operations

Load-Balanced Partitioning [3]

Block cooperative Advance of large neighbor lists;
Warp cooperative Advance of medium neighbor lists;
Pre-thread Advance of small neighbor lists.
Per-thread fine-grained, Per-warp and per-CTA coarse-grained [4]
Optimizations: Idempotence

P: Concurrent discovery conflict (v5,8)
S: Idempotent operations (frontier reorganization)
- Allow multiple concurrent discoveries on the same output element
- Avoid atomic operations
First appeared in BFS [4], now available to other primitives
Optimizations: **Pull vs. push traversal**

**P:** From many to very few (v5,6,7,8,9,10 -> v11, 12)

**S:** Pull vs. push operations (frontier generation)
- Automatic selection of advance direction based on ratio of undiscovered vertices

First appeared in DO-BFS [5], now available to other primitives
Optimizations: Priority queue

P: A lot of redundant work in SSSP-like primitives
S: Priority queue (frontier reorganization)
- Expand high-priority vertices first
First appeared in SSSP[3], now available to other primitives
Idea: Multiple GPUs

P: Single GPU is not big and fast enough
S: use multiple GPUs
-> larger combined memory space and computing power

P: Multi-GPU program is very difficult to develop and optimize
S: Make algorithm-independent parts into a multi-GPU framework
-> Hide implementation details, and save user's valuable time

P: Single GPU primitives can’t run on multi-GPU
S: Partition the graph, renumber the vertices in individual sub-graphs and do data exchange between super steps
-> Primitives can run on multi-GPUs as it is on single GPU
Multi-GPU Framework (for programmers)

Recap: Gunrock on single GPU

![Diagram showing input and output frontiers with associative data and single GPU]
Multi-GPU Framework (for programmers)

Dream: just duplicate the single GPU implementation
Reality: it won’t work, but good try!

GPU 0

Input frontier

Associative data (label, parent, etc.)

Output frontier

GPU 1

Input frontier

Output frontier

Associative data (label, parent, etc.)

Iterate till convergence

Gunrock @ GTC 2016, Apr. 6, 2016
Multi-GPU Framework (for programmers)

Now it works

Iterate till all GPUs convergence

Partition

Local input frontier

Remote input frontier

Remote input frontier

Remote input frontier

Remote input frontier

Local input frontier

Local output frontier

Remote output frontier

Remote output frontier

Remote output frontier

Local output frontier

Associative data (label, parent, etc.)

GPU 0

GPU 1

Gunrock @ GTC 2016, Apr. 6, 2016
Multi-GPU Framework (for end users)

gunrock_executable input_graph --device=0,1,2,3 other_parameters
Graph partitioning

- Distribute the vertices
- Host edges on their sources’ host GPU
- Duplicate remote adjacent vertices locally
- Renumber vertices on each GPU (optional)

-> Primitives no need to know peer GPUs
-> Local and remote vertices are separated
-> Partitioning algorithm not fixed

P: Still looking for good partitioning algorithm /scheme
Optimizations: Multi-GPU Support & Memory Allocation

**P:** Serialized GPU operation dispatch and execution

**S:** Multi CPU threads and multiple GPU streams

- ≥1 CPU threads with multiple GPU streams to control each individual GPUs
- -> overlap computation and transmission
- -> avoid false dependency

**P:** Memory requirement only known after advance / filter

**S:** Just-enough memory allocation

- check space requirement before every possible overflow
- -> minimize memory usage
- -> can be turned off for performance, if requirements are known (e.g. from previous runs on similar graphs)
Results:

Single GPU Gunrock vs. Others

<table>
<thead>
<tr>
<th>BGL</th>
<th>h09</th>
<th>i04</th>
<th>kron</th>
<th>rga</th>
<th>roadnet</th>
<th>soc</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td><img src="image1.png" alt="Graph" /></td>
<td><img src="image2.png" alt="Graph" /></td>
<td><img src="image3.png" alt="Graph" /></td>
<td><img src="image4.png" alt="Graph" /></td>
<td><img src="image5.png" alt="Graph" /></td>
<td><img src="image6.png" alt="Graph" /></td>
</tr>
<tr>
<td>BFS</td>
<td><img src="image7.png" alt="Graph" /></td>
<td><img src="image8.png" alt="Graph" /></td>
<td><img src="image9.png" alt="Graph" /></td>
<td><img src="image10.png" alt="Graph" /></td>
<td><img src="image11.png" alt="Graph" /></td>
<td><img src="image12.png" alt="Graph" /></td>
</tr>
<tr>
<td>CC</td>
<td><img src="image13.png" alt="Graph" /></td>
<td><img src="image14.png" alt="Graph" /></td>
<td><img src="image15.png" alt="Graph" /></td>
<td><img src="image16.png" alt="Graph" /></td>
<td><img src="image17.png" alt="Graph" /></td>
<td><img src="image18.png" alt="Graph" /></td>
</tr>
<tr>
<td>PR</td>
<td><img src="image19.png" alt="Graph" /></td>
<td><img src="image20.png" alt="Graph" /></td>
<td><img src="image21.png" alt="Graph" /></td>
<td><img src="image22.png" alt="Graph" /></td>
<td><img src="image23.png" alt="Graph" /></td>
<td><img src="image24.png" alt="Graph" /></td>
</tr>
<tr>
<td>SSSP</td>
<td><img src="image25.png" alt="Graph" /></td>
<td><img src="image26.png" alt="Graph" /></td>
<td><img src="image27.png" alt="Graph" /></td>
<td><img src="image28.png" alt="Graph" /></td>
<td><img src="image29.png" alt="Graph" /></td>
<td><img src="image30.png" alt="Graph" /></td>
</tr>
</tbody>
</table>

Outperforms BGL compared to BGL and PowerGraph.

BGL (Boosted Graph Library) is a C++ template library for graph algorithms.

CuShma (CUDA Shma) is a CUDA-based implementation of graph algorithms.

GraphBL (Graph Building Library) is a C++ template library for graph algorithms.

Light (Lightweight Graph Library) is a C++ template library for graph algorithms.

MapGraph (Map Graph Library) is a C++ template library for graph algorithms.

Performance comparison across various graph algorithms and datasets.
Results: Multi-GPU Scaling

* Primitives (except DOBFS) get good speedups (averaged over 16 datasets of various types)
  BFS: 2.74x, SSSP: 2.92x, CC: 2.39x, BC: 2.22x, PR: 4.03x using 6 GPUs

* Peak DOBFS performance: 514 GTEPS with rmat_n20_512

* Gunrock is able to process graph with 3.6B edges (full-friendster graph, undirected, DOBFS in 339ms, 10.7 GTEPS using 4 K40s), 50 PR iterations on the directed version (2.6B edges) took ~51 seconds
Results: Multi-GPU Scaling

*Strong: Rmat_n24_32
*Weak edge: Rmat_n19_256 * #GPUs
*Weak vertex: Rmat_2^{19} * #GPUs_256

Mostly linear, except for DOBFS strong scaling
**Results: Multi-GPU Gunrock vs. Others (BFS)**

<table>
<thead>
<tr>
<th>graph</th>
<th>algo</th>
<th>ref.</th>
<th>ref. hw.</th>
<th>ref. perf.</th>
<th>our hw.</th>
<th>our perf.</th>
<th>comp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>com-Friendster (66M, 1.81B, UD)</td>
<td>BFS</td>
<td>Bisson [5]</td>
<td>1xK20Xx64</td>
<td>15.68 GTEPS</td>
<td>4xK40</td>
<td>14.1 GTEPS</td>
<td>0.90X</td>
</tr>
<tr>
<td>kron_n23_16 (8M, 256M, UD)</td>
<td>BFS</td>
<td>Bernaschi [4]</td>
<td>1xK20Xx4</td>
<td>~1.3 GTEPS</td>
<td>4xK40</td>
<td>30.8 GTEPS</td>
<td>23.7X</td>
</tr>
<tr>
<td>kron_n25_16 (32M, 1.07G, UD)</td>
<td>BFS</td>
<td>Bernaschi [4]</td>
<td>1xK20Xx16</td>
<td>~3.2 GTEPS</td>
<td>6xK40</td>
<td>31.0 GTEPS</td>
<td>9.69X</td>
</tr>
<tr>
<td>kron_n25_32 (32M, 1.07G, D)</td>
<td>BFS</td>
<td>Fu [13]</td>
<td>2xK20x32</td>
<td>22.7 GTEPS</td>
<td>4xK40</td>
<td>32.0 GTEPS</td>
<td>1.41X</td>
</tr>
<tr>
<td>kron_n23_32 (8M, 256M, D)</td>
<td>BFS</td>
<td>Fu [13]</td>
<td>2xK20x2</td>
<td>6.3 GTEPS</td>
<td>4xK40</td>
<td>27.9 GTEPS</td>
<td>4.43X</td>
</tr>
<tr>
<td>kron_n24_32 (16.8M, 1.07G, UD)</td>
<td>BFS</td>
<td>Liu [23]</td>
<td>2xK40</td>
<td>15 GTEPS</td>
<td>2xK40</td>
<td>77.7 GTEPS</td>
<td>5.18X</td>
</tr>
<tr>
<td>kron_n24_32 (16.8M, 1.07G, UD)</td>
<td>BFS</td>
<td>Liu [23]</td>
<td>8xK40</td>
<td>18.4 GTEPS</td>
<td>4xK80</td>
<td>40.2 GTEPS</td>
<td>2.18X</td>
</tr>
<tr>
<td>twitter-mpi (52.6M, 1.96G, D)</td>
<td>BFS</td>
<td>Bebee [3]</td>
<td>1xK40x16</td>
<td>0.2242 sec</td>
<td>3xK40</td>
<td>94.31 ms</td>
<td>2.38X</td>
</tr>
</tbody>
</table>

* graph format: name (|V|, |E|, directed (D) or undirected (UD))
* ref. hw. format: #GPU per node x GPU model x #nodes
* Gunrock out-performs or close to small GPU clusters using 4 ~ 64 GPUs, on both real and generated graphs
* a few times faster than Enterprise (Liu et al., SC15), a dedicated multi-GPU DOBFS implementation
Current Status

It has over 10 graph primitives
* traversal-based, node-ranking, global (CC, MST)
* LOC ≤ 10 to use a primitive
* LOC ≤ 300 to program a new primitive
* Good balance between performance and programmability

Multi-GPU framework going to support multi-node GPU cluster
* use circular-queue for better scheduling and smaller overhead
* extendable onto multi-node usage

More graph primitives are coming
* graph coloring, maximum independent set, community detection, subgraph matching

Open source, available at
http://gunrock.github.io/
Future Work

* Multi-node support with NVLink
* Performance analysis and optimization
* Graph BLAS
* Asynchronized graph algorithms
* Fixed partitioning / 2D partitioning
* Global, neighborhood, and sampling operations
* More graph primitives
* Dynamic graphs
* ...
Acknowledgment

The Gunrock team

Onu Technology and Royal Caliber team
  Erich Elsen, Vishal Vaidyanathan, Oded Green and others
  For their discussion on library development and dataset generating code

All code contributors to the Gunrock library

NVIDIA
  For hardware support, GPU cluster access, and all other supports and discussions

The Gunrock project is funded by
* DARPA XDATA program under AFRL Contract FA8750-13-C-0002
* NSF awards CCF-1017399 and OCI-1032859
* DARPA STTR award D14PC00023
References


Questions?


Q: Papers, slides, etc.? A: [https://github.com/gunrock/gunrock#publications](https://github.com/gunrock/gunrock#publications)

Q: Requirements? A: CUDA ≥ 7.5, GPU compute capability ≥ 3.0, Linux || Mac OS

Q: Language? A: C/C++, with a simple wrapper connects to Python

Q: Is it free and open? A: Absolutely (under Apache License v2.0)

Q: … (continue)
Example python interface - breadth-first search

```python
from ctypes import *

### load gunrock shared library - libgunrock
gunrock = cdll.LoadLibrary('..../build/lib/libgunrock.so')

### read in input CSR arrays from files
row_list = [int(x.strip()) for x in open('toy_graph/row.txt')]
col_list = [int(x.strip()) for x in open('toy_graph/col.txt')]

### 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
labels = pointer((c_int * nodes)())

### call gunrock function on device
gunrock.bfs(labels, nodes, edges, row, col, 0)

### sample results
print ' bfs labels (depth):',
for idx in range(nodes): print labels[0][idx],
```
Example: BFS with Gunrock

Advance + Compute (+1, AtomicCAS)
Example: BFS with Gunrock

Advance + Compute (+1, AtomicCAS)

Filter

3 4 2

1 2 3 4 5 6 7 8 9 10 11 12 13

$\infty$
Example: BFS with Gunrock

1
Advance + Compute (+1, AtomicCAS)

3 4 2
Filter

3 4 2
Advance + Compute (+1, AtomicCAS)
Example: BFS with Gunrock

P: uneven neighbor list lengths (v4 vs. v3)
P: Concurrent discovery conflict (v5,8)

Advance + Compute

3 4 2

Filter

3 4 2

Advance + Compute (+1, AtomicCAS)

1 2 5 6 7 8 9 10 1 8 1 3 5 8

Gunrock @ GTC 2016, Apr. 6, 2016
Example: BFS with Gunrock

1. Advance + Compute
   3  4  2

2. Filter
   3  4  2

3. Advance + Compute (+1, AtomicCAS)
   1  2  5  6  7  8  9  10  1  8  1  3  5  8

4. Filter
   6  7  9  10  8  5

P: uneven neighbor list lengths (v4 vs. v3)
P: Concurrent discovery conflict (v5,8)
Example: BFS with Gunrock

1
Advance + Compute
3 4 2
Filter
3 4 2
Advance + Compute (+1, AtomicCAS)
1 2 5 6 7 8 9 10 | 1 8 | 1 3 5 8
Filter
6 7 9 10 8 5
Advance + Compute, Filter
11 12

P: uneven neighbor list lengths (v4 vs. v3)
P: Concurrent discovery conflict (v5,8)
P: From many to very few (v5,6,7,8,9,10 -> v11, 12)

Gunrock @ GTC 2016, Apr. 6, 2016
Multi-GPU Framework (for programmers)
Graph partitioning

\[|V| = 13\]
\[|E| = 44\]

\[|V| = 11\]
\[|E| = 23\]

\[|V| = 12\]
\[|E| = 21\]

\[|V| = 13\]
\[|E| = 44\]