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

Objectives
For large-scale graph analytics on the GPU, the irregularity of data access and control flow and the complexity of programming GPUs have been two significant challenges for developing a programmable high-performance graph library. We describe Gunrock, our system for graph processing on the GPU. Our goal with Gunrock is to deliver the performance of GPU hardwired graph primitives with a high-level programming model that allows programmers to quickly develop new graph primitives.

Introduction
The superior performance, price-performance, and power-performance capabilities of the modern GPU over the traditional CPU make it a strong candidate for data-intensive applications like graph processing. Previous GPU-based large graph analytics work either uses a serial or coarse-grained parallel programming model (single-node systems) or has substantial communication cost (distributed systems). GPU low-level implementations of specific graph primitives (“hardwired” primitives) require expert knowledge of GPU programming and optimization. Existing high-level GPU graph processing systems often recapitulate CPU programming models and do not compare favorably in performance with hardwired primitives.

With Gunrock, we design and implement a set of simple and flexible APIs that significantly reduce the code size and the development time and apply to a wide range of graph processing primitives. We also implement several GPU-specific optimization strategies for memory efficiency, load balancing, and workload saving that together achieve high performance.

The Gunrock Abstraction
Gunrock targets graph operations that can be expressed as iterative convergent processes. Each step operates on a frontier of active vertices or edges in the graph. Steps are bulk-synchronous parallel (BSP): different steps may have dependencies between them, but individual operations within a step can be processed in parallel.

- A traverse step generates a new frontier from the current frontier.
- An advance generates a new frontier by visiting the neighbors of the current frontier. According to the direction of the edges, advance can perform both push-style traversal (scatter) and pull-style traversal (gather).
- A filter chooses a subset of the current frontier based on programmer-specified criteria.
- A computation step defines an operation on all elements (vertices or edges) in the current frontier; Gunrock then performs that operation in parallel across all elements.

Applications
By reusing Gunrock’s efficient operators and combining different functors, users can build new graph primitives with minimal extra work. Currently, Gunrock supports graph traversal-based algorithms (Breadth-First Search (BFS) and Single-Source Shortest Path (SSSP)), node ranking algorithms (Betweenness Centrality (BC), PageRank, HITS, SALSA, and Twitter’s “Money” [which requires bipartite graph support]), and subgraph-based algorithms (Connected Component Labeling, Minimum Spanning Tree). We are moving forward to more complex graph primitives as well as extending our operators within the current traversal-computation programming model.

Example: Comparing Abstractions on Single-Source Shortest Path

Table 1: Gunrock’s runtime comparison with other graph libraries and hardware GPU implementations. Ligra’s timings for PageRank and Gunrock’s one-iteration PageRank are in bold. Hardwired GPU implementations for each primitive are bolded BFS (Merritt et al., PPoPP ’12), deltaStep SSSP (Davidson et al., IPDPS ’14), gunrock-BC (Sarajlic et al., GPUGPU ’16), and connected component labeling (Soman et al., IPDPSW ’10).

Results

Next Steps
- Scalability to multiple GPUs/nodes;
- Higher-level graph primitives;
- In-depth comparison to GAS and Graph BLAS;
- Support for mutable graph/time-series graph.

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

Funding
DoD XDATA, STTR ST13B-004; NSF OCI-1032859, CCF-1017399.