Linear Algebra is the Right Way to Think About Graphs

Carl Yang
University of California, Davis
Lawrence Berkeley National Laboratory
ctcyang@ucdavis.edu

ABSTRACT

Graph algorithms are challenging to implement on new accelerators such as GPUs. To address this problem, GraphBLAS is an innovative on-going effort by the graph analytics community to formulate graph algorithms as sparse linear algebra, so that they can be expressed in a performant, succinct and in a backend-agnostic manner. Initial research efforts in implementing GraphBLAS on GPUs for graph processing and analytics have been promising, but challenges such as feature-incompleteness and poor performance still exist compared to their vertex-centric ("think like a vertex") graph framework counterparts. For our thesis, we propose a multi-language graph framework aiming to simplify the development of graph algorithms, which 1) provides a multi-language GraphBLAS interface for the end-users to express, develop, and refine graph algorithms more succinctly than existing distributed graph frameworks; 2) abstracts away from the end-users performance-tuning decisions; 3) utilizes the advantages of existing low-level GPU computing primitives to maintain high performance.

KEYWORDS

sparse matrix multiplication, breadth-first search, graph algorithms

1 PROBLEM STATEMENT

High-performance graph processing is very important in many problems including anomaly detection (intrusions into computer network), recommender systems (which movie to watch) and community detection (identifying proteins within a cell with similar functionality). As the graphics processor (GPU) has become common in personal computers, supercomputers, and datacenters, its advantages in raw performance and price-performance have motivated its use in graph computation. Recent research has demonstrated the performance advantages of GPUs over CPUs on a spectrum of graph problems. However, the GPU is notorious for being hard to program, which raises the following questions:

- Is it possible to design a graph framework that is expressible enough so that one does not need to be a GPU ninja to take advantage of its performance advantage?
- What is the right set of primitives for expressing graph algorithms?

SC '18, Dallas, Texas

DOI: 0000/0000

We will define the right set of primitives as one that maximizes the following metrics:

(1) Conciseness
(2) Expressibility
(3) Performance
(4) Backend-agnosticism

First and foremost, their application code ought to be concise. Secondly, the graph primitives ought to be highly expressible, so that the end user ought not need to descend to writing low-level code in order to describe their graph application. Thirdly, a set of graph primitives ought to enable the end user to reach state-of-the-art performance on the graph application they are building. Finally, the application code ought to require little to no change when targeting different backends. That is to say, the same application code ought to work just as well for single-threaded CPU as for GPU. The objective of my research is to investigate the following statement:

Linear algebra is the right way to think about graph algorithms. Graph primitives based in linear algebra are superior to existing vertex-centric graph frameworks in conciseness, expressibility, performance and backend-agnosticism.

To determine the validity of this statement, we will build a graph framework that supports GPU as a backend. It will have graph primitives that are based in linear algebra. We will compare our graph framework against existing graph frameworks using the metrics given above. Our linear algebra-based graph framework will be grounded in GraphBLAS.

GraphBLAS is an innovative on-going effort by the graph analytics community to formulate graph algorithms as sparse linear algebra. The core idea behind GraphBLAS is that by representing a graph as an adjacency matrix and the vertex to be traversed as a vector, there is an equivalence between graph traversal and matvec (matrix-vector multiplication–Level 2 BLAS). Relying on this fact, GraphBLAS then generalizes naturally to batches of frontiers (matrix-matrix multiplication–Level 3 BLAS) in a way that is unnatural and challenging to implement with traditional vertex-centric frameworks.

Initial research efforts in implementing GraphBLAS on GPUs for graph processing and analytics have been promising, but challenges such as feature-incompleteness and poor performance still exist compared to their vertex-centric graph framework counterparts. Therefore, the primary focus of our work will be the key optimizations required to make a linear algebra-based framework on the GPU based in GraphBLAS performant on a broad spectrum of graph algorithms including: breadth-first-search (BFS), single-source shortest-path (SSSP), connected components (CC), betweenness centrality (BC) and PageRank (PR). In doing so, we will also tackle challenges of expressibility, conciseness and backend-agnostic design.

2 RESEARCH HIGHLIGHTS

2.1 Push-Based BFS on GPU using Sparse Matrix-Vector Multiplication

In [10], we present the first GPU linear algebra-based implementation of BFS to show parity with push-only version of the state-of-the-art GPU graph framework Gunrock [7]. We introduce a primitive called sparse matrix-sparse vector multiplication (SpM-SpV), and show that it is crucial to getting good BFS performance. By using scan-based load-balancing and sorting, we are up to 3.3× faster than using a highly-optimized sparse matrix-dense vector (SpMV) implementation.

2.2 Direction-Optimized BFS

In [9], we overcome the expressibility and performance bottlenecks present in previous implementations of GraphBLAS for the GPU [11] by adding direction-optimized BFS capability. Our speed-ups over previous single-threaded and GPU implementations are 122× and 18.3× respectively. Our introduction of the direction-optimized capability allows us to match the performance of push-pull Gunrock on scale-free graphs. Furthermore, due to our decomposition of direction-optimized BFS into three optimizations using linear algebra, the superior generality offered by linear algebra lets us determine what other graph algorithms they can speed up.

2.3 Sparse Matrix-Dense Matrix Multiplication

We have also had success implementing the building blocks (sparse matrix-dense matrix multiplication or SpMM) that are needed for other algorithms such as batched BC [8]. In this work, we derive a peak speed-up of 4.1× over vendor-shipped library and implementations of SpMM in literature. Our speed-up comes from our use of a multi-algorithm that combines a novel memory access pattern with a merge-based memory access pattern.

3 EXPERIMENTAL SETUP

We ran all experiments on a Linux workstation with 2× 3.50 GHz Intel 4-core E5-2637 v2 Xeon CPUs, 556 GB of main memory, and an NVIDIA K40c GPU with 12 GB on-board memory. The GPU programs were compiled with NVIDIA's nvcc compiler (version 8.0.61). The C code was compiled using gcc 4.9.4. Ligra was compiled using icpc 15.0.1 with CilkPlus. All results ignore transfer time (from disk-to-memory and CPU-to-GPU). The gather, scan, and row-based mxv (without mask) operations are from the ModernGPU library [2]. The radix sort and reduce operations are from the CUDA UnBound library (CUB) [6]. All BFS tests were run 10 times with the average runtime and MTEPS used for results. Software artifacts are available of our most recent work [8, 9] on GitHub[1, 2].

4 CURRENT AND FUTURE WORK

We have had success in breadth-first-search (BFS). However, we need to obtain more datapoints to demonstrate or falsify our problem statement. Therefore, we are currently focusing on single-source shortest-path (SSSP), connected components (CC), betweenness centrality (BC) and PageRank (PR). By the time I graduate, I hope to have completed these four applications and performed a comprehensive evaluation of linear algebra-based graph frameworks and vertex-centric frameworks.

Greiner and Jacob have proven theoretically [5] that as the number of nonzeroes per row exceeds some hardware threshold, namely $\frac{m}{M}$ where m is the number of rows in the sparse matrix and M is the size of the fast memory of the device, tiling will become more efficient than the access pattern described in this paper (i.e. going across the sparse matrix and selecting nonzeroes in the dense matrix). Indeed, they claim that tiling both the sparse matrix **A** and **B** in a manner akin to tiling dense matrix-matrix multiplication is optimal. In future work, it would be interesting to find out whether doing this tiling will extend SpMM's effectiveness range beyond 9% sparsity (i.e. range in which SpMM is better than dense-dense matrix multiplication).

An interesting future direction for research is designing a library around load-balancing techniques such as merge path. While merge path is already present in two libraries–Modern GPU and CUB [2, 6]—they are not designed as layers separated from computation. Similarly in our code, computation and load-balancing are very tightly knit. It would be interesting to discover how to abstract out the load balancing from the computation. Ideally, the user would have to identify the quantities that are desirable for load balancing separately from the computation. Then the load-balancing library would handle the rest making load-balanced GPU kernels much easier to write. The impact of our improved SpMM kernels on application codes is also worth investigating in the future. In particular, we expect a co-design approach to provide more pronounced performance benefits to applications compared to drop-down kernel replacement.

[1] https://github.com/owensgroup/push-pull
[2] https://github.com/owensgroup/merge-spmm

REFERENCES

[1] H. Metin Aktulga, Aydın Buluç, Samuel Williams, and Chao Yang. 2014. Optimizing Sparse Matrix-Multiple Vectors Multiplication for Nuclear Configuration Interaction Calculations. In *Proceedings of the IPDPS*. IEEE Computer Society.
[2] S. Baxter. 2015. Modern GPU Library. http://nvlabs.github.io/moderngpu/. (2015).
[3] Aydin Buluc, Timothy Mattson, Scott McMillan, Jose Moreira, and Carl Yang. 2017. *The GraphBLAS C API Specification.* Rev. 1.1.
[4] Aydın Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. 2009. Parallel Sparse Matrix-Vector and Matrix-Transpose-Vector Multiplication Using Compressed Sparse Blocks. In *Proceedings of SPAA*.
[5] Gero Greiner and Riko Jacob. 2010. The I/O complexity of sparse matrix dense matrix multiplication. In *LATIN*. 143–156.
[6] Duane Merrill. 2015. CUB Library. http://nvlabs.github.io/cub. (2015).
[7] Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John D Owens. 2016. Gunrock: A high-performance graph processing library on the GPU. In *ACM SIGPLAN Notices*, Vol. 51. ACM, 11.
[8] Carl Yang, Aydın Buluç, and John D. Owens. 2018. Design Principles for Sparse Matrix Multiplication on the GPU. In *Proceedings of the 24th International European Conference on Parallel and Distributed Computing (Euro-Par 2018).* https://escholarship.org/uc/item/5h35w3b7
[9] Carl Yang, Aydın Buluç, and John D. Owens. 2018. Implementing Push-Pull Efficiently in GraphBLAS. In *Proceedings of the International Conference on Parallel Processing (ICPP 2018).* https://escholarship.org/uc/item/021076bn
[10] Carl Yang, Yangzihao Wang, and John D. Owens. 2015. Fast sparse matrix and sparse vector multiplication algorithm on the GPU. In *IPDPSW*. IEEE, 841–847.
[11] Peter Zhang, Marcin Zalewski, Andrew Lumsdaine, Samantha Misurda, and Scott McMillan. 2016. GBTL-CUDA: Graph algorithms and primitives for GPUs. In *IPDPSW*. IEEE, 912–920.