Synchronous vs. Asynchronous GPU Graph Frameworks

Yuechao Pan, Muhammad Osama, John D. Owens
University of California, Davis
1 Shields Ave.
Davis, CA 95616 USA
{ychpan,mosama,jowens}@ucdavis.edu

ABSTRACT

Recent node-level GPU accelerated graph processing frameworks have separately chosen synchronous and asynchronous architectures. Which is better under which circumstances, and why? We focus on Gunrock (a synchronous framework) vs. Groute (an asynchronous framework) with 3 primitives on 3 different datasets. We identify load balance, kernel count, and communication latency and bandwidth as quantities of particular interest.

KEYWORDS

Graph processing, GPU, Performance Analysis

ACM Reference format:
Yuechao Pan, Muhammad Osama, John D. Owens. 2017. Synchronous vs. Asynchronous GPU Graph Frameworks. In *Proceedings of The 7th Workshop on Multi-core and Rack Scale Systems, Belgrade, Serbia, April 2017 (MaRS 2017),* 3 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This paper focuses on multi-GPU implementations of graph primitives. Synchronous approaches to this problem [3, 6] follow the BSP model [9] and require a global synchronization between algorithmic iterations. Asynchronous approaches [1, 10] allow direct GPU-to-GPU communication without a global barrier. In this positioning paper, we focus on the Gunrock (synchronous) and Groute (asynchronous) frameworks for GPU graph computation to gain insight into which framework works the best in which situations. We show experiments using large real-world graphs, on different GPU generations, to illuminate performance bottlenecks and how they may change over time.

2 ANALYSIS OF SYNCHRONOUS FRAMEWORKS

The BSP model that underpins existing synchronous graph frameworks is a natural fit for the GPU execution model. More specifically, results output from a streaming multiprocessor (SM) are only guaranteed to be viewable on another SM after a kernel boundary, which is a hard synchronization point on a single GPU. In general, BSP approaches on GPUs, and synchronous graph frameworks, are best suited for large workloads on every kernel launch. Having a large workload per kernel motivates an implementation focus on load balancing, a critical optimization when dealing with irregular workloads like sparse graphs.

MaRS 2017, Belgrade, Serbia
2017. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

However, this approach also has drawbacks. Load balancing operations are not free and may not justify the cost when working on more regular datasets. An inter-GPU synchronization point inhibits more fine-grained (and possibly useful) data communication and may introduce waiting time when workloads are not perfectly balanced.

Using the BSP model and Gunrock as an example, we summarize the cost of computation, communication, and synchronization on different graph primitives in Table 1 of our previous work [6]. For Breadth-First Search (BFS), Connected Components (CC), and PageRank (PR), the communication volume is at most on the order of the number of vertices in a GPU's local sub-graph, and the computation is at least on the order of the number of edges of such sub-graph. As a result, for graphs with large average vertex degrees, a synchronous framework should achieve good scaling across multiple GPUs. However, we will observe poor scalability for workloads in which the per-iteration overhead (e.g., kernel launch overheads and communication latencies) is significant compared to other parts of the framework, which happens when the per-iteration workload is too small.

3 ANALYSIS OF ASYNCHRONOUS FRAMEWORKS

Asynchronous frameworks must run atop a synchronous GPU substrate. One common approach is to launch small batches of workloads using many individual kernels, mirroring a continuous data flow. Small kernels may not be able to utilize the full computing power of GPUs, and kernel launches have a few microseconds of overhead. With small workloads per kernel, load balancing is proportionally more expensive and may not be helpful.

The principal advantage of asynchrony is its lack of a global barrier, which allows faster communication: the graph framework does not need to wait for all the data from its peers before performing further computation, significantly reducing synchronization costs. Also because data flows more freely and in smaller packages, smart asynchronous algorithms can propagate data more quickly than synchronous frameworks, and thus allow solutions to converge sooner, yielding better performance on some algorithms such as PR and CC.

4 EXPERIMENTS

To understand how the frameworks perform on real data, we test them using three huge graphs (Table 1). These graphs have a similar number of edges, but have a wide variety of graph properties: *soc-twitter-2010* is a large scale-free social network with a short diameter and a irregular degree distribution; *osm-eur* is the road network of Europe, with a large graph diameter, and a very regular degree

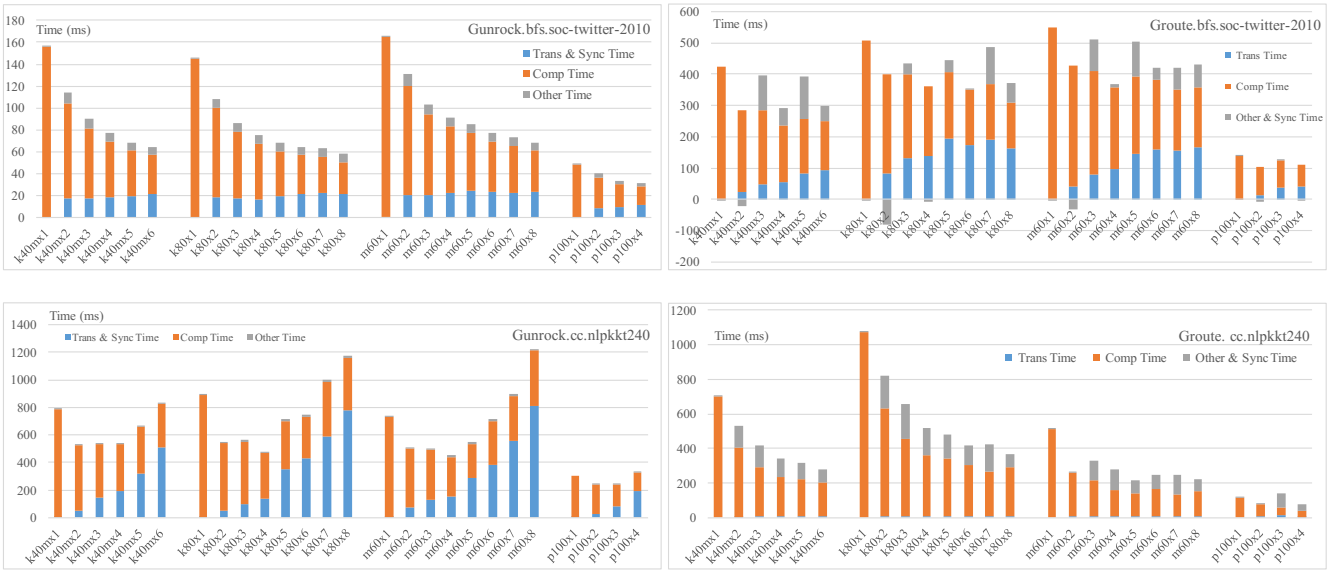

Figure 1: Performance of Gunrock (left) and Groute (right). BFS using soc-twitter-2010 on top, CC using snlpkkt240 at bottom. Negative timings indicate successful overlap between compute and communication.

Graph	$\|V\|$	$\|E\|$	D*	Note
soc-twitter-2010	21.3M	530M	15	[2], undirected
nlpkkt240	28M	746M	~ 200	[7], undirected
osm-eur	174M	348M	~ 22k	[5], directed

Table 1: Graphs used for evaluation. * We use the avg. depth of multiple BFS search from randomly chosen source vertices to approximate graph diameter.

distribution (more than 90 percent of vertices have out-degree 2 or 3); *nlpktt240* is a graph from a optimization problem, with a medium diameter and a regular degree distribution.

We also selected three primitives, BFS, PR, and CC. BFS is a strictly level-synchronous primitive that requires processing at one depth to complete before starting the next; Soman's CC algorithm [8] does not have any ordering requirement on the hooking or the pointer jumping steps, which makes it very asynchronous-friendly; PR has high tolerance for varying the order of operations, but for performance, PR does need to accumulate some ranking changes on a vertex before propagating to its neighbors. We use the latest version of Groute [1] and Gunrock [4], and GPUs across 4 generations. We summarize performance in Fig. 1.

For BFS on *soc-twitter-2010*, a large irregular graph, Gunrock's computation displays clear advantages over Groute's, mostly because of Gunrock's load balancing strategies. Gunrock's computation also scales nicely across multiple GPUs and different GPU generations. Gunrock's communication for BFS is bounded by border size, and when the time for computation is at least as much as for communication, the framework can maintain good scalability. Groute's asynchronous communication takes up considerably more time for the large irregular graph, which harms its scalability. This comparison generally stays true for other BSP-like primitives on large irregular graphs.

On the other end of the spectrum, for an asynchronous-friendly primitive, CC, on a more regular graph, *nlpkkt240*, Groute's asynchronous communication shows its strength in keeping communication volume at a very low level, yielding impressive performance and scalability. Although Gunrock's computation still manages to scale, Gunrock suffers from communication volume increases resulting from its all-to-all label updates in between iterations. After reaching 3–4 GPUs, the additional cost of communication time outweighs the savings in computation time. Gunrock's load balancing methods increase per-iteration overheads; synchronization time becomes a significant component of runtime when the per-iteration workload is small; together these considerations make Gunrock's performance on regular, large-diameter graphs inferior to Groute's.

5 FUTURE HARDWARE

Technology roadmaps indicate that computing power will continue to grow faster than inter-processor bandwidth, and that inter-processor communication latency may not see any significant reduction in the near future. This may require future frameworks of both kinds to pay more attention to reducing communication cost, especially in not allowing the latency to be a bottleneck (this is a particular concern when routing a large number of small data packets). The different successes of synchronous and asynchronous workloads in this set of experiments motivates exploring a hybrid between both models in an effort to build a framework that can run workloads with varying regularity and size on single- or multiple-GPU configurations.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the DARPA XDATA program (US Army award W911QX-12-C-0059); DARPA STTR awards D14PC00023 and D15PC00010; and NSF award CCF-1629657.

REFERENCES

[1] Tal Ben-Nun, Michael Sutton, Sreepathi Pai, and Keshav Pingali. 2017. Groute: An Asynchronous Multi-GPU Programming Model for Irregular Computations. In *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '17)*. ACM, New York, NY, USA, 235–248. DOI: http://dx.doi.org/10.1145/3018743.3018756

[2] Timothy A. Davis. 1994. The University of Florida Sparse Matrix Collection. *NA Digest* 92, 42 (16 Oct. 1994). http://www.cise.ufl.edu/research/sparse/matrices.

[3] Hang Liu and H. Howie Huang. 2015. Enterprise: Breadth-first Graph Traversal on GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15)*. Article 68, 12 pages. DOI: http://dx.doi.org/10.1145/2807591.2807594

[4] University of California Davis. 2017. Gunrock:High-Performance Graph Primitives for the GPU. (2017). http://gunrock.github.io.

[5] Karlsruhe Institute of Technology. 2014. OSM Europe Graph. (2014). http://i11www.iti.uni-karlsruhe.de/ressources/roadgraphs.php.

[6] Yuechao Pan, Yangzihao Wang, Yuduo Wu, Carl Yang, and John D. Owens. 2017. Multi-GPU Graph Analytics. In *Proceedings of the 31st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2017)*. http://escholarship.org/uc/item/39r145g1

[7] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 4292–4293. http://networkrepository.com/

[8] Jyothish Soman, Kothapalli Kishore, and P J Narayanan. 2010. A Fast GPU Algorithm for Graph Connectivity. In *24th IEEE International Symposium on Parallel and Distributed Processing, Workshops and PhD Forum (IPDPSW 2010)*. 1–8. DOI: http://dx.doi.org/10.1109/IPDPSW.2010.5470817

[9] Leslie G Valiant. 1990. A bridging model for parallel computation. *Commun. ACM* 33, 8 (1990), 103–111.

[10] Guozhang Wang, Wenlei Xie, Alan J. Demers, and Johannes Gehrke. 2013. Asynchronous Large-Scale Graph Processing Made Easy. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. http://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper58.pdf