Multi-GPU Graph Analytics

[Extended Abstract]

Yuechao Pan, Yangzihao Wang, Yuduo Wu, Carl Yang, and John D. Owens
University of California, Davis
{ychpan, yzhwang, yudwu, ctcyang, jowens}@ucdavis.edu

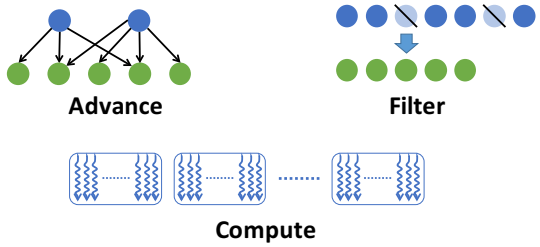

Figure 1: Graph algorithm as a data-centric process

Keywords

GPU, multi GPU, parallel graph processing

We present "Gunrock," a multi-GPU graph processing library that allows programmers to easily implement graph algorithms and extend onto multiple GPUs for scalable performance on large graphs with billions of edges.

Our high-level data-centric abstraction focuses on frontier operations, as shown in Fig. 1. A frontier is a compact queue of vertices or edges. It can be generated by visiting the neighbors of an existing frontier (advance), or by selecting and reorganizing elements from an existing frontier using programmer specified criteria (filter). Per-element computations on frontiers are implemented as parallel GPU kernels, and can be combined with advance or filter operations.

With this abstraction, Gunrock balances between performance and low programming complexity by coupling high performance GPU computing primitives and optimization strategies. Gunrock implements the breadth-first search algorithm by Merrill et al. [5], the delta-stepping single-source shortest path algorithm by Davidson et al. [2], the betweenness centrality algorithms by Jia et al. [4] and Sariyüce et al. [6], the connected component algorithm by Soman et al. [7], and our own PR algorithm. It also incorporates several optimization strategies, including dynamic workload mapping and idempotence by Merrill et al. [5], load balancing and priority queue by Davidson et al. [2], pull style traversal by Beamer et al. [1], plus our own improvements. More importantly, once an optimization is realized, it is not limited to a single algorithm, but can be applied to all applicable primitives.

As shown in Fig. 2, our multi-GPU framework only requires programmers to specify a few algorithm-dependent blocks, hiding most multi-GPU related implementation details. For the sub-queue or full-queue kernels, a single-GPU graph primitive can be used with small changes. Other than that, programmers only need to specify: 1) which vertex associative data need to be carried together with the sub-frontier during inter-GPU communication, 2) how to merge the received data with the local data, and 3) what is the convergent condition. The framework will take care of the rest. Programmers can also select the partitioning methods, or even implement their own. Our multi-GPU framework aims to provide programmability, algorithm generality, hardware compatibility, performance and scalability.

The multi-GPU framework also embeds several optimizations. It uses a dedicated CPU thread with multiple GPU streams to control each GPU. This effectively overlaps computation and inter-GPU data communication. It implements a just-enough GPU memory allocation scheme, which can start with a minimum pre-allocation, detect the actual memory requirement each time a larger space is needed, and reallocate according to such requirement. This scheme allows GPU memory usage to scale with more GPUs.

We have the best single-GPU and multi-GPU-single-node performance results among similar works. We achieve 22 GTEPS peak performance for BFS, and demonstrate a 6X speed-up with 2X total GPU memory consumption on 8 GPUs. Our edge traversal rate outperforms all previous work on GPU BFS within a single node and is comparable to work on GPU clusters using more GPUs (Table 1).

We identify synchronization / data communication patterns, graph topology, and partitioning algorithms as limiting factors to further scalability.

We are actively improving Gunrock, by carrying out in depth performance analysis and optimization, extending it

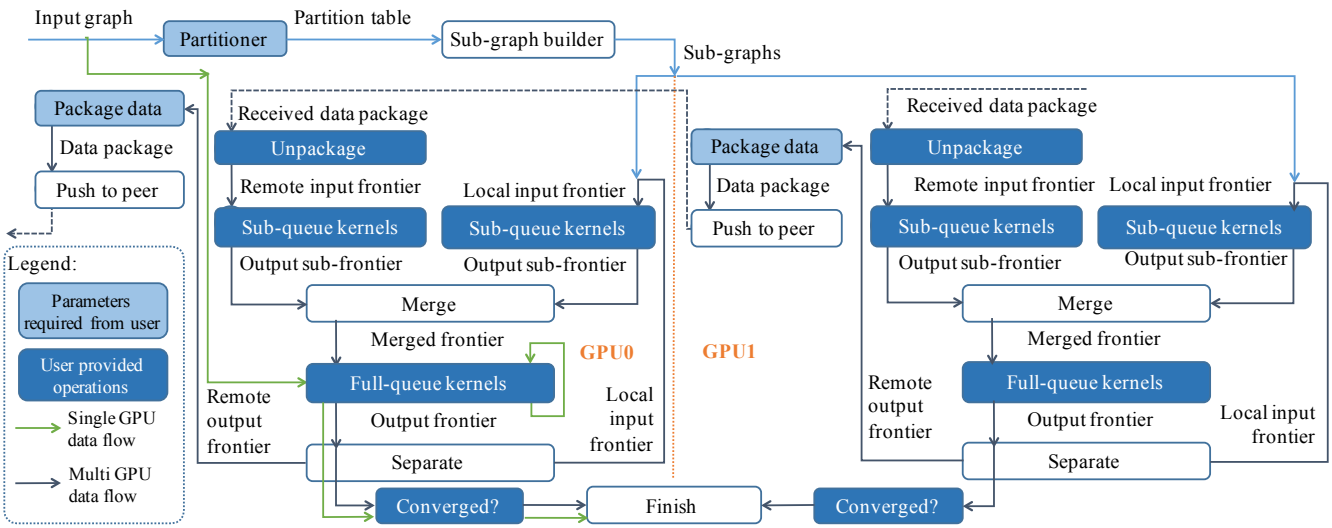

Figure 2: Block design of the Multi-GPU framework

	ref.	ref. hardware	ref. performance	our hardware	our performance
rmat_n20_128	Merrill et al. [5]	4x Tesla C2050	8.3 GTEPS	4x Tesla K40	11.2 GTEPS
rmat_n20_16	Zhong et al. [8]	4x Tesla C2050	15.4 ms	4x Tesla K40	9.29 ms
peak performance	Fu et al. [3]	16x Tesla K20 (cluster)	15 GTEPS	6x Tesla K40	22.3 GTEPS
peak performance	Fu et al. [3]	64x Tesla K20 (cluster)	29.1 GTEPS	6x Tesla K40	22.3 GTEPS

Table 1: Comparison with previous work on GPU BFS. Merrill et al.'s results on 3-year-old hardware are particularly impressive, though we note their implementation, as is Fu et al.'s, was customized only to BFS. Medusa (Zhong et al.), like Gunrock, is a programmable framework.

onto multiple nodes, and enabling more partitioning options, such as 2D partitioning and fixed partitioning, as well as more graph algorithms, including asynchronized ones. These works will appear in future submissions / publications.

References

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 12:1–12:10, Nov. 2012.

[2] A. Davidson, S. Baxter, M. Garland, and J. D. Owens. Work-efficient parallel GPU methods for single source shortest paths. In *Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium*, pages 349–359, May 2014.

[3] Z. Fu, H. K. Dasari, B. Bebee, M. Berzins, and B. Thompson. Parallel breadth first search on GPU clusters. In *IEEE International Conference on Big Data*, pages 110–118, Oct. 2014.

[4] Y. Jia, V. Lu, J. Hoberock, M. Garland, and J. C. Hart. Edge v. node parallelism for graph centrality metrics. In W. W. Hwu, editor, *GPU Computing Gems Jade Edition*, chapter 2, pages 15–28. Morgan Kaufmann, Oct. 2011.

[5] D. Merrill, M. Garland, and A. Grimshaw. Scalable GPU graph traversal. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 117–128, Feb. 2012.

[6] A. E. Sariyüce, K. Kaya, E. Saule, and U. V. Çatalyürek. Betweenness centrality on GPUs and heterogeneous architectures. In *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, GPGPU-6, pages 76–85, Mar. 2013.

[7] J. Soman, K. Kishore, and P. J. Narayanan. A fast GPU algorithm for graph connectivity. In *24th IEEE International Symposium on Parallel and Distributed Processing, Workshops and PhD Forum*, IPDPSW 2010, pages 1–8, Apr. 2010.

[8] J. Zhong and B. He. Medusa: Simplified graph processing on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1543–1552, June 2014.