Multi-GPU Graph Analytics

[Extended Abstract]

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

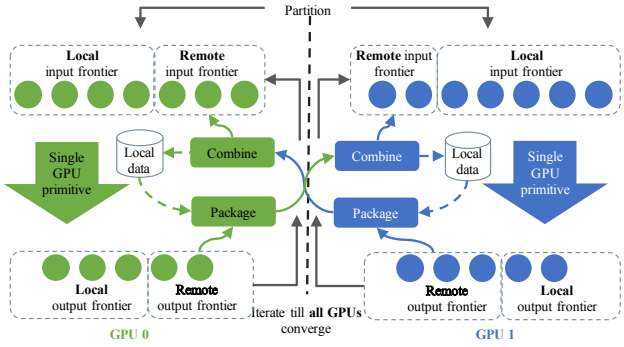

Fig. 1: mGPU Framework highlighting inter-GPU communications

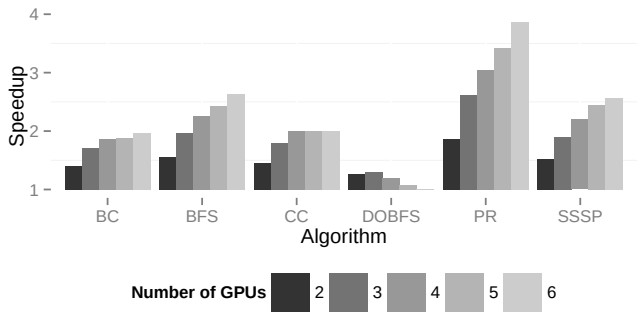

Fig. 2: mGPU speedup over single-GPU performance for BC, BFS, CC, DOBFS, PR and SSSP. All numbers shown are geometric means of runtime speedup over all soc, web and rmat datasets.

I. Introduction

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 [7] focuses on frontier operations. 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. More importantly, once an optimization is realized, it is not limited to a single algorithm, but can be applied to all applicable primitives.

II. Framework

In order to harvest more computing power and combined memory space, we extended Gunrock onto multiple GPUs. As shown in Fig. 1, our multi-GPU framework performs inter-GPU communication at the iteration boundary, and provides a transparent view to the single GPU primitive. The framework hides implementation level details from programmers for better programmability. To achieve this without loosing algorithm generality, programmers need to specify the following necessary information:

Core single-GPU primitive Use Gunrock operators to define a series of operations on input frontiers.

Data to communicate What kinds of data associated with vertices must be pushed to remote GPUs?

Combining remote and local data Specify the operation (e.g., minimum) to combine local and (possibly multiple) received data at the beginning of an iteration, except the first one.

Stop condition Define the local and/or global stop condition so that each local GPU will properly exit its computing iteration when the algorithm finishes.

The framework handles all other aspects of the computation:

Split frontier Split the output frontier of a iteration into local and remote sub-frontiers.

Package data Package the remote sub-frontiers with the associated data that specified by the programmer. Data packaging can be done together with frontier splitting.

Push to remote GPUs Manage communication so that each GPU pushes the right information to the right GPU for use in the next iteration.

Merge local and received sub-frontiers Using the programmer-specified combiner, efficiently merge the local sub-frontier with all received sub-frontiers to get the input frontier for the next iteration.

Manage GPUs Each GPU is managed by a dedicated CPU thread and GPU streams in the framework, to avoid false dependency between GPUs.

graph	algo	ref.	ref. hw.	ref. perf.	our hw.	our perf.	comp.
com-orkut (3M, 117M, UD)	BFS	Bisson [3]	1×K20X×4	2.67 GTEPS	4×K40	14.22 GTEPS	5.33X
com-Friendster (66M, 1.81B, UD)	BFS	Bisson [3]	1×K20X×64	15.68 GTEPS	4×K40	14.1 GTEPS	0.90X
kron_n23_16 (8M, 256M, UD)	BFS	Bernaschi [2]	1×K20X×4	∼1.3 GTEPS	4×K40	30.8 GTEPS	23.7X
kron_n25_16 (32M, 1.07G, UD)	BFS	Bernaschi [2]	1×K20X×16	∼3.2 GTEPS	6×K40	31.0 GTEPS	9.69X
kron_n25_32 (32M, 1.07G, D)	BFS	Fu [4]	2×K20×32	22.7 GTEPS	4×K40	32.0 GTEPS	1.41X
kron_n23_32 (8M, 256M, D)	BFS	Fu [4]	2×K20×2	6.3 GTEPS	4×K40	27.9 GTEPS	4.43X
kron_n24_32 (16.8M, 1.07G, UD)	BFS	Liu [5]	2×K40×1	15 GTEPS	2×K40	77.7 GTEPS	5.18X
kron_n24_32 (16.8M, 1.07G, UD)	BFS	Liu [5]	4×K40×1	18 GTEPS	4×K40	67.7 GTEPS	3.76X
kron_n24_32 (16.8M, 1.07G, UD)	BFS	Liu [5]	8×K40×1	18.4 GTEPS	4×K80	40.2 GTEPS	2.18X
twitter-mpi (52.6M, 1.96G, D)	BFS	Bebee [1]	1×K40×16	0.2242 sec	3×K40	94.31 ms	2.38X
rmat_n21_64 (2M, 128M, D)	BFS	Merrill [6]	4×C2050×1	8.3 GTEPS	4×K40	23.7 GTEPS	2.86X

TABLE I: Comparison with previous in-core GPU graph processing work. Ref. hardware is denoted by intra-node GPU count×GPU model×node count. We use the same number of GPUs whenever possible within the constraints of a single node.

III. Optimizations

We incorporated several optimizations into the framework, to increase performance, or to decrease GPU memory usage.

Direction-optimizing traversal We implemented a new multi-GPU friendly direction-optimizing breadth first search (DOBFS) with three optimizations: a new direction choosing method based on estimated forward and backward edge visits, a new backward advance kernel and a separated frontier for unvisited vertices.

Idempotence We extended the idempotence support from 1GPU to multi-GPU, by marking the bitmask when remote and local data are combined. It's key to avoiding atomic operations and achieving good BFS scalibility.

Kernel Fusion We fused an advance+filter kernel pair if the filter directly follows the advance. This eliminated the memory space requirement for the intermediate frontier.

Compute/communicate overlap Our multi-GPU framework separates primitive computation and communication tasks into different GPU streams. We synchronize and establish dependencies between GPUs without CPU intervention by using `cudaStreamWaitEvent()`.

Just enough memory allocation The resulted frontier size is only known at the start or even finish of a advance or filter kernel, which often leads to over-provision of already limited GPU memory space to accommodate the largest possible output size. We solved this issue by making a reasonable estimation to pre-allocate memory before computation, and reallocating when allocated space is insufficient.

IV. Results

The overall speedup of all the primitives is shown in Fig. 2, normalized to the performance of a single GPU as 1. The speedup of a given primitive using a given number of GPUs is the averaged speedup from 16 datasets (including social, rmat and web graphs, with the number of vertices in [1M, 118M], the number of edges in [85.7M, 1.71B] and the graph diameter in [5, 379].) tested for that configuration. Most of the primitives scale well from 1 to 6 GPUs, resulting in 2.63X, 2.57X, 2.00X, 1.96X and 3.86X speedup for BFS, SSSP (single source shortest path), CC (connected component), BC (betweeness centrality) and PR (Page rank) respectively using 6 GPUs. The performance of DOBFS mostly stays flat, as it is limited by communication overhead.

When comparing to similar works (Table I), we achieved significant speedups when using the same or similar hardware (single node multi-GPU systems), or at least comparable performance with small GPU clusters.

V. Future work

We are actively improving Gunrock, by carrying out in depth performance analysis and optimization, extending it onto multiple nodes, and enabling more partitioning options, as well as more graph algorithms, including asynchronized ones. These works will appear in future submissions/publications.

References

[1] B. Bebee, "What to do with all that bandwidth? GPUs for graph and predictive analytics," 21 Mar. 2016, https://devblogs.nvidia.com/parallelforall/gpus-graph-predictive-analytics/.

[2] M. Bernaschi, G. Carbone, E. Mastrostefano, M. Bisson, and M. Fatica, "Enhanced GPU-based distributed breadth first search," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ser. CF '15, 2015, pp. 10:1–10:8.

[3] M. Bisson, M. Bernaschi, and E. Mastrostefano, "Parallel distributed breadth first search on the Kepler architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, Sep. 2015.

[4] 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*, Oct. 2014, pp. 110–118.

[5] H. Liu and H. H. Huang, "Enterprise: Breadth-first graph traversal on GPUs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15, Nov. 2015, pp. 68:1–68:12.

[6] 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*, ser. PPoPP '12, Feb. 2012, pp. 117–128.

[7] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the GPU," in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP 2016, Mar. 2016. [Online]. Available: http://escholarship.org/uc/item/6xz7z9k0