Literature Review

Cheng Liu

2016-10-27

1 Gunrock

Gunrock [5] is a data-centric graph processing framework which mainly handles graphs that can be expressed with iterative convergent processes. It is also based on the BSP model. Basically, the graph problems are divided into steps and each step must be synchronized. Advance-filter-compute are the typical primitive steps used in Gunrock. These operations are performed on top of graph frontiers which is used in many graph processing framework. Major optimization strategies used in Gunrock are listed below.

- Kernel fusion: Inspired by the primitive GPU optimization strategy which leverages the producer-consumer locality between operations, Gunrock tries to integrate advance, filter and user-specific functor into a thread to improve performance as well as memory access efficiency.

- Workload balance: Gunrock's advance step which is applied on graph frontier result in severe load balancing problem especially for graphs with power-law distribution just as any other similar framework. To solve this problem, Gunrock roughly divides the vertices in the frontier into three groups based on the size of the associated neighbor lists. Vertices with larger size of neighbor lists will be executed in parallel in a cooperative thread array (CTA). Vertices with medium size will be executed in parallel in a warp. Vertices with small size will be executed in separate threads. When multiple vertices are assigned to a CTA or warp, the vertex with largest size of associated neighbor list will be distributed to all the threads of the CTA or warp and be executed first. Then all the vertices will be executed sequentially with the same logic.

- Idempotent vs. non-idempotent operations: As vertices in current frontier may share the same neighbors, there are duplicate vertices when producing the next frontier. Gunrock removes some of the duplicate vertices with inexpensive heuristics for applications that allows duplicated vertices in frontier. This strategy helps to improve performance. For applications that can't tolerate duplicated vertices, it can also remove duplicate vertices completely.

- Pull and push traversal: This is the same with the top-down and bottom-up traversal strategy used in Ligra.

- Priority queue: Usually all the vertices in the frontier are treated equally in BSP model while Gunrock divides them into two queues based on a criterion to save work.

2 Work-efficient Parallel GPU Methods for SSSP

The algorithm developed in this work [1] is based on delta-stepping algorithm. It consists of the following steps.

- The vertices are divided into one or multiple buckets. Vertices within a bucket is processed in parallel.
- Traverse the vertices in a bucket. Load balancing is the key challenge in this step.
- Decide vertices to be processed in next iteration.

This work comes from the same group of Gunrock and it focuses on SSSP optimization on GPUs. Although the optimization techniques used target SSSP, they are generalized and ported to Gunrock. Thus this work is reviewed for more details about the graph optimization techniques. Here are the highlights of this SSSP optimization techniques.

- load balanced graph traversal:

  Group blocking: The edges of the vertices within a block are stripped from each vertex's edge list and processed by a cooperative thread array (CTA) in parallel. Threads within a block is load-balanced, but load-imbalance may still exist between the blocks. Particularly, when the vertex degree is small, this method is not efficient.

  CTA+Warp+Scan [3]: The basic idea is to divide the vertices into three categories based on the size of the edge list. The method is applied in Gunrock as described in last section.

  Edge partition: Instead of grouping equal number of vertices in each block, this method organizes the groups of edges with equal length to ensure strict load balance within a block.

- Work organization which essentially decides the vertices to be processed in next iteration. Again three different methods are proposed.

  xxx

  xxx

  xxx

Here is a summary of SSSP algorithms.

Algorithms	**Wk. Complexity**	**Type**	**Parallelism**
Dijkstra	$O(v \log v + e)$	General	Serial
Bellman-Ford	$O(ve)$	High Degree	Parallel
Delta Step	$O(v \log v + e)$	General	Coarse Parallel
PHAST	$O(v \log v + e)$	Low Degree	Preprocessing Parallel

Table 1: SSSP Algorithms

Dijkstra algorithm implemented with a priority queue is efficient as a sequential algorithm but expose little parallelism for parallel computing architectures.

PHAST [2]: It has a Dijkstra-like preprocessing step to pre-compute distances to vertices of high-degree. Then the Dijkstra algorithm can start from these highly ranked vertices in parallel. This algorithm works well on low-degree and high-diameter graphs.

Delta Step [4]: Instead of processing one vertex at a time in Dijkstra algorithm, it groups vertices in buckets and process vertices in a bucket in parallel. In delta-stepping, the vertices are grouped into buckets depending on distances of the vertices from the source.

Major challenges for Delta step algorithm on GPUs.

- Delta-stepping's bucket implementation requires dynamic array that can be quickly resized in parallel.

- Fine-grained renaming and moving vertices between buckets are difficult and inefficient.

- GPU memory hierarchy is not well explored.

Bellman-Ford: It is a standard parallel algorithm for SSSP problem. Each vertex maintains the distance to the source and has neighbor vertices information updated iteratively. The algorithm completes when the algorithm converges. This algorithm suffers load imbalance for graphs with power-law distribution. Race condition occurs when the update is parallelized and atomic update is needed.

References

[1] Andrew Davidson, Sean Baxter, Michael Garland, and John D. Owens. Work-efficient parallel gpu methods for single-source shortest paths. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, IPDPS '14, pages 349–359, Washington, DC, USA, 2014. IEEE Computer Society.

[2] Daniel Delling, Andrew V Goldberg, Andreas Nowatzyk, and Renato F Werneck. Phast: Hardware-accelerated shortest path trees. *Journal of Parallel and Distributed Computing*, 73(7):940–952, 2013.

[3] Duane Merrill, Michael Garland, and Andrew Grimshaw. Scalable gpu graph traversal. In *ACM SIGPLAN Notices*, volume 47, pages 117–128. ACM, 2012.

[4] Ulrich Meyer and Peter Sanders. δ-stepping: a parallelizable shortest path algorithm. *Journal of Algorithms*, 49(1):114–152, 2003.

[5] Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John 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*, page 11. ACM, 2016.