Graph Acceleator Review

Cheng Liu

November 10, 2016

Basic Vertex Processing Framework

Here is the basic vertex-centric processing flow.

- Initialize Active Vertex List
- For each active vertex, get associated Edge List
- For each edge in the Edge List, update associated vertices
- Update Active Vertex List

Iteratively perform the vertex processing with BSP model until the computing converges. Graph operation abstraction in different graph frameworks: Advance-Compute-filter, Gather-Scatter-Apply, EdgeMap-VertexMap, ProcessEdge-Reduce-Apply ...

General Graph Optimization Rules

- Exploit locality of the graph data
- Reduce amount of data tranfer or computing
- Explore more parallelism
- Explore load balancing strategies

Graph Partition

The graph are divided into sub-graphs based on various strategies.

- **Graph slicing:** Graph IDs are used in [3] to divide the vertices and edges into different groups. Dependent vertices are replicated.
- **Interval and Shard:** Graph IDs are reordered and equally divided as intervals. Based on the vertex intervals, the edges are further split into shards. Basically the graph are divided in two dimensions [2, 1]. Note that it needs pre-processing.

Vertex Related Optimization Techniques

The optimization techniques concentrates on the vertices in the active vertex list. The basic idea is to divide the vertices in the list into groups with different priority and process them separately. This helps on both balance the load on parallel computing architectures and reduce the computing work in some of the applications.

- Bottom-up (push) and Top-down (pull) Vertex Traversal [7, 6]
- Remove Redundant Vertices in Vertex Active List [7]
- Priority Queue for Active Vertex List [6]
- Near-Far pile, delta bucketing [7, 5]

Edge Related Optimization Techniques

The edge optimization techniques mostly aim to achieve load balancing on the parallel architectures.

- Naive vertex based allocation
- Group Blocking
- CTA (Cooperative Thread Array) + WARP [7, 4]
- Equally edge partition [7]

Memory Access Optimization

- **Double-Buffer:** Iteratively swap vertex buffers of vertex data in current iteration and next iteration.
- **eDRAM:** For random edge or vertex access, a large eDRAM can be used as a on-chip scratchpad memory [3].
- **Pre-Fetch:** When all the edges or vertices are to be accessed, sequentially accessing all the data with pre-fetching can hide memory access latency completely [3].

Reference

Y. Chi, G. Dai, Y. Wang, G. Sun, G. Li, and H. Yang.

Nxgraph: An efficient graph processing system on a single machine.
CoRR, abs/1510.06916, 2015.

G. Dai, Y. Chi, Y. Wang, and H. Yang.

Fpgp: Graph processing framework on fpga a case study of breadth-first search.
In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 105–110, New York, NY, USA, 2016. ACM.

T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi.

Graphicionado: A high-performance and energy-efficient accelerator for graph analytics.

D. Merrill, M. Garland, and A. Grimshaw.

Scalable gpu graph traversal.
In *ACM SIGPLAN Notices*, volume 47, pages 117–128. ACM, 2012.

U. Meyer and P. Sanders.

δ-stepping: a parallelizable shortest path algorithm.
Journal of Algorithms, 49(1):114–152, 2003.

J. Shun and G. E. Blelloch.

Ligra: A lightweight graph processing framework for shared memory.
SIGPLAN Not., 48(8):135–146, Feb. 2013.

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*, page 11. ACM, 2016.