Towards Realizing Topology Mutation for Iterative Graph Processing on a GPU

Yasuaki Mitani, Fumihiko Ino and Kenichi Hagihara
Graduate School of Information Science and Technology, Osaka University
m-yasuak@ist.osaka-u.ac.jp

Abstract

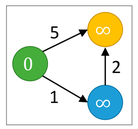
We present and discuss how a dynamic graph can be processed efficiently on a GPU. In particular, **irregular dynamic data structure** is required to realize topology mutation, which is a fundamental operation for solving some important classes of graph problems. We propose **an iterative graph processing framework capable of efficient addition and deletion of edges needed for topology mutation**. We compare our framework with an existing framework, namely Medusa, to understand the impact of our dynamic data structure in several graph algorithms.

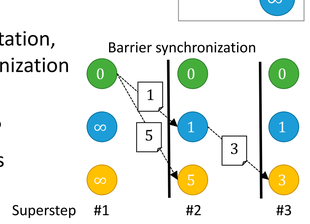
Motivation

Existing iterative graph processing frameworks [1] have demonstrated that large-scale graph processing can be successfully accelerated using GPUs; however, the previous approach assumes that the graph is given as a fixed data. This static data structure facilitates maximizing the execution efficiency on the GPU: the graph can be effectively stored in global memory such that vertices in the graph can receive messages with memory coalescing. However, such a data structure heavily relies on the connectivity of vertices of the graph. Thus, a more flexible data structure is needed to deal with dynamic graphs that can change their connectivity at run time.

Pregel-like iterative graph processing [2]

- Superstep-based parallel processing
  - A superstep consists of local computation, communication and barrier synchronization
- Description needed for supersteps
  - Which vertices exchange messages?
  - How per-vertex and per-edge values should be updated?

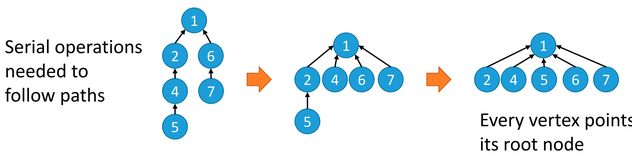




A snapshot of Bellman-Ford algorithm

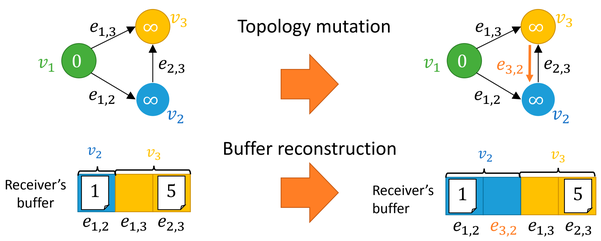
Topology mutation

- Useful to realize pointer jumping employed in many parallel graph algorithms
  - Pointer jumping follows all paths simultaneously and shares results among dependent operations
  - ex. Minimum spanning tree problem



Medusa [1]

- Each vertex has **a fix-sized array** to receive messages
  - Each array element is dedicated to an incoming edge
  - Array elements are sorted by vertex ID to realize **memory coalescing** when retrieving messages
- Buffer reconstruction is needed when adding/deleting a vertex to/from the graph



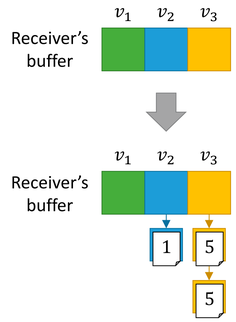
Proposed Method

Contrary to the previous approach, which employs array-based data structure, our framework allows vertices to have own list for receiving messages. By using this list-based data structure, a message can be sent by insertion into the receiver's list. Our data structure is independent from the connectivity of vertices of the graph, so that edges can be rapidly added and deleted without reconstruction of message buffer.

List-based data structure for message passing

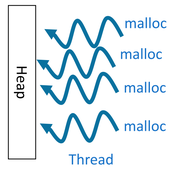
- Each vertex has **a variable-sized list** to receive messages
- The sender adds its messages to the receiver's list

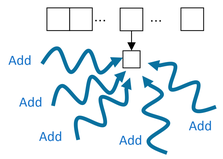
➡ **No reconstruction needed when adding/deleting a vertex**



Challenging issues to implement list structure on a highly-threaded GPU

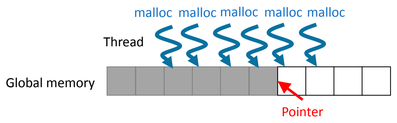
- Realizing fast, dynamic memory allocation/deallocation
  - Thousands of threads can send messages at every superstep
- Realizing exclusive but rapid access to the list
  - Thousands of threads can access the same list simultaneously



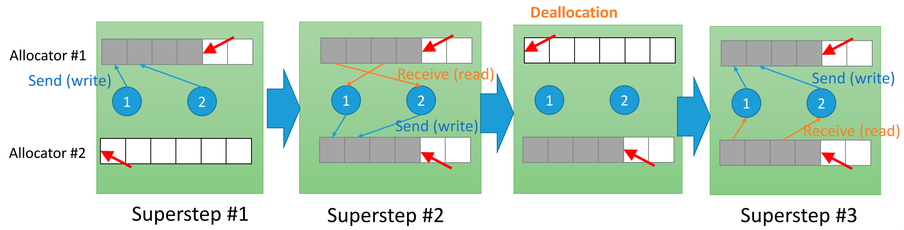


Dynamic memory allocation

- We extend a memory allocator designed for MapCG [4]
  - A pointer is used to indicate the boundary of used/unused region
  - O(*w*) time for allocation of memory region, where *w* is the number of warps
  - O(1) time for deallocation of all of the allocated memory region
  - Restriction: **partial deallocation not available**

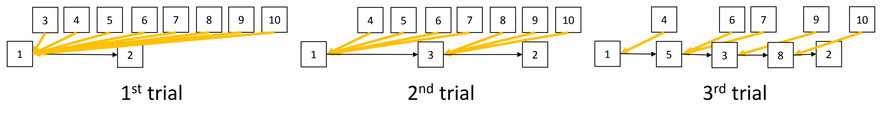


- Considering this restriction, we integrate this allocator into **a double buffering technique**
  - Two allocators are alternatively used at every superstep to avoid duplication of incoming messages



Exclusive access to the list

- We use **a tree-based parallel insertion scheme** to realize fast message passing with a less number of conflicts
  - A half of threads follows the next pointer at every trial
- Messages can be added in O(log *m*) time in average, where *m* represents the number of messages to be sent to the same vertex



Experimental Results

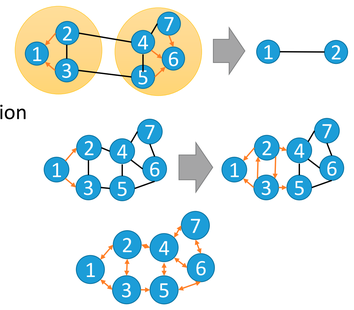
We compared our framework with Medusa, a GPU-accelerated framework that uses a static data structure for fixed graphs. For the Borůvka algorithm, our framework was three times faster than Medusa. In contrast, our framework was 50-90% slower than Medusa for the PageRank algorithm, which does not mutate graph topology. Similarly, the Bellman-Ford algorithm never requires topology mutation. However, we unexpectedly found that our framework achieved the best speedup of 1.3x. This higher performance came from our list-based data structure, which allocates memory region only for actually sent messages. In contrast, an array-based data structure allocates memory region for incoming edges (i.e., possible messages), and thus, the memory bandwidth was wasted due to this unfilled buffer.

Setup

CPU	Intel Core i7-4770K CPU @ 3.50GHz
GPU	GeForce GTX 780
OS	Ubuntu 12.04.3 LTS
CUDA	5.5

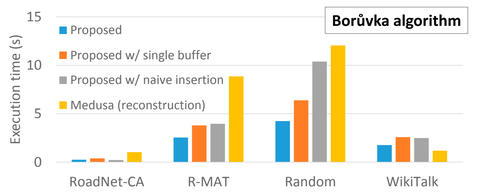
Dataset	|V| (10^6)	|E| (10^6)	Max degree	Avg degree	Variance
RoadNet-CA	2.0	5.5	12	2.8	1.0
R-MAT	1.0	16.0	1,742	16	32.9
Random	1.0	16.0	28	16	4.0
WikiTalk	2.4	5.0	100,022	2.1	99.9

- Borůvka algorithm [4] with topology mutation
  - The minimum spanning tree problem
  - A few vertex has many incoming messages
- Bellman-Ford algorithm without topology mutation
  - The single-source shortest path problem
  - A few vertex transmits messages to others
- PageRank algorithm without topology mutation
  - The website ranking problem
  - All vertices exchange messages



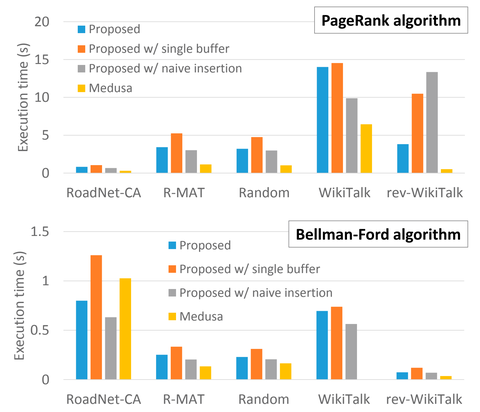
Performance evaluation

- Double buffering technique achieves 1.5x speedup at best
- Tree-based insertion achieves 3x speedup at best, but decreases performance if messages do not gather at a specific vertex



Comparison with Medusa

- For Borůvka algorithm, our method was 3x faster because it avoids buffer reconstruction
- For PageRank algorithm, our method was 10-50% slower because many vertices send messages at every superstep
- For Bellman-Ford algorithm, our performance varies according to communication pattern
  - 1.3x faster at best
  - 50% slower at worst



References

[1] Jianlong Zhong and Bingsheng He, "Medusa: Simplfied graph processing on GPUs," IEEE Trans. Parallel and Distributed System, 25(6):1543-1556, June 2014.
[2] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser and Grzegorz Czajkowski, "Pregel: A System for Large-Scale Graph Processing," Proc. *SIGMOD'10*, pp.135-145, June 2010.
[3] Chuntao Hong, Dehao Chen, Wenguang Chen, Weimin Zheng, Haibo Lin, "MapCG: Writing parallel program portable between CPU and GPU," Proc. PACT '10, pp.217-226, Sept. 2010.
[4] Semih Salihoglu and Jennifer Widom, "Optimizing Graph Algorithms on Pregel-like Systems," Proc. VLDB'14, pp.577-588, Sept. 2014.

This study was partly supported by the JST CREST program "An Evolutionary Approach to Construction of a Software Development Environment for Massively-Parallel Computing Systems."