LITERATURE REVIEW:
Dynamic Graphs on the GPU

Jimmy Lord
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
jimmylord@cmail.carleton.ca

October 13, 2020

1 Introduction

Graphs are data structures fundamental to many areas of computer science. They are used in the study of countless real-world problems including, but not limited to, physical network design (telecommunications, electrical grids, water/sewer networks, etc), message routing, recommendation engines, fraud detection, advertising and machine learning.

Many graph algorithms exist to aid in these applications. These algorithms include search algorithms, such as breadth-first and depth-first search, shortest path algorithms, such as Dijkstra's algorithm and the A* algorithm, subgraph generation algorithms, such as Kruskal's algorithm for generating a minimum spanning tree, connected components, sorting algorithms, minimum spanning tree, page rank, centrality measurements and many more. These algorithms have been implemented and optimised for a variety of system architectures, be it a sequential design for a single CPU or designed for parallelism with multiple local CPUs or multiple CPUs connected via a network.

In recent years, researchers have been working to improve the performance of these graph algorithms on GPUs, which has it's own set of challenges due to memory restrictions and massive parallelisation.

Also, for many of these applications it is not enough to rely on static graphs, they need to support insertion and deletion of nodes within the graph structure either during the execution of a query or otherwise.

This brings us to the topic of this paper, efficient dynamic graph data structures that support quick insertion and deletion of nodes as well as continuing to provide fast queries for the various graph algorithms used in common practice.

2 Literature Review

When it comes to building a data structure to support the creation and querying of graphs, a lot of factors come into play, such as the density of connections between the nodes in the graph and whether the graph nodes and edges are static or able to change over time. This section will explain some of choices that have been made in the past to help support the needs of the graph algorithms, focusing on choices made for the parallelism offered by GPUs.

2.1 Graph Density

When the graph density is high, with many of the nodes of a graph connected to many of the others, the list of edges is often maintained in an adjacency matrix. Due to the memory requirements of these matrices, the maximum number of vertices can be severely restricted as seen in experiments by Buluc et al. [5].

More often, the graphs being studied are sparse, with few connections between their nodes, which by nature is generally true of large graphs. These graphs store their edges in adjacency lists, which in turn leads to much more variety of possible data structures, each with their own trade-offs between query performance, insertion/deletion speed and other properties.

2.2 Static Graph Data Structures

Much of the literature on graphs algorithms on the GPU is focused on storing and querying static graph data structures. These data structures don't accommodate for edges to be inserted and removed from the graph after it has been built and often require a complete rebuild of the structure if any changes are needed. Many implementations of static graphs use data structures such as Compressed Sparse Row (CSR) and other similar structures to store graph edge lists. These formats are chosen for their compactness and speed of query. Static graphs are often built on the CPU and later transferred to GPU memory. An example of this can be found this paper by Bisson et al. [4]

2.3 Dynamic Graph Data Structures

In recent years, more and more work has been done on dynamic graphs, which unlike static graphs allow for the addition and removal of vertices and edges without rebuilding the entire graph. This leads us to the focus of this paper, data structures for use in dynamic graphs on the GPU. There have been three important frameworks focused on working with dynamic graphs. The following subsections detail some of the efforts.

As summary of the features of the following frameworks can be found in Table 1.

2.3.1 STINGER, cuSTINGER and Hornet

In 2009, the STINGER [3] algorithm for dynamic graphs was proposed. STINGER is a dynamic data structure that allows for the insertion and removal of both vertices and edges. Later, in 2012, STINGER [7] was updated with some optimizations, including parallel implementations of the insertion and removal procedures for edges which led to a massive increase in performance of the algorithm.

This later led to the introduction of cuSTINGER [8] in 2016, which offered a GPU extension of the STINGER data structure designed for CUDA. cuSTINGER provides memory management for dynamic or static graph applications with multiple different allocators that can be configured per application.

In 2018, the Hornet [6] data structure was introduced, a GPU data structure designed for dynamic graphs representing sparse data sets. Hornet is a custom data structure designed for the GPU that offers full dynamic batched updates to both the nodes and the edges of a graph while offering a level of stability after large changes to the graph that previous efforts didn't.

2.3.2 GPMA and GPMA+

In their paper [9], the authors introduce a new data structure for storing graphs on the GPU named GPMA. GPMA is a GPU variation of the Packed Memory Array data structure, a self-balancing binary tree, which maintains the adjacencies in a sorted order. This technique has advantages for certain graph algorithms that benefit from having sorted adjacency lists. Though this comes at the cost of a more expensive mechanism for modifying the edge list due to memory locking that prevents modifications of multiple entries in similar positions in the same pass.

Also presented in the same paper is GPMA+, which is a lock-free version of GPMA that sorts the graph changes on the CPU and sends them to the GPU in batches to avoid the locking mechanism.

2.3.3 aimGraph and faimGraph

In 2017, aimGraph [11] was introduced, similar to cuSTINGER at the time, it allowed for dynamic insertion and removal of edges in its adjacency lists, but not for vertices. One big innovation of aimGraph was that it managed memory for the adjacency lists on the GPU, allowing for faster reallocations when the lists got full.

Winter et al. followed up their work on aimGraph with faimGraph [10] in 2018, a fully dynamic data structure that allows for insertions and removal of both edges and vertices from a graph, previous work only allowed for dynamic edges, not vertices. This new implementation performs all memory management on the GPU

2.3.4 Dynamic Graphs on the GPU

In 2020, Awad et al. released a new framework [2] which implements a dynamic graph structure using the SlabHash [1] dynamic GPU hash table to store the edge lists in a manner that supports fast insertions and deletions. The authors of this paper run a triangle counting algorithm to test the speed of their dynamic graph compared to faimGraph and Hornet. They compared both the speed of creating and maintaining the graph after many insertions and deletions as well as the speed of the triangle counting algorithm.

Name	GPU	Vertices	Edges	Memory Management
GPMA	Yes	Fixed	Dynamic	CPU/GPU
STINGER	No	Fixed	Dynamic	CPU
cuSTINGER	Yes	Fixed	Dynamic	CPU/GPU
Hornet	Yes	Dynamic	Dynamic	CPU/GPU
aimGraph	Yes	Fixed	Dynamic	GPU
faimGraph	Yes	Dynamic	Dynamic	GPU
Awad et al.	Yes	Dynamic	Dynamic	GPU

Table 1: Summary of features of various graph implementations.

References

[1] S. Ashkiani, M. Farach-Colton, and J. D. Owens. A dynamic hash table for the gpu. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 419–429, May 2018.

[2] M. A. Awad, S. Ashkiani, S. D. Porumbescu, and J. D. Owens. Dynamic graphs on the gpu. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 739–748, May 2020.

[3] D. Bader, J. Berry, A. Amos-Binks, D. Chavarria-Miranda, C. Hastings, and K. Madduri. "STINGER: Spatio-temporal interaction networks and graphs (STING) extensible representation", 2009.

[4] M. Bisson and M. Fatica. Static graph challenge on gpu. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8, Sep. 2017.

[5] Aydın Buluç, John R. Gilbert, and Ceren Budak. Solving path problems on the gpu. *Parallel Computing*, 36(5):241 – 253, 2010. Parallel Matrix Algorithms and Applications.

[6] F. Busato, O. Green, N. Bombieri, and D. A. Bader. Hornet: An efficient data structure for dynamic sparse graphs and matrices on gpus. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–7, Sep. 2018.

[7] D. Ediger, R. McColl, J. Riedy, and D. A. Bader. Stinger: High performance data structure for streaming graphs. In *2012 IEEE Conference on High Performance Extreme Computing*, pages 1–5, Sep. 2012.

[8] O. Green and D. A. Bader. custinger: Supporting dynamic graph algorithms for gpus. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, Sep. 2016.

[9] Mo Sha, Yuchen Li, Bingsheng He, and Kian-Lee Tan. Accelerating dynamic graph analytics on gpus. *Proc. VLDB Endow.*, 11(1):107–120, September 2017.

[10] M. Winter, D. Mlakar, R. Zayer, H. Seidel, and M. Steinberger. faimgraph: High performance management of fully-dynamic graphs under tight memory constraints on the gpu. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 754–766, Nov 2018.

[11] M. Winter, R. Zayer, and M. Steinberger. Autonomous, independent management of dynamic graphs on gpus. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, Sep. 2017.