Chapter 1

HavoqGT

Contributed by Brian Page

1.1 Introduction

In recent years, much attention has been paid to the graphs and graphs analytic. With climbing data set size and complexity, efficient graph algorithms are becoming increasingly important to researchers for a variety of reasons. Vertex-centric algorithms [7] have been at the forefront of scalable graph algorithm research and have lead to many advancements. With respect to scale free graphs and distributed memory graph traversal implementations the vertex-centric paradigm, such as HavoqGT [10], have proven quite successful in furthering progress towards increase scalablity and performance.

1.2 Background

HavoqGT is developed at Lawrence Livermore National Laboratory (LLNL) and follows the now common vertex-centric framework [3, 8, 9, 10]. It is intended for the development of efficient and scalable graph traversal algorithms. Using HavoqGT a developer can create separate behaviors for edge traversal and vertex interaction while performing load balancing of vertices with extremely high degree. This is case with scale free graphs in which vertex degree is proportional to the number of vertices with that degree [4].

In essence this means that while most vertices will have very few edges associated with them, a few vertices will have the vast majority of edges associated with them. This disparity is what makes scale free graphs challenging, since optimal workload distributions often involve splitting up an high degree vertex onto multiple compute nodes. HavoqGT seeks to alleviate some of the complexity and overhead associated with this process by utilizing the asyncrhonous visitor model (AVM) for graph traversal [9]. AVM allows a visitor, designed by the programmer, to traverse the graph and spawn or kill visitors as it works its way through [8, 9, 10].

1.3 Expressing Graphs

In its current form, HavoqGT can generate synthetic RMAT based graphs for evaluation purposes, or allow for input of graphs from file. HavoqGT is a framework written in the C++ programming language and allows the developer to implement a near endless number of possibilities [8, 9, 10].

The selection of input handling, while limited in the current development tree, can be amended to suite a multitude of options. For instance, the RMAT graphs generated are output as binary variants of a simple edgelist. Subsequent tweaking could easily allow for inputing sparse matrices in the form of coordinate format from a common benchmark library like the Suite Sparse Matrix Collection or SNAP [6, 5]

1.4 Syntax

Written in C++ and incorporating the Boost library [1] HavoqGT is a highly templated framework, intended to simplify implementation and to mitigate the precise properties of the graphs being evaluated. A class for type of visitor must be written, including methods to control behavior of the *visitor* as it traverses the graph. Calls to class objects and member methods are called in the standard manner for C++ 11 and later variants.

1.5 Key Graph Primitives

Graphs in HavoqGT has 3 primary components: vertices, edges, visitors. Vertices are distributed across the compute nodes, with the work associated with a given vertex calculated based on the size of its associated edge list. If a vertex has a very large $associatededgelist$, this vertex becomes a candidate for being split or distributed among multiple nodes. When a vertex is split in this fashion multiple copies of the vertex are made and distributed. These copies are called *delagates* and have the same properties as the original vertex, except that their local associated edge list contains only portion the entire edge list associated with that vertex as a whole.

A visitor is an instantiation of the traversal algorithm developed by the programmer and walks through the graph as designed. When a visitor comes to a vertex that has delegates, meaning the vertex has been split up and distributed, multiple copies of the visitor are generated and traverse the vertex locally on each node. The process in which a visitor checks a vertex to determine what course of action to take, is known as a *visit*. It is even possible to define the behavior actions prior to visiting a vertex, $pre - visit$, as well as after or $post - visit$. Often $post - visit$ involves terminating visitor object due to a reduction in edges to traverse.

Visitors traverse a graph as designed until the specific circumstances of their algorithm determine that a traversal has completed, or a particular metric has been calculated.

1.6 Execution Model

Execution of a HavoqGT based application will require either graph generation or the input of an existing graph source from file. Once the graph is loaded, identification and partitioning of delegate vertices is performed so that load balancing can occur. Next visitors are created and put in the work queue for the initial traversal. Once traversal has begun, each visitor performs according to its algorithm design. It is important to note that for optimal performance work load balancing and visitor algorithm design relies on minimizing movement to vertices not assigned to the current working node. Once all visitor queues are empty, meaning there are no other vertices which need to be traversed, traversal is terminated and any compute metric is returned.

1.7 Examples

Development of HavoqGT is ongoing and new visitor algorithms are being evaluated. Currently only a handful of traversal algorithms have been implemented including Breadth First Search (BFS) and Triangle Counting [2].

1.8 Conclusion

Discuss other topics that may make sense.

Bibliography

[1] Boost. https://www.boost.org.

[2] Havoqgt documentation. https://llnl.github.io/havoqgt/index.html.

[3] Havoqgt repository. https://github.com/LLNL/HavoqGT.

[4] Scale free graphs. https://en.wikipedia.org/wiki/Scale-free_network.

[5] Stanford large network dataset collection (snap). https://snap.stanford.edu/data/.

[6] Suite spare matrix collection. https://sparse.tamu.edu.

[7] Robert Ryan McCune, Tim Weninger, and Gregory R. Madey. Thinking like a vertex: a survey of vertex-centric frameworks for distributed graph processing. *CoRR*, abs/1507.04405, 2015.

[8] Roger Pearce, Maya Gokhale, and Nancy M. Amato. Multithreaded asynchronous graph traversal for in-memory and semi-external memory. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[9] Roger Pearce, Maya Gokhale, and Nancy M. Amato. Scaling techniques for massive scale-free graphs in distributed (external) memory. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, pages 825–836, Washington, DC, USA, 2013. IEEE Computer Society.

[10] Roger Pearce, Maya Gokhale, and Nancy M. Amato. Faster parallel traversal of scale free graphs at extreme scale with vertex delegates. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 549–559, Piscataway, NJ, USA, 2014. IEEE Press.