FemtoGraph: A Pregel Based Shared-memory Graph Processing Library

Alex Ballmer
Illinois Institute of Technology
alexandersballmer@gmail.com

Benjamin Walters
Illinois Institute of Technology
bwalter4@hawk.iit.edu

Ioan Raicu
Illinois Institute of Technology
iraicu@cs.iit.edu

ABSTRACT

The emerging applications for large graphs in big data science and social networks has led to the development of numerous parallel or distributed graph processing applications. The need for faster manipulation of graphs has driven the need to scale across large core counts and many parallel machines. While distributed memory parallel systems continue to be used for high performance computing, some smaller systems make use of shared memory (SMP) and larger core counts. We have implemented a graph processing framework for shared memory systems capable of scaling past 48 parallel cores. This system leverages and scale to large core counts and provide a framework for later incorporating distributed processing across multiple nodes.

CCS Concepts

•**Theory of computation** → **Shared memory algorithms;**

Keywords

Graph processing; Parallel Algorithms; Shared Memory

1. INTRODUCTION

FemtoGraph is a graph processing application which will use a vertex-centric approach. This approach involves calling a function in the context of a vertex for each and every vertex. This function can modify or read from edges and other vertices. Computation occurs in intervals or steps, with some form of communication between steps. Vertex-centric algorithms can be either synchronous, meaning every vertex function must finish before the next step begins, or asynchronous, which means that the next step can begin in the context of one vertex immediately. [1]

FemtoGraph is based off of the pregel model. Pregel is a vertex-centric graph processing model. [5] Pregel is synchronous, with computation occurring in steps called supersteps. There is a messaging system for sending data and state to vertices in the next superstep, but not to any vertex in the current superstep. In each step, vertices can do computations, modify neighbor vertices and edges, send messages to vertices in the next superstep, and vote to halt, meaning it cannot run its compute function until it receives a message or all vertices have voted to halt. When all vertices have voted to halt, the simulation ends. [5]

ACM ISBN 978-1-4503-2138-9…$-24.32
DOI: 10.1145/1235

2. RELATED WORK

One of the main similar applications in this area is GraphLab. GraphLab is another vertex-centric graph processing framework that can either run on a single node using shared memory, or as a distributed application. GraphLab is the simplest to compare to FemtoGraph as it can run with shared memory without the harsh overhead of frameworks like Hadoop or Spark. GraphLab is asynchronous, which gives it the edge of not having to wait for the fist superstep to complete before starting the next superstep. [6] Graphlab was the main point of comparison of FemtoGraph

3. IMPLEMENTATION

FemtoGraph is implemented in C++ using parts of the Boost C++ library. There are 3 main parts vertex storage, the message queue, and the compute function. Vertex storage also stores edges and uses an adjacency list using C++ vectors. The message queue is implemented using Boost lock-free queues, [7] one for each vertex. The compute function is a user defined function that is run in the context of each vertex during a pregel superstep. Update functions are called in parallel with a user defined number of threads.

4. DIFFICULTIES ENCOUNTERED

In the pregel model for graph processing, the main bottleneck encountered when scaling to many cores is the message queue. The message queue and the vertex storage are the only two data structures accessed from multiple threads. Vertices are not modified enough to warrant anything more than simple mutex based locking. The message queue, made clear by profiling with the callgrind function call analysis tool, is accessed at a very high rate from vertex compute functions from all threads. This can lead to race conditions, slowdowns, and deadlocks. A mutex based locking system resulted in an extreme slowdown to the point where the system scaled in reverse (Figure 1).

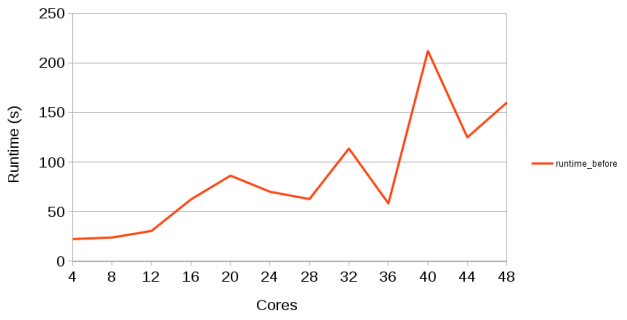

Figure 1: FemtoGraph scaling using mutex based message queue

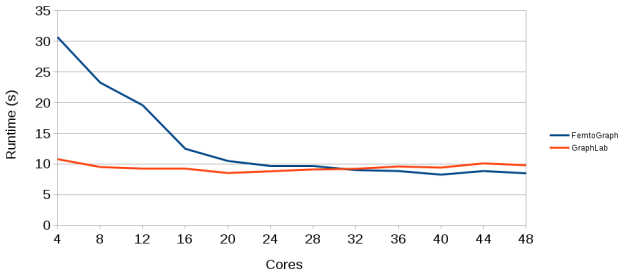

Figure 2: FemtoGraph in comparison to GraphLab on a single node

5. SOLUTION

My main solution for the message queue was to use a vector of Boost::lockfree queues for the message queue. [7] The vector was used to presort messages by vertex in order to minimize compute time sorting message when they were received. Adding new vertices still requires a mutex, as the vector is not lockfree. This mutex is not a problem for the algorithms that I tested, as they spend most of their time updating current vertices. The lockfree queues minimize the total bottleneck in the message queue.

6. RESULTS

We compared FemtoGraph and GraphLab on running Pagerank on a large Stanford SNAP graph (Figure 2). The graph used was the Wikipedia Talk Network, a directed graph based off of discussion about Wikipedia article edits with 2394385 Vertices and 5021410 edges. [2] The final version of FemtoGraph is capable of scaling to 48 cores on a single large system. At 28 cores, it begins to overtake graphlab in terms of runtime. GraphLab scales very weakly, only increasing in performace below 20 cores. After 20 cores, GraphLab scales in reverse (Figure 3). In the case of both applications, only compute time was counted. reading in data and initializing graphs were outside of the measured data.

7. CONCLUSION

FemtoGraph is a lightweight, single node graph processing system capable of scaling to large core counts and outperforming some of the current graph processing standards.

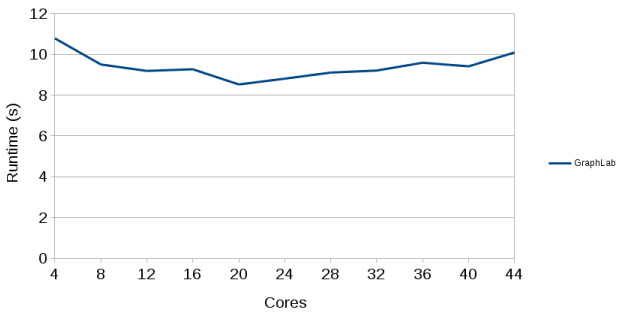

Figure 3: Graphlab scaling performance

FemtoGraph shows that the pregel model performs well under shared memory situations at scale.

8. REFERENCES

[1] O. Batarfi, R. El Shawi, A. G. Fayoumi, R. Nouri, A. Barnawi, S. Sakr, et al. Large scale graph processing systems: survey and an experimental evaluation. *Cluster Computing*, 18(3):1189–1213, 2015.

[2] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[3] T. Li, C. Ma, J. Li, X. Zhou, K. Wang, D. Zhao, I. Sadooghi, and I. Raicu. Graph/z: A key-value store based scalable graph processing system. *IEEE Cluster 2015*.

[4] T. Li, X. Zhou, K. Wang, D. Zhao, I. Sadooghi, Z. Zhang, and I. Raicu. A convergence of key-value storage systems from clouds to supercomputers. *Concurrency and Computation: Practice and Experience (CCPE) Journal 2015*.

[5] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[6] R. R. McCune, T. Weninger, and G. Madey. Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM Comput. Surv.*, 48(2):25:1–25:39, Oct. 2015.

[7] B. Schling. *The Boost C++ Libraries*. XML Press, 2011.

[8] D. Zhao, C. Shou, T. Malik, and I. Raicu. Distributed data provenance for large-scale data-intensive computing. *IEEE Cluster 2013*.