GPU Accelerated Graph Analytics Using Abstract Sparse Linear Algebra

Stephen T. Kozacik, Aaron L. Paolini, Paul Fox, James L. Bonnett, Evenie M. Chao, Eric Kelmelis

EM Photonics
Newark, DE

Dennis W. Prather
Department of Electrical and Computer Engineering
University of Delaware
Newark, DE

***Abstract*— Large-scale graph analytics using conventional processing systems often involves prohibitively excessive runtimes. Phrasing graph operations as abstract linear algebra operations offers potential for leveraging backend processing engines that scale better on modern heterogeneous computing platforms. To achieve this goal, we present a language for graph analytics that allows users to write in a familiar, vertex-centric API while leveraging the computational power of many-core accelerators. Our prototype toolchain automatically employs abstract sparse linear algebra (ASLA) operations using custom semi-rings in order to maximize performance.**

We use a C++ EDSL to map the user’s algorithm to efficient, fused ASLA operations at compile time with no runtime overhead. Using this technique, we have implemented several algorithms, including single-source shortest path, PageRank and single-source widest path. With a GPU-accelerated linear algebra backend, we can achieve better than 500% speedup over a multi-core ASLA library, using real and synthetic datasets.

Keywords—Graph Analytics, GPU Acceleration, Linear Algebra

I. Introduction

Graph analytics is a key component in identifying emerging trends and threats in many real-world applications in that it allows the analysis of interconnected systems such as social networks, commercial relationships, computer networks, logistics, and more. Large-scale graph analytics frameworks, such as Apache Giraph and Pregel [1], provide a convenient means of developing algorithms for distributed analytics on extremely large datasets. The “think like a vertex” programming model used by these frameworks allows algorithm developers to base their software on an intuitive understanding of relationships in the graph.

These frameworks use the bulk synchronous parallelism model and as such scale nicely to conventional computing clusters of many cores and machines that communicate by passing messages. However, in order to achieve maximum performance in an exascale computing environment the framework must be able to leverage the computing power of GPUs.

We and others have shown that linear algebra is able to efficiently use heterogeneous clusters with GPUs [2]. To take advantage of the fact that linear algebra is a well-studied and scalable problem on hybrid computers the GraphBLAS [3] [4] standard was created. GraphBLAS can be used to perform graph calculations as sparse matrix calculations on semirings. While this approach is computationally efficient, for many analysts, thinking of these calculations in terms of abstract matrix calculations is not as intuitive as the think like a vertex model.

In this paper we present an embedded domain-specific language in C++ for creating vertex-centric graph algorithms, converting them to abstract sparse linear algebra computations and accelerating these computations on the GPU.

II. Contributions

We have implemented a vertex-centric embedded domain specific language using expression templates [5]. This language is then processed using a series of overloaded functions. The resulting code is converted from a per vertex operation to whole graph operations. Next this operation is fused into abstract sparse linear algebra operations.

To leverage the processing power of the GPU, we have also created a GPU accelerated abstract sparse linear algebra library [6]. This is possible through use of the C++ template system and overloading the addition and multiplication operators to be the semiring operators. For performance we leverage the CUB library and their merge-based decomposition CSR format sparse matrix vector product [7].

III. Representative Results

A. Experimental Setup

To test the efficacy of both our frontend language and our backend library, we created implementations of three common graph algorithms, PageRank, single-source shortest-path, and single-source widest path. We tested these algorithms on three real-world datasets as well as on synthetic data of 9 different sizes.

We benchmarked our software on the following hardware:

- CPU: Intel Core i7-5930K CPU @ 3.50GHz (6 Cores, 12 Threads)
- RAM: 16 GB DDR4 2133 MHz
- GPU: Nvidia GeForce GTX Titan X with CUDA 7.5
- Software: Ubuntu 14.04, g++ 5.3, mpich2 3.0.4-6, CombBLAS 1.5.0 (9 Processes used)

Table 1 - Datasets used for benchmarking

Data Set	Vertices	Edges
Wikipedia [8]	3,566,907	45,030,389
LiveJournal [9]	4,847,571	68,993,773
cage15 [10]	5,154,859	99,199,551

Table 1 shows the three real-world datasets that we used for benchmarking. Tables 2, 3, and 4 show the results of the three tested algorithms on each of these data sets.

Table 2 - PageRank Runtimes in Seconds

	Data Set		
Platform	**Wikipedia**	**LiveJournal**	**cage15**
CPU	0.160325	0.262824	0.129189
GPU	0.0837797	0.12686	---
GPU w/Fusion	0.0266712	0.0264352	0.0100554

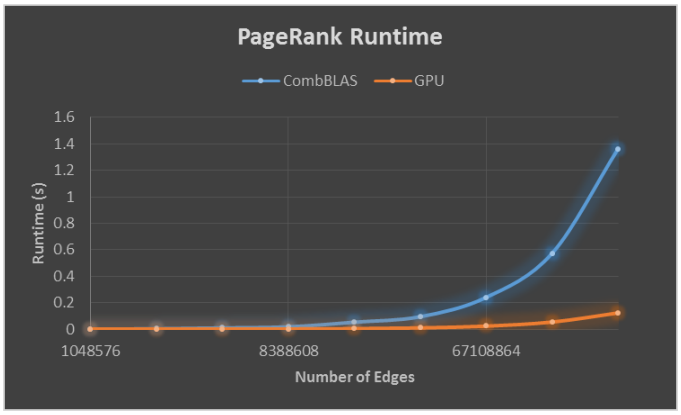

Figure 1 – PageRank runtimes on synthetic data generated using the R-MAT model

Table 3 – Single-Source Shortest Path Runtimes in Seconds

	DataSet		
Platform	**Wikipedia**	**LiveJournal**	**cage15**
CPU	0.201485	0.279654	0.15647
GPU	0.0701992	0.0916699	---
GPU w/Fusion	0.0244974	0.0234983	0.00801

Table 4 – Single-Source Widest Path Runtimes in Seconds

	DataSet		
Platform	**Wikipedia**	**LiveJournal**	**cage15**
CPU	0.220781	0.332494	0.135978
GPU	0.0882223	---	---
GPU w/Fusion	0.0271287	0.02976	0.0138946

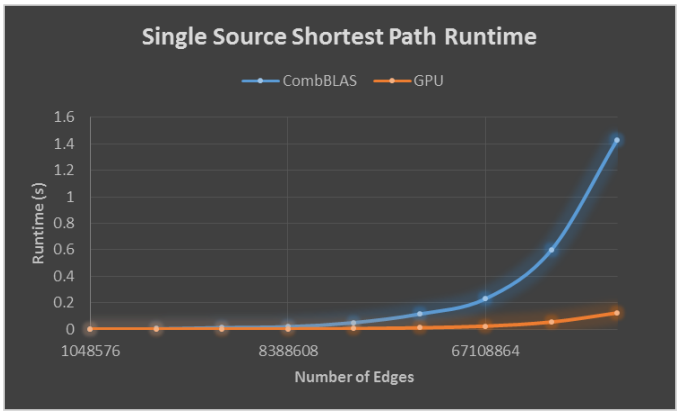

Figure 2 – Single-Source Shortest Path runtimes on synthetic data generated using the R-MAT model

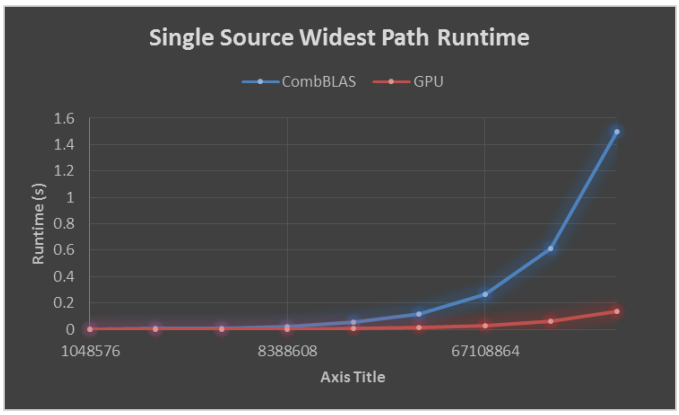

Figure 3 – Singe-Source Widest Path runtimes on synthetic data generated using the R-MAT model

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

For scaling analysis we used the Recursive-Matrix (R-MAT) model [11] with the “a” parameter set to .57 and the “b” and “c” parameters set to .19. Using this model, we generated directed graphs with edge counts from 2^{20} through 2^{28}, where there were 16 edges per vertex. To generate these datasets we used the PaRMAT multithreaded RMAT graph generator [12] with generation of duplicate edges disabled. Figures 8, 9, and 10 show the scaling results of the three algorithms.

For all tested datasets and algorithms we saw a speedup of at least 500% over the CPU implementation.

REFERENCES

[1] 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*, New York, NY, USA, 2010, pp. 135–146.

[2] J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis, “CULA: hybrid GPU accelerated linear algebra routines,” 2010, vol. 7705, pp. 770502-770502–7.

[3] J. Kepner, D. Bade, A. Buluc, J. Gilbert, T. Mattson, and H. Meyerhenke, “Graphs, Matrices, and the GraphBLAS: Seven Good Reasons,” *ArXiv150401039 Cs*, Apr. 2015.

[4] D. Bader, A. Buluc, J. Gilbert, J. Gonzalez, J. Kepner, and T. Mattson, “The Graph BLAS effort and its implications for Exascale,” presented at the SIAM workshop on Exascale Applied Mathematics Challenges and Opportunities, Chicago, IL, 2014.

[5] T. Veldhuizen, “C++ Gems,” S. B. Lippman, Ed. New York, NY, USA: SIGS Publications, Inc., 1996, pp. 475–487.

[6] S. Kozacik, A. L. Paolini, P. Fox, and E. Kelmelis, “Many-core graph analytics using accelerated sparse linear algebra routines,” 2016, vol. 9848, pp. 984808-984808–6.

[7] D. Merrill and M. Garland, “Merge-based Sparse Matrix-vector Multiplication (SpMV) Using the CSR Storage Format,” in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, New York, NY, USA, 2016, p. 43:1–43:2.

[8] “Gleich/wikipedia-20070206 sparse matrix.” [Online]. Available: http://www.cise.ufl.edu/research/sparse/matrices/Gleich/wikipedia-20070206.html. [Accessed: 16-Mar-2016].

[9] “SNAP/soc-LiveJournal1 sparse matrix.” [Online]. Available: http://www.cise.ufl.edu/research/sparse/matrices/SNAP/soc-LiveJournal1.html. [Accessed: 16-Mar-2016].

[10] T. A. Davis and Y. Hu, “The University of Florida Sparse Matrix Collection,” *ACM Trans Math Softw*, vol. 38, no. 1, p. 1:1–1:25, Dec. 2011.

[11] D. Chakrabarti, Y. Zhan, and C. Faloutsos, “R-MAT: A Recursive Model for Graph Mining,” in *Proceedings of the 2004 SIAM International Conference on Data Mining*, 0 vols., Society for Industrial and Applied Mathematics, 2004, pp. 442–446.

[12] “farkhor/PaRMAT,” *GitHub*. [Online]. Available: https://github.com/farkhor/PaRMAT. [Accessed: 06-May-2016].