Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

Proposal for Thesis Research in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Title: Unified Graph Framework: Achieving High-Performance across Algorithms, Graph Types, and Architectures

Submitted by: Ajay Brahmakshatriya
32, Vassar Street, 32-G788
Cambridge, MA 02139

Date of Submission: August 9, 2020

Expected Date of Completion: September 1, 2020

Laboratory: Computer Science and Artificial Intelligence Library

Brief Statement of the Problem:

Graph applications are difficult to optimize for different architectures like CPUs, GPUs and domain specific accelerators. Current compiler implementations (like GraphIt) perform a series of transformations and analyses to generate the most optimized code for a single target architecture. There is a need for a framework that can clearly separate the target dependent optimizations from the target indepedent optimizations in such a way that the target independent optimizations can be reused across all the targets with little to no changes and the implementation of the target dependent optimizations is simple and modular. I propose a new framework called UGF and a new intermediate representation GraphIR to solve this problem.

Supervision Agreement:

The program outlined in this proposal is adequate for a Master's thesis. The supplies and facilities required are available, and I am willing to supervise the research and evaluate the thesis report.

Saman Amarasinghe, Professor, EECS, MIT

Unified Graph Framework: Achieving High-Performance across Algorithms, Graph Types, and Architectures

Ajay Brahmakshatriya
Massachusetts Institute of Technology
ajaybr@mit.edu

August 9, 2020

Abstract

Graph applications are difficult to optimize because of their irregular memory accesses, different sparsity patterns, complex datastructures and variations in sizes, cache utilization and memory bandwidth with changing graph inputs and applications. The problem is further compounded when writing graph applications for different architectures like CPUs, GPUs and domain specific accelerators. Current compiler implementations (like GraphIt) perform a series of transformations and analyses to generate the most optimized code for a single target architecture. There is a need for a framework that can clearly separate the target dependent optimizations from the target indepedent optimizations in such a way that the target independent optimizations can be reused across all the targets with little to no changes and the implementation of the target dependent optimizations is simple and modular. I propose a new framework called UGF and a new intermediate representation GraphIR to solve this problem.

1 Introduction

Graph processing is at the heart of many modern applications, such as recommendation engines [1,2], social networks [3, 4], and map services [5]. Achieving high performance is important because these applications often need to process large graphs with trillions of edges [6] or have strict latency requirements [1].

However, the performance of graph programs is notoriously difficult to optimize [7]. Graph programs exhibit irregular memory access patterns that are difficult to execute efficiently on modern hardware platforms, which are optimized for regular memory accesses. Performance bottlenecks of graph programs depend on the algorithm, the size and structure of the input graphs, and the underlying hardware.

No single hardware platform performs best for all graph applications. Some applications perform better on CPUs and others perform better on GPUs or Domain-Specific Accelerators (DSAs). Shared-memory CPUs have out-of-order execution, which helps hide the long latency of irregular memory accesses that miss in the last-level cache. CPUs also have larger memories than GPUs and other accelerators, which enables processing larger graphs. By contrast, GPUs have up to an order of magnitude more compute power and memory bandwidth than CPUs [8] and can better exploit the data parallelism of some graph programs when the graph fits in the GPU memory.

A large number of Domain-Specific Accelerators for sparse computations have emerged recently [9–17]. They provide hardware features such as efficient speculative execution that can drastically improve graph performance. However, the program must be transformed specifically for each DSA to take advantage of the new hardware features, yet it is infeasible to build a new compiler for each DSA. As a result, building a portable compiler infrastructure is crucial to exploit the diverse hardware of different DSAs.

Existing graph processing frameworks cannot achieve high performance across multiple hardware platforms with one unified programming abstraction. To get the highest performance, CPU and GPU processing libraries have adopted abstractions and optimizations specific to their hardware platforms [18–22]. Graph domain-specific languages (DSLs) [23–25], such as GraphIt [7, 26], are either unable to support platforms other than CPUs or slower than state-of-the-art libraries on the other platforms. Existing work cannot easily incorporate hardware-specific optimizations while maintaining a unified programming model.

To achieve portability across CPUs, GPUs, and DSAs, one needs to separate hardware-independent and hardware-specific optimizations. I propose a new domain-specific intermediate representation (IR), Graph Intermediate Representation (GraphIR), to encode hardware-independent optimizations and serve as a high-level interface to different hardware backends. This way, hardware-independent optimizations can be reused across backends, and high-level data structures and operators can be mapped to different efficient hardware-optimized implementations.

I also propose a new graph processing framework, the Unified Graph Framework (UGF), that unifies the creation of compiler backends for different architectures. The framework would enable users to write high-performance graph algorithms once and run across GPUs, CPUs, and DSAs. UGF would be built on top of the GraphIt DSL [7, 26], which decouples the algorithm from the performance optimizations (schedules) for graph algorithms. UGF would utilize a new scheduling language that combines load balancing, edge traversal direction, active vertex set creation, active vertex set ordering, and kernel fusion optimizations on GPUs and can be extended to support CPU and DSA-specific optimizations.

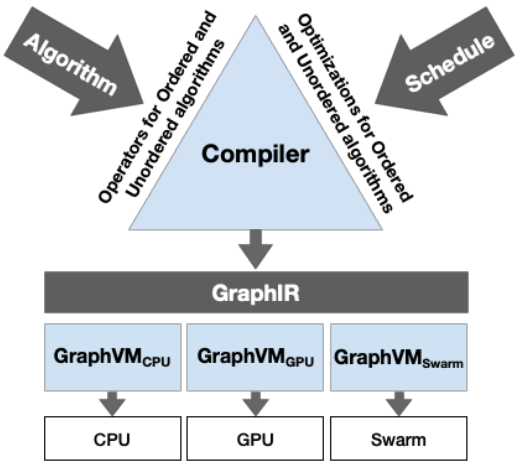

Figure 1: The Unified Graph Framework (UGF) provides a three-way decoupling between algorithm, schedule, and target hardware.

UGF would use the new GraphIR to support the different hardware backends. As shown in Figure 1, the compiler would perform various analyses and lowering passes to generate GraphIR, and the GraphIR would be lowered into code for different architectures using an architecture-specific Graph Virtual Machine (GraphVM). GraphVMs would perform hardware-specific transformations and code generation.

With the help of UGF I will build GraphIt GraphVMs for GPUs and the Swarm architecture to supplement the current CPU backend. I would also evaluate the performance of these new GraphVMs on various graph algorithms and graph inputs and compare them against the fastest

Work item	Status/Proposed completition date
Design the specification for the GraphIR with a complete list of operators and their respective arguments	**Completed**
Design an extensible scheduling language with target independent scheduling and support for extensions for respective targets	**Completed**
Explore the optimization space on GPUs by hand-implementing different versions and evaluating the performance	**Completed**
Implement the GraphVM and the scheduling language for GPUs	**Completed**
Implement the GraphVM and the scheduling language for Swarm	To be completed by August 14th 2020
Evaluate the performance of the 2 GraphVMs and compare against the related works	To be completed by August 27th 2020
Prepare a detailed report describing the design decisions and the performance results of the implementations	To be completed by August 28th 2020

Table 1: Proposed timeline for the UGF project implementation and evaluation

state-of-the-art GPU graph frameworks, including Gunrock [22], GSwitch [19], and SEP-graph [20].

2 Procedure

The creation of the GraphIR and the GraphVMs requires carefully designing the operators and types in the GraphIR. These operators have to capture all the algorithmic details of typical bulk synchronous graph algorithms. Further, for building each GraphVM, I would carefully need to understand the optimization space for the architecture and build a scheduling language that exposes all these decisions to the user. Following that I would have to implement all the analyses and transformations required to implement these optimizations in the respective GraphVMs. Based on this, I propose a timeline in Table 1.

References

[1] C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec, "Pixie: A system for recommending 3+ billion items to 200+ million users in real-time," in *Proceedings of the 2018 World Wide Web Conference (WWW)*, 2018, pp. 1775–1784.

[2] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018, pp. 974–983.

[3] A. Sharma, J. Jiang, P. Bommannavar, B. Larson, and J. Lin, "Graphjet: Real-time content recommendations at twitter," *Proc. VLDB Endow.*, vol. 9, no. 13, pp. 1281–1292, Sep. 2016.

[4] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, M. Marchukov, D. Petrov, L. Puzar, Y. J. Song, and V. Venkataramani,

"TAO: Facebook's distributed data store for the social graph," in *USENIX Annual Technical Conference (USENIX ATC)*, 2013, pp. 49–60.

[5] S. Pallottino and M. G. Scutellà, *Shortest Path Algorithms In Transportation Models: Classical and Innovative Aspects*, 1998, pp. 245–281.

[6] S. Maass, C. Min, S. Kashyap, W. Kang, M. Kumar, and T. Kim, "Mosaic: Processing a trillion-edge graph on a single machine," in *Proceedings of the Twelfth European Conference on Computer Systems*, 2017, pp. 527–543.

[7] Y. Zhang, M. Yang, R. Baghdadi, S. Kamil, J. Shun, and S. Amarasinghe, "Graphit: A high-performance graph dsl," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, pp. 121:1–121:30, Oct. 2018.

[8] NVIDIA, "Cuda c++ programming guide," https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, Aug. 2019.

[9] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.

[10] M. C. Jeffrey, S. Subramanian, C. Yan, J. Emer, and D. Sanchez, "A scalable architecture for ordered parallelism," in *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2015, pp. 228–241.

[11] A. Segura, J.-M. Arnau, and A. González, "SCU: A GPU stream compaction unit for graph processing," in *Proceedings of the 46th International Symposium on Computer Architecture (ISCA)*, 2019, pp. 424–435.

[12] G. Li, G. Dai, S. Li, Y. Wang, and Y. Xie, "GraphIA: An in-situ accelerator for large-scale graph processing," in *Proceedings of the International Symposium on Memory Systems (MEMSYS)*, 2018, pp. 79–84.

[13] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, and H. Yang, "GraphH: A processing-in-memory architecture for large-scale graph processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 640–653, April 2019.

[14] A. Mukkara, N. Beckmann, and D. Sanchez, "Phi: Architectural support for synchronization- and bandwidth-efficient commutative scatter updates," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019, pp. 1009–1022.

[15] A. Addisie, H. Kassa, O. Matthews, and V. Bertacco, "Heterogeneous memory subsystem for natural graph analytics," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, Sep. 2018, pp. 134–145.

[16] P. Yao, L. Zheng, X. Liao, H. Jin, and B. He, "An efficient graph accelerator with parallel data conflict management," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2018, pp. 8:1–8:12.

[17] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 105–117.

[18] T. Ben-Nun, M. Sutton, S. Pai, and K. Pingali, "Groute: An asynchronous multi-gpu programming model for irregular computations," in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2017, pp. 235–248.

[19] K. Meng, J. Li, G. Tan, and N. Sun, "A pattern based algorithmic autotuner for graph processing on GPUs," in *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2019, pp. 201–213.

[20] H. Wang, L. Geng, R. Lee, K. Hou, Y. Zhang, and X. Zhang, "Sep-graph: finding shortest execution paths for graph processing under a hybrid framework on gpu," in *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2019, pp. 38–52.

[21] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2013, pp. 135–146.

[22] Y. Wang, Y. Pan, A. Davidson, Y. Wu, C. Yang, L. Wang, M. Osama, C. Yuan, W. Liu, A. T. Riffel *et al.*, "Gunrock: Gpu graph analytics," *ACM Transactions on Parallel Computing (TOPC)*, vol. 4, no. 1, p. 3, 2017.

[23] S. Hong, H. Chafi, E. Sedlar, and K. Olukotun, "Green-marl: A dsl for easy and efficient graph analysis," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012, pp. 349–362.

[24] G. Gill, R. Dathathri, L. Hoang, A. Lenharth, and K. Pingali, "Abelian: A compiler for graph analytics on distributed, heterogeneous platforms," in *Euro-Par*, 2018, pp. 249–264.

[25] C. R. Aberger, A. Lamb, S. Tu, A. Nötzli, K. Olukotun, and C. Ré, "Emptyheaded: A relational engine for graph processing," vol. 42, no. 4, Oct. 2017, pp. 20:1–20:44.

[26] Y. Zhang, A. Brahmakshatriya, X. Chen, L. Dhulipala, S. Kamil, S. Amarasinghe, and J. Shun, "Optimizing ordered graph algorithms with graphit," in *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*, ser. CGO 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 158âĂŞ170. [Online]. Available: https://doi.org/10.1145/3368826.3377909