High-Performance GraphBLAS API Implementation in Functional Style

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

Semyon Grigorev
Saint Petersburg State University,
JetBrains Research,
St. Petersburg, Russia
s.v.grigoriev@spbu.ru,
semyon.grigorev@jetbrains.com

***Abstract*—Abstract is very abstract. Abstract is very abstract.**

***Index Terms*—graph analysis, sparse linear algebra, GraphBLAS API, GPGPU, parallel programming, functional programming, .NET, OpenCL**

I. INTRODUCTION

One of the promising ways to high-performance graph analysis is based on utilization of linear algebra: operations over vectors and matrices can be efficiently implemented on modern parallel hardware, and once we reduce given graph analysis problem to composition of such operations, we get high-performance solution for our problem. Well-known example of such reduction is a reduction of all-pairs shortest path (APSP) problem to matrix multiplication over appropriate *semiring*. GraphBLAS API standard [1] provides formalization, generalization of this observation and make it useful in practice. GraphBLAS API introduces appropriate algebraic structures (monoid, semiring), objects (scalar, vector, matrix), and operations over them to provides building blocks to create graph analysis algorithms. It was shown, that sparse linear algebra over specific semirings is useful not only for graph analysis, but also in other areas, such as computational biology [2] and machine learning [3].

Reference implementation SuiteSarse:GraphBLAS [4] which is used in a number of projects, such as RedisGraph, !!!. Other implementations: CombBLAS, !!! .

GPGPU for high-performance analysis of huge amount of data. GraphBLAST [5] — GraphBLAST in CUDA C. GPGPU implementation of GraphBLAS API is challenging. Problems with polymorphic operations in OpenCL C and, as a result, with support of user-defined semirings. Memory traffic. Global optimizations (kernel fusion).

High-level programming languages for application development vs low-level for high-performance programming. Moreover, specific languages for GPGPU programming: CUDA C, OpenCL C. Problems with types, compositionality, optimizations (kernel fusion).

Functional programming. Type systems. Optimizations. Futhark [6], kernel fusion, specialization, deforestation etc.

In this work we discuss a way to implement GraphBLAS API which combines high-performance computations on GPGPU and power of high-level programming languages in both application development and possible code optimizations. Our solution is based on metaprogramming techniques: we propose to generate code for GPGPU from high-level programming language. Namely, we plan to generate OpenCL C form a subset of F# programming language. Usage of F# simplifies both implementation of GraphBLAS API, and its utilization in application development. Moreover it should makes possible to use advanced optimization techniques, like kernel fusion, on the same way, as proposed, for example, in the Futhark programming language. Choice of OpenCL C as a target language is motivated by its portability: it is possible to run OpenCL C code on multythread CPU, on different GPGPUs (not only Nvidia), and even on FPGA [7], [8]. Utilization of FPGAs may open a way to hardware acceleration of sparse linear algebra and, as a result, of many solutions in different areas such as graph analysis, computational biology, machine learning. Brahma.FSharp[1]. Our preliminary evaluation shows that !!!

Identify applicable funding agency here. If none, delete this.

[1] !!!

II. Design Principles

Functional style, types, optimizations, etc.

Code example with description and explanations.

III. Implementation Details

Details on implementation.

A few worlds on Brahma.FSharp.

Architecture.

IV. Evaluation

Evaluation of the proposed implemenation.

Hardware configuration description.

SuiteSparse, Math.NET Numerics[2], GraphBLAST, ???, and our solution on CPU and GPGPU.

Dataset description.

Results.

Results analysis. and conclusion.

V. Conclusion

Conclusion, current state, results.

Future work. Library extension up to full GraphBLAS API implementation.

Evaluation. Comparison with other implementations. Manual implementation versus translation.

Brahma.FSharp improvements. Algebraic data types. Example with Min-Plus or other semiring.

Can we use it on FPGA?

References

[1] J. Kepner, P. Aaltonen, D. Bader, A. Buluc, F. Franchetti, J. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, H. Meyerhenke, S. McMillan, C. Yang, J. D. Owens, M. Zalewski, T. Mattson, and J. Moreira, "Mathematical foundations of the graphblas," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2016, pp. 1–9.

[2] O. Selvitopi, S. Ekanayake, G. Guidi, G. A. Pavlopoulos, A. Azad, and A. Buluç, "Distributed many-to-many protein sequence alignment using sparse matrices," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.

[3] J. Kepner, M. Kumar, J. Moreira, P. Pattnaik, M. Serrano, and H. Tufo, "Enabling massive deep neural networks with the graphblas," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, 2017, pp. 1–10.

[4] T. A. Davis, "Algorithm 1000: Suitesparse:graphblas: Graph algorithms in the language of sparse linear algebra," *ACM Trans. Math. Softw.*, vol. 45, no. 4, Dec. 2019. [Online]. Available: https://doi.org/10.1145/3322125

[5] C. Yang, A. Buluç, and J. D. Owens, "GraphBLAST: A high-performance linear algebra-based graph framework on the GPU," *arXiv preprint*, 2019.

[6] T. Henriksen, N. G. W. Serup, M. Elsman, F. Henglein, and C. E. Oancea, "Futhark: Purely functional gpu-programming with nested parallelism and in-place array updates," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2017. New York, NY, USA: ACM, 2017, pp. 556–571. [Online]. Available: http://doi.acm.org/10.1145/3062341.3062354

[7] T. Kenter, "Invited tutorial: Opencl design flows for intel and xilinx fpgas: Using common design patterns and dealing with vendor-specific differences," in *FSP Workshop 2019; Sixth International Workshop on FPGAs for Software Programmers*. VDE, 2019, pp. 1–8.

[8] K. Shagrithaya, K. Kepa, and P. Athanas, "Enabling development of opencl applications on fpga platforms," in *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*, 2013, pp. 26–30.

[2]Library which provides numerical computations primitives for .NET: https://numerics.mathdotnet.com/. Access date: 12.01.2021.