SPLA: Portable Multi-GPU Implementation of GraphBLAS API

1st Egor Orachev
Faculty of Mathematics and Mechanics
St. Petersburg State University,
Programming Languages and Tools Lab
JetBrains Research
St. Petersburg, Russia
egor.orachev@gmail.com
!!!OCRID

2nd Gleb
!!!
!!!
St. Petersburg, Russia
!!!email
!!!OCRID

3rd Semyon Grigorev
Faculty of Mathematics and Mechanics
St. Petersburg State University,
Programming Languages and Tools Lab
JetBrains Research
St. Petersburg, Russia
s.v.grigoriev@spbu.ru
semyon.grigorev@jetbrains.com
0000-0002-7966-0698

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

***Index Terms*—graphs, algorithms, graph analysis, sparse linear algebra, GraphBLAS, GPGPU, OpenCL**

I. Introduction

Big graphs are everywhere. High-performance graph analysis is an actual challenge. [1]

To utilize general purpose graphic processing units (GPGPU-s) for graph analysis is a promising way. Existing solutions demonstrate good performance. Scalable. But existing solutions provide a very low level API for graph algorithms construction. As a result, Verbosity, so on

One of promising ways to high performance graph analysis is a GraphBLAS API[1] [**?**] provides a linear algebra based building blocks [2] to create graph analysis algorithms. The idea of GraphBLAS is based on is well-known fact that linear algebra operations can be efficiently implemented on parallel hardware. Within it, a graph can be natively represented using matrices: adjacency matrix, incidence matrix, etc. But real data often sparse, thus underlying matrices and vectors also sparse, and classical data structures and algorithms became inefficient. This fact leads to sparse linear algebra. Generic sparse linear algebra. SuiteSparse:GraphBLAS[2] [**?**].

Though such well-known libraries as cuSparse show that sparse linear algebra operations can be efficiently implemented for GPGPU-s, it is not so trivial to implement GraphBLAS on GPGPU. First of all, it requires *generic* sparse linear algebra, thus it is impossible jus tu reuse existing libraries which almost specified for operations over floats and doubles. The second problem is specific optimizations, such as maskings fusion, which can not be natively implemented on top of existing kernels. GraphBLAS on GPGPU. GraphBLAST: [3], GBTL [**?**], !!!. But these solutions are not portable (because thea are based on Nvidia Cuda stack). Moreover, scalability problem is not solved: all these solutions support only single-GPU, not multi-GPU computations.

To provide portable multi-GPU implementation of GraphBLAS API we developed a *SPLA*[3] library. Contribution

- Design
- Implemented
- Evaluation

II. Solution Description

A. Design Principles

Multi-GPU

Portability across multiple devices and vendors.

C interface to simplify interaction with other languages.

Tasks for better load-balancing.

Extensibility: one can easily implement a specific algorithm for primitive operation.

....

B. Architecture Overview

Fig. 1. Architecture

C. Matrices and Vectors Representation

Block formats. Pros and cons.

Currently COO only.

D. Algebraic Operations

Implemented operations and algorithms for it.

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

[1]GraphBLAS home page: https://graphblas.org/. Accass date: 07.01.2022.

[2]!!!

[3]SPLA home page: https://jetbrains-research.github.io/spla/.

E. Implementation Details

OpenCL,
Used libraries (Boost.Compute[4], taskflow[5] [4]), etc

III. Evaluation

What and how.

A. Graph Algorithms

BFS, SSSP, ... implemented

B. Dataset Description

SuiteSparse matrix collection[6] [5]. Table with specific graphs.

C. Evaluation Setup

Hardware description, two GPU, ...
Single GPU
Two GPU

D. Evaluation Results

Single GPU

TABLE I
TABLE TYPE STYLES

Table Head	Table Column Head		
	Table column subhead	*Subhead*	*Subhead*
copy	More table copy[a]		

[a]Sample of a Table footnote.

Two GPU

TABLE II
TABLE TYPE STYLES

Table Head	Table Column Head		
	Table column subhead	*Subhead*	*Subhead*
copy	More table copy[a]		

[a]Sample of a Table footnote.

Analysis and conclusion.

IV. Conclusion

Work in progress!
And future work
Advanced algorithms for LA operations
More formats for sparse matrices/vectors
C interface
Python package
Graphalytics
!!!

[4]https://github.com/boostorg/compute.

[5]Taskflow project home page: https://taskflow.github.io/. Access date: 07.01.2022.

[6]https://sparse.tamu.edu/.

References

[1] M. E. Coimbra, A. P. Francisco, and L. Veiga, "An analysis of the graph processing landscape," *Journal of Big Data*, vol. 8, no. 1, Apr. 2021. [Online]. Available: https://doi.org/10.1186/s40537-021-00443-9

[2] J. Kepner, P. Aaltonen, D. Bader, A. Buluç, 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)*, 2016, pp. 1–9.

[3] C. Yang, A. Buluc, and J. D. Owens, "Graphblast: A high-performance linear algebra-based graph framework on the gpu," 2019.

[4] T.-W. Huang, D.-L. Lin, C.-X. Lin, and Y. Lin, "Taskflow: A lightweight parallel and heterogeneous task graph computing system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, pp. 1303–1320, 2022.

[5] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, dec 2011. [Online]. Available: https://doi.org/10.1145/2049662.2049663