The Library of GPGPU-Powered Sparse Boolean Linear Algebra Operations

Egor Orachev
Saint Petersburg State University,
St. Petersburg, Russia
egororachyov@gmail.com

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

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

Artem Khoroshev
Computation Biology Department
BIOCAD
St. Petersburg, Russia
arthoroshev@gmail.com

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

***Abstract*—Sparse matrices are widely applicable in data analysis, and the theory of matrix processing is well-established and introduces a wide range of different algorithms for basic operations such as matrix-matrix and matrix-vector multiplication, factorization, etc. To make this observation practical, GraphBLAS API provides a set of respective building blocks, allows one to reduce algorithms to sparse linear algebra operations. While GPGPU utilization for high-performance linear algebra is a common practice, the high complexity of GPGPU programming makes the implementation of GraphBLAS API on GPGPU challenging. In this work, we present a GPGPU library of sparse operations for an important case — Boolean algebra —, which is based on modern algorithms for sparse matrix processing. We provide Python !!! Our evaluation shows that !!! We hope that our results help to move the development of the GPGPU version of GraphBLAS API forward.**

***Index Terms*—sparse linear algebra, GPGPU, boolean semiring, sparse boolean matrix**

I. Introduction

One of the techniques to efficiently solve a data analysis problem is to formulate it in terms of linear algebra (in terms of operations over vectors and matrices). That gives one well studied for years mathematical tools and solutions, as well as the possibility to evaluate this problem with *zero-cost* by high-performance linear algebra libraries, which utilize modern hardware, provide various optimization techniques, and allow quickly and safely prototype solution in code with predefined building blocks. GraphBLAS API[1] [1] is one of the standards that introduce such building blocks. GraphBLAS take into account sparsity of data by using sparse formats of matrices and vectors, and operates with arbitrary *monoids* and *semirings* to make provided building blocs generic. While initially GraphBLAS was focused on graph analysis, it was shown that the proposed approach can be successfully used for data analysis in other areas, such as computational biology [2] and machine learning [3].

GPGPU utilization for data analysis and for linear algebra operations is a promising way to high-performance data analysis because GPGPU gives much more power in parallel data processing. But the implementation of appropriate libraries is very challenging. GPGPU programming introduces heterogeneous device model into the system, memory traffic, and data operations limitations, as well as requires taking into account vendor-specific capabilities. Thus, there is no, best to our knowledge, full implementation of GraphBLAS API on GPGPU, except GraphBLAST project[2] [4], which currently in active development.

The sparsity of data introduces problems with load balancing, irregular data access, thus sparsity makes the implementation of high-performance algorithms for sparse linear algebra on GPGPU even more challenging. As a result, there is a huge number of different formats for sparse matrices and vectors representation, such as CSR, COO, Quad-tree, and a huge number of algorithms for operations over these formats. For example, one can look at the significant survey of sparse matrix-matrix multiplication algorithms [5]. Unfortunately, algorithms for different operations, such as matrix-matrix multiplication, matrix-vector multiplication, etc. are developed independently. Thus, there are no sparse linear algebra libraries based on state-of-the-art algorithms. Moreover, existing libraries, such as cuSparse[3], clSparse[4] [6],

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

[1]GraphBLAS project web page: https://graphblas.github.io/. Access date: 19.01.2021.

[2]GraphBLAST project: https://github.com/gunrock/graphblast. Access date: 19.01.2021.

[3]NVIDIA sparse matrix library (in Cuda) https://docs.nvidia.com/cuda/cusparse/. Access date: 19.01.2021.

[4]Sparse linear library functions in OpenCL: http://clmathlibraries.github.io/clSPARSE/. Access date: 19.01.2021.

or more modern CUSP[5] or bhSparse[6] [7], are focused on numerical computations over floats or doubles, not on generic data processing over arbitrary semirings which required for GraphBLAS API implementation.

An important partial case of linear algebra is as sparse Boolean linear algebra. Boolean algebra allows to address problems over a finite set of values, for example, transitive closure of relation or graph, regular and context-free path queries for graphs [8], parsing for different classes of languages, such as Context-Free [9], Boolean and Conjunctive [10], Multiple Context-Free(MCFL) [11]. Moreover, some operations over Boolean semiring may be used as building blocks for algorithms over other semirings. For example, to compute the shape of the result of the operation. Thus, sparse Boolean linear algebra is an important partial case both as a way to solve applied problems and as a building block for other algorithms. However, sparse Boolean linear algebra on GPGPU is still not presented, because of its high specificity.

In this work, we present the sparse boolean linear algebra operations implementation as stand-alone self-sufficient programming libraries for the two most popular GPGPU platforms: NVIDIA Cuda[7]and OpenCL[8]. Cuda is a GPGPU technology for NVIDIA devices, which allows to employ of some platform-specific facilities, such as unified memory mechanism, and make architectural assumptions, which gives more optimizations space at cost of portability. OpenCL is a platform-agnostic API standard, which allows running computations on different platforms, such as multi-threaded CPUs, GPUs, and FPGAs. Our implementation relies on modern sparse matrices processing techniques, as well as exploits some optimizations, related to the boolean data processing. A few words on Python ALI and evaluation results!!!

II. Related work

Existing libraries, algorithns, frameworks.

III. Implementation Details

Implemented sparse boolean linear algebra libraries for OpenCL and NVIDIA Cuda platforms are called *clBool* and *cuBool* respectively. Source code and related artefacts are available at GitHub hosting [**?**], [**?**]. Libraries are written in the C++ programming language, which is well-suited performance and resource critical computational tasks. Libraries expose C compatible API with relatively small amount of functions and primitives, what gives expressiveness and allows to embed that API into other execution environments via interoperability mechanisms, for example, into Python or .NET runtimes. Libraries source code compilation is configured with CMake tool. Compilation process is straightforward and requires setup of only basic components and instruments, such as compiler, build configuration tools and platform-specific development kits.

Libraries operate on boolean semiring with values set $\{$*true*, *false*$\}$ with *false* as a neutral element, '+' operation defined as logical *or* and '*' defined as logical *and*. Values are also denoted as $\{1,\ 0\}$ respectively, and the abbreviation *NNZ(M)* gives the number of non-zero cells of the matrix M.

Main primitive is sparse matrix of boolean values, stored in one of the sparse formats. Sparse vector primitive is not presented, since its utilization is relatively rare presented in practical computational tasks. But its support is something to be added in far future. Primary available operations and functions are following.

- Create sparse matrix M of size $m \times n$.
- Delete sparse matrix M and release all its internal resources.
- Fill the matrix M with values $L = \{(i,j)_k\}_k$. The result of this operation is $M_{i,j} = \textit{true}$ for each $(i,j) \in L$, and $M_{i,j} = \textit{false}$ for the rest of matrix values.
- Read matrix M values $L = \{(i,j) \mid M_{i,j} = \textit{true}\}$.
- Matrix-matrix multiply-add operation $C \mathrel{+}= M \times N$.
- Matrix-matrix add operation $M \mathrel{+}= N$.
- Matrix-matrix Kronecker product $K = M \otimes N$.

A. cuBool

cuBool is sparse boolean linear algebra implementation specifically for NVIDIA Cuda platform. Core of this library relies on Cuda C/C++ language and API, what with NVCC compiler allows intermix C++ with Cuda specifics. Also cuBool employs NVIDIA Thrust auxiliary library, which provides implementation for generic data containers and operations, such as *iterating*, *exclusive or inclusive scan*, *map* and etc., which are executed on Cuda device. That allows express algorithms in terms of high-level optimised primitives, what increases code readability and reduces time for development.

Sparse matrix primitive is stored in the *compressed sparse row* (CSR) format with only two arrays: $rowspt$ for row offset indices and $cols$ for columns indices. Boolean matrices has no actual values, thus $true$ values are encoded only as (i,j) pairs. It allows to store matrix M of size $m \times n$ in $(m + \textit{NNZ(M)}) \times \textit{sizeof(IndexType)}$ bytes of GPU memory, where *IndexType* is type of stored indices, for simplicity can be selected as *uint32_t*.

The algorithm Nspasrse [12] is used for matrix-matrix multiplication. This algorithm is a boolean values case adaptation of the state-of-the-art, efficient and memory saving sparse general matrix multiplication (SpGEMM) algorithm, proposed in Yusuke Nagasaka et al. research [13]. This algorithm was selected because it gives promising relatively small memory footprint for large matrices processing, as well as it competes with other major Cuda SpGEMM implementations, such as cuSPARSE or CUSP.

Matrix-matrix addition is based on GPU Merge Path algorithm [14] with dynamic work balancing and two pass process-

[5]CUSP sparse linear algebra library: https://cusplibrary.github.io/modules.html. Access date: 19.01.2021.

[6]bhSparse sparse matrix multiplication library: https://github.com/weifengliu-ssslab/bhSPARSE. Access date: 19.01.2021.

[7]CUDA is a platform and programming model for NVIDIA devices. Home page: https://developer.nvidia.com/CUDA-zone. Access date: 19.01.2021.

[8]OpenCL is an open standard for parallel programming of heterogeneous systems. Home page: https://www.khronos.org/opencl/. Access date: 19.01.2021.

ing. These optimizations give better workload dispatch among execution blocks and allow more precise memory allocations in order to keep memory footprint small respectively.

As an example of library C API embedding, cuBool provides python wrapper, called Pycubool. This module exports library functionality via default CTypes module for native functions calling and provides safe and automated management for native resources.

B. clBool

clBool is sparse boolean linear algebra implementation for OpenCL platform. This library is implemented in the C++ with OpenCL kernels, stored as separate source files, loaded on demand at runtime.

Sparse matrix primitive is stored in *coordinate format* (COO) with two arrays: $rows$ and $cols$ for row and column indices of the stored non-zero values. For the matrix M of size $m \times n$ memory consumption is $2 \times$*NNZ(M)*$\times$*sizeof(IndexType)*. This format was selected instead of CSR, because COO gives better memory footprint for very sparse matrices with a lot of empty rows.

!!! Matrix-matrix multiplication !!!

Matrix-matrix addition is based on GPU Merge Path algorithm as well. Since all COO matrix values are stored in the single array, its merge can be completed at single time, compared to CSR matrix merge computed on a per row basis. This operation is implemented in a classic one pass fashion: it allocates single merge buffer of size *NNZ(A)*$+$*NNZ(B)*$)$ before actual merge of matrices A and B, what can negatively affect memory consumption for large matrices with lots of duplicated non-zero values at the same positions.

!!! Something about managed environment wrapper !!!

IV. Evaluation

Evaluation of the proposed implemenation(s).

V. Conclusion

Conclusion and future work.

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

[5] J. Gao, W. Ji, Z. Tan, and Y. Zhao, "A systematic survey of general sparse matrix-matrix multiplication," *ArXiv*, vol. abs/2002.11273, 2020.

[6] J. L. Greathouse, K. Knox, J. Poła, K. Varaganti, and M. Daga, "Clsparse: A vendor-optimized open-source sparse blas library," in *Proceedings of the 4th International Workshop on OpenCL*, ser. IWOCL '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2909437.2909442

[7] W. Liu and B. Vinter, "A framework for general sparse matrix-matrix multiplication on gpus and heterogeneous processors," *J. Parallel Distrib. Comput.*, vol. 85, no. C, pp. 47–61, Nov. 2015. [Online]. Available: https://doi.org/10.1016/j.jpdc.2015.06.010

[8] R. Azimov and S. Grigorev, "Context-free path querying by matrix multiplication," in *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, ser. GRADES-NDA '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3210259.3210264

[9] L. G. Valiant, "General context-free recognition in less than cubic time," *J. Comput. Syst. Sci.*, vol. 10, no. 2, pp. 308–315, Apr. 1975. [Online]. Available: https://doi.org/10.1016/S0022-0000(75)80046-8

[10] A. Okhotin, "Parsing by matrix multiplication generalized to boolean grammars," *Theoretical Computer Science*, vol. 516, pp. 101–120, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397513006919

[11] G. Satta, "Tree-adjoining grammar parsing and boolean matrix multiplication," *Comput. Linguist.*, vol. 20, no. 2, pp. 173–191, Jun. 1994.

[12] A. Terekhov, A. Khoroshev, R. Azimov, and S. Grigorev, "Context-free path querying with single-path semantics by matrix multiplication," 06 2020, pp. 1–12.

[13] Y. Nagasaka, A. Nukada, and S. Matsuoka, "High-performance and memory-saving sparse general matrix-matrix multiplication for nvidia pascal gpu," in *2017 46th International Conference on Parallel Processing (ICPP)*, 2017, pp. 101–110.

[14] O. Green, R. Mccoll, and D. Bader, "Gpu merge path: a gpu merging algorithm," 11 2014.