The Library of GPGPU-Powered Sparse Boolean Linear Algebra Operations

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

Maria Karpenko
ITMO University
St. Petersburg, Russia
mkarpenko.spb@gmail.com

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 for developed library to simplify its utilization in applied solutions. Our evaluation shows that operations specialized for Boolean matrices cam be up to 2 times faster and consume up to YYY times less memory then generic operations from modern libraries. 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], 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, transi-

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.

[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.

tive 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. Moreover, we provide a Python API to simplify utilization of our library. Preliminary evaluation shows that such operation as matrix-matrix multiplication specialized for Boolean matrices can be up to 2 times faster and consume up to YYY times less memory in comparison with generic operations from such libraries as CUSP or CuSparse.

II. About Libraries

Implemented sparse boolean linear algebra libraries for OpenCL and NVIDIA Cuda platforms are called *clBool*[9] and *cuBool*[10] respectively. Projects are hosted at GitHub platform. The source code is licensed under MIT license. The build process is straightforward: it is configured with CMake tool and requires extra setup only of platform-specific development kits.

Libraries architecture is briefly depicted at figure 1. The core of the libraries is written in the C++ programming languages, which is well-suited for performance and resource critical computational tasks. Actual GPU related logic is presented in platform specific backends: Cuda and OpenCL, which use respective technologies for resources and GPU executable code management. cuBool library exposes C compatible API, what gives expressiveness and allows to embed that API into other execution environments by interoperability mechanisms.

[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.

[9]clBool project: https://replace/me/with/actual/url. Access date: 03.02.2021.

[10]cuBool project: https://github.com/JetBrains-Research/cuBool. Access date: 03.02.2021.

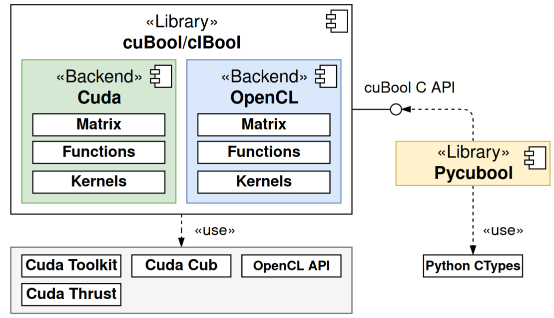

Fig. 1. Conceptual sparse boolean linear algebra library architecture

Pycubool module encapsulates such functionality and provides it for high-level Python runtime.

It is worth mention, that it is convenient to create the single library with common interface and several backends for different execution targets. At this time clBool and cuBool are distinct libraries, but they can be integrated into single library. This integration is something to be done in near future. This process requires careful selection of the interface to allow the end user properly configure the library for specific tasks, as well as provide the option to automatically select a specific implementation depending on the capabilities of the target device.

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 $M_{i,j} = 1$ for each $(i,j) \in L$, and $M_{i,j} = 0$ for the rest of matrix values.
- Read matrix M values $L = \{(i,j) \mid M_{i,j} = 1\}$.
- Matrix-matrix multiply-add operation $C \mathrel{+}= M \times N$.
- Matrix-matrix add operation $M \mathrel{+}= N$.
- Matrix-matrix Kronecker product $K = M \otimes N$.

III. Implementation Details

In this section we discuss the particular implementation details of the proposed libraries. Although general and architectural specifics are similar, the actual internal storage formats and algorithms are different. With this development strategy we address the potential problem of processing the sparse data with different values distribution, as well as the problem of proper balancing between time of the execution and memory consumption.

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 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 1 values are encoded only as (i, j) pairs. It allows to store matrix M of size $m \times n$ in $(m + \textit{nnz(M)}) \times \textit{sizeof(index\_t)}$ bytes of GPU memory, where *index_t* 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 processing. These optimizations give better workload dispatch among execution blocks and allow more precise memory allocations in order to keep memory footprint small respectively.

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 $2 \times \textit{nnz(M)} \times \textit{sizeof(index\_t)}$. 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 implementation is based on the algorithm, proposed in Weifeng Liu et al. research [15]. It is multi-step algorithm with dynamic workload balancing, which operates on CSR matrices. Since clBool primary primitive is COO matrix, before actual matrix-matrix multiplication the input matrices are converted into *doubly compressed sparse row* (DCSR) format, described in A. Buluc et al. work [16].

TABLE I
MATRIX DATA

Matrix M	$\#rows$	Nnz of M	Nnz of M^2	Nnz of $M + M^2$
wing	62032	243088	714200	917178
luxembourg_osm	114599	239332	393261	632185
roadNet-PA	1090920	3083796	7238920	9931528
roadNet-TX	1393383	3843320	8903897	12264987
belgium_osm	1441295	3099940	5323073	8408599
roadNet-CA	1971281	5533214	12908450	17743342

This algorithm is suitable for OpenCL implementation, what is confirmed with its utilisation in clSPARSE library.

Matrix-matrix addition is based on GPU Merge Path algorithm as well. Since all COO matrix values are stored in the continuous manner, 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 $\textit{nnz(A)} + \textit{nnz(B)})$ 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.

IV. EVALUATION

We evaluate the applicability of the proposed libraries for analysis of some real-world matrix data. The experiments are designed as a computational tasks, that arise as stand-alone or intermediate steps in the solving of practical problems.

For evaluation, we used a PC with Ubuntu 20.04 installed. It has Intel core i7-6700 CPU, 3.4GHz, DDR4 64Gb RAM and Geforce 1070Ti GPU with 8Gb VRAM. We only measure the execution time of the operations themselves. The actual data is assumed to be loaded into the VRAM or RAM respectively in the appropriate format, required for the target tested framework. Time to load data from disc and prepare initial matrices state is excluded from the time measurements.

We use four sparse matrix libraries, CUSP, cuSPARSE, clSPARSE for GPU and SuiteSparse for CPU. CUSP provides template based implementation for operations, however it does not provide extra optimizations especially for boolean case values. cuSPARSE and clSPARSE both provide operations only for general types, such as float or double. But this limitation can be ignored, if we consider non-zero float values as *true* one. SuiteSparse is a GraphBLAS API reference implementation for CPU, which allows to use build-in boolean semiring.

For performance evaluations, we selected N various square matrices which are widely used for sparse matrices benchmarks from the Sparse Matrix Collection at University of Florida[11]. The name and size of the matrix data are summarized in the table I.

The results of the evaluation are summarized in the tables below. Time is measured in seconds unless specified otherwise. The result for each experiment is average over 20 runs. The cell is left blank if the time limit is exceeded, or if there is

[11]T. Davis. The SuiteSparse Matrix Collection (the University of Florida Sparse Matrix Collection). Home page: https://sparse.tamu.edu/. Access date: 23.01.2021.

TABLE II
MATRIX-MATRIX MULTIPLICATION EVALUATION RESULTS, TIME IS MEASURED IN SECONDS.

Matrix M	CuBool	CUSP	CuSprs	ClBool	ClSprs	SuiteSprs
wing	0.003	0.007	0.021	0.007	-	0.007
luxembourg_osm	0.005	0.005	0.002	0.010	-	0.003
roadNet-PA	0.023	0.043	0.037	0.047	-	0.067
roadNet-TX	0.030	0.053	0.047	0.057	-	0.084
belgium_osm	0.027	0.034	0.030	0.054	-	0.061
roadNet-CA	0.039	0.076	0.071	0.078	-	0.121

TABLE III
ELEMENT-WISE MATRIX-MATRIX ADDITION, TIME IS MEASURED IN SECONDS.

Matrix M	CuBool	CUSP	CuSprs	ClBool	ClSprs	SuiteSprs
wing	0.001	0.002	0.003	0.006	-	0.003
luxembourg_osm	0.002	0.002	0.001	0.006	-	0.002
roadNet-PA	0.015	0.011	0.013	0.073	-	0.035
roadNet-TX	0.019	0.014	0.015	0.076	-	0.045
belgium_osm	0.019	0.010	0.011	0.064	-	0.027
roadNet-CA	0.027	0.019	0.020	0.141	-	0.062

not enough memory to allocate the data and internal auxiliary structures.

The first experiment is intended to measure performance of the matrix-matrix multiplication as $M \times M$. The results are presented in the table II. We can see that for a relatively small matrices, the results of all libraries are comparable. However, cuBool shows generally better performance among competitors. clBool gives good performance, constantly better than SuiteSparse, and comparable to cuSPRASE or CUSP in some cases. However, there is still space for optimizations, so it requires an in deep investigation in our further research.

The second experiment is intended to measure performance of the element-wise matrix-matrix addition as $M+M^2$, where evaluation of matrix M^2 is excluded from measurements. The results are presented in the table III. The numbers obtained in this experiment are less unambiguous than in the previous experiment. Although libraries are still comparable for small matrices, results vary greatly for large matrices. CUSP shows nearly best performance among almost all experiments. cuSPRASE is also comparable with it in this aspect. Libraries cuBool and clBool lag behind insignificantly, and generally keep their results within acceptable limits. It is worth mention, that CUSP matrix-matrix addition implementation has significant memory consumption, what can negatively affect on processing of huge data. cuSPARSE performance can degraded in such case as well, since its implementation is based on hashing, what is very sensitive for out-of shared blocks memory access for large data processing.

V. CONCLUSION

In this work we present a library for sparse Boolean linear algebra which implements such basic operations, as matrix-matrix multiplication and element-wise matrix-matrix addition in both, Cuda C and OpenCL C.

The first direction of the future work is to integrate all parts (OpenCL and Cuda backends) to a single library and improve its documentation and prepare to publish. Also, it is necessary to publish a Python package.

Another important step is to evaluate the library on different algorithms and devices. Namely, algorithms for RPQ and CFPQ should be implemented and evaluated on related data sets. Also, it is necessary to evaluate OpenCL version on FPGA which may require additional technical effort and code changes.

Finally, we plan to discuss with GraphBLAS community possible ways to use our library as a backend for GraphBLAST or SuiteSparse in case of Boolean computations. Moreover, it may be possible to use implemented algorithms as a base for generalization to arbitrary semirings.

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," in *Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences; Systems (GRADES) and Network Data Analytics (NDA)*, ser. GRADES-NDA'20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3398682.3399163

[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. A. Bader, "Gpu merge path: A gpu merging algorithm," in *Proceedings of the 26th ACM International Conference on Supercomputing*, ser. ICS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 331–340. [Online]. Available: https://doi.org/10.1145/2304576.2304621

[15] W. Liu and B. Vinter, "A framework for general sparse matrix-matrix multiplication on gpus and heterogeneous processors," *CoRR*, vol. abs/1504.05022, 2015. [Online]. Available: http://arxiv.org/abs/1504.05022

[16] A. Buluc and J. R. Gilbert, "On the representation and multiplication of hypersparse matrices," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, 2008, pp. 1–11.