

# 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 while the theory of matrix processing is well-established. There are a wide range of algorithms for basic operations such as matrix-matrix and matrix-vector multiplication, factorization, etc. To facilitate data analysis, GraphBLAS API provides a set of building blocks and allows for reducing algorithms to sparse linear algebra operations. While GPGPU utilization for high-performance linear algebra is common, 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. The library is based on modern algorithms for sparse matrix processing. We provide a Python wrapper for the library to simplify its use in applied solutions. Our evaluation shows that operations specialized for Boolean matrices can be up to 2 times faster and consume up to  $YYY$  times less memory than generic operations from modern libraries. We hope that our results help to move the development of a GPGPU version of GraphBLAS API forward.

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

## I. INTRODUCTION

One technique to efficiently solve a data analysis problem is to formulate it in terms of operations over vectors and matrices (in terms of linear algebra). This way it is possible to employ a set of reliable mathematical tools and solutions. Another advantage of this approach is the ability to evaluate the problem with *zero-cost* by high-performance linear algebra libraries, which utilize modern hardware, provide various optimization techniques, and allow one to prototype a solution in code with predefined building blocks quickly and safely. GraphBLAS API<sup>1</sup> [1] is one of the standards that introduce such building blocks. GraphBLAS takes into account the sparsity of data by using sparse formats of matrices and vectors, and generalizes the building blocks by operating with arbitrary *monoids* and *semirings*. 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].

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

<sup>1</sup>GraphBLAS project web page: <https://graphblas.github.io/>. Access date: 19.01.2021.

GPGPU utilization for data analysis and for linear algebra operations is a promising way to high-performance data analysis because GPGPU is much more powerful in parallel data processing. Unfortunately, GPGPU programming is very challenging. It introduces heterogeneous device model into the system, memory traffic, and data operations limitations, as well as requires taking into account vendor-specific capabilities. Best to our knowledge, there is no complete implementation of GraphBLAS API on GPGPU, except for the GraphBLAST project<sup>2</sup> [4], which is currently in active development.

The sparsity of data introduces issues with load balancing, irregular data access, thus sparsity complicates the implementation of high-performance algorithms for sparse linear algebra on GPGPU even more. 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. See [5] for a significant survey of sparse matrix-matrix multiplication algorithms. Algorithms for different operations, such as matrix-matrix multiplication and matrix-vector multiplication are developed independently. Thus, there are no sparse linear algebra libraries based on the state-of-the-art algorithms. Moreover, existing libraries, such as cuSparse<sup>3</sup>, clSparse<sup>4</sup> [6], or more modern CUSP<sup>5</sup> or bhSparse<sup>6</sup> [7], are focused on numerical computations over floats or doubles, not on generic data processing over arbitrary semirings which is required for GraphBLAS API implementation.

An important partial case of linear algebra is the sparse Boolean linear algebra. Boolean algebra is suitable for problems over a finite set of values, such as transitive closure of a relation or a graph, regular and context-free path queries for graphs [8], as well as parsing for different classes of languages,

<sup>2</sup>GraphBLAST project: <https://github.com/gunrock/graphblast>. Access date: 19.01.2021.

<sup>3</sup>NVIDIA sparse matrix library (in Cuda) <https://docs.nvidia.com/cuda/cusparsel/>. Access date: 19.01.2021.

<sup>4</sup>Sparse linear library functions in OpenCL: <http://clmathlibraries.github.io/clSPARSE/>. Access date: 19.01.2021.

<sup>5</sup>CUSP sparse linear algebra library: <https://cusplibrary.github.io/modules.html>. Access date: 19.01.2021.

<sup>6</sup>bhSparse sparse matrix multiplication library: <https://github.com/weifengliu-ssslab/bhSPARSE>. Access date: 19.01.2021.

such as Context-Free [9], Boolean and Conjunctive [10], Multiple Context-Free(MCFL) [11]. Moreover, some operations over the Boolean semiring can be used as building blocks for algorithms over other semirings. 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, a library for sparse Boolean linear algebra on GPGPU does not exist.

In this paper, we present the implementation of sparse Boolean linear algebra operations as two stand-alone self-sufficient programming libraries for two most popular GPGPU platforms: NVIDIA Cuda<sup>7</sup> and OpenCL<sup>8</sup>. Cuda is a GPGPU technology for NVIDIA devices which employs some platform-specific facilities, such as unified memory mechanism, and make architectural assumptions which gives more optimizations space at the cost of portability. OpenCL is a platform-agnostic API standard, which allows for running computations on different platforms, such as multi-threaded CPUs, GPUs, and FPGAs. Our implementation relies on modern techniques of sparse matrices processing and 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. GENERAL INFORMATION

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

The architecture of the libraries is depicted in figure 1. The core of the libraries is written in the C++ programming language, which is well-suited for performance and resource critical computational tasks. The GPU related logic is in the platform specific backends: Cuda and OpenCL, which use respective technologies for resources and GPU executable code management. The cuBool library exposes C compatible API, which gives expressiveness and allows one to embed that API into other execution environments by interoperability mechanisms. Pycubool module encapsulates such functionality and provides it for the high-level Python runtime.

It is worth to mention, that it is convenient to create the single library with common interface and several backends

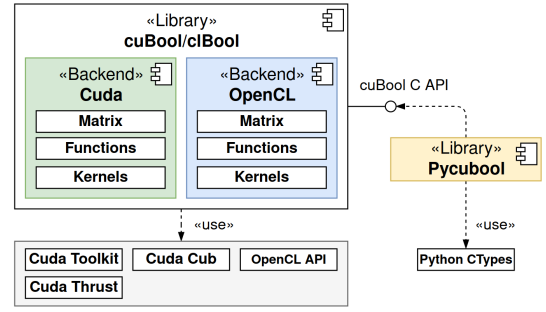


Fig. 1. Conceptual sparse boolean linear algebra library architecture

for different execution targets. At this time clBool and cuBool are distinct libraries, but they can be integrated into a single library. This integration is planned for the near future. This process requires careful selection of the interface to allow the end user to properly configure the library for specific tasks, as well as to provide the option to automatically select a specific implementation depending on the capabilities of the target device.

Libraries operate on the 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  $M$ .

The main primitive is a sparse matrix of boolean values, stored in one of the sparse formats. The sparse vector primitive is not supported, since it is rarely used in practical computational tasks. Other available operations and functions are the following.

- Create sparse matrix  $M$  of size  $m \times n$ .
- Delete sparse matrix  $M$ .
- Fill matrix with values  $\{(i, j)_k\}_k$ .
- Read matrix values  $\{(i, j) \mid M_{i,j} = 1\}$ .
- Matrix-matrix multiplication  $C += M \times N$ .
- Matrix-matrix element-wise addition  $M += N$ .
- 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 structure and architecture are similar, the 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. Library cuBool

cuBool is a sparse boolean linear algebra implementation developed specially for NVIDIA Cuda platform. The core of this library relies on Cuda C/C++ language and API. Also cuBool employs NVIDIA Thrust auxiliary library which provides implementation for generic data containers and operations, such as *iterating*, *exclusive or inclusive scan*, *map*, etc., which

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

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

<sup>9</sup>clBool project: [https://github.com/mkarpenkospb/sparse\\_boolean\\_matrix\\_operations](https://github.com/mkarpenkospb/sparse_boolean_matrix_operations). Access date: 03.02.2021.

<sup>10</sup>cuBool project: <https://github.com/JetBrains-Research/cuBool>. Access date: 03.02.2021.

are executed on Cuda device. The algorithms can be expressed in terms of high-level optimized primitives, which increases code readability and reduces time for development.

Sparse matrices are stored in the *compressed sparse row* (CSR) format with only two arrays: *rowspt* for row offset indices and *cols* for columns indices. There is no need to store any values in boolean sparse matrices, thus 1 values are encoded only as  $(i, j)$  pairs. This means that it is possible to store a matrix  $M$  of size  $m \times n$  in  $(m + \text{nnz}(M)) \times \text{sizeof}(\text{index\_t})$  bytes of GPU memory, where *index\_t* is the type of stored indices, which can be selected to be *uint32\_t* for simplicity.

We use the algorithm Nsparsrse [12] for matrix-matrix multiplication. This algorithm is an 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] for boolean values. This algorithm was selected because it has a 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 for more precise memory allocations in order to keep memory footprint small.

#### B. Library clBool

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

Sparse matrix primitive is stored in the *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  $m \times n$ , the memory consumption is  $2 \times \text{nnz}(M) \times \text{sizeof}(\text{index\_t})$ . This format was selected instead of CSR, because COO gives better memory footprint for very sparse matrices with many empty rows.

Matrix-matrix multiplication implementation is based on the algorithm, proposed in the paper by Weifeng Liu et al. [15]. It is a multi-step algorithm with dynamic workload balancing which operates on CSR matrices. Since clBool primary primitive is COO matrix, the input matrices are converted into *doubly compressed sparse row* (DCSR) format, described in A. Buluc et al. work [16], before matrix-matrix multiplication. This algorithm is suitable for OpenCL implementation, which is confirmed by its utilization in clSPARSE library.

Matrix-matrix addition is based on the GPU Merge Path algorithm as well. It is implemented in the classic fashion: it allocates a single merge buffer of size  $\text{nnz}(A) + \text{nnz}(B)$  before the actual merge of matrices  $A$  and  $B$ , what can negatively affect the 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

TABLE I  
SPARSE MATRIX DATA FOR EVALUATION

Matrix $M$	#Rows	Nnz of $M$	Nnz of $M^2$	Nnz of $M + M^2$
1. wing	62,032	243,088	714,200	917,178
2. luxembourg_osm	114,599	239,332	393,261	632,185
3. amazon0312	400,727	3,200,400	14,390,544	14,968,909
4. amazon-2008	735,323	5,158,388	25,366,745	26,402,678
5. web-Google	916,428	5,105,039	29,710,164	30,811,855
6. roadNet-PA	1,090,920	3,083,796	7,238,920	9,931,528
7. roadNet-TX	1,393,383	3,843,320	8,903,897	12,264,987
8. belgium_osm	1,441,295	3,099,940	5,323,073	8,408,599
9. roadNet-CA	1,971,281	5,533,214	12,908,450	17,743,342
10. netherlands_osm	2,216,688	4,882,476	8,755,758	13,626,132

designed as 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 i7-4790 CPU, 3.6GHz, DDR4 32Gb RAM and GeForce GTX 1070 GPU with 8Gb VRAM. We measure only 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 the 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 a 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. However this limitation can be ignored, if we consider non-zero float values as *true*. SuiteSparse is a GraphBLAS API reference implementation for CPU with built-in boolean semiring.

For performance evaluations, we selected 10 various square matrices, which are widely used for sparse matrices benchmarks, from the Sparse Matrix Collection at University of Florida<sup>11</sup>. Information about matrices is summarized in table I.

The results of the evaluation are summarized in tables II and III. Time is measured in milliseconds. Peak VRAM memory usage is measured in megabytes. The result for each experiment is averaged over 10 runs. The cell is left blank if the operation is not implemented by a library.

The first experiment is intended to measure the performance of the matrix-matrix multiplication as  $M \times M$ . The results are presented in the table II. We can see that cuBool shows nearly best performance among competitors. clBool has good performance as well, comparable to major Cuda competitors or clSPARSE in some cases. However, CUSP and clSPARSE have significant memory consumption, which can negatively affect on processing of large data.

The second experiment is intended to measure performance of the element-wise matrix-matrix addition as  $M + M^2$ , where evaluation of the matrix  $M^2$  is excluded from measurements. The results are presented in the table III. The numbers obtained in this experiment are more ambiguous than in the previous

<sup>11</sup>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  
(MILLISECONDS, MEGABYTES).

M	CuBool		CUSP		CuSprs		ClBool		ClSprs		SuiteSprs
	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time
1.	2.25	215	5.83	125	20.2	155	60.5	95	127	109	10.0
2.	2.97	213	4.17	111	1.78	149	16.0	91	10.8	99	2.53
3.	24.6	215	110	897	411	301	97.6	279	65.7	459	238
4.	38.9	341	173	1409	182	407	110	401	104	701	339
5.	50.1	341	240	1717	4756	439	277	491	409	1085	644
6.	21.7	215	43.3	481	37.6	247	45.6	203	85.5	283	63.0
7.	26.6	215	52.1	581	46.8	271	55.8	229	107	329	74.9
8.	26.9	215	33.8	397	26.7	235	68.6	183	104	259	57.8
9.	37.6	215	76.4	771	67.2	325	77.7	279	151	433	110
10.	40.4	215	51.9	585	51.1	291	78.1	251	158	361	93.0

TABLE III  
ELEMENT-WISE MATRIX-MATRIX ADDITION EVALUATION RESULTS  
(MILLISECONDS, MEGABYTES).

M	CuBool		CUSP		CuSprs		ClBool		ClSprs		SuiteSprs
	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time
1.	1.14	97	1.58	103	2.47	163	22.8	105	-	-	4.06
2.	1.79	103	1.18	103	0.92	159	4.52	103	-	-	1.56
3.	9.44	237	16.5	455	24.1	405	84.0	543	-	-	35.1
4.	16.2	347	29.1	723	23.6	595	163	877	-	-	61.2
5.	18.7	379	32.3	815	88.7	659	176	989	-	-	72.5
6.	15.4	207	11.6	329	11.8	317	66.4	359	-	-	34.0
7.	19.3	231	14.0	385	14.7	357	73.2	429	-	-	41.8
8.	19.6	197	10.2	303	10.3	297	61.8	321	-	-	26.8
9.	27.1	289	19.5	513	20.3	447	135	579	-	-	61.4
10.	33.1	263	15.2	423	18.2	385	76.1	457	-	-	47.0

experiment. cuBool, CUSP and cuSPARSE show nearly best performance among almost all experiments. Memory consumption for cuBool is relatively small compared to CUSP and cuSPARSE. clBool generally keeps its results within acceptable limits. However, it still lags behind SuiteSparse. There is still space for optimizations, so it requires a deep investigation in our future research.

## V. CONCLUSION

In this paper 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. Evaluation shows that our Boolean-specific implementations faster and require less memory than generic operations from state-of-the-art libraries.

The first direction of the future work is to integrate all parts (OpenCL and Cuda backends) into a single library and improve its documentation and prepare to publish. Moreover, it is necessary to extend the library with other operations, including matrix-vector operations, masking, and so on. As a result a Python package should be published.

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. Buluc, "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. Buluc, 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. Pola, K. Varaganti, and M. Daga, "Cisparse: A vendor-optimized open-source sparse blas library," in *Proceedings of the 4th International Workshop on OpenCL*, ser. IWOC '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](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-adjointing 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.