SPbLA: The Library of GPGPU-Powered Sparse Boolean Linear Algebra Operations

Egor Orachev
Saint Petersburg State University
JetBrains Research,
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 5 times faster and consume up to 4 times less memory than generic, not the Boolean optimized, 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 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 [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].

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 [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. Gao et al. [5] provides 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 [6], clSPARSE [7], or more modern CUSP [8] or bhSPARSE [9], 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 [10], as well as parsing for different classes of languages, such as Context-Free [11], Boolean and Conjunctive [12], Multiple Context-Free (MCFL) [13]. 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 (SPbLA) library as two stand-alone self-sufficient computational backends for two most popular

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

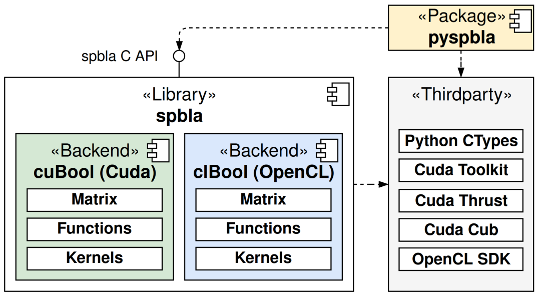

Fig. 1. Sparse Boolean linear algebra library architecture.

GPGPU platforms: NVIDIA Cuda and OpenCL. 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 5 times faster and consume up to 4 times less memory in comparison with general-purpose, not the Boolean optimized, operations from such libraries as CUSP or cuSPARSE.

II. Libraries Design

Implemented SPbLA library backends for NVIDIA Cuda and OpenCL platforms are called *cuBool* and *clBool* respectively. The general architecture of the SPbLA is depicted in figure 1. The core of the library 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. The library exposes C compatible API, which gives expressiveness and allows one to embed that API into other execution environments by interoperability mechanisms. Pyspbla package encapsulates such functionality and provides it for the high-level Python runtime.

It is worth to mention, that the library is still being worked on. At this time clBool and cuBool are distinct backends, but they will 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.

However, cuBool already provides all the functionality described in the figure 1. It has a C compatible API, multiple backends for Cuda and CPU computations, a Python wrapper, and it is the lightweight version of the SPbLA without OpenCL computations.

Library operates on the boolean semiring with values set $\{true, false\}$ with *false* as an identity element, '$+$' operation is defined as logical *or* and '$\times$' is 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 is partially presented. Its full support will be added in the future. All 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 $\{(i,j) \mid M_{i,j} = 1\}$.
- Transpose $M = N^T$.
- Sub-matrix extraction $M = N[i..m, j..n]$.
- Matrix to vector reduce $V = \mathit{reduceToColumn}(M)$.
- Matrix-matrix multiplication $C \mathrel{+}= M \times N$.
- Matrix-matrix element-wise addition $M \mathrel{+}= N$.
- Matrix-matrix Kronecker product $K = M \otimes N$.

III. Implementation Details

In this section we discuss the particular implementation details of the proposed backends. 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. Backend cuBool

cuBool[1] is a sparse boolean linear algebra implementation developed specially for NVIDIA Cuda platform. The core of this backend relies on Cuda 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 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 $(m + \mathit{nnz(M)}) \times \mathit{sizeof(index\_t)}$ 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 Nsparse [14] 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

[1] cuBool project: https://github.com/JetBrains-Research/cuBool. Access date: 10.02.2021.

et al. research [15] for boolean values. This algorithm was selected because it has a relatively small memory footprint for large matrices processing, as well as it compares favorably with other major Cuda SpGEMM implementations, such as cuSPARSE or CUSP.

Matrix-matrix addition is based on GPU Merge Path algorithm [16] 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. Backend clBool

clBool[2] is a sparse boolean linear algebra implementation for OpenCL platform. This backend is implemented in C++ with OpenCL kernels, packed into executable code at compile time.

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 $2 \times$*nnz(M)*$\times$*sizeof(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 Yusuke Nagasaka et al. [15] as well. Since this algorithm originally was designed for processing CSR matrices on Cuda devices, it is modified to work on OpenCL platform. In particular, input matrices are converted from COO into *doubly compressed sparse row* (DCSR) [17] format, because a redundant COO rows array slows down the rows indexing process.

Matrix-matrix addition is based on the GPU Merge Path algorithm as well. Since all COO matrix values are stored continuously, its addition can be treated as the merge of two sorted arrays, whereas the matrix merge in CSR is computed on a per row basis. This operation is implemented in two steps: merge and duplicates reduce. In the first step it allocates a single merge buffer of size *nnz(A)*$+$*nnz(B)*, where merge result is stored with possible duplicates. Although this approach is simple and straightforward, it 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 backends for analysis of some real-world matrix data. The experiments are designed as computational tasks, that arise as stand-alone or intermediate steps in the solving of practical problems. The purpose of the evaluation is to show the performance gain between the Boolean optimized and general-purpose operations. The comparison is not entirely fair, but the Boolean optimized libraries for GPU have not be introduced yet.

For evaluation, we used a PC with Ubuntu 20.04 installed. It has Intel Core i7-6700 CPU, 3.40Hz, DDR4 64Gb 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.

TABLE I
Sparse matrix data for evaluation.

№	Matrix M	$\#Rows$	Nnz of M	Nnz of M^2	Nnz of $M+M^2$
0	wing	62,032	243,088	714,200	917,178
1	luxembourg_osm	114,599	239,332	393,261	632,185
2	amazon0312	400,727	3,200,400	14,390,544	14,968,909
3	amazon-2008	735,323	5,158,388	25,366,745	26,402,678
4	web-Google	916,428	5,105,039	29,710,164	30,811,855
5	roadNet-PA	1,090,920	3,083,796	7,238,920	9,931,528
6	roadNet-TX	1,393,383	3,843,320	8,903,897	12,264,987
7	belgium_osm	1,441,295	3,099,940	5,323,073	8,408,599
8	roadNet-CA	1,971,281	5,533,214	12,908,450	17,743,342
9	netherlands_osm	2,216,688	4,882,476	8,755,758	13,626,132

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 [18]. Information about matrices is summarized in table I. These matrices were selected because they correspond to (un)directed graphs and they are suitable for a correct application of multiplication and addition operations. For a detailed study, it is necessary to carry out measurements on specific algorithms and data.

The results of the evaluation are summarized in tables II and III. Time is measured in milliseconds. Peak VRAM usage for GPU targets and peak RAM usage for SuiteSparse 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 and clBool show best performance among competitors. CUSP, cuSPARSE and clSPARSE have good performance as well. However, they 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. CUSP and cuSPARSE show nearly best performance among almost all runs. cuBool and clBool show good performance as well. Memory consumption for cuBool is relatively small compared to other GPU

[2]clBool project: https://github.com/mkarpenkospb/sparse_boolean_matrix_operations. Access date: 10.02.2021.

TABLE II
MATRIX-MATRIX MULTIPLICATION EVALUATION RESULTS
(TIME IN MILLISECONDS, MEMORY IN MEGABYTES).

M	cuBool		clBool		CUSP		cuSPRS		clSPRS		SuiteSprs	
№	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem
0	1.9	93	1.9	89	5.2	125	20.1	155	4.2	105	7.9	22
1	2.4	91	2.0	89	3.7	111	1.7	151	6.9	97	3.1	169
2	23.2	165	55.5	163	108.5	897	412.8	301	52.2	459	257.6	283
3	33.3	225	82.1	221	172.0	1409	184.8	407	77.4	701	369.5	319
4	41.8	241	127.6	239	246.2	1717	4761.3	439	207.5	1085	673.3	318
5	18.1	157	14.2	153	42.1	481	37.5	247	56.6	283	66.6	294
6	22.6	167	16.9	165	53.1	581	46.7	271	70.4	329	80.7	328
7	23.2	151	16.9	159	32.9	397	26.7	235	68.2	259	56.9	302
8	32.0	199	23.4	211	74.4	771	65.8	325	98.2	433	114.5	344
9	35.3	191	24.9	189	51.0	585	51.4	291	102.8	361	90.9	311

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

M	cuBool		clBool		CUSP		cuSPRS		clSPRS		SuiteSprs	
№	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem
0	1.1	95	1.9	105	1.4	105	2.4	163	-	-	2.3	176
1	1.7	95	1.6	109	1.0	97	0.8	151	-	-	1.6	174
2	11.4	221	23.8	543	16.2	455	24.3	405	-	-	37.2	297
3	17.5	323	35.4	877	29.5	723	27.2	595	-	-	64.8	319
4	24.8	355	43.1	989	31.9	815	89.0	659	-	-	77.2	318
5	16.9	189	12.5	359	11.2	329	11.6	317	-	-	36.6	287
6	19.6	209	15.4	429	14.5	385	16.9	357	-	-	45.3	319
7	19.5	179	10.5	321	10.2	303	10.5	297	-	-	28.5	302
8	30.5	259	22.4	579	19.4	513	20.2	447	-	-	65.2	331
9	30.1	233	18.6	457	14.8	423	18.3	385	-	-	50.2	311

libraries. Thus, there is still space for clBool 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 and OpenCL. Evaluation shows that our Boolean-specific implementations faster and require less memory than generic, not the Boolean optimized, operations from state-of-the-art libraries. Thus, the specialization of operations for this data type makes sense.

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 foundation 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] "Sparse matrix library (in cuda)." [Online]. Available: https://docs.nvidia.com/cuda/cusparse/

[7] 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

[8] S. Dalton, N. Bell, L. Olson, and M. Garland, "Cusp: Generic parallel algorithms for sparse matrix and graph computations," 2014, version 0.5.0. [Online]. Available: http://cusplibrary.github.io/

[9] 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

[10] 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

[11] 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

[12] 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

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

[14] 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

[15] 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.

[16] 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

[17] 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.

[18] T. Davis, "Suitesparse matrix collection (the university of florida sparse matrix collection)." [Online]. Available: https://sparse.tamu.edu/