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

Egor Orachev [1,3], **Maria Karpenko** [2], **Pavel Alimov** [1], **and Semyon Grigorev** [1,3]

1 Saint Petersburg State University **2** ITMO University **3** JetBrains Research

DOI: 10.21105/joss.03743

Software

- Review
- Repository
- Archive

Editor: Nikoleta Glynatsi

Reviewers:

- @mlxd
- @bencardoen

Submitted: 07 September 2021
Published: 19 August 2022

Summary

`SPbLA` is a sparse Boolean linear algebra primitives and operations for GPGPU computations. It comes as a stand-alone self-sufficient library with C API for high-performance computing with multiple backends for Nvidia Cuda, OpenCL and CPU-only platforms. The library has PyPI `pyspbla` package (Orachev et al., 2021) for work within a Python runtime. The primary library primitive is a sparse matrix of Boolean values. The library provides the most popular operations for matrix manipulation, such as construction from values, transpose, sub-matrix extraction, matrix-to-vector reduce, matrix-matrix element-wise addition, multiplication and Kronecker product.

Statement of need

Answering research questions in data analysis often involves expressing the solution in terms of matrix/vector operations. This way it is possible to leverage a set of powerful sparse linear algebra libraries.
`GraphBLAS API` provides a set of unified linear algebra based building blocks for reducing analysis 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 the complete set of sparse operations on GPGPU challenging. Thus, it is worth addressing this problem by focusing on a basic but still important case — sparse Boolean algebra.

The primary goal of the `SPbLA` is to provide a base for the implementation, testing and profiling high-performance algorithms for solving data analysis problems, such as RDF analysis (X. Zhang et al., 2015), RNA secondary structure analysis (Anderson et al., 2013), static code analysis (such as points-to or alias analysis) (Q. Zhang et al., 2013) and evaluation of regular and CFL-reachability queries (Azimov & Grigorev, 2018; Orachev et al., 2020).

Thus, we can offload different language-constrained path querying related problems, and other problems that can be reduced to manipulation of Boolean matrices, to GPGPU uniformly.

Moreover, real world data analysis leads to huge matrix processing that can not be efficiently handled on a single GPGPU. The creation of the library which supports multi-GPU and out-of-VRAM computations helps to create an efficient solution for a wide range of applied problems. The creation of such a solution is an open problem while ad-hoc solutions exist in specific areas. We propose an `SPbLA` as a base for such a solution.

Also, we hope that the library is a small step to move the implementation of the fully-featured sparse linear algebra as specified in `GraphBLAS` forward multi-GPU computations.

Related tools

`GraphBLAS API` (Kepner et al., 2016) is one of the first standards that formalize the mathematical building blocks in the form of the programming interface for implementing algorithms in the language of linear algebra. `SuiteSparse` (T. A. Davis, 2019) is a reference implementation of the `GraphBLAS API` for CPU computation. It is a mature and fully featured library with a number of bindings for other programming languages, such as `pygraphblas` (*Pygraphblas*, 2021) for Python programming.

GPGPU's 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. However, GPGPU programming is still challenging. To the best of our knowledge, there is no complete `GraphBLAS API` implementation for GPGPU computations, except `GraphBLAST` (Yang et al., 2019), which is currently in active development. Some work is also done to move `SuiteSparse` forward GPGPU computations.

However, 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. There are a number of open-source and proprietary libraries, which implement independently different sparse formats and operations. Thus, there is no single solid sparse linear algebra framework. Libraries such as `cuSPARSE` (*Sparse Matrix Library in Cuda*, n.d.), `bhSPARSE` (Liu & Vinter, 2015), `clSPARSE` (Greathouse et al., 2016) and `CUSP` (Dalton et al., 2014) have limited type and operators customization features with major focus on numerical types only.

Performance

We evaluate the utility of the proposed library for some real-world matrix data. The experiment itself is designed as a computational task, which arises as a stand-alone or intermediate step in the solving of practical problems. Results of the evaluation compared to CPU `SuiteSparse` and existing GPU sparse linear algebra libraries. The comparison is not entirely fair, since there are still no Boolean linear algebra libraries for GPU computations.

Machine for performance evaluation has the following configuration: PC with OS Ubuntu 20.04 installed, Intel Core i7-6700 3.4Hz CPU, 64Gb DDR4 RAM, GeForce GTX 1070 GPU with 8Gb VRAM.

Matrix name	# Rows	Nnz M	Nnz/row	Max Nnz/row	Nnz M^2
amazon0312	400,727	3,200,440	7.9	10	14,390,544
amazon-2008	735,323	5,158,388	7.0	10	25,366,745
web-Google	916,428	5,105,039	5.5	456	29,710,164
roadNet-PA	1,090,920	3,083,796	2.8	9	7,238,920
roadNet-TX	1,393,383	3,843,320	2.7	12	8,903,897
roadNet-CA	1,971,281	5,533,214	2.8	12	12,908,450
netherlands_osm	2,216,688	4,882,476	2.2	7	8,755,758

For evaluation, we selected a number of square real-world matrices, widely applicable for sparse matrices benchmarks, from the Sparse Matrix Collection at University of Florida (T. Davis, n.d.). Information about matrices summarized above. Table contains matrix name, number of rows in the matrix (the same as number of columns), number of non-zero elements (Nnz) in the matrix, average and maximum nnz in row, nnz in the result matrix.

The experiment is intended to measure the performance of matrix-matrix multiplication as $M \times M$. Results of the evaluation presented in Figure 1 and Figure 2. Results averaged among 10 runs. The deviation of results does not exceed 10%. Best and worst results highlighted. Extra warm-up run, required for initialization and kernels compilation, is excluded from measurements.

`SPbLA` library shows the best performance among competitors for both OpenCL and Nvidia Cuda backends. `CUSP` and `cuSPARSE` show good performance as well. However, they have significant memory consumption in some cases, which can be a critical limitation in practical analysis tasks. `SuiteSparse` library on CPU has acceptable performance characteristics, and it is still a good alternative for CPU-only computations.

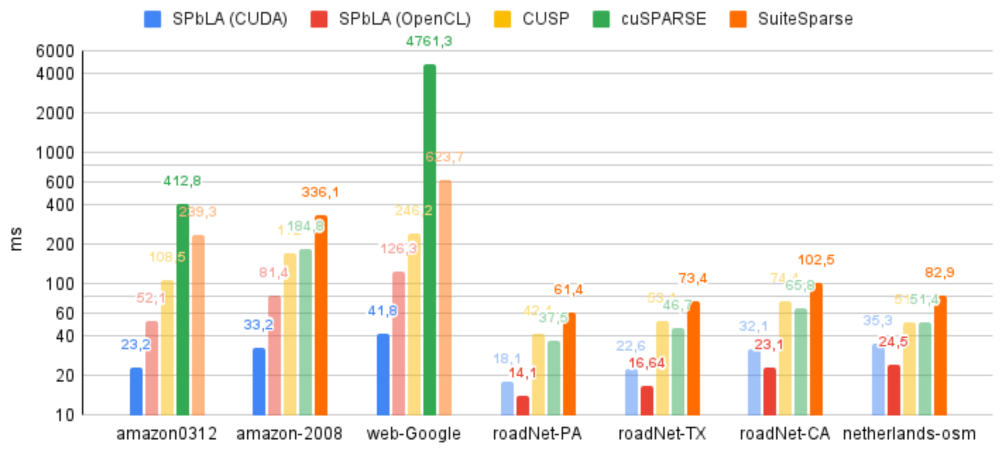

Figure 1: Matrix-matrix multiplication time consumption. Time in milliseconds. Lower is better.

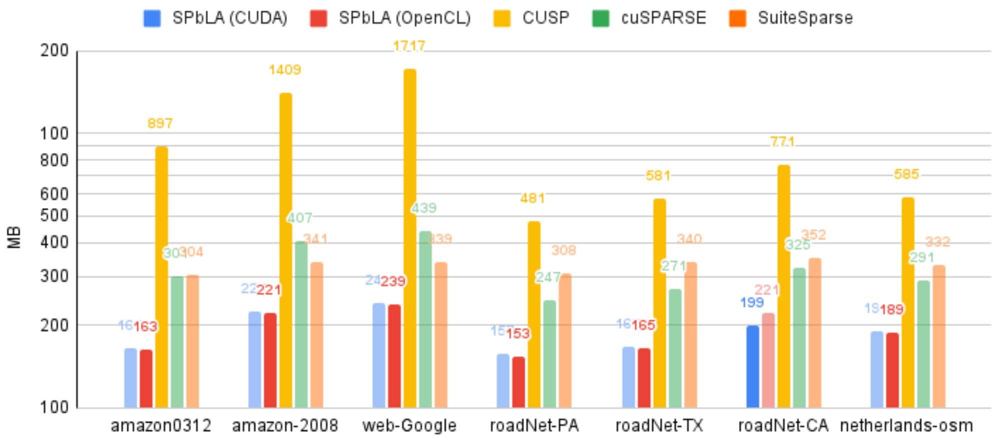

Figure 2: Matrix-matrix multiplication memory consumption. Memory in megabytes. Lower is better.

Future research

First direction of the future research is library extension to multi-GPU environment support. This step introduces a number of issues, such as memory management among computational units as well as proper workload dispatch and granularity of parallel tasks. A potential solution is to use a hybrid sparse matrix format, such as quadtree or blocked storage, and utilize virtual memory. It is necessary to expose more control over expressions evaluations to the user in order to support matrix and expression level granularity among computational units.

Finally, we plan to generalize computational kernels and primitives in order to support arbitrary types and operations, defined by the user. This step will allow defining custom elements and functions, which will be executed on GPU similarly as it is done for predefined Boolean values.

Acknowledgements

Work on the SPbLA project was supported by a grant from JetBrains Research.

References

Anderson, J., Novák, A., Sükösd, Z., Golden, M., Arunapuram, P., Edvardsson, I., & Hein, J. (2013). Quantifying variances in comparative RNA secondary structure prediction. *BMC Bioinformatics*, *14*, 149. https://doi.org/10.1186/1471-2105-14-149

Azimov, R., & Grigorev, S. (2018). *Context-free path querying by matrix multiplication*. 1–10. https://doi.org/10.1145/3210259.3210264

Dalton, S., Bell, N., Olson, L., & Garland, M. (2014). *Cusp: Generic parallel algorithms for sparse matrix and graph computations*. http://cusplibrary.github.io/

Davis, T. (n.d.). *SuiteSparse matrix collection (the university of florida sparse matrix collection)*. https://sparse.tamu.edu/

Davis, T. A. (2019). Algorithm 1000: SuiteSparse:GraphBLAS: Graph algorithms in the language of sparse linear algebra. *ACM Trans. Math. Softw.*, *45*(4). https://doi.org/10.1145/3322125

Greathouse, J. L., Knox, K., Poła, J., Varaganti, K., & Daga, M. (2016). CISPARSE: A vendor-optimized open-source sparse BLAS library. *Proceedings of the 4th International Workshop on OpenCL*. https://doi.org/10.1145/2909437.2909442

Kepner, J., Aaltonen, P., Bader, D., Buluc, A., Franchetti, F., Gilbert, J., Hutchison, D., Kumar, M., Lumsdaine, A., Meyerhenke, H., McMillan, S., Yang, C., Owens, J. D., Zalewski, M., Mattson, T., & Moreira, J. (2016). Mathematical foundations of the GraphBLAS. *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, 1–9. https://doi.org/10.1109/HPEC.2016.7761646

Liu, W., & Vinter, B. (2015). A framework for general sparse matrix-matrix multiplication on GPUs and heterogeneous processors. *J. Parallel Distrib. Comput.*, *85*(C), 47–61. https://doi.org/10.1016/j.jpdc.2015.06.010

Orachev, E., Epelbaum, I., Azimov, R., & Grigorev, S. (2020). *Context-free path querying by kronecker product* (pp. 49–59). https://doi.org/10.1007/978-3-030-54832-2_6

Orachev, E., Karpenko, M., Alimov, P., & Grigorev, S. (2021). *SPbLA: Sparse boolean linear algebra for CPU, cuda and OpenCL computations*. https://pypi.org/project/pyspbla/

Pygraphblas: A python wrapper around the GraphBLAS API. (2021). Github. https://github.com/Graphegon/pygraphblas

Sparse matrix library in cuda. (n.d.). https://docs.nvidia.com/cuda/cusparse/

Yang, C., Buluç, A., & Owens, J. D. (2019). GraphBLAST: A high-performance linear algebra-based graph framework on the GPU. *arXiv Preprint*.

Zhang, Q., Lyu, M. R., Yuan, H., & Su, Z. (2013). Fast algorithms for dyck-CFL-reachability with applications to alias analysis. *SIGPLAN Not.*, *48*(6), 435–446. https://doi.org/10.1145/2499370.2462159

Zhang, X., Feng, Z., Wang, X., Rao, G., & Wu, W. (2015). Context-free path queries on RDF graphs. *CoRR*, *abs/1506.00743*. https://doi.org/10.1007/978-3-319-46523-4_38