

Hardware and Software Co-Design for Sparse Linear Algebra Operations Acceleration

Aleksey Tyurin
Saint Petersburg State University
Saint Petersburg, Russia
alekseytyurinspb@gmail.com

Daniil Berezun
Saint Petersburg State University
JetBrains Research
Saint Petersburg, Russia
d.berezun@2009.spbu.ru

Semyon Grigorev
Saint Petersburg State University
JetBrains Research
Saint Petersburg, Russia
s.v.grigoriev@spbu.ru

Abstract—In the era of big data, computations are expected to be faster and less power-consuming in order to become more effective and affordable. And when modern CPUs and GPUs appear to be underutilized due to being too general-purposed for some problems, application-specific hardware accelerators are able to provide the needed performance. This is particularly the case for sparse linear algebra problems for which a number of hardware accelerators have been designed. In this work, we elaborate on the optimizations that are missed out in present accelerators due to the lacking of a software part of an accelerator. We propose a co-design approach that could potentially exploit such optimizations and still have decent hardware acceleration.

INTRODUCTION

Linear algebra is a great instrument for solving a wide variety of problems utilizing matrices and vectors for data representation and analysis with the help of highly optimized routines. And whilst the matrices involved in a vast diversity of modern applications, e.g., recommender systems [1], [2] and graph analysis [3], [4], consist of a large number of elements, the major part of them are zeros. Such a high sparsity incurs both computational and storage inefficiencies, requiring unnecessarily large storage, occupied by zero elements, and a large number of operations on zeroes, where the result is obviously known beforehand. The traditional approach to address these inefficiencies is to compress the matrix and store only the non-zero elements. Thus, the effect of matrices tending to be sparse in many applications makes the techniques of matrix compressed representation and sparse linear algebra to be the effective way of tackling problems in areas including but not limited to graph analysis [5], computational biology [6], and machine learning [7].

Sparse linear algebra defines primitives for expressing algorithms for the mentioned areas in a uniform way in terms of sparse matrix and vector operations parameterized by a semiring. Such uniform representation allows tuning the whole bunch of expressible algorithms through optimizing the primitives solely. One of the most used primitive is a sparse matrix-sparse matrix multiplication (spMspM) operation. It has finely-tuned implementations for both CPU and GPU, which, however, are proven to be underutilized due to the memory-bound nature of sparse computations induced by compressed representation [8]–[11]. Further, the pipeline of spMspM is patchy, which makes some of the computational

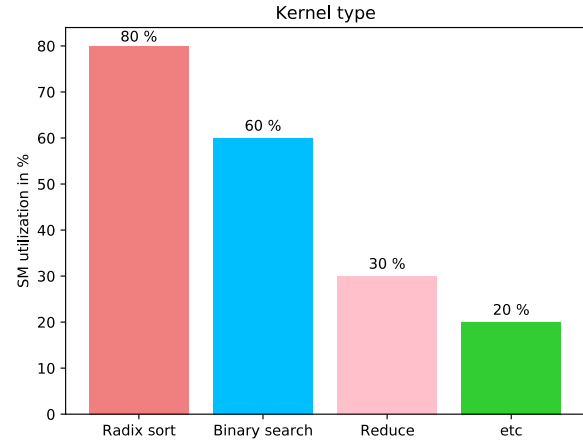


Fig. 1. GPU's SM utilization for spMspM pipeline from cuSPARSE²

```
-- A,B,C,D are sparse matrices
-- M is a mask
D<M> = A eWiseAdd B eWiseMult C
```

Listing 1: Sequence of sparse operations example

units to be idle from time to time as it could be seen in figure 1, while the peak FLOPS is less than 1% of maximum available¹.

This makes the typical CPUs and GPUs not well-suited hardware for sparse computations and gives a rise to specialized hardware accelerators, which are primarily concerned with spMspM. However, for a sparse framework to be useful, it should incorporate not only spMspM, but also other sparse operations like in listing 1, where masking, which filters the matrix elements, and element-wise operations (possibly parameterized by a semiring) needed for, e.g., PageRank and bread-first search (BFS) algorithms [12], are presented. Such operations are assembled in *GraphBLAS* [13] specification and could be used as building blocks to construct many algorithms [5]–[7]. When such operations are chained explicitly or implicitly, via a loop body, certain optimizations could be

¹<https://hanlab.mit.edu/projects/sparch/> (Accessed 09.02.2021)

²<https://developer.nvidia.com/cusparse> (Accessed 09.02.2021)

applied, like the one that eliminates intermediate matrices in sequence from listing 1. Unfortunately, some of such optimizations (e.g., the one mentioned) are only expressible at a software level, i.e., in a programming language, hence modern spMSPM accelerators could be impractical for accelerating the whole program representing a sparse linear algebra-based algorithm like PageRank or BFS, due to the lack of a software part and to a too narrow hardware specialization.

Thus a co-design of dedicated hardware and software components, i.e., domain-specific processor (DSP) and a corresponding domain-specific language (DSL), could provide a system that is not more efficient for spMSPM than present hardware accelerators, but appear to be more effective in terms of speed and power consumption for holistic pieces of program, i.e., for chained operations, than current CPUs and GPUs implementations. The ongoing work is devoted to the design of respective DSL, DSP, and an optimizing compiler, and this work, in particular, gives a brief overview of the field, discusses the ideas and challenges behind the design.

I. PROBLEM STATEMENT

Since sparse linear algebra applications are mostly concerned with graph problems, some of the optimizations are graph-specific [12], [14], e.g., direction optimization. However, in memory-bound applications optimizations that minimize data transfer are essential. For example, *kernel fusion* is a wide-addressed optimization that fuses multiple operations into one, avoiding intermediate memory accesses, utilizing, e.g., registers to pass the data between the operations. In the case of sparse linear algebra frameworks kernel fusion is not yet widely implemented, but most often related to fusing chained operations like from listing 1 to avoid intermediate matrices construction and reduce memory accesses with masking [12].

The problem of intermediate data structures is common for functional programming and there have been developed a number of optimization techniques that try to reduce intermediate data structures or computations, namely partial evaluation [15], deforestation [16], supercompilation [17], and distillation [18].

These fusion family optimizations are implementation-dependent, meaning that the optimizer should have an access to the source code, which is impossible when the function is implemented solely in hardware, hence a software part is inevitable in the design of a system that tries to put together fusion and hardware. Thus, present hardware accelerators are not general enough to implement and accelerate a whole sparse linear algebra-based program, e.g., do not provide arbitrary semiring support, and lack a software part hence leaving out essential optimizations. General-purpose devices such as GPUs are hard to perform some optimizations, e.g., fusion requires two GPU kernels to have the same memory access pattern and partial evaluation could induce some thread divergence due to SIMD nature of a GPU. Further, present systems that support fusion are domain-specific³, or perform

the optimization on top of data structures that may not be suitable for sparse operations (mainly for effective compressed representation), e.g., streams and lists [19], [20], while array-based fusion systems do not perform fusion with index arithmetic [21]. Finally, some data needed for optimizations [15] is only available in runtime, thus there should be support for JIT optimizations. The design of the proposed hardware-software system should take these challenges into consideration. Next, we discuss hardware accelerators from adjacent areas and elaborate on the software optimization we want to exploit.

II. GRAPH PROCESSORS

Sparse linear algebra is often concerned with graph problems, and since graph processing is mature enough to be considered as a distinct computation paradigm, there have been emerged a number of graph processors [22] to mitigate the issues of graph processing on general-purpose processors. Presently, many proposed accelerator models are not generic, in the sense, they are optimized for specific classes of graph algorithms and lack scalability. Further, sparse linear algebra makes graph computations more amenable to effective parallelization, ease scalability, and even spreads beyond graph computations [6].

There is definitely a research interest in the design of sparse linear algebra-based instruction set processor [10], but the design in [10] at the moment provides only spMSPM operation and does not mention a software part as one of its key components. Other sparse linear algebra accelerators [11], [23], [24] are applicable only to certain operations like spMSPM. Notably, in [25] the way of compressed representation of a sparse matrix is identified as the main bottleneck and an approach for hardware acceleration for compressed representation indexing is proposed, which is able to employ existing optimizations and could be integrated into existing software frameworks. Hence, the underlying hardware is better to support the indexing needed by a compressed representation of choice to be effective. However, we believe that to fully leverage fusion the indexing should be transparent, i.e., constructive, in the software, which is not the case in [25].

III. SUPERCOMPILE

Supercompilation [17] could be thought of as a combination of many optimizations designed to reduce the number of data structures and the number of computations in general for functional programming languages [15], [16], [18]. In particular, partial evaluation [15] stands for precomputing the parts of a program that depend on statically known data. Interestingly, some data may become static in runtime, e.g., it could remain fixed between different iterations of a loop, thus it may be profitable to perform such optimization even in runtime. Despite such optimizations are common in the world of functional programming, they are hard to perform in the case of imperative languages widely used in high performance computing. However, a fully working supercompiler at the moment is developed only for a small functional language [26], since being too hard to implement for a rich

³<https://www.tensorflow.org/xla> (Accessed 09.02.2021)

language like Haskell. But we believe that such a small language is enough to provide a DSL for sparse linear algebra with the help of syntactic sugar and custom fusion rules. Also, such optimization may produce a burden of `case` statements, which slow down the hardware that exploits SIMD model, hence they could not be executed in parallel. Finally, fusion performed, e.g., by a supercompiler heavily depends on the compressed representation, i.e. presently quadtree representation seems to fit more for fusion than, e.g., coordinate format. Some operations could not be fused in principle, e.g. sorting, but the need for some of such operations could be induced by the compressed representation once again, e.g., quadtree representation does not require sorting, so it is reasonable to implement non-fusable operations in hardware.

IV. FUNCTIONAL LANGUAGE PROCESSORS

The functional DSL is not of much use until effectively compiled for DSP, therefore it is tempting to utilize state-of-the-art processors developed to add hardware support for functional programming [27]–[29]. Where [27] leverages the parallelization of independent function calls and also provides optimized support for lists and `map` operation specifically, which may not be the desired property due to the compressed representation of choice. The general-purpose nature of such processors could also hurt sparse operations performance due to non-specialized cache. Further, they focus on hardware support for graph reduction via parallelization, specialized memory, and pipelining, thus even more parallelism could be extracted from a compressed representation with their help.

However, present functional processors could be too complex to integrate with, e.g., to add hardware support for indexing or another computation that cannot be optimized effectively in software. Instead, we plan to opt for a more flexible solution and want to leverage the approach of dataflow representation of functional programs with indirect memory accesses [30], which then could be effectively represented in hardware in a highly parallel way. The approach currently generates application-specific RTL code, but we hope to adapt it for dataflow processor architecture, e.g., transport-triggered-architecture (TTA), in order to have a highly scalable and parallel processor with hardware support for a compressed representation, able to run functional programs and easily extensible with custom operations.

V. RESEARCH QUESTIONS AND ROADMAP

We propose the co-design of a hardware/software system for sparse linear algebra acceleration to address the mentioned problems and challenges of optimizeability and expressibility. We believe that a functional DSL, where an arbitrary semiring could be concisely and conveniently expressed, powered by a set of optimizers and compilable to some DSP with enough parallelism could be a good starting point. The solution should include the simultaneous design of the following components.

- Functional DSL to develop a library of sparse linear algebra operations, since functional programming language is appear to be well-suitable for supercompilation and

specialized hardware creation. The language should be minimized to simplify optimizations and compilation, but expressive enough to implement a sufficient subset of GraphpBLAS specification.

- Compiler from the functional DSL to the specialized hardware.
- Supercompiler for the functional DSL. The supercompiler should be designed for multistage processing. For example, the first stage could be a compile-time fusion of functions, and the second stage could be a partial evaluation on some data. This allows one to exploit dynamic data as static, and minimize overhead in running time.
- Library of sparse linear algebra operations, i.e., GraphBLAS specification implementation, in the functional DSL. We should choose such representation of vectors and matrices, and such principles of code design that both allow software optimizations and efficient hardware design should be possible.
- Special hardware, which should be general enough to execute all the code written in the functional DSL that utilizes the library of sparse linear algebra operations.

Such a multicomponent solution with nontrivial connections between components requires deep analysis in a number of areas. So, at the first stage, we plan to focus on the following questions.

- 1) Is supercompilation suitable for required optimizations? For example, is it able to remove intermediate matrices creation in chained operations, like presented in listing 1.
- 2) Can we generate program-specific hardware from the sparse linear algebra-based program written in a functional programming language such that it will be efficient in comparison with current solutions?
- 3) Is supercompilation suitable in functional-program-to-hardware workflow? Does a supercompiled program compile to more efficient hardware than the original one?
- 4) Does staged supercompilation allow to minimize partial evaluation time enough to apply partial evaluation in runtime?

As a first step, we plan to write a prototype comprising operations like multiplication, element-wise addition, Kronecker product, and masking in a minimalistic functional programming language (Haskell-core) and try to apply HOSC [26] to it with the focus on a more amenable to supercompilation quadtree compressed representation. It would allow us to estimate supercompilation effect in terms of expected optimizations like fusion, and time required for partial evaluation. Also, at this step, we should establish some principles of the library design that maximize supercompilation effect.

At the same time, we plan to evaluate functional language to hardware compiler [30] on the developed library to estimate the performance of the implemented operations and to compare basic and supercompiled versions.

As a result of this step, we expect to estimate applicability

of functional programming language and related optimization techniques for the creation of high-performance sparse linear algebra-based solutions as well as the suitability of dataflow approach to DSP architecture. In the end, we should provide a solution for program-specific accelerators creation for sparse linear algebra-based subprograms.

As far as the final goal is to create not a program-specific hardware accelerator, but a processor unit for high-performance sparse linear algebra-based computations, the second stage would be focused on the following strongly connected questions.

- 1) Can we generalize ideas from functional languages to hardware compilation to create a specific processor unit for sparse linear algebra? In other words, is that possible to generalize the dataflow approach of compiling a functional program to RTL description to compiling a functional program to a dataflow processor?
- 2) Can the designed language be efficiently compiled to the designed processor unit?

As a result of this stage, we hope to create a specialized processor unit for sparse linear algebra and a compiler from the functional DSL to it. We plan, that generalization will be based on the transformation of the program-specific dataflow diagram to TTA-like processor by reusing functional units with respective compiler development.

The final stage is the creation of SDK to integrate the DSL into an application. The SDK should provide tools for staged optimizations, communication with the special processor unit, and interoperability with general-purpose programming languages.

REFERENCES

- [1] U. Gupta, C.-J. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cotel, K. Hazelwood, B. Jia, H.-H. S. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. Zhang, "The architectural implications of facebook's dnn-based personalized recommendation," 2020.
- [2] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, no. 1, p. 76–80, Jan. 2003. [Online]. Available: <https://doi.org/10.1109/MIC.2003.1167344>
- [3] M. Besta, F. Marending, E. Solomonik, and T. Hoefer, "Slimsell: A vectorizable graph representation for breadth-first search," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017, pp. 32–41.
- [4] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1, pp. 107 – 117, 1998, proceedings of the Seventh International World Wide Web Conference. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S016975529800110X>
- [5] J. Kepner and J. Gilbert, *Graph Algorithms in the Language of Linear Algebra*. USA: Society for Industrial and Applied Mathematics, 2011.
- [6] O. Selvitopi, S. Ekanayake, G. Guidi, G. Pavlopoulos, A. Azad, and A. Buluc, "Distributed many-to-many protein sequence alignment using sparse matrices," 2020.
- [7] J. Kepner, M. Kumar, J. Moreira, P. Pattnaik, M. Serrano, and H. Tufo, "Enabling massive deep neural networks with the graphblas," *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep 2017. [Online]. Available: <http://dx.doi.org/10.1109/HPEC.2017.8091098>
- [8] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, Dec. 2011. [Online]. Available: <https://doi.org/10.1145/2049662.2049663>
- [9] J. Leskovec and R. Sosis, "Snap: A general purpose network analysis and graph mining library," 2016.
- [10] W. S. Song, V. Gleyzer, A. Lomakin, and J. Kepner, "Novel graph processor architecture, prototype system, and results," *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep 2016. [Online]. Available: <http://dx.doi.org/10.1109/HPEC.2016.7761635>
- [11] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "Sparch: Efficient architecture for sparse matrix multiplication," in *26th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.
- [12] C. Yang, A. Buluc, and J. D. Owens, "Graphblast: A high-performance linear algebra-based graph framework on the gpu," 2020.
- [13] A. Buluc, T. Mattson, S. McMillan, J. Moreira, and C. Yang, "The graphblas c api specification," *GraphBLAS. org. Tech. Rep.*, 2017.
- [14] Y. Zhang, M. Yang, R. Baghdadi, S. Kamil, J. Shun, and S. Amarasinghe, "Graphit: A high-performance graph dsl," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, Oct. 2018. [Online]. Available: <https://doi.org/10.1145/3276491>
- [15] N. D. Jones, C. K. Gomard, and P. Sestoft, *Partial Evaluation and Automatic Program Generation*. USA: Prentice-Hall, Inc., 1993.
- [16] P. Wadler, "Deforestation: transforming programs to eliminate trees," *Theoretical Computer Science*, vol. 73, no. 2, pp. 231–248, 1990. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/030439759090147A>
- [17] M. Sørensen, R. Glück, and N. Jones, "A positive supercompiler," *Journal of Functional Programming*, vol. 6, pp. 811 – 838, 11 1996.
- [18] G. Hamilton, "Extracting the essence of distillation," 06 2009, pp. 151–164.
- [19] O. Kiselyov, A. Biboudis, N. Palladinos, and Y. Smaragdakis, "Stream fusion, to completeness," *SIGPLAN Not.*, vol. 52, no. 1, p. 285–299, Jan. 2017. [Online]. Available: <https://doi.org/10.1145/3093333.3009880>
- [20] D. Coutts, R. Leshchinskiy, and D. Stewart, "Stream fusion. from lists to streams to nothing at all," vol. 42, 09 2007, pp. 315–326.
- [21] T. Henriksen, N. G. W. Serup, M. Elsmann, F. Henglein, and C. E. Oancea, "Futhark: Purely functional gpu-programming with nested parallelism and in-place array updates," *SIGPLAN Not.*, vol. 52, no. 6, p. 556–571, Jun. 2017. [Online]. Available: <https://doi.org/10.1145/3140587.3062354>
- [22] Y. Horawalavithana, "On the design of an efficient hardware accelerator for large scale graph analytics," 2016.
- [23] M. Soltaniyeh, R. P. Martin, and S. Nagarakatte, "Synergistic cpu-fpga acceleration of sparse linear algebra," 2020.
- [24] S. Pal, J. Beaumont, D. Park, A. Amarnath, S. Feng, C. Chakrabarti, H. Kim, D. Blaauw, T. Mudge, and R. Dreslinski, "Outerspace: An outer product based sparse matrix multiplication accelerator," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 724–736.
- [25] K. Kanellopoulos, N. Vijaykumar, C. Giannoula, R. Azizi, S. Koppula, N. M. Ghiasi, T. Shahroodi, J. G. Luna, and O. Mutlu, "Smash: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 600–614. [Online]. Available: <https://doi.org/10.1145/3352460.3358286>
- [26] I. Klyuchnikov, "Supercompiler hosc 1.5: homeomorphic embedding and generalization in a higher-order setting," 01 2010.
- [27] R. Coelho, F. Tanus, A. Moreira, and G. Nazar, "Acqua: A parallel accelerator architecture for pure functional programs," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 346–351.
- [28] M. Naylor and C. Runciman, "The reduceron: Widening the von neumann bottleneck for graph reduction using an fpga," in *Implementation and Application of Functional Languages*, O. Chitil, Z. Horváth, and V. Zsóka, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 129–146.
- [29] A. Boeijink, P. K. F. Hölzenspies, and J. Kuper, "Introducing the pilgrim: A processor for executing lazy functional languages," in *Implementation and Application of Functional Languages*, J. Hage and M. T. Morazán, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 54–71.
- [30] R. Townsend, "Compiling irregular software to specialized hardware," 2019.