Hardware and Software Codesign 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
daniil.berezun@jetbrains.com

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.**

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

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

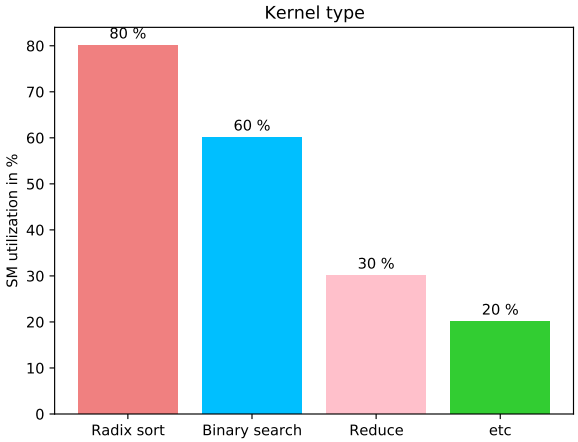

Fig. 1. GPU's SM utilization for spMspM pipeline from cuSPARSE[1]

with spMspM. However, for a sparse framework to be useful, it should incorporate not only spMspM, but also other sparse operations parameterized by a semiring, e.g., masking, which filters the matrix elements or element-wise operations needed for, e.g., PageRank and bread-first search (BFS) algorithms [12]. And when such operations are chained explicitly or implicitly, via a loop body, certain optimizations could be applied. Unfortunately, some of such optimizations are only expressible at software level, i.e., in programming language, hence modern spMspM accelerators could be impractical for accelerating the whole program representing a 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 which is not more effective 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,

[0]https://developer.nvidia.com/cusparse (Accessed 09.02.2021)

and this work in particular gives a brief overview of the filed, 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], [13], 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

```
D<M> = A eWiseAdd B eWiseMult C
```

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 [14], deforestation [15], supercompilation [16], and distillation [17].

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. 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. We propose the co-design of ??? to address the problems 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 in acceleration of sparse linear algebra based programs. Next we discuss some possible hardware architectures and incurred challenges as well as high-level architecture of the proposed solution.

Static vs dynamic data. Specialization requires MIMD.

Representation for data (formats)

AOT vs JIT optimizations. Some data become static in running time.

II. Sparse LA arch

III. Graph processors

Sparse Algrabra processors? Graph algorithms constitute a too large set to be implemented in one processor without a proper unification like sparse linear algebra. Nevertheless, there are a number of algorithm-specific accelerators [18], but the only sparse linear algebra instruction set processor is seems to be still unfinished [10], while other sparse linear algebra accelerators [10], [11], [19], [20] are applicable only to certain operations like spMspM. Notably, in [21] an approach for hardware acceleration for compressed representation indexing is proposed, which is able to employ existing optimizations and could be integrated in existing software frameworks. However, we believe that to fully leverage fusion the indexing should be transparent in the software, which is not the case in [21].

IV. Partial Evaluation and Supercompilation

Specialization on static data. For GPU.

Partial cases for different models (array programming, stream fusion, kernels fusion)

Easy for functional languages, hard for imperative which are widely used for HPC.

Supercompiler provides all optimizations, but too hard to implement in general case. We can use small language to design LA library.

V. Lambda processors

VI. Dataflow processors

TTA and other dataflow + MIMD

VII. Roadmap

Library + language + compiler + hardware co-design.

Integration with application level.

Runtime code generation, staged compilation. Allows one to exploit dynamic data as static. For example, first round of supercompilation in compile time to fuse functions, and the second round in running time to specialize on some data.

References

[1] U. Gupta, C.-J. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, 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, p. 76–80, Jan. 2003.

[3] M. Besta, F. Marending, E. Solomonik, and T. Hoefler, "Slimsell: A vectorizable graph representation for breadth-first search," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 32–41, 2017.

[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.

[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.

[8] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, Dec. 2011.

[9] J. Leskovec and R. Sosic, "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.
[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] 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, Oct. 2018.
[14] N. D. Jones, C. K. Gomard, and P. Sestoft, *Partial Evaluation and Automatic Program Generation*. USA: Prentice-Hall, Inc., 1993.
[15] P. Wadler, "Deforestation: transforming programs to eliminate trees," *Theoretical Computer Science*, vol. 73, no. 2, pp. 231–248, 1990.
[16] M. Sørensen, R. Glück, and N. Jones, "A positive supercompiler," *Journal of Functional Programming*, vol. 6, pp. 811 – 838, 11 1996.
[17] G. Hamilton, "Extracting the essence of distillation," pp. 151–164, 06 2009.
[18] M. Besta, D. Stanojevic, J. D. F. Licht, T. Ben-Nun, and T. Hoefler, "Graph processing on fpgas: Taxonomy, survey, challenges," 2019.
[19] M. Soltaniyeh, R. P. Martin, and S. Nagarakatte, "Synergistic cpu-fpga acceleration of sparse linear algebra," 2020.
[20] 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)*, pp. 724–736, 2018.
[21] 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*, MICRO '52, (New York, NY, USA), p. 600–614, Association for Computing Machinery, 2019.