Challenges with Hardware-Software Co-design for Sparse Machine Learning on Streaming Dataflow

Rubens Lacouture
Stanford University, USA

Olivia Hsu
Stanford University, USA

Kunle Olukotun
Stanford University, USA

Fredrik Kjolstad
Stanford University, USA

Abstract

This paper details the problem landscape that arises from using a general tensor algebra accelerator framework to compute real-world end-to-end machine learning applications. We identify three key challenges for correctness and performance, which include support for tensor reshaping and nonlinear operations, dataflow optimization (kernel fusion, optimal dataflow order), and leveraging sparsity structure. This paper motivates the need to address these problems in the domain-specific language, compiler framework, and architectural design for sparse machine learning. We extended a general tensor algebra compiler and architectural model, the Sparse Abstract Machine, to real-world sparse machine learning models in order to identify the key challenges above.

Introduction

Prior work on machine learning (ML) accelerator design has mainly revolved around designing highly optimized handwritten kernels that are abundant in ML models. Oftentimes, architects build fixed-function ASICs for accelerating bottleneck ML operations [18, 19]. This state-of-the-art execution approach has been most successful for dense ML applications, namely, to reduce models to a sequence of data transformations that feed to a dense matrix multiplication accelerator [15, 23]. However, this approach will work poorly for sparse models. Factorizing the computation to fixed kernels is computationally expensive because it reorganizes sparse data structures and prevents kernel fusion, which can lead to worse asymptotic complexity (arbitrary slowdowns) [10].

More general systems, on the other hand, have the ability to generate fused kernels but are often limited to a smaller domain of computation. Specifically, sparse tensor algebra is the backbone of sparse ML but cannot fully express most ML models. The more general problem to solve is how to leverage sparse tensor algebra systems to compute end-to-end real-world applications. There have been many general software and hardware systems—like compilers and reconfigurable dataflow accelerators (like CGRAs)—for tensor algebra [3, 6, 16, 17, 21, 24], and we would like to investigate how to apply them to modern ML applications.

Three challenges arise when we attempt to integrate sparse tensor algebra accelerators into sparse ML: designing efficient hardware features for non-tensor algebra operations, dataflow optimization (automatic fusion and dataflow order selection), and structured sparsity. An entire machine learning model cannot be developed without these additional features, which lie beyond linear and multi-linear operations.

Even though it is vital to perform kernel fusion across several subgraphs of the application, it comes at the cost of programmability and debuggability. A host of other unsolved problems also arise from fusion. For example, choosing the best schedule (tiling and iteration order) for the fully fused application is difficult since one iteration order may be better for some sub-computations but worse for others, especially since sparse applications are data-dependent. Additionally, determining when fusion must be broken in order to maintain correctness across such a large computation is tedious.

We identified the above missing feature and performance challenges when adding sparse machine learning applications—especially transformers, and graph neural networks (GNN)—to a general sparse tensor algebra hardware framework from prior work, the Sparse Abstract Machine (SAM) [17]. The Sparse Abstract Machine is a spatial streaming dataflow machine with various types of primitives that can compose to express any tensor algebra expression. Moreover, it can express many algorithms (schedules) for each expression and has a compiler from a high-level API. Our goal is to co-design the SAM hardware and software such that machine learning experts and hardware architects alike can quickly iterate over novel ideas for new state-of-the-art sparse ML accelerators, giving them the flexibility to explore hardware accelerator design for new sparse models with a robust framework.

Challenge 1: New Hardware Features

Although SAM can compute any tensor algebra expression, modern ML models also need operations that transform and filter tensors and perform non-linear computations. For example, in the masked multi-head attention module in the decoder blocks of a transformer, a triangular lower mask is used to prevent the decoder from looking ahead at ground truth during training. Table 1 shows a list of operations used by various modern ML algorithms. Therefore, hardware designs also need to handle these operations efficiently, which mainly consist of fine-grained memory management such as data movement, data generation, and tiling operations. The data movement, in particular, includes tensor reshaping, concatenation, splitting, and transpositions, is crucial

Name	Nonlinear	Mask Generation	Reshaping
Transformer [26]	relu, softmax, sin, cos	tri-lower, dropout	split, concat
BEIT [2]	relu, softmax	block random	split, concat
BERT [11]	gelu, relu, softmax	random	split, concat
FlashAttention [9]	max, exp	diag, block random	
Sparse Transformer [5]	relu, softmax	random	split, concat
GCN [20]	relu, max, softmax	diag	
GraphSage [14]	relu, max, mean	diag	concat
GAT [27]	leaky relu, softmax	diag	

Table 1. Sparse machine learning algorithms along with operations that are beyond tensor algebra and SAM.

to representing any sparse ML models. Another challenge is making general enough blocks so that they can be reused for more than one specific operation. We have identified three new SAM primitives for sparse ML, a filter block, a tiling block and a unary ALU block. The filter block is primarily used to mask tensors, this can include structured masking such as lower triangular and diagonal, or random masking such as dropout (see Figure 1). The tiling block can be used to perform tensor reshaping operations such as split and concat. For non-linear operations, the unary ALU introduces element-wise sparse array algebra operations such as maximum which can be used in computing the ReLU.

Challenge 2: Dataflow Optimization

Kernel fusion is necessary to maximize performance [1, 17, 28]. For end-to-end ML applications, the number of kernels can grow quite large, making it infeasible to fuse operations by hand. For example, the original transformer model [26] calls several hundred kernels (15 unique kernels) that operate on one or two input operands. Fully fusing across such a large number of kernels is practically infeasible and inefficient, making it necessary to have a hardware language that can express an automatic system that can optimize fusion. This framework would also need to determine cases where fusion is not feasible such as cases where two kernels are not in concordant traversal or cases where an addition is broadcast into a product in the wrong iteration order. Current deep learning compilers such as TVM [4] and XLA [25] only target dense fusion and mostly target operator fusion. Sparse kernel fusion is necessary to represent cross-layer fusion.

Dataflow order selection is another key part of the optimization space when mapping ML applications to dataflow. First, the same dataflow order has to be used across each fully fused expression, which increases the search space for larger expressions. This constraint may also significantly worsen performance since a sub-optimal ordering may compose slow sub-computations to further bottleneck performance. Second, current compilers require concordant traversal of all input tensors for a given iteration ordering; otherwise, fusion is not possible. This is one of the main challenges that makes it non-trivial to arbitrarily fuse multiple kernels together. In the case where two subsequent kernels do not follow the same iteration order, a transpose or reordering of

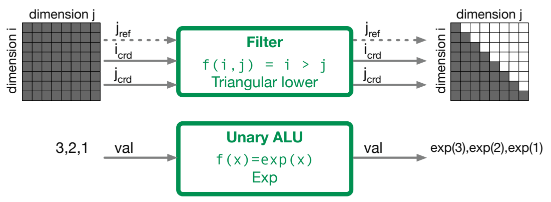

Figure 1. Top: The filter primitive, where the boolean function f defines whether to remove the inner reference token j_{ref}, bottom: the unary ALU primitive, where the function f defines the elementwise operation.

modes (tensor reshape) has to be imposed. Ideally, we would need a way to automatically search for and choose the optimal dataflow ordering for all intermediate tensors over an entire computation pipeline to maximize fusion benefits. To do so, a large search space of possible combinations of orders needs to be explored and the hardware language has to be able to express this optimization space.

Challenge 3: Structured Sparsity

Prior work on sparse machine learning hardware accelerators has primarily focused on accelerating unstructured sparse computation. Due to the need for efficient computation and reuse, modern ML models mix irregular sparsity with structured sparsity where some parts of the model have a predefined sparsity pattern, such as convolutional filters or attention masks. Recent work has shown that structured sparsity is a promising approach to accelerating machine learning models [7, 8]. For example, Megablocks [13] reduces the computations of Mixture-of-Experts (MOE) models into block sparse computation. Both models are able to achieve substantial speedups. Although there are a few hardware accelerators that support some structured sparsity [12, 22, 24], they are constrained to sparsity patterns that users have to adhere to and cannot be generalized for the larger landscape of sparsity, whether that is unstructured or structured. There is a need for a hardware accelerator coupled with a language that can handle the full range of sparsity patterns encountered in modern machine-learning models.

Conclusion

We identified three key challenges and highlight the need for addressing these challenges through domain-specific languages, compiler frameworks, and architectural designs for sparse machine learning hardware accelerators. We believe that hardware-software co-design is about designing the programming languages that the hardware executes. With appropriate and composable operations that are rich enough to represent large applications, we believe one can build efficient domain-specific hardware that can compute whole applications. We hope this paper will spark discussions around coming up with solutions to tackle these key challenges.

References

[1] Willow Ahrens, Daniel Donenfeld, Fredrik Kjolstad, and Saman Amarasinghe. 2023. Looplets: A Language for Structured Coiteration. In *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*. 41–54.

[2] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. 2021. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254* (2021).

[3] Aart Bik, Penporn Koanantakool, Tatiana Shpeisman, Nicolas Vasilache, Bixia Zheng, and Fredrik Kjolstad. 2022. Compiler support for sparse tensor computations in MLIR. *ACM Transactions on Architecture and Code Optimization (TACO)* 19, 4 (2022), 1–25.

[4] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. *arXiv preprint arXiv:1802.04799* (2018).

[5] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating Long Sequences with Sparse Transformers. arXiv:1904.10509 [cs.LG]

[6] Vidushi Dadu, Jian Weng, Sihao Liu, and Tony Nowatzki. 2019. Towards General Purpose Acceleration by Exploiting Common Data-Dependence Forms. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 924–939. https://doi.org/10.1145/3352460.3358276

[7] Tri Dao, Beidi Chen, Kaizhao Liang, Jiaming Yang, Zhao Song, Atri Rudra, and Christopher Re. 2021. Pixelated butterfly: Simple and efficient sparse training for neural network models. *arXiv preprint arXiv:2112.00029* (2021).

[8] Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. 2022. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*. PMLR, 4690–4721.

[9] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.

[10] Shail Dave, Riyadh Baghdadi, Tony Nowatzki, Sasikanth Avancha, Aviral Shrivastava, and Baoxin Li. 2021. Hardware Acceleration of Sparse and Irregular Tensor Computations of ML Models: A Survey and Insights. *Proc. IEEE* 109, 10 (oct 2021), 1706–1752. https://doi.org/10.1109/jproc.2021.3098483

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[12] Chao Fang, Shouliang Guo, Wei Wu, Jun Lin, Zhongfeng Wang, Ming Kai Hsu, and Lingzhi Liu. 2022. An Efficient Hardware Accelerator for Sparse Transformer Neural Networks. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2670–2674. https://doi.org/10.1109/ISCAS48785.2022.9937659

[13] Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. 2022. MegaBlocks: Efficient Sparse Training with Mixture-of-Experts. *arXiv preprint arXiv:2211.15841* (2022).

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 1025–1035.

[15] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 620–629. https://doi.org/10.1109/HPCA.2018.00059

[16] Kartik Hegde, Hadi Asghari-Moghaddam, Michael Pellauer, Neal Crago, Aamer Jaleel, Edgar Solomonik, Joel Emer, and Christopher W Fletcher. 2019. Extensor: An accelerator for sparse tensor algebra. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 319–333.

[17] Olivia Hsu, Maxwell Strange, Jaeyeon Won, Ritvik Sharma, Kunle Olukotun, Joel Emer, Mark Horowitz, and Fredrik Kjolstad. 2022. The Sparse Abstract Machine. https://doi.org/10.48550/ARXIV.2208.14610

[18] Norman P. Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. arXiv:2304.01433 [cs.AR]

[19] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[20] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJU4ayYgl

[21] Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato, and Saman Amarasinghe. 2017. The tensor algebra compiler. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–29.

[22] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. 2021. Accelerating Sparse Deep Neural Networks. arXiv:2104.08378 [cs.LG]

[23] Eriko Nurvitadhi, Jaewoong Sim, David Sheffield, Asit Mishra, Srivatsan Krishnan, and Debbie Marr. 2016. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. 1–4. https://doi.org/10.1109/FPL.2016.7577314

[24] Alexander Rucker, Matthew Vilim, Tian Zhao, Yaqi Zhang, Raghu Prabhakar, and Kunle Olukotun. 2021. Capstan: A Vector RDA for Sparsity. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 1022–1035. https://doi.org/10.1145/3466752.3480047

[25] Amit Sabne. 2020. XLA : Compiling Machine Learning for Peak Performance.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. arXiv:1710.10903 [stat.ML]

[28] Rohan Yadav, Alex Aiken, and Fredrik Kjolstad. 2022. DISTAL: the distributed tensor algebra compiler. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 286–300.