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 hand-written 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
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 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Nonlinear</th>
<th>Mask Generation</th>
<th>Reshaping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer [26]</td>
<td>relu, softmax, sin, cos</td>
<td>tri-youer, dropout</td>
<td>split, concat</td>
</tr>
<tr>
<td>BEIT [2]</td>
<td>relu, softmax</td>
<td>block random</td>
<td>split, concat</td>
</tr>
<tr>
<td>BERT [1]</td>
<td>gelu, relu, softmax</td>
<td>random</td>
<td>split, concat</td>
</tr>
<tr>
<td>FlashAttention [9]</td>
<td>max, exp</td>
<td>diag, block random</td>
<td>split, concat</td>
</tr>
<tr>
<td>Sparse Transformer [5]</td>
<td>relu, softmax</td>
<td>block random</td>
<td>split, concat</td>
</tr>
<tr>
<td>GCN [20]</td>
<td>relu, max, softmax</td>
<td>diag</td>
<td></td>
</tr>
<tr>
<td>GraphSage [14]</td>
<td>relu, max, mean</td>
<td>diag</td>
<td>concat</td>
</tr>
<tr>
<td>GAT [27]</td>
<td>leaky relu, softmax</td>
<td>diag</td>
<td></td>
</tr>
</tbody>
</table>

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

**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.
References


