HighLight: Efficient and Flexible DNN Acceleration with Hierarchical Structured Sparsity

Yannan Nellie Wu
MIT
Cambridge, MA, USA
nelliewu@mit.edu

Po-An Tsai
NVIDIA
Westford, MA, USA
poant@nvidia.com

Saurav Muralidharan
NVIDIA
Santa Clara, CA, USA
sauravm@nvidia.com

Angshuman Parashar
NVIDIA
Westford, MA, USA
aparashar@nvidia.com

Vivienne Sze
MIT
Cambridge, MA, USA
sze@mit.edu

Joel S. Emer
MIT/NVIDIA
Cambridge/Westford, MA, USA
jsemer@mit.edu

ABSTRACT

Due to complex interactions among various deep neural network (DNN) optimization techniques, modern DNNs can have weights and activations that are dense or sparse with diverse sparsity degrees. To offer a good trade-off between accuracy and hardware performance, an ideal DNN accelerator should have high flexibility to efficiently translate DNN sparsity into reductions in energy and/or latency without incurring significant complexity overhead.

This paper introduces hierarchical structured sparsity (HSS), with the key insight that we can systematically represent diverse sparsity degrees by having them hierarchically composed from multiple simple sparsity patterns. As a result, HSS simplifies the underlying hardware since it only needs to support simple sparsity patterns; this significantly reduces the sparsity acceleration overhead, which improves efficiency. Motivated by such opportunities, we propose a simultaneously efficient and flexible accelerator, named HighLight, to accelerate DNNs that have diverse sparsity degrees (including dense). Due to the flexibility of HSS, different HSS patterns can be introduced to DNNs to meet different applications' accuracy requirements. Compared to existing works, HighLight achieves a geomean of up to 6.4× better energy-delay product (EDP) across workloads with diverse sparsity degrees, and always sits on the EDP-accuracy Pareto frontier for representative DNNs.

KEYWORDS

Deep learning accelerator, structured sparsity, hardware-software co-design, computer architecture

ACM Reference Format:
Yannan Nellie Wu, Po-An Tsai, Saurav Muralidharan, Angshuman Parashar, Vivienne Sze, and Joel S. Emer. 2023. HighLight: Efficient and Flexible DNN Acceleration with Hierarchical Structured Sparsity . In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23), October 28-November 1, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3613424.3623786

MICRO '23, October 28-November 1, 2023, Toronto, ON, Canada

ACM ISBN 979-8-4007-0329-4/23/10.
https://doi.org/10.1145/3613424.3623786

1 INTRODUCTION

Modern deep neural networks (DNNs) can have weight and activation tensors with diverse discrete amounts of sparsity, *i.e.*, *sparsity degrees*, where sparsity is the percentage of zeros out of the total number of values in the tensor. This phenomenon is a result of complex interactions among various DNN optimization techniques in a large DNN model design space. For example, activations can be dense or sparse based on the choice of activation functions (*e.g.*, ReLU [1] introduces sparse activations, whereas Mish [33] can result in much denser activations). Similarly, *weight pruning* is often applied to over-parameterized DNN models, leading to zero-valued weights within the network. Sparsity degrees for pruned DNNs vary depending on how amenable the given network is to sparsification (*e.g.*, large models such as ResNet50 [16] can sometimes be pruned to 80% sparsity while still maintaining accuracy, while compact models such as EfficientNet [45] cannot be pruned as aggressively).

As a result, it is desirable to have a DNN accelerator that can translate any sparsity into efficiency, resulting in a good accuracy-efficiency trade-off. Specifically, the accelerator should be:

- ***Efficient***: incurs low latency, energy, and area overhead cost, referred to as having *low sparsity tax*, to implement the sparsity-related acceleration features. The sparsity tax can come from extra control logic, lack of data reuse, etc.
- ***Flexible***: supports *diverse* sparsity degrees (including dense). "Support" refers to two capabilities: **(i)** process the DNN to produce functionally correct results; **(ii)** translate weight and activation sparsity into reductions in energy and/or latency.

Specifically, the accelerator has two goals:

- for medium/high-sparsity DNNs, eliminate *ineffectual operations* (*i.e.*, compute and data movement involving zeros) [19] to introduce energy and/or latency savings;
- for low-sparsity DNNs, have similar energy efficiency and latency as a dense accelerator (*i.e.*, have a low sparsity tax).

However, to the best of our knowledge, none of the existing DNN accelerators achieve both goals [8, 14, 23, 25, 30, 36, 37, 41, 42, 52, 58, 60]. Table 1 describes the incurred sparsity tax and sparsity degree diversity for representative accelerators across different tensor accelerator categories. *Dense* accelerators [4, 25, 36] have no sparsity tax, but never exploit sparsity. *Structured sparse* accelerators [30, 31, 37, 60] target DNNs whose sparsity is spatially constrained and introduce low-to-medium sparsity tax. However,

Categories	Representative Designs	Sparsity Tax	Sparsity Degree Diversity
Dense	TC [36]	N/A	N/A
Structured Sparse	STC [37]	Very Low	Low
	S2TA [30]	Medium	Medium
Unstructured Sparse	DSTC [52]	High	Very High
HSS	**Our Work**	Low	High

Table 1: Comparison of designs from different DNN accelerator design categories. *HSS* stands for hierarchical structured sparsity. An ideal design should have a low sparsity tax to achieve high efficiency and a very high number of supported sparsity degrees to achieve high flexibility.

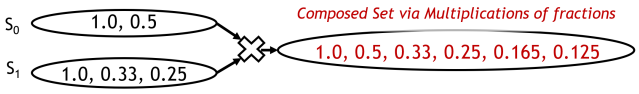

Figure 1: Composing two sets of density degrees, S_0 and S_1, by multiplying the fractions in each set.

they often only recognize a limited set of sparsity degrees. *Unstructured sparse* DNN accelerators [39, 42, 52, 57] provide support for arbitrarily distributed zeros with diverse sparsity degrees, but pay a considerable sparsity tax (*e.g.*, employ costly intersection units to locate nonzeros) for that flexibility. Thus, they are often inefficient for low-sparsity DNNs. *In short, the trend of DNNs containing tensors with diverse sparsity degrees challenges the fundamental design premise of many DNN accelerators.*

To address the limitations of existing work, we present a novel class of sparsity patterns named *hierarchical structured sparsity* (HSS), with the insight that we can systematically represent diverse sparsity degrees by hierarchically composing them from simple sparsity patterns. Such simple sparsity patterns help us maintain a low sparsity tax by correspondingly simplifying the hardware that implements acceleration features for translating sparsity into reductions in energy/latency. Since there are many different ways of composing various simple sparsity patterns and designing their associated acceleration hardware, HSS opens up a promising design space. We evaluate the impact of various design decisions in such an HSS-based design space and propose an efficient and flexible accelerator, HighLight[1].

The key insight of our design is that we leverage *the properties of the multiplication of fractions* to: i) represent diverse structured sparsity degrees and ii) enable modularized low-sparsity-tax hardware support for each set of fractions to exploit the structured sparsity degrees. Fig. 1 illustrates the idea with two *composable* sets of density degrees (where *density = 1−sparsity*) represented as fractions. Composing the densities from S_0 and S_1 results in six density degrees. Thus, hardware with modularized support for each set naturally supports all six derived degrees.

[1]More information on HighLight can be found at http://emze.csail.mit.edu/highlight

This work makes the following key contributions:

(1) Proposes *hierarchical structured sparsity (HSS)*, which allows a flexible representation of diverse sparsity degrees by hierarchically composing different sets of simple sparsity patterns. In addition, we propose a precise fibertree-based [44] sparsity specification to distinguish HSS from existing sparsity patterns.
(2) Proposes a simultaneously efficient and flexible hardware accelerator design, named HighLight, that:
   - leverages modularity in HSS to enable modularized sparsity acceleration to exploit different sets of simple sparsity patterns in HSS at different architecture levels.
   - introduces low-overhead sparsity acceleration hardware at each architecture level to provide efficient processing with a low sparsity tax.
(3) Demonstrates that DNNs can use HSS to meet various accuracy/efficiency requirements at various sparsity degrees.
(4) To demonstrate efficiency and flexibility, HighLight outperforms existing works with better overall hardware efficiency across workloads with diverse degrees in terms of both energy-delay-product (EDP) and energy-delay-squared (ED^2).
   - Compared to dense accelerators, HighLight achieves a geomean of 6.4× (and up to 20.4×) lower EDP across DNN layers with diverse sparsity degrees (including dense) and is at EDP parity for dense DNN layers.
   - Compared to sparse accelerators, HighLight achieves a geomean of 2.7× (and up to 5.9×) lower EDP and is at EDP parity for sparse DNN layers.

2 BACKGROUND & MOTIVATION

This section introduces the basics of sparse DNN acceleration, discusses the limitations of accelerators designed for different sparsity patterns (*i.e.*, the distribution of zero and nonzero value locations), and motivates the need for a simultaneously efficient and flexible sparse DNN accelerator.

2.1 Opportunities and Challenges

The zero values in sparse DNNs can introduce a significant number of *ineffectual computations*, whose results can be easily derived by applying the simple algebraic equalities of $X \times 0 = 0$ and $X + 0 = X$, without reading all the operands or doing the computations [19, 54]. Thus, ineffectual computations introduce promising opportunities for accelerators to eliminate unnecessary hardware operations (*i.e.*, buffer accesses and arithmetic calculations) and improve efficiency.

However, to translate such opportunities into hardware savings, the accelerator faces the challenge of providing hardware support to identify nonzero values and evenly distribute them to parallel hardware components, referred to as *workload balancing*. Workload balancing is important to ensure high utilization of the available resources, thus achieving maximum speedup. Often, the sparsity tax associated with such hardware support is highly related to the sparsity patterns that the accelerator aims to exploit.

2.2 Limitations of Existing Accelerators

In recent years, many sparse DNN accelerators have been proposed to exploit ineffectual computations to reduce data movement and compute for different sparsity patterns [6, 8, 11, 14, 19, 30, 37, 39,

41, 42, 52, 57, 58, 60]. At a high level, we can classify them into *unstructured sparse accelerators* and *structured sparse accelerators*. In the following sections, we will discuss their limitations both qualitatively and quantitatively.

2.2.1 Unstructured Sparse Accelerators. Unstructured sparse accelerators target DNNs with unstructured sparsity, which refers to an unconstrained distribution of zeros. Such patterns can be introduced to activations by activation functions or to weights by unstructured pruning that removes weights regardless of their locations in the tensor.

Unstructured sparse accelerators have high flexibility to exploit arbitrarily distributed zeros with any sparsity degree. However, the hardware support for unstructured sparsity introduces a high sparsity tax since it cannot make any assumptions about the locations of nonzero values when trying to identify and distribute the effectual computations. Existing unstructured sparse accelerators either pay for expensive intersections to identify the effectual computations (*e.g.*, SparTen [14] employs a prefix sum logic that occupies 55% of its processing element area), or employ dataflows that identify effectual computations without intersections but require large, and thus expensive, accumulation buffers to hold the now randomly distributed output (*e.g.*, the costly dataflow employed by DSTC [52]). Furthermore, since the number of effectual computations varies across sub-tensors within and across workloads, these accelerators can often only ensure perfect workload balance for a limited set of sparsity (*e.g.*, DSTC [52] only ensures perfect workload balancing among columns of compute units when a sub-tensor's occupancy is a multiple of 32).

Takeaway: unstructured sparse accelerators often support diverse sparsity degrees with a high sparsity tax.

2.2.2 Structured Sparse Accelerators. Structured sparse accelerators target DNNs with structured sparsity, which refers to distributions of zeros with spatial constraints and is often introduced via *structured pruning* [17, 30, 32, 35]. Structured sparsity can have different spatial constraints for nonzero value locations. For example, one of the most popular structured sparsity patterns is the *G:H sparsity pattern*, which mandates (at most) G elements to be nonzero within a block of H elements, and thus results in a density of G/H. For example, NVIDIA's Sparse Tensor Core (STC) [37] employs a 2:4 pattern, which sparsifies two elements in every block of four elements [32], resulting in 50% sparsity.

The predetermined constraints for nonzero value locations in structured sparsity make it much easier for hardware to identify the locations of the nonzeros and evenly distribute them to parallel hardware components (*e.g.*, for G:H sparsity, the hardware can evenly assign G nonzeros to G compute units to balance the workload). As a result, accelerating a specific pattern is often efficient with a very low sparsity tax. However, existing structured sparse tensor accelerators often only accelerate a very limited set of sparsity patterns (*i.e.*, a few sparsity degrees). For example, the STC [37] is only able to exploit the 2:4 pattern, whereas S2TA [30] exploits a few $G : 8$ patterns with design-specific constraints.

Takeaway: structured sparse accelerators often incur a low sparsity tax but only support a few sparsity degrees.

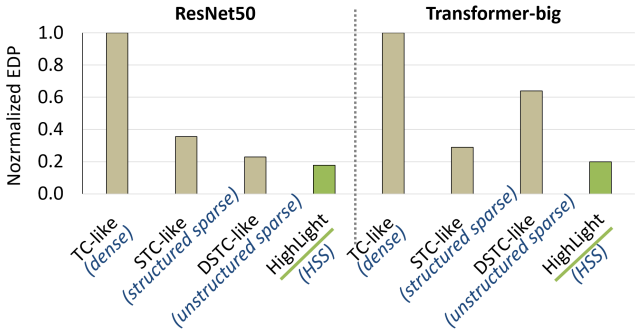

Figure 2: Normalized energy-delay product (EDP) of accelerators running two types of DNNs, pruned Transformer-Big [50] and pruned ResNet50 [16]. While ensuring similar accuracy (within 0.5% difference), the DNNs were structured pruned for STC [37] and HighLight (our work) and unstructured pruned for DSTC. For both models, HighLight achieves the lowest EDP while ensuring similar accuracy.

2.2.3 Quantitative Comparison. To concretely demonstrate the limitations of each class of accelerators, without loss of generalizability, we quantitatively compare representative designs allocated with similar hardware resources: **(i) *DSTC-like* [52]**: targets unstructured sparse DNNs with a high sparsity tax introduced by its costly dataflow; **(ii) *STC-like* [37]**: targets DNNs with weights that are dense or 2:4 sparse, introducing a low sparsity tax.

To compare the designs, we normalize their energy-delay-product (EDP) to a dense accelerator, *TC-like* [36]. In particular, we evaluate their EDP running two different DNN architectures, *Transformer-Big* [50] and *ResNet50* [16]. For each DNN architecture, while ensuring similar accuracy (which we define as <0.5% difference), *TC-like* runs the dense version of the model, *STC-like* runs a structured pruned version, and *DSTC-like* runs an unstructured pruned version. Fig. 2 shows the EDP of the three designs running all the GEMM layers in the DNNs.

Inflexibility of Structured Sparse Designs: As shown in Fig. 2, *STC-like* is outperformed by *DSTC-like* when running *ResNet50*. This is because *STC-like* is designed to only allow a maximum of 2× speedup with the 2:4 sparsity pattern. Furthermore, even if *ResNet50* has ~60% sparse activations, *STC-like* cannot exploit activation sparsity for speedup. On the other hand, *DSTC-like* is able to translate the sparsity in both weights and activations into reductions in both processing speed and energy consumption. Thus, even if *STC-like* has a low sparsity tax, when running *ResNet50*, its inability to translate various sparsity degrees into hardware savings results in higher EDP than *DSTC-like*.

Inefficiency of Unstructured Sparse Designs: *DSTC-like* is outperformed by *STC-like* on *Transformer-Big*, as shown in Fig. 2. This is because *DSTC-like*'s outer-product-based dataflow with expensive accumulation buffer has a high sparsity tax. Since *Transformer-Big* has less than 10% average sparsity in activations, *DSTC-like*'s savings are overshadowed by its hardware support with high sparsity tax. Thus, even though *DSTC-like* has high flexibility, when

running *Transformer-Big*, its inefficient sparsity support with high overhead results in higher EDP than *STC-like* does.
Takeaway: there is no existing sparse accelerator that always has lower EDP for both evaluated DNNs because of their respective limitations.

2.3 Need for a Flexible and Efficient Design

To develop a flexible accelerator that is efficient for various DNNs with diverse sparsity degrees, *the community can benefit from a general design that is simultaneously flexible and efficient.* However, as already demonstrated by the *DSTC-like* and *STC-like* comparisons above, many existing sparse DNN designs tend to trade flexibility for efficiency or vice versa, thus facing the challenge of not being able to meet both requirements.

To address this problem, we introduce a hardware-software co-design approach motivated by a novel class of sparsity patterns: **hierarchical structured sparsity (HSS)**, which leverages multiple levels of G:H structured sparsity to express diverse sparsity degrees in a multiplicative fashion. As shown in Fig. 2, while maintaining accuracy with our HSS-based sparsification, our low-sparsity-tax HSS-based hardware accelerator, HighLight, provides lower EDP in both scenarios.

3 PRECISE SPARSITY SPECIFICATION

In order to clearly compare existing sparsity patterns and distinguish our proposed HSS from existing works, it is necessary to precisely describe various sparsity patterns. However, conventional sparsity pattern classification approaches are often based on names that provide an informal characterization of just the dimensions on which the pattern is imposed, and thus fail to distinguish between different sparsity pattern proposals [2, 21] (*e.g.*, the term *sub-channel* is repeatedly used to describe many different patterns presented in Table 2).

To solve the problem, we propose a precise way of specifying various sparsity patterns based on the fibertree abstraction [44]. As shown in Table 2, our specification can easily distinguish between existing sparsity patterns and cleanly reflects the properties of an example sparsity pattern that belongs to our proposed hierarchical structured sparsity (HSS).

3.1 Fibertree Abstraction

The fibertree abstraction [34, 38, 44] provides a systematic and precise way to express the content of tensors, *without getting into the complexities related to its layout when the tensor is actually stored in buffers* (e.g., *compressed or uncompressed*). Since the sparsity specification focuses on understanding the nature of the sparsity patterns rather than how they can be compressed, we use fibertrees as a basis for our proposed methodology.

For ease of presentation, we use the three-dimensional weight tensor in Fig. 3(a) as an example, which has C channels, R rows, and S columns. Fig. 3(b) shows the fibertree representation of the tensor. The fibertree has three levels, each of which is referred to as a *rank* and corresponds to a dimension of the tensor (*e.g.*, the lowest rank, *Rank0*, corresponds to dimension S). Each rank contains multiple *fibers*, each of which contains a set of *coordinates* and their associated *payloads*. For intermediate ranks, the payload is a fiber from a lower rank (*e.g.*, the first coordinate in *Rank1* has the first fiber in *Rank0* as its payload); for a coordinate in *Rank0*, the payload is a simple value (*e.g.*, the first coordinate in *Rank0* has the value a_0 as its payload).

Example Pattern	Conventional Classification	Fibertree-based Specification *Rank* (<rule>)...
[15]	Unstructured	CRS(Unconstrained)
[17] (Fig. 4(a))	Channel	C(Unconstrained)→R→S
[35]	Sub-kernel	C→RS(G:H), *with any G, H*
[32] (Fig. 4(b))	Sub-channel	RS→C_1→C_0(2:4)
[60]	Sub-channel	RS→C_1→C_0(4:16)
[30]	Sub-channel	RS→C_1→C_0(G≤8:8)
Example Two-rank HSS (Fig. 5)	Sub-channel	RS→C_{N-1}→ C_{N-2}(3:4)→...→C_0(2:4)

Table 2: Informal conventional classification and the precise fibertree-based specifications for example sparsity patterns. For fibertree-based specificatons, ranks without pruning rules, *i.e.*, N/A rules in the figures, do not have (). Partitioned ranks are indicated by appending a number to the rank name, e.g., C is split into C_1 and C_0. Note that there could be multiple G and H values allowed for each rank.

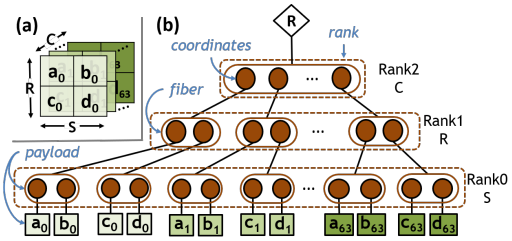

Figure 3: (a) Example dense weight tensor. *C*: channels, *R*: height, *S*: width. (b) Corresponding fibertree-based abstraction of the tensor. Each tensor dimension corresponds to a level of the tree, referred to as a *Rank*.

3.2 Fibertree-based Sparsity Specification

With the fibertree fundamentals presented above, we now discuss how to use such an abstraction to describe sparsity in a tensor. Sparsity is introduced via pruning away the *coordinates* in the dense fibertree. At a high level, to define a specific sparsity pattern, a rank order needs to be specified and each *rank* is assigned a *pruning rule*. The rule specifies if the coordinates in each of its *fibers* can be pruned away; and if so, whether there is a pattern that the per-fiber pruning should follow.

Coordinates in arbitrary ranks can be pruned away. Pruning a coordinate at the lowest rank simply removes values, whereas pruning a coordinate at intermediate ranks removes its fiber payload (*i.e.*, the entire subtree associated with the coordinate), implicitly pruning away all the associated lower-level coordinates. Because of this chained effect, the introduced sparsity is conventionally known as *structured sparsity*. Structured sparsity can have structures at different granularities, which are impacted by the rank at which the pruning rules are defined. For example, Fig. 4(a) shows the

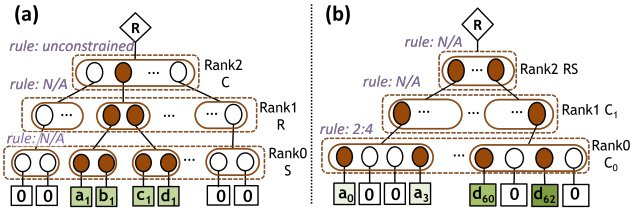

Figure 4: Fibertree-based specification for popular sparsity patterns applied to the tensor in Fig. 3(a). (a) channel-based structured [18]: C(unconstrained)→R→S. (b) 2:4 structured [32]: RS→C_1→C_0(2:4). Please note that the abstraction describes the exact sparsity, and is orthogonal to how the tensor is stored in the buffer.

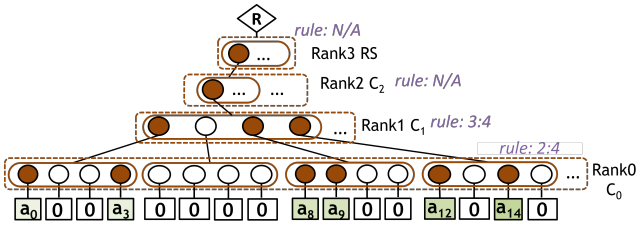

Figure 5: Fibertree-based specification for an example two-rank HSS with RS→C_2→C_1(3:4)→C_0(2:4). Please note that, for general HSS patterns, the choices of number of ranks, rank ordering, flattening, and splitting are not limited to the ones presented in this example.

fibertree-based specification of the conventionally known *channel-based structured sparsity*, which demands arbitrary channels to be completely removed. The fibertree-based representation specifies the *unconstrained* pruning rule at the top rank *C*, as each removed coordinate corresponds to a removed channel, which is indicated by the empty circles. We specify such a structure as C(unconstrained)→R→S as shown in Table 2 and Fig. 4(a). The → defines the higher to lower rank order, and ranks with pruning rules carry (<rule>). The removal of the highest-rank coordinates results in all zeros in the corresponding channels. Since the removal of lower ranks is always implicit, the R and S lower ranks are not associated with any explicit pruning rules and are thus not followed by (<rule>) in the specification.

Furthermore, a sparsity pattern specification may involve first applying *content-preserving transformations* to the tensor, such as *reordering*, *flattening*, or *partitioning* the ranks [34]. For example, this is done for creating the conventionally known sub-channel-based 2:4 structured sparsity [37]. Fig. 4(b) shows the 2:4 structured sparsity that's supported by NVIDIA's sparse tensor core. The original ranks are first reordered to have C as the lowest rank, and the C rank is then partitioned into two ranks, C_1 and C_0. The G:H-style sparsity structure (as described in Section 2.2) is manifest by allowing at most 2 non-zero values in each fiber of the C_0 rank, which due to the partitioning have exactly four coordinates. For a specific fiber, we refer to the total number of coordinates as its *shape* and the number of coordinates associated with nonzeros as its *occupancy*. Thus, the fiber shape in C_0 rank is defined by the denominator of the fraction, *i.e.*, H in C_0(G:H) structured sparsity, and the max fiber occupancy is defined by the numerator in the fraction. This sparsity pattern is thus specified as RS→C_1→C_0(2:4).

As illustrated in Table 2, our proposed fibertree-based specification allows precise descriptions of many existing works that were not easily distinguishable using conventional sparsity pattern classification techniques.

4 HSS

With the fibertree-based sparsity specifications, it is clear that existing works in Table 2 all propose to apply different sparsity pattern(s) to *one* rank. A natural approach to enhance such works to represent more sparsity degrees would be to introduce sparsity patterns to *multiple* ranks, motivating our proposed concept of *hierarchical structured sparsity* (HSS).

4.1 General Concept

HSS allows *multiple ranks* to be sparse, where each rank has its own sparsity pattern(s). Such a hierarchy of sparsity patterns could lead to structured sparsity with more sparsity degrees than only one rank with a very flexible sparsity pattern. Since HSS allows more than one rank to be assigned with sparsity patterns, we introduce the parameter N which describes *the number of ranks with sparsity patterns assigned*. For each rank n where 0≤n≤N-1 and N≥1, a G:H pattern is assigned. G:H ratios can be different for different ranks. We call an instance of HSS that contains N ranks with sparsity patterns assigned an N-rank HSS (*e.g.*, in Fig. 5, there are four ranks in the fibertree, and two of them have dedicated G:H sparsity patterns, so we call it a two-rank HSS).

4.1.1 Fibertree-based Specification. To more concretely illustrate the idea of HSS, we will refer to the fibertree-based specification of the two-rank HSS pattern shown in Fig. 5. However, for general HSS patterns, the choices of rank ordering, flattening, and splitting are not limited to the ones shown in this example.

The example HSS pattern orders the ranks in R, S, C fashion, similar to the one in Fig. 4(b). Unlike in Fig. 4(b), where the original C rank is partitioned into two ranks, the example HSS pattern partitions the original C rank into three ranks C_2, C_1, and C_0, and assigns 3:4 and 2:4 to the lowest two ranks, *i.e.*, C_1 and C_0. Such a two-rank HSS pattern can be described as RS→C_2→C_1(3:4)→C_0(2:4). As shown in Table 4, the specification has more than one rank with pruning rules assigned, resulting in qualitatively different sparsity patterns compared to existing DNN sparsity patterns.

4.1.2 Sparsity Degrees. Different ranks in a multi-rank HSS have sparsity patterns with different granularity. For example, in Fig. 5, the lowest C_0 rank's 2:4 structure is based on a single-value granularity. Whereas the higher C_1 rank's 3:4 structure is based on a larger granularity that's the shape of the lower fiber. Thus, the 3:4 ratio describes whether a fiber payload of each coordinate must contain all zeros or can contain nonzeros.

The overall sparsity degree of an HSS tensor can be derived from its sparsity structures at each rank. For example, the two-rank HSS RS→C_2→C_1(3:4)→C_0(2:4) in Fig. 5 has a sparsity of

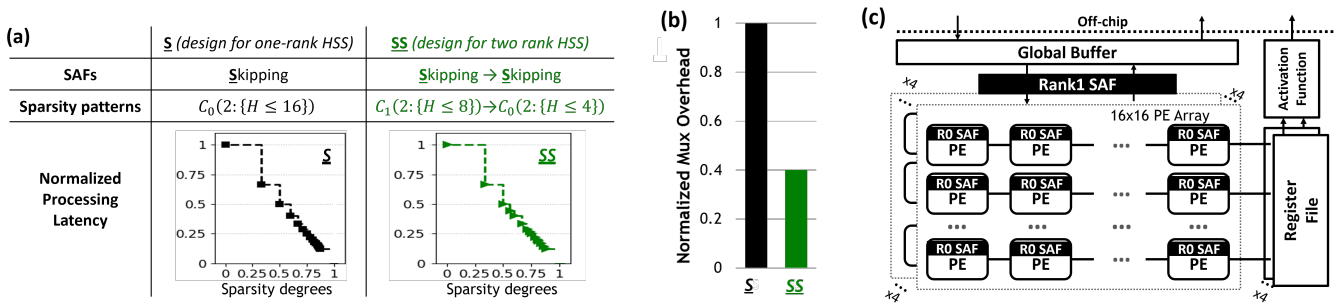

Figure 6: Comparison of designs with the same flexibility (15 sparsity degrees across 0%-87.5%) but different numbers of ranks. SS shows great potential for high flexibility and efficiency. (a) Design attributes and normalized processing latency (markers indicate the discrete sparsity degrees.) (b) Normalized muxing overhead. (c) High-level architecture of HighLight, with modularized SAFs for each rank at different architecture levels.

$1 - \frac{3}{4} \times \frac{2}{4} = 0.625$. In general, the overall sparsity degree can be expressed as $sparsity = 1 - \prod_{n=0}^{N-1} \frac{G_n}{H_n}$, where G_n:H_n is the ratio assigned to rank n. Thus, by assigning a different number of ranks and different G:H ratios at each rank, HSS allows a flexible and systematic expression of various overall sparsity degrees.

Note: for ease of presentation, we will succinctly specify all sparsity patterns with only the ranks with sparsity patterns, *e.g.*, RS→C_1→C_0(2:4) is simplified to C_0(2:4).

4.2 DNN Sparsification with HSS

Similar to all existing DNN sparsity patterns, HSS patterns can be introduced into DNNs to produce sparse DNN models. The goal for HSS-based sparsification is to ensure the most important nonzero values are preserved as much as possible.

To achieve the goal, we sparsify a dense tensor rank-by-rank in a lower-to-higher fashion. For example, for a C_1(3:4)→C_0(2:4) HSS, we first apply the rank C_0's 2:4 pattern and then rank C_1's 3:4 pattern. For the lowest rank, we sparsify the values with the smallest magnitude. For an intermediate rank, we prune coordinates whose fiber payload has the smallest *scaled L2 norm*, defined as the average magnitude of all values in the payload. Depending on the per-rank sparsity patterns, the flexibility of HSS allows us to obtain sparse models with diverse sparsity degrees.

Since the introduced sparsity pattern is orthogonal to the pruning algorithm choice (*e.g.*, pruning on trained dense model [32], pruning from scratch [59], pruning with value revival [29], etc.), it is the algorithm designer's freedom to decide whether the ranks are sparsified at once or gradually sparsified over the process. As we will show in Sec. 7, even with a traditional pruning algorithm, a DNN with HSS patterns can maintain reasonable accuracy.

5 HIGHLIGHT OVERVIEW

HSS unveils an organized HSS-based hardware design space, where each design can be systematically developed by considering three aspects for each HSS operand:

- G:H patterns supported at a rank.
- the number of HSS ranks supported by the hardware.
- the acceleration techniques, referred to as *sparse acceleration features (SAFs)* [54], supported at each rank.

In this section, we motivate HighLight's high-level architecture by discussing the impact of making different design decisions for the above design aspects.

5.1 Impact of Supported SAF at Each Rank

The accelerator can have different supported SAFs at a rank to translate sparsity into different savings. Specifically, when there are ineffectual operations, the hardware can employ

- **Gating:** lets the hardware stay idle to save energy. Gating often involves a trivial sparsity tax, *e.g.*, an AND gate.
- **Skipping**: fast forwards to the next effectual operation to save energy and time. Skipping incurs a higher sparsity tax, *e.g.*, muxing logic for leader-follower intersections.

Since gating is undesirable for many latency-sensitive applications, we will focus on discussing the impact of various design aspects assuming skipping as the supported SAF.

Skipping is highly reliant on high utilization of the components to achieve the desired speedup, so it is desirable to support G:H patterns with a fixed (set of) G that is a factor of the number of parallel hardware units (*e.g.*, with four processing elements (PEs), it is desirable to support G=4 patterns, as all PEs can be utilized with the four nonzeros in the block of H values, regardless of what H is). As a result, as shown in Fig. 6(a), the example designs with skipping SAFs support sparsity patterns with a fixed G value of two at each rank.

Takeaway: it's more desirable to support skipping, which favors G:H patterns with a G that's a factor of the available number of hardware instances to more easily ensure workload balancing with a high hardware utilization.

5.2 Impact of Per-rank Supported Patterns

To implement skipping for a G:H pattern, additional hardware is needed. For example, Fig. 7(a) shows a dot product workload with two vectors: vector 0 (*v0*) is 4:8 sparse, and vector 1 (*v1*) is dense. In order to *only* perform effectual computations, *i.e.*, skip the ineffectual computations, the accelerator needs 8-to-4 muxing logic to select the correct *v1* values. Specifically, as shown in Fig. 7(b), the 8-to-4 muxing logic can be implemented with four 8-to-1 muxes. For example, the first 8-to-1 mux selects A based on a's coordinate

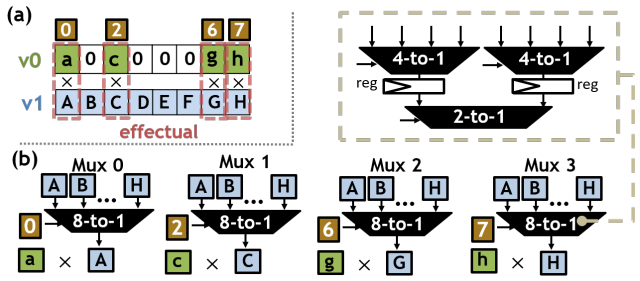

Figure 7: (a) Effectual computations in a dot product with 2:4 sparse vector 0 (v0) and dense vector 1 (v1). (b) Implementation of the muxing logic for selecting four values from a block of eight values.

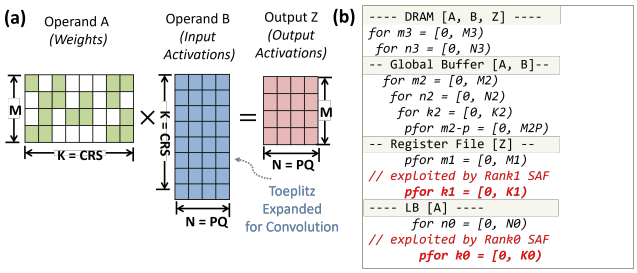

Figure 8: (a) Convolution represented as matrix multiplication with flattened operand A (weights) and Toeplitz expanded operand B (input activations) [44]. *M: number of filters; C: # of channels; R,S: height and width of filter kernels; P,Q: height and width of outputs.* (b) Loopnest representation of HighLight's dataflow.

0. To ensure low latency, an 8-to-1 mux can be implemented with two 4-to-1 muxes pipelined with a 2-to-1 mux.

Since an accelerator can be designed to support multiple G:H patterns with different H values, the muxing sparsity tax increases as the largest supported H value, *i.e.*, H_{max}, increases. Specifically, in order to support all possible patterns, the accelerator needs G number of H_{max}-to-1 muxes.

Takeaway: with a fixed G, the energy and area sparsity tax increases approximately linearly with H_{max}.

5.3 Impact of Supported Number of Ranks

Given a target number of sparsity degrees to represent, supporting more HSS ranks reduces the H_{max} at each rank, reducing the sparsity tax. Specifically, due to the nature of fraction multiplications, multi-rank HSS can easily represent a large number of sparsity degrees with a much smaller H_{max} at each rank by exploiting the composability of sparsity patterns. As shown in Fig. 6(a), with both designs supporting 15 different sparsity degrees across 0% to 87.5%, compared to the one-rank HSS design S, which requires a H_{max} of 16, the two-rank HSS design SS only requires H_{max} of 8 at Rank1 and a H_{max} of 4 at Rank0.

The sparsity support of a multi-rank HSS design is implemented at different architecture levels, each of which targets a specific rank (*e.g.*, in Fig. 6(c), the processing element (PE) array level implements *Rank1 SAF* to exploit Rank1 patterns and the PE level implement *Rank0 SAF* to exploit Rank0 patterns). Fig. 6(b) shows the normalized sparsity tax for the two HSS designs in terms of their muxing overhead. Due to the reduced H_{max} values at each rank, SS introduces $> 2\times$ less muxing overhead, *i.e.*, lower sparsity tax, while representing the same number of sparsity degrees as S.

Takeaway: Compared to the popular one-rank HSS supported in many existing works, hardware designed for multi-rank HSS can represent the same number of sparsity degrees with much lower sparsity tax.

5.4 High Level Architecture

Fig. 6(c) shows the high-level architecture organization of our proposed HighLight accelerator, a simultaneously efficient and flexible accelerator consisting of a memory hierarchy and 1024 *MAC*s grouped into four PE arrays. HighLight supports DNNs with two-rank HSS weights and unstructured sparse input activations. In terms of SAF choices, HighLight implements modularized skipping SAFs at different architecture levels to translate the two-rank HSS into energy and latency savings, and performs gating on input activation's sparsity to further reduce energy consumption.

6 A DEEPER DIVE INTO HIGHLIGHT

In this section, we present more details on HighLight's micro-architecture implementations. **For ease of presentation, we will use a down-sized architecture with two PEs and sparsity support for $C_1(2{:}\{2{\le}H{\le}4\}){\rightarrow}C_0(2{:}4)$ to discuss the core ideas of the HighLight micro-architecture.**

6.1 DNNs Processed as Matrix Multiplication

We design HighLight to process various layers in DNNs as matrix multiplications (MM) workloads (as shown in Fig. 8(a), convolutional layers are flattened with Toeplitz expansion on the inputs [44] before sending to the accelerator for processing), as many existing works do [25, 30, 36, 37, 48, 60]. Thus, DNN layers with MM kernels (*e.g.*, fully connected layers) are represented as they originally are. Whereas, as shown in Fig. 8(a), convolutional layers are represented as MM by flattening the weight dimensions and performing a Toeplitz expansion on the inputs [44] before sending to the accelerator for processing. Processing all layers as matrix multiplications implies interchangeable operands. Hence, instead of referring to the operands as weights and input activations, we refer to them as operands A and B, where operand A is dense or HSS, and operand B is either dense or unstructured sparse.

6.2 Compression Format for HSS

To correctly eliminate ineffectual hardware operations, *i.e.*, buffer accesses and computes associated with zeros, it is important to capture both ranks' sparsity structure with metadata. HighLight uses an offset-based coordinate representation (CP) [54] format to describe the position of nonzero values/non-empty blocks at each rank. Fig. 9 shows the metadata for an example $C_1(2{:}4){\rightarrow}C_0(2{:}4)$ operand A tensor. For Rank0, each nonzero value carries a CP to

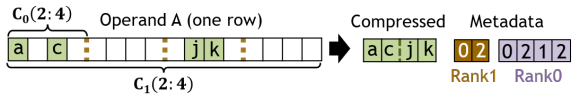

Figure 9: Hierarchical CP compression for operand A row.

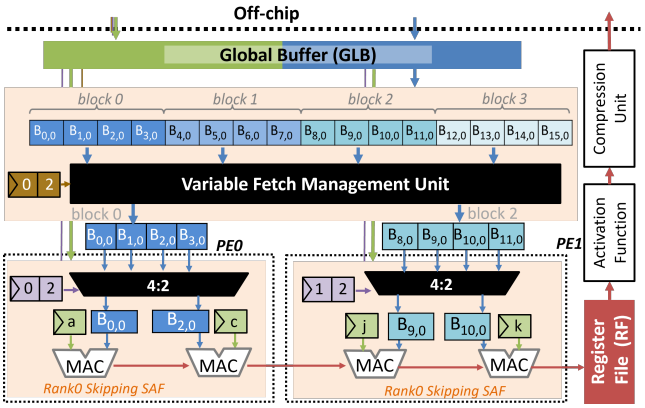

Figure 10: Down-sized architecture organization of HighLight with hierarchical skipping SAF. The showcased processing flow is for the example $C_1(2{:}4) \rightarrow C_0(2{:}4)$ operand A in Fig. 9 and a dense operand B. Matched capitalized and lower case letters indicate corresponding values. Boxes with triangles are registers.

indicate its position in its block of H_0 values (*e.g.*, since a is at the first position in its block, it carries a 0 metadata). For Rank1, each nonzero block carries a CP to indicate its relative position in the H_1 blocks (*e.g.*, the first and third blocks have nonzeros and thus carry upper-level metadata 0 and 2.)

6.3 Hierarchical Skipping

To achieve high utilization of the hardware, and thus fast processing speed, HighLight employs a hierarchical skipping technique, *i.e.*, both *Rank1 SAF* and *Rank0 SAF* perform skipping based on their target rank's sparsity structure, as shown in Fig. 10. **Thus, HighLight's total speedup is the product of the speedup introduced at each rank.** To illustrate the ideas, we use the $C_1(2{:}4) \rightarrow C_0(2{:}4)$ operand A shown in Fig. 9 and a dense operand B as an example workload. We will discuss sparse B operand support in Sec. 6.4.

6.3.1 HSS-Operand Stationary Dataflow. Before diving into the SAFs, we discuss the general processing flow of HighLight by presenting its *dataflow*, which defines an accelerator's scheduling of data movement and compute in space and time [5, 40]. To exploit the statically known sparsity structure to introduce desirable workload balancing, HighLight employs an HSS-operand stationary dataflow, where each Rank0 block of A is held stationary in each PE for reuse across different operand B values. As shown in Fig. 10, PE0 holds stationary in registers the two nonzero values a, c in the first block of operand A. Each MAC in the PE is responsible for working on one of the G nonzeros in its assigned block, specifically, the MAC on the left in PE0 works on a, and the right MAC works on c. The partial sums calculated at each MAC are first spatially accumulated across the PEs in the same row and updated to the *Register File*. More dataflow details are described in Fig. 8(b) based on the well-known loopnest representation [5, 40, 44].

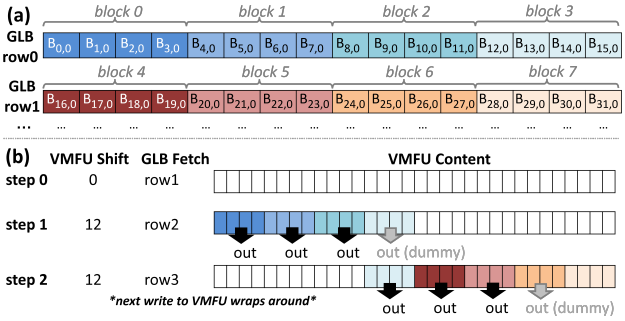

Figure 11: (a) Operand B memory layout in GLB. (b) Operand B datamovement between GLB and *Variable Fetch Management Unit (VFMU)* for the first three processing steps when operand A has a $C_1(2{:}3)$ sparsity. To output the correct operand B blocks, *VFMU* is configured to shift by 12 values (three blocks) per read. *out* points to blocks that VFMU outputs to the muxing logic in the step.

6.3.2 Skipping SAF at Rank1. HighLight's *Rank1 Skipping SAF* exploits the sparsity structure in *Rank1* only. Specifically, it is responsible for only distributing non-empty Rank1 blocks in operand A and the corresponding blocks in operand B to the PEs for parallel processing, *e.g.*, as shown in Fig. 10, only the nonzeros in the first block (*i.e.*, a, c) and the third block (*i.e.*, j, k) in operand A are transferred to be processed at the PEs. Since only half of the *Rank1* blocks are non-empty in the example tensor, *Rank1 Skipping SAF* introduces a 2× speedup. Other sparsity patterns (degrees) in rank 1 can be exploited similarly to achieve different speedups.

Recall that *Rank1 Skipping SAF* is required to support sparsity patterns defined by $C_1(2{:}\{2 \leq H \leq 4\})$. To maintain high utilization for different sparsity patterns (degrees), each of the two *PEs* in Fig. 10 should always get a non-empty block of operand A. To ensure correctness, different operand B data need to be selected for computation for different supported sparsity patterns. For different H values at Rank1, a different number of operand B blocks need to be selected from at each processing step. For example, as shown in Fig. 10, four blocks (b_0, b_1, b_2, *and* b_3) need to be selected for operand A having HSS patterns with H=4).

However, due to the fixed physical dimensions of the GLB, each GLB fetch has to be fixed to a certain number of blocks. As shown in the example GLB memory layout in Fig. 11, each GLB row contains 16 data words from a dense operand B (*i.e.*, four Rank1 blocks). To avoid unaligned *GLB* fetches for different H_1 values, as shown in Fig. 10, HighLight's *Rank1 Skipping SAF* employs a *Variable Fetch Management Unit* (*VFMU*) to allow variable length streaming access, which is a technique commonly used for bitstream parsing (*e.g.*, in the entropy coding for video compression [7, 43]). More specifically, *VFMU* includes a small buffer that stores the $2{\times}H_{max}$ blocks of operand B. The buffer is written with data that are fetched

from *GLB* in an aligned fashion and can be configured with a *shift* signal to determine the offset position for the current read to start. Fig. 11 describes the data movement between GLB and *VMPU* for the first three processing steps when operand A has a G:H=2:3 sparsity at C_1. The *shift* signal is configured to three blocks (*i.e.*, 12 values) to allow the correct operand B blocks to be read out. Note that to keep the output width uniform, there are always four blocks read out of the VMPU. However, in the case of G:H=2:3, the last block is just a dummy padding that will never be selected by the muxing logic in *Rank1 Skipping SAF*. The *VMPU* processing is trivial to show for operand A with G:H=2:4, as all accesses are well aligned. Such variable fetch support allows correct operand B to be fetched for different 2:H structures.

With the correctly shifted blocks, to avoid implementing wide muxes that select from the entire blocks of data, VFMU employs 4-to-2 muxes to select the correct pair of start and end addresses using the metadata from operand A. The addresses are used to index into VFMU's internal registers.

6.3.3 Skipping SAF at Rank0. As discussed above, each *PE* in HighLight works on a non-empty *Rank1* block with $C_0(2:4)$. To keep the two *MACs* busy in each *PE*, HighLight employs Rank0 skipping SAF with 4-to-2 muxing logic. As shown in Fig. 10, based on *Rank0*'s CP metadata, the 4:2 mux in each *PE* selects the correct operand B for each *MAC*.

6.4 Exploiting Operand B Sparsity

So far, we have used a dense operand B in our example workloads to illustrate HighLight's processing flow. However, in DNN workloads, operand B for can be sparse due to non-linear activation functions (*e.g.*, ReLU) and/or activation pruning [30, 55]. HighLight exploits unstructured sparse operand B through compression and gating.

We again use the $C_1(2:3) \rightarrow C_0(2:4)$ example to present the hardware support. As shown in Fig. 12(a), when compressed, only the nonzero Operand B values are stored in the GLB. Operand B carries three levels of metadata that hierarchically encodes the nonzero value locations. Specifically, the metadata includes: **(1)** the total number of nonzeros for every set of Rank1 blocks (three in $C_1(2:3)$); **(2)** end addresses of each Rank1 block; **(3)** the intra-Rank0-block offset for each nonzero value. For intermediate layers, such compression on a previous layer's output activation is performed by the compression unit after the activation function unit in Fig. 10 to prepare for the processing for the next layer.

Since different Operand B blocks have different occupancy, instead of always assuming a fixed shift amount as in the case of a dense operand B, the VFMU assigns the shift to be the encoded offset for each set of Rank1 blocks. For example, as shown in Fig. 12(b), the shift at *step1* is configured to 8 as the first three Rank1 blocks have a total of 8 nonzero values. Furthermore, if there are enough data words stored in VMFU for the next processing step, the GLB fetch is not performed (*e.g.*, at step 2 in Fig. 12, VFMU has 13 valid entries, and the next processing step only needs 8, so no GLB fetch is performed at the step). Such a mechanism allows the metadata information to catch up with the fetched nonzeros from GLB.

The gating SAF is applied to Operand B's Rank0 sparsity to save energy by letting the *MAC* unit in each PE stay idle when there is no effectual operation to perform. Note that since the gating SAF does not change the number of cycles spent, it still keeps the PEs in sync, and thus the partial sum accumulation inside each PE and across PEs is not impacted by the support for sparse Operand B.

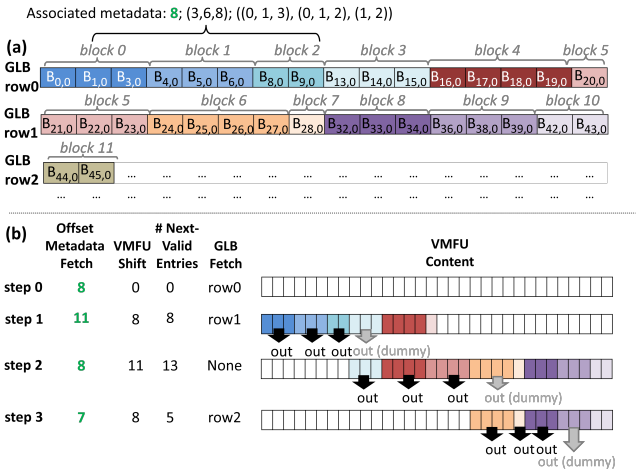

Figure 12: (a) Compressed operand B memory layout in GLB and the metadata associated with the first three blocks. (b) Operand B datamovement between GLB and VFMU for the first four processing steps when operand A has a $C-1$*(2:3)* sparsity. VFMU shifts based on the number of nonzeros encoded in the metadata for each set of Rank1 blocks. GLB fetches are only performed when the number of valid data in VFMU cannot meet the need for the next set of Rank1 blocks.

7 EXPERIMENTAL RESULTS

In this section, we discuss our experimental setup and present results related to the co-designed hardware (*i.e.*, HighLight) and the software (*i.e.*, HSS-based sparsification).

7.1 Methodology

7.1.1 Baseline designs. Given the abundant prior designs, we compare HighLight to state-of-the-art representative designs that capture the key properties of each category described in Table 1.

- TC [36] represents dense accelerators (*e.g.*, [4, 25, 36]), which are oblivious of the potential benefits introduced by sparsity.
- STC [37] represents single-sided G:H structured sparse accelerators (*e.g.*, [37, 60]), which introduce considerable efficiency benefits at a reasonable sparsity tax, but often support a limited number of sparsity degrees.
- S2TA [30] represents dual-sided (*i.e.*, both operands) G:H structured sparse accelerators, which improve on the single-sided designs with additional efficiency gains from the second sparse operand, but can introduce higher sparsity tax.
- DSTC [52] represents dual-sided unstructured sparse accelerators (*e.g.*, [8, 14, 41, 52, 57]), which often have high flexibility but introduce considerable sparsity tax.

Table 3 describes more details on each design's supported sparsity structures (if any) for each operand.

Design	Supported Sparsity Patterns	
	Operand A	Operand B
TC [36]	dense	
STC [37]	dense; C_0({G≤2}:4)	dense
DSTC [52]	dense; unstructured sparse	
S2TA [30]	C_0({G≤4}:8)	C_0({G≤8}:8)
HighLight (our work)	C_1(4:{4≤ H≤8})→C_0(2:{2≤ H≤4})	dense; unstructured sparse

Table 3: Supported sparsity patterns for each design.

Design	Storage		Compute
	GLB	RF	
TC [36]	320KB	4 × 2 KB	4 × 256
STC [37]	256 + 64KB		
DSTC [52]	256 + 64KB		
S2TA [30]	256 + 64KB	64 × 64B	64 × 16
HighLight (our work)	256 + 64KB	4 × 2KB	4 × 256

Table 4: Hardware resource allocation. GLB is partitioned to data and metadata storage for sparse designs.

To ensure fairness, as shown in Table 4, we allocate similar storage and compute resources to all designs. For designs that support compression of sparse input activations (*i.e.*, DSTC, S2TA, and HighLight), the same-style compression unit support as shown in Fig. 10 is applied to reduce DRAM traffic. Furthermore, all accelerators are designs that process DNNs as matrix multiplications. Since matrix multiplications accelerators treat the two operands interchangeably, we allow them to swap operands and report the best hardware performance (*e.g.*, since STC benefits from sparse operand A, we swap the operands if operand B is sparse and A is dense).

7.1.2 Workloads. We evaluate two classes of workloads: **(i) Synthetic matrix multiplication** with operand A and B matrices that are 1024-by-1024, a common shape in DNN workloads. A and B are of various sparsity degrees: three different degrees for A: 0%, 50%, 75%, and four different degrees for B: 0%, 25%, 50%, 75%. Synthetic workloads allow us to capture the diverse sparsity characteristics in the DNN design space. **(ii) Representative DNN models** with distinct network architectures and target applications: the convolutional ResNet50 [16] and attention-based Deit-small [47] for image classification trained on ImageNet [12] and the attention-based Transformer-Big [50] for language translation trained on WMT16 EN-DE [3]. Actual DNN models allow us to take both accuracy and hardware performance impact into the picture.

7.1.3 Evaluation Frameworks. **Accelerator Modeling:** we use the Sparseloop-Accelergy infrastructure [53, 54] to model the accelerators. Sparseloop captures each accelerator's cycle counts and component runtime activities. We added a new density model to Sparseloop to capture the characteristics of HSS.

To characterize energy and area costs, we built 65nm Accelergy estimation plug-ins to characterize various components:

- HighLight's design-specific SAF implementations (*i.e.*, muxing logic and VFMU) and datapath components (*e.g.*, adders, multipliers): synthesized RTL.
- Small SRAMs: SRAM compiler.
- Large SRAMs not supported by compiler: CACTI [13].
- DRAM: propriety commercial data for LPDDR4.

All accelerator designs are evaluated with the same evaluation framework to ensure fairness.

DNN Pruning: We use Condensa [24] to introduce various sparsity patterns to various DNNs, structured or unstructured. Since a good set of sparsity patterns should allow reasonable accuracy recovery even without novel or advanced pruning algorithms (*e.g.*, special ways to perform hyper-parameter searches), we reuse the pruning algorithm proposed for sparse tensor core (STC) [32]. Specifically, the algorithm for STC first statically prunes a pre-trained dense DNN by masking the appropriate weights and their gradients to zeros based on sparsity-pattern-specific sparsification rules (*e.g.*, the HSS-based rules in Sec. 4.2), and it then fine-tunes the masked DNN to regain accuracy. *To ensure fairness, all of our performed pruning follows the same algorithm, and the same set of hyperparameters is used for all sparsity patterns.*

7.2 Outperforms Prior Work

We compare HighLight to existing designs running synthetic workloads to demonstrate its flexibility and efficiency. Specifically, its ability to always achieve high processing efficiency for workloads with varying sparsity degrees.

Fig. 13 compares the processing latency, energy consumption, and energy-delay product (EDP), a widely used metric for evaluating overall hardware performance in many existing works [20, 22, 28]. As shown in Fig. 13, different existing designs introduce inefficient processing at different sparsity degrees. Specifically, **(i) STC** employs simple acceleration with low sparsity tax for dense and 50% sparse workloads. However, STC's limited sparsity support fails to exploit the available opportunities for both speedup and energy for high sparsity workloads. **(ii) DSTC** introduces significant sparsity tax to identify effectual operations. Specifically, it employs a dataflow that requires a costly accumulation buffer that is frequently accessed. Thus, DSTC's high sparsity tax masks the sparsity-related savings for workloads with low sparsity. Furthermore, DSTC also suffers from a not perfectly balanced workload due to the unpredictable nature of unstructured sparsity, *i.e.*, not all compute units are active. **(iii) S2TA** requires both operands to be structured sparse and has limited flexibility on the G values supported for each operand. For example, as shown in Table 4, S2TA requires one of the operands to have {G≤4}:8, *i.e.*, cannot have more than 50% sparsity. Thus, S2TA often fails to support or does not fully exploit the available speedup for workloads with operands that have low or medium sparsity.

On the other hand, HighLight is always able to efficiently exploit various sparsity degrees. HighLight's per-rank skipping SAF and low-overhead hierarchical compression format introduce brings low sparsity tax, specifically low energy overhead. Furthermore, due to the structured sparsity, HighLight always achieves theoretical speedup with perfect workload balancing. Thus, as shown

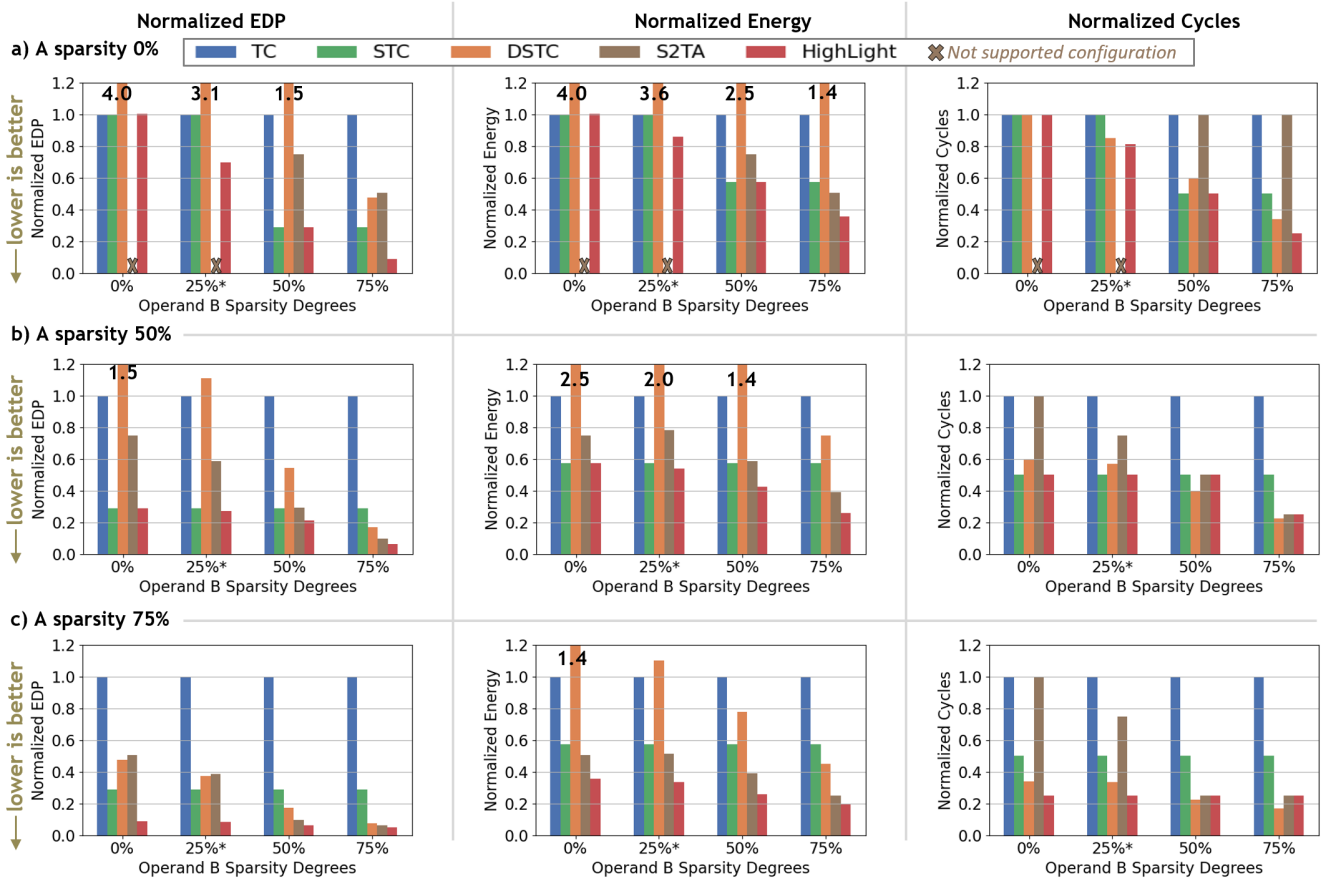

Figure 13: Comparison of existing designs running workloads with operands with different sparsity degrees. We compare the overall hardware efficiency energy-delay-product (EDP), energy and speed of the designs. S2TA [30] assumes both operands are structured. HighLight is always able to effectively exploit diverse sparsity degrees. **HighLight evaluated with 20% sparsity for conservative estimations.*

in Fig. 13, HighLight always achieves the best EDP and comparable-to-best processing speed for all evaluated sparsity degrees. HighLight achieves the best geomean for all evaluated metrics.

Fig. 14 shows each metric's geomean across the evaluated workloads. **Compared to existing designs, HighLight achieves better geomean for all evaluated metrics.**

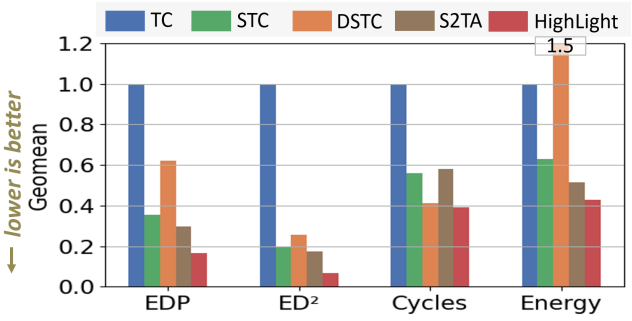

Figure 14: Geomean of various metrics. HighLight achieves the best geomean across all evaluated metrics.

7.3 Good Accuracy-Efficiency Trade-offs

To demonstrate HighLight provides good trade-offs between accuracy and efficiency, we compare the **EDP-accuracy loss relationship** of various design approaches, as shown in Fig. 15. Specifically, we compare HighLight to multiple popular existing co-design approaches: **1)** dense (represented by the *TC* data points); **2)** unstructured sparse (represented by the *DSTC* data points); **3)** C_0(G:H) sparse (represented by the *STC* and *S2TA* data points).

We evaluate three representative DNNs: ResNet50 [16], Deit-small [47], and Transformer-Big [50]. For ResNet50, we prune all convolutional and fully-connected layers. For Deit-small, we pruned the feed-forward block and the output projection weights. For Transformer-big, we prune the feed-forward block and all projection weights. To ensure fairness, we use the same pruning algorithm as described in Sec. 7.1.3 for all of the evaluated sparsity patterns, including both structured and unstructured.

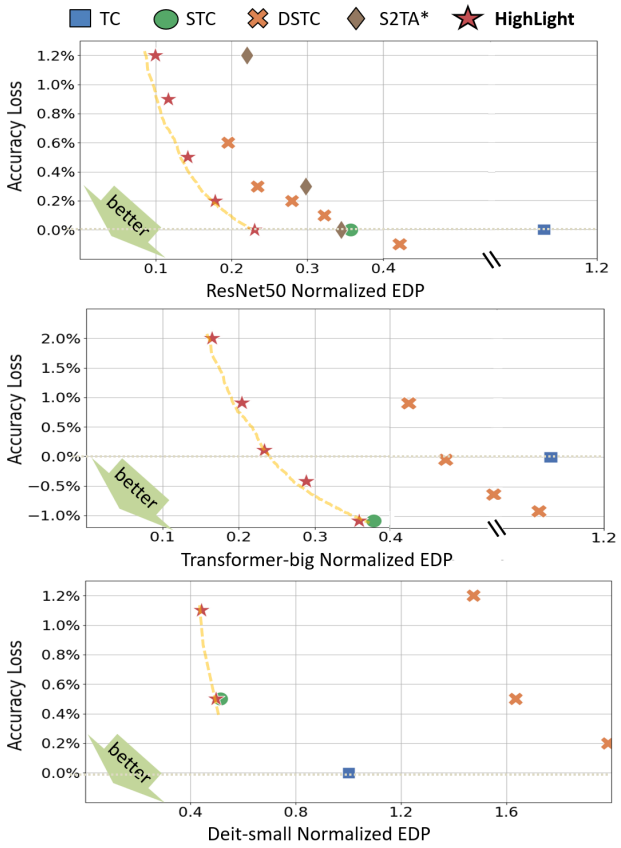

Figure 15: EDP-Accuracy Loss Pareto frontier for ResNet50 [16], Transformer-Big [50], and Deit-small [47]. Different markers refer to different accelerators. HighLight is always on the accuracy-EDP Pareto frontier and thus serves as a great candidate to support diverse DNNs with high hardware efficiency while maintaining accuracy.

Fig. 15 shows the EDP-accuracy loss relationship for the three DNN models, with their weights pruned to different sparsity degrees. Ideally, we would like to always have very low EDP and accuracy loss. Unfortunately, low EDP often requires higher sparsity and thus leads to higher accuracy loss. The best design should excel at balancing the trade-off, thus always sitting on the Pareto frontier of the EDP-accuracy loss relationship. Furthermore, the design should excel at the evaluated DNNs, which have different sparsity characteristics. Specifically, ResNet50 has much sparser activations than Transformer-big and Deit-small, and Deit-small has much fewer layers being pruned due to its already small parameter count (compared to other vision transformers).

As shown in Fig. 15, HighLight always sits on the Pareto frontiers. STC only delivers great accuracy-efficiency trade-off only at a single sparsity degree (*i.e.*, 50% sparse), S2TA fails to support attention-based models due to its incapability to process purely dense layers, and DSTC can introduce worse-than-dense EDP due to its high sparsity tax for the relatively dense models. **Thus, HighLight serves as a great accelerator candidate to support diverse DNNs with high hardware efficiency while maintaining a reasonable accuracy loss.**

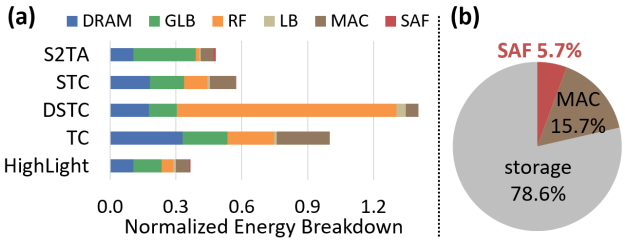

Figure 16: (a) Energy consumption breakdown for a workload with 75% sparse operand A and dense operand B. (b) HighLight area breakdown. HighLight introduces low sparsity tax in terms of both energy and area.

7.4 Sparsity Tax Evaluation

Sparse DNN accelerators involve two types of sparsity tax: energy and area. Fig. 16(a) shows the energy cost breakdown across different components in different architectures processing an example workload with 75% operand A and a dense operand B (*i.e.*, one example set of bars from Fig. 13). Existing designs either do not fully exploit the sparsity for energy savings (*e.g.*, STC only recognizes upto 50% sparsity) or introduce inefficient dataflows to trade-off its insignificant SAF cost (*e.g.*, DSTC suffers from significant accumulation traffic at RF due to its outer produce style dataflow). Fig. 16(b) shows the area breakdown of HighLight, with the SAFs accounting for only 5.7% of the design's area. Note that since sparsity tax is intrinsic to the hardware design, different workloads would not change the general amount of sparsity tax introduced[2]. **Thus, HighLight has low sparsity tax.**

7.5 Potential Benefits of Dual-Side HSS

Exploiting sparsity in both operands for speedup, *i.e.*, dual-side speedup, is highly desirable but often requires complex intersection hardware and workload balancing techniques. To address such a challenge, we make the observation that, with low intersection sparsity tax and easy workload balancing, hierarchical structured sparsity can also potentially be used to achieve dual-side speedup. In this section, we will discuss the potential benefits such improvements could bring to motivate more studies on fully supporting dual-side HSS workloads.

An HSS-based accelerator can achieve dual-side speedup with easy workload balancing by supporting multi-rank HSS operands with alternating dense ranks, *e.g.*, weights with C_1(dense)→C_0(2:4) and input activations (iacts) with C_1(2:4)→ C_0(dense). To identify the nonzero value locations, each operand only needs to carry the offset metadata for the rank with G:H sparsity. For example, weights with C_1(dense)→C_0(2:4) carry offset metadata for each nonzero value to identify its relative position in its C_0 block, and iacts with C_1(2:4)→ C_0(dense) carry offset metadata for each C_0 block to identify its relative position in each C_1 block.

[2]The metadata costs differ based on workload sparsity, but such cost is not the dominant source of sparsity tax in the evaluated designs.

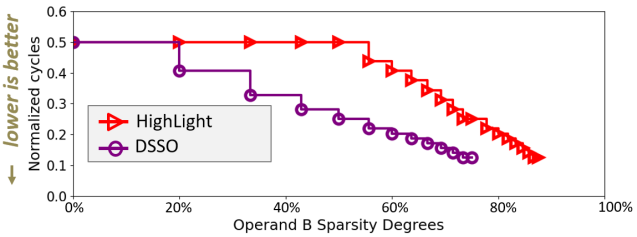

Figure 17: Normalized processing speed of HighLight and the dual structured sparse operand (*DSSO*) design. The *DSSO* design supports dual-side HSS with alternating dense ranks and allows dual-side speedup.

With the alternating dense ranks, both operands are never sparse at the same rank; therefore, the SAF at each rank only needs to perform dense-sparse intersections. For instance, for C_1, skipping can be performed by intersecting C_1(2:4) in iacts with C_1(dense) in weights. For C_0, skipping can be performed by intersecting C_0(2:4) in weights with C_0(dense) in iacts. Such dense-sparse intersections by nature lead to a perfectly balanced workload [11]. We refer to the accelerator design that supports dual-side HSS as the *dual structured sparse operands (DSSO)* design.

Fig. 17 compares the processing speed of HighLight and *DSSO* for workloads with operand A (weights) with C_1(dense)→C_0(2:4) and operand B that follows $C_1(2{:}2 \leq H \leq 8)$→C_0(dense). *DSSO* demonstrates interesting trade-offs compared to singled-sided HSS. As shown in Fig. 17, ***DSSO* achieves 2× better processing speed compared to HighLight for the commonly supported sparsity degrees.** However, since *DSSO* requires one rank to be dense to enable perfect workload balancing, there are fewer sparsity degrees supported for operand B.

As currently proposed, the HighLight design does not naively support the dual-side HSS configuration. Specifically, HighLight does not discuss the hardware support needed to prune and compress the output activations of each layer into the desired HSS format. In addition, although existing works have shown that it is possible for DNNs with dual structured sparse operands to maintain accuracy [9, 30, 55], such DNNs may still require more advanced pruning techniques to recover accuracy. Thus, there remain many interesting research questions to answer regarding complete dual-side HSS support in both hardware and pruning algorithms.

8 RELATED WORK

There is ample prior work in designing accelerators for efficiently processing sparse DNNs. These works either focus on co-designing sparsity patterns and hardware or solely focus on hardware for existing pruned models. Co-design approaches involve pruning DNNs to structured sparsity patterns that can be easily exploited by the underlying hardware. The target underlying system can be existing dense systems (maybe with relatively minor ISA updates) [10, 17, 35, 46, 59], *e.g.*, GPUs, or custom accelerators [27, 30, 37, 49, 60] designed for the sparsity structure. The accelerators can be designed either with conventional digital technology or emerging technology, *e.g.*, processing-in-memory accelerators [49]. To better recover accuracy loss due to the enforced structure, some proposals have relatively relaxed structures and pre-process the pruned models into more compact structures before sending them to hardware, *e.g.*, pack unstructured columns into compact blocks [27]. Since structured sparsity has static nonzero values locations, the accelerators often have a low sparsity tax but low flexibility.

On the other hand, accelerators designed for existing pruned models or general sparse matrix multiplications often involve designing flexible sparsity support for unstructured sparsity [6, 8, 14, 19, 26, 39, 41, 42, 56–58]. Since supporting dynamic nonzero value locations requires extremely flexible hardware, these designs often focus on different dataflows that reduce complexity, efficient auxiliary components (*e.g.*, fast intersection unit [19]) that alleviate the significant control overhead, etc. Nonetheless, such accelerators often rely on the assumption that unstructured pruning can introduce very high (> 80%) sparsity, which can cancel out the cost of high sparsity tax.

The concept of hierarchy is also used in compressed data representations [19, 26, 51]. However, this line of work often focuses on better workload partitioning to enable efficient hardware processing, instead of using the hierarchy to provide flexibility and/or modularity, which is the goal of HSS. In fact, their proposed sparsity patterns are often unstructured or one-rank structured sparse (*e.g.*, SMASH [26] employs two levels of bitmask to represent unstructured sparse tensors), and target HPC/Graph analytics applications, which often have much higher sparsity degrees, and thus are less sensitive to high sparsity tax than DNNs.

9 CONCLUSION

Various optimization techniques introduce DNNs with diverse sparsity degrees. The diversity challenges the assumptions made by existing DNN accelerators, which often trade flexibility for efficiency, or vice versa. This paper addresses the importance of balancing accelerator flexibility and efficiency by proposing a novel class of DNN sparsity patterns, hierarchical structured sparsity (HSS), which leverages the multiplication of fractions to systematically represent diverse sparsity degrees. Leveraging the modularity of HSS, we developed HighLight to achieve flexible sparsity support with low sparsity tax. In conjunction, we show that HSS allows DNN developers to prune DNNs to diverse degrees while maintaining desired accuracy levels. Compared to dense accelerators, HighLight achieves a geomean of 6.4× (and up to 20.4×) better energy-delay product (EDP) across layers with diverse sparsity degrees (including dense) and is at parity for dense DNN layers. Compared to sparse accelerators, HighLight achieves a geomean of 2.7× (and up to 5.9×) better EDP and is at parity for sparse layers.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive feedback. Part of the research was done during Yannan Nellie Wu's internship at NVIDIA Research. This research was funded by the MIT AI Hardware Program. We would also like to thank Andrew Feldman, Michael Gilbert, Tanner Andrulis, Yifan Yang, and Zi Yu (Fisher) Xue for their valuable feedback on improving the clarity of the paper.

REFERENCES

[1] Abien Fred Agarap. 2018. Deep Learning using Rectified Linear Units (ReLU). *CoRR* abs/1803.08375 (2018). arXiv:1803.08375 http://arxiv.org/abs/1803.08375

[2] Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. 2020. What is the State of Neural Network Pruning?. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze (Eds.). mlsys.org. https://proceedings.mlsys.org/book/296.pdf

[3] Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 Conference on Machine Translation. In *Proceedings of the First Conference on Machine Translation*. Association for Computational Linguistics, Berlin, Germany, 131–198. http://www.aclweb.org/anthology/W/W16/W16-2301

[4] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 269–284. https://doi.org/10.1145/2541940.2541967

[5] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 367–379. https://doi.org/10.1109/ISCA.2016.40

[6] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138. https://doi.org/10.1109/JSSC.2016.2616357

[7] Yu-Hsin Chen and Vivienne Sze. 2015. A Deeply Pipelined CABAC Decoder for HEVC Supporting Level 6.2 High-Tier Applications. *IEEE Transactions on Circuits and Systems for Video Technology* 25, 5 (2015), 856–868. https://doi.org/10.1109/TCSVT.2014.2363748

[8] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308. https://doi.org/10.1109/JETCAS.2019.2910232

[9] Brian Chmiel, Itay Hubara, Ron Banner, and Daniel Soudry. 2022. Optimal Fine-Grained N:M sparsity for Activations and Neural Gradients. https://doi.org/10.48550/ARXIV.2203.10991

[10] Kyusik Choi and Hoeseok Yang. 2021. A GPU Architecture Aware Fine-Grain Pruning Technique for Deep Neural Networks. In *Euro-Par 2021: Parallel Processing: 27th International Conference on Parallel and Distributed Computing, Lisbon, Portugal, September 1–3, 2021, Proceedings* (Lisbon, Portugal). Springer-Verlag, Berlin, Heidelberg, 217–231. https://doi.org/10.1007/978-3-030-85665-6_14

[11] Chunhua Deng, Yang Sui, Siyu Liao, Xuehai Qian, and Bo Yuan. 2021. GoSPA: An Energy-efficient High-performance Globally Optimized SParse Convolutional Neural Network Accelerator. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1110–1123. https://doi.org/10.1109/ISCA52012.2021.00090

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[13] Xiangyu Dong, Xiaoxia Wu, Guangyu Sun, Yuan Xie, Helen Li, and Yiran Chen. 2008. Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *2008 45th ACM/IEEE Design Automation Conference*. 554–559.

[14] Ashish Gondimalla, Noah Chesnut, Mithuna Thottethodi, and T. N. Vijaykumar. 2019. SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks. 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, 151–165. https://doi.org/10.1145/3352460.3358291

[15] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)* (2016).

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[17] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. In *The IEEE International Conference on Computer Vision (ICCV)*.

[18] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*. 1389–1397.

[19] 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* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 319–333. https://doi.org/10.1145/3352460.3358275

[20] Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W. Fletcher. 2021. Mind Mappings: Enabling Efficient Algorithm-Accelerator Mapping Space Search. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS 2021)*. Association for Computing Machinery, New York, NY, USA, 943–958. https://doi.org/10.1145/3445814.3446762

[21] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research* 22, 241 (2021), 1–124.

[22] Mark Horeni, Pooria Taheri, Po-An Tsai, Angshu Parashar, Joel Emer, and Siddharth Joshi. 2022. Ruby: Improving Hardware Efficiency for Tensor Algebra Accelerators Through Imperfect Factorization. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*.

[23] Jun-Woo Jang, Sehwan Lee, Dongyoung Kim, Hyunsun Park, Ali Shafiee Ardestani, Yeongjae Choi, Channoh Kim, Yoojin Kim, Hyeongseok Yu, Hamzah Abdel-Aziz, Jun-Seok Park, Heonsoo Lee, Dongwoo Lee, Myeong Woo Kim, Hanwoong Jung, Heewoo Nam, Dongguen Lim, Seungwon Lee, Joon-Ho Song, Suknam Kwon, Joseph Hassoun, SukHwan Lim, and Changkyu Choi. 2021. Sparsity-Aware and Re-configurable NPU Architecture for Samsung Flagship Mobile SoC. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 15–28. https://doi.org/10.1109/ISCA52012.2021.00011

[24] V. Joseph, G. L. Gopalakrishnan, S. Muralidharan, M. Garland, and A. Garg. 2020. A Programmable Approach to Neural Network Compression. *IEEE Micro* 40, 5 (2020), 17–25. https://doi.org/10.1109/MM.2020.3012391

[25] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. *SIGARCH Comput. Archit. News* 45, 2 (jun 2017), 1–12. https://doi.org/10.1145/3140659.3080246

[26] Konstantinos Kanellopoulos, Nandita Vijaykumar, Christina Giannoula, Roknoddin Azizi, Skanda Koppula, Nika Mansouri Ghiasi, Taha Shahroodi, Juan Gomez Luna, and Onur Mutlu. 2019. 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* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 600–614. https://doi.org/10.1145/3352460.3358286

[27] H.T. Kung, Bradley McDanel, and Sai Qian Zhang. 2019. Packing Sparse Convolutional Neural Networks for Efficient Systolic Array Implementations: Column Combining Under Joint Optimization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 821–834. https://doi.org/10.1145/3297858.3304028

[28] Hyoukjun Kwon, Liangzhen Lai, Michael Pellauer, Tushar Krishna, Yu-Hsin Chen, and Vikas Chandra. [n. d.]. Heterogeneous dataflow accelerators for multi-DNN workloads. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 71–83.

[29] Zi Lin, Jeremiah Zhe Liu, Zi Yang, Nan Hua, and Dan Roth. 2020. Pruning Redundant Mappings in Transformer Models via Spectral-Normalized Identity Prior. arXiv:2010.01791 [cs.CL]

[30] Z. Liu, P. N. Whatmough, Y. Zhu, and M. Mattina. 2022. S2TA: Exploiting Structured Sparsity for Energy-Efficient Mobile CNN Acceleration. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE Computer Society, Los Alamitos, CA, USA, 573–586. https://doi.org/10.1109/HPCA53966.2022.00049

[31] Liqiang Lu, Jiaming Xie, Ruirui Huang, Jiansong Zhang, Wei Lin, and Yun Liang. 2019. An efficient hardware accelerator for sparse convolutional neural networks on FPGAs. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 17–25.

[32] Asit K. Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. 2021. Accelerating Sparse Deep Neural Networks. *CoRR* abs/2104.08378 (2021). arXiv:2104.08378 https://arxiv.org/abs/2104.08378

[33] Diganta Misra. 2019. Mish: A Self Regularized Non-Monotonic Neural Activation Function. *CoRR* abs/1908.08681 (2019). arXiv:1908.08681 http://arxiv.org/abs/1908.08681

[34] Nandeeka Nayak, Toluwanimi O Odemuyiwa, Shubham Ugare, Christopher W. Fletcher, Michael Pellauer, and Joel S. Emer. 2023. TeAAL: A Declarative Framework for Modeling Sparse Tensor Accelerators. (2023).

[35] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. *PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-Based Weight Pruning*. Association for Computing Machinery, New York, NY, USA, 907–922. https://doi.org/10.1145/3373376.3378534

[36] NVIDIA. 2017. *NVIDIA TESLA V100 GPU ARCHITECTURE*. Technical Report.

[37] NVIDIA. 2020. *NVIDIA AMPERE GA102 GPU ARCHITECTURE*. Technical Report.

[38] Toluwanimi O. Odemuyiwa, Hadi Asghari-Moghaddam, Michael Pellauer, Kartik Hegde, Po-An Tsai, Neal C. Crago, Aamer Jaleel, John D. Owens, Edgar Solomonik, Joel S. Emer, and Christopher W. Fletcher. 2023. Accelerating Sparse Data Orchestration via Dynamic Reflexive Tiling. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 18–32. https://doi.org/10.1145/3582016.3582064

[39] Subhankar Pal, Jonathan Beaumont, Dong-Hyeon Park, Aporva Amarnath, Siying Feng, Chaitali Chakrabarti, Hun-Seok Kim, David Blaauw, Trevor Mudge, and Ronald Dreslinski. 2018. OuterSPACE: An Outer Product Based Sparse Matrix Multiplication Accelerator. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 724–736. https://doi.org/10.1109/HPCA.2018.00067

[40] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangarajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*.

[41] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada) *(ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 27–40. https://doi.org/10.1145/3079856.3080254

[42] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna. 2020. SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 58–70. https://doi.org/10.1109/HPCA47549.2020.00015

[43] Vivienne Sze and Anantha P. Chandrakasan. 2012. A Highly Parallel and Scalable CABAC Decoder for Next Generation Video Coding. *IEEE Journal of Solid-State Circuits* 47, 1 (2012), 8–22. https://doi.org/10.1109/JSSC.2011.2169310

[44] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2020. Efficient Processing of Deep Neural Networks. *Synthesis Lectures on Computer Architecture* 15, 2 (2020), 1–341. https://doi.org/10.2200/S01004ED1V01Y202004CAC050

[45] Mingxing Tan and Quoc V. Le. 2020. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv:1905.11946 [cs.LG]

[46] Yijun Tan, Kai Han, Kang Zhao, Xianzhi Yu, Zidong Du, Yunji Chen, Yunhe Wang, and Jun Yao. [n. d.]. Accelerating Sparse Convolution with Column Vector-Wise Sparsity. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[47] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. 2021. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, Vol. 139. 10347–10357.

[48] Fengbin Tu, Yiqi Wang, Ling Liang, Yufei Ding, Leibo Liu, Shaojun Wei, Shouyi Yin, and Yuan Xie. 2022. SDP: Co-Designing Algorithm, Dataflow, and Architecture for in-SRAM Sparse NN Acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022), 1–1. https://doi.org/10.1109/TCAD.2022.3172600

[49] Fengbin Tu, Yiqi Wang, Ling Liang, Yufei Ding, Leibo Liu, Shaojun Wei, Shouyi Yin, and Yuan Xie. 2022. SDP: Co-Designing Algorithm, Dataflow, and Architecture for in-SRAM Sparse NN Acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022), 1–1. https://doi.org/10.1109/TCAD.2022.3172600

[50] 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/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[51] Richard W Vuduc and Hyun-Jin Moon. 2005. Fast sparse matrix-vector multiplication by exploiting variable block structure. In *High Performance Computing and Communications: First International Conference, HPCC 2005, Sorrento, Italy, September 21-23, 2005. Proceedings 1*. Springer, 807–816.

[52] Yang Wang, Chen Zhang, Zhiqiang Xie, Cong Guo, Yunxin Liu, and Jingwen Leng. 2021. Dual-Side Sparse Tensor Core. In *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA)* (Virtual Event, Spain) *(ISCA '21)*. IEEE Press, 1083–1095. https://doi.org/10.1109/ISCA52012.2021.00088

[53] Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. 2019. Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942149

[54] Yannan N. Wu, Po-An Tsai, Angshuman Parashar, Vivienne Sze, and Joel S. Emer. 2022. Sparseloop: An Analytical Approach To Sparse Tensor Accelerator Modeling . In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*.

[55] Qing Yang, Jiachen Mao, Zuoguan Wang, and Hai Li. 2019. DASNet: Dynamic Activation Sparsity for Neural Network Efficiency Improvement. *CoRR* abs/1909.06964 (2019). arXiv:1909.06964 http://arxiv.org/abs/1909.06964

[56] Tzu-Hsien Yang, Hsiang-Yun Cheng, Chia-Lin Yang, I-Ching Tseng, Han-Wen Hu, Hung-Sheng Chang, and Hsiang-Pang Li. 2019. Sparse ReRAM Engine: Joint Exploration of Activation and Weight Sparsity in Compressed Neural Networks. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 236–249. https://doi.org/10.1145/3307650.3322271

[57] Guowei Zhang, Nithya Attaluri, Joel S. Emer, and Daniel Sanchez. 2021. Gamma: Leveraging Gustavson's Algorithm to Accelerate Sparse Matrix Multiplication. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS 2021)*. Association for Computing Machinery, New York, NY, USA, 687–701. https://doi.org/10.1145/3445814.3446702

[58] Z. Zhang, H. Wang, S. Han, and W. J. Dally. 2020. SpArch: Efficient Architecture for Sparse Matrix Multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE Computer Society, Los Alamitos, CA, USA, 261–274. https://doi.org/10.1109/HPCA47549.2020.00030

[59] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. 2021. Learning N: M fine-grained structured sparse neural networks from scratch. *arXiv preprint arXiv:2102.04010* (2021).

[60] Maohua Zhu, Tao Zhang, Zhenyu Gu, and Yuan Xie. 2019. Sparse Tensor Core: Algorithm and Hardware Co-Design for Vector-Wise Sparse Neural Networks on Modern GPUs. 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, 359–371. https://doi.org/10.1145/3352460.3358269