TB-STC: Transposable Block-wise N:M Structured Sparse Tensor Core

Jun Liu*[1,3], Shulin Zeng*[2,3], Junbo Zhao[2], Li Ding[1], Zeyu Wang[1], Jinhao Li[1], Zhenhua Zhu[2], Xuefei Ning[2], Chen Zhang[1], Yu Wang[2], Guohao Dai†[1,3]
[1]Shanghai Jiao Tong University, China [2]Tsinghua University, China, [3]Infinigence-AI, China
*Equal contributions, †Corresponding author: daiguohao@sjtu.edu.cn

***Abstract*—The computational and memory demands of Deep Learning (DL) models, from convolutional neural networks to Large Language Models (LLMs), are experiencing a notable surge. The sparsification (*e.g.*, weight pruning and sparse attention) represents a significant approach to reducing latency and energy consumption. However, it is non-trivial to identify a good trade-off between model accuracy and hardware efficiency. Existing work has sought to mitigate the hardware complexity overhead through structured sparsity, yet the resulting accuracy loss remains considerable (*e.g.*, more than 6% accuracy drop with 50% structured sparsity on OPT-6.7B and Llama2-7B).**

To address the above challenges, this paper proposes Transposable Block-wise Structured Sparsity (TBS). Our key insight is that the weight matrices of the forward and backward pass are *transposed* to each other during DL training. Exploiting this transposition property facilitates obtaining a structured sparsity pattern that is closer to the unstructured sparsity. In contrast, existing studies explore only one-dimensional structured sparsity. In light of these observations, we propose the transposable block-wise structured sparsity pattern with an efficient end-to-end sparse training method. This method improves accuracy by up to 2.58% over other structured sparsity studies under the same sparsity degree. At the micro-architecture level, we propose TB-STC, a Transposable Block-wise N:M Sparse Tensor Core to efficiently and flexibly facilitate the TBS pattern. TB-STC introduces an adaptive codec architecture for on-the-fly storage format conversion with a higher bandwidth utilization (1.47×), and implements an I/O-aware configurable architecture for sparsity-aware scheduling with a better computational utilization (1.57×). Compared with existing work, TB-STC improves the Energy-Delay Product (EDP) by an average of 3.82× and offers an enhanced accuracy-EDP Pareto frontier across various sparse DL models.

I. Introduction

Deep Learning (DL), from Convolutional Neural Networks (CNNs) to transformer-based Large Language Models (LLMs), has achieved remarkable success across various domains [20], [31], [63], [64]. This success is accompanied by the exponential growth in computation and model size by several hundred times per year, which has far outpaced the advancements in hardware's capability (about 2-3× per two years) [13], [66]. As a result, the surge in computational and memory demands has led to a significant increase in latency and energy consumption for the inference of DL models [20], [34], [63], [77].

The sparsification (*e.g.*, weight pruning [1], [19], [42] and sparse attention [50], [68]) represents a significant approach to reducing latency and energy consumption [81]. However, it is non-trivial to identify a good trade-off between model accuracy and hardware efficiency. Existing sparsity patterns are broadly classified into the Unstructured Sparsity (US) and Structured Sparsity (SS) patterns. On the one hand, the US pattern provides good accuracy but introduces significant hardware complexity overhead. For example, the US-based architecture introduces the area overhead by over 80% handling randomly distributed non-zero elements [57]. On the other hand, existing work has sought to mitigate the hardware complexity overhead through the N:M structured sparsity [11], [25], [27], [37], [70], [83], which is to retain at most N elements from every M elements (the remaining elements are zero) of the reduction dimension[1]. However, existing SS patterns still introduce considerable accuracy loss, *e.g.*, >6% average accuracy drop using the 50% structured sparsity on OPT-6.7B [79] and Llama2-7B [41], [63], due to the following challenges:

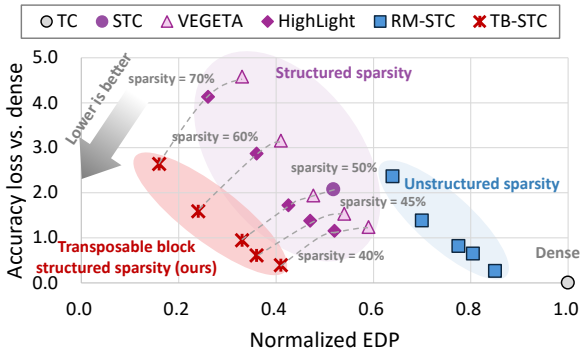


Fig. 1. Compared with other work, TB-STC offers an enhanced accuracy-EDP Pareto frontier (results from BERT [8] model on sst-2 [55] dataset).

Challenge-1*: Existing N:M structured sparsity has limited mask-space with only one-dimensional sparsity, leading to considerable accuracy loss.** Our key insight is that the weight matrices of the forward and backward pass are ***transposed to each other during DL training. Specifically, sparse training [26], [38], [62], [75] yields superior accuracy compared to the fine-tuning approach [14], [80]. During sparse training, the sparse pattern is not fixed during the forward

[1]To illustrate the dimensions of the input matrices (*e.g.*, $\mathbb{A}$ and $\mathbb{B}$), we refer to the dimensions that remain in the resulting matrix (*e.g.*, $\mathbb{D}$) as independent dimension, and another as reduction dimension (as shown in Fig. 3(a)).

2378-203X/25/$31.00 ©2025 IEEE
DOI 10.1109/HPCA61900.2025.00075

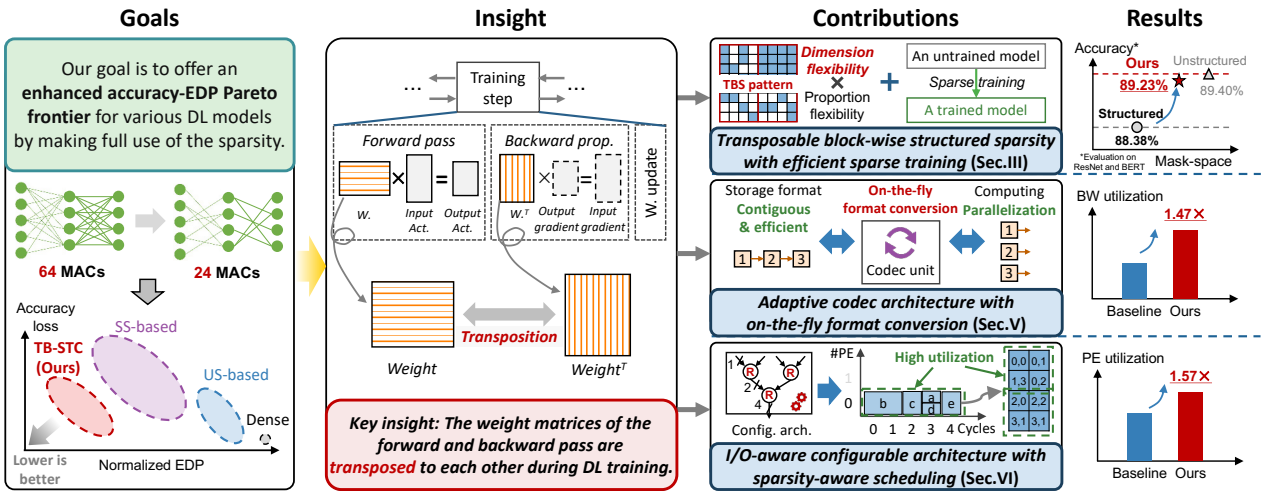


Fig. 2. Overview of TB-STC, a Transposable Block-wise N:M Sparse Tensor Core, which is to fully unleash the potential of the proposed TBS pattern for an enhanced accuracy-EDP (Energy-Delay Product) Pareto frontier under various sparse DL workloads.

and backward stages and often changes in both the row and column dimensions [1], [82]. Therefore, sparse training encompasses a vast mask-space (*i.e.*, sparse representation space). The transposable nature between the two stages results in sparse training exhibiting sparsity in both dimensions. In contrast, existing SS patterns only explore one-dimensional sparsity. Specifically, current N:M sparsity considers only the SS patterns in the reduction dimension (*i.e.*, row-wise), but not the independent dimension (*i.e.*, column-wise) simultaneously.

To fully exploit the transposition property of DL training, this paper proposes *Transposable Block-wise Structured Sparsity* (**TBS**), a novel sparsity pattern that considers both the reduction and independent dimensions. By fully utilizing this transposition property, TBS provides a vast mask-space and obtains a structured sparsity pattern that is closer to the unstructured sparsity. As shown in Fig. 4(b), the TBS pattern is much more similar to the US pattern. Motivated by these observations, we propose **TB-STC**, a Transposable Block-wise N:M Sparse Tensor Core. TB-STC is to fully unleash the potential of the proposed TBS pattern for an enhanced accuracy-EDP (Energy-Delay Product) Pareto frontier under various sparse DL workloads (Fig. 1). In particular, TB-STC aims at improving the bandwidth and computational utilization at the micro-architecture level to tackle the following challenges:

***Challenge-2*: Existing storage formats introduce non-contiguous and redundant memory accesses for the TBS pattern, resulting in low bandwidth utilization.** On the one hand, the single-dimensional compression (SDC) [37], [42] format compresses the sparse matrix according to the maximum number of non-zero elements in each row. However, the TBS pattern introduces N:M sparsity in the independent direction (*i.e.*, row-wise), whereby the number of non-zero elements in each row may not be identical, leading to a considerable amount of redundant memory access overhead. On the other hand, the compressed sparse row (CSR) [3] format stores the non-zero elements and corresponding sparse indices with minimal redundancy, but introduces non-continuous memory accesses. These storage formats result in a memory bandwidth of a mere 38.2% when processing the TBS pattern.

***Challenge-3*: The sparsity degree differences within and between the blocks cause workload imbalance, leading to low computation utilization.** Existing sparse acceleration studies focus only on the one-dimension workload balancing (*e.g.*, row-wise reordering [27], [56]), leaving no support for both the reduction and independent dimensions. On the one hand, *inter-PE* (processing elements) imbalance occurs as computational tasks are often more intensive for blocks with a higher concentration of non-zero entries, resulting in some PEs being overburdened while others are underutilized. On the other hand, *intra-PE* imbalance occurs as the non-zero distribution differs in the reduction and independent dimensions. Results show that the workload imbalance leads to a low computation utilization of 45.50%.

To address the above challenges, this paper makes the following contributions:

1) We propose the **transposable block-wise structured sparsity** pattern with an efficient end-to-end sparse training method for *Challenge-1*. TBS improves the accuracy by up to 2.58% compared with existing SS pattern studies, thanks to its vast mask-space. To the best of our knowledge, we are the first to explore N:M sparsity in both reduction and independent dimensions.
2) We propose an **adaptive codec architecture** to achieve *on-the-fly adaptive format conversion* with a *dual-dimensional compression format* for *Challenge-2*. Ensuring continuous memory access while removing redundancy in different dimensions, TB-STC boosts the bandwidth utilization by 1.47×.

3) We propose an **I/O-aware configurable architecture** with *hierarchical sparsity-aware scheduling* for *Challenge-3*. TB-STC leverages the intrinsic properties of TBS with dataflow-hardware co-design at both inter- and intra-block levels. Compared with direct mapping, TB-STC enhances the computing utilization by 1.57×.

We evaluate TB-STC on a variety of DL models (*e.g.*, ResNet, BERT, and OPT-6.7B). Experimental results show that, TB-STC improves the Energy-Delay Product (EDP) by an average of 3.82×, and offers an enhanced accuracy-EDP Pareto frontier across various sparse DL models.

II. Preliminary and Motivation

A. The N:M Sparsity

Sparse matrix-matrix multiplication (SpMM, $\mathbb{D} = \mathbb{A} \times \mathbb{B} + \mathbb{C}$), which is a fundamental operator in sparse computation, is widely used in many sparse neural networks [54], [61], [65]. In SpMM, a sparse matrix $\mathbb{A}$ is multiplied by a dense matrix $\mathbb{B}$ and added by a matrix $\mathbb{C}$ to generate a dense matrix $\mathbb{D}$.

The N:M sparsity (*i.e.*, structured sparsity) [4], [11], [25], [27], [70], [83] achieves a better trade-off in terms of hardware efficiency compared to US [24], [49], [56], [57], [78], due to the consideration of both sparsity and hardware overhead. To better explain the properties of N:M sparsity, we make a few statements here [27].

- **Sparsity pattern.** It refers to the position and number of non-zero elements in the sparse matrix. For example, the tile-wise N:M sparsity pattern is that at most N non-zero elements out of every M elements (Fig. 4(a)).
- **Sparsity dimension.** It refers to the dimension that implements the N:M sparsity. It should be noted that this concept does not exist in the US pattern. For example, in the sparse matrix of Fig. 3(b), the sparsity dimension is the reduction dimension (*e.g.*, row-wise in here).
- **Sparsity degree.** It refers to the proportion of zero elements in the whole sparse matrix. Sometimes, we also use density degree to represent sparsity degree (density degree $= 1-$ sparsity degree).

As shown in Fig. 4(a), tile-wise N:M sparsity (TS) [4], [83] is the first proposed N:M sparsity pattern, which is currently applied in the NVIDIA Sparse Tensor Core [6], [47]. However, TS is challenging to explore more flexible sparsity due to the limited sparsity degrees. The granularity of row-wise N:M sparsity (RS) is finer than TS, so it has a larger representation space and expresses more sparsity degrees. The typical work includes VEGETA (RS-V) [27] and HighLight (RS-H) [70]. VEGETA explores the flexibility between different rows, and HighLight utilizes a hierarchical sparsity ratio to achieve multiple sparsity degrees. US is the most flexible sparsity pattern, achieving element-wise sparsity. However, due to the random and irregular data distribution, it is challenging in practice to achieve performance improvements that correspond to the level of sparsity. Therefore, the US pattern is not the focus of this paper.

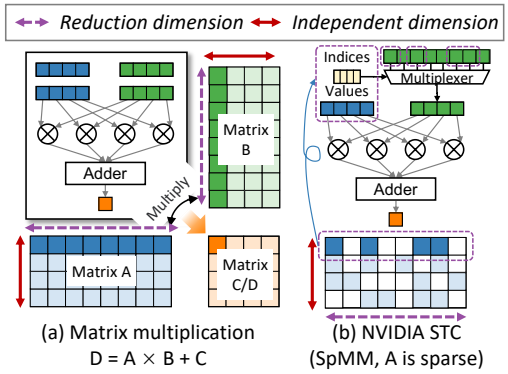


Fig. 3. (a) NVIDIA Tensor Core computes matrix multiplication $\mathbb{D} = \mathbb{A} \times \mathbb{B} + \mathbb{C}$ in the form of inner products. (b) NVIDIA Sparse Tensor Core (STC) supports the SpMM operator by adding the multiplexer.

B. NVIDIA Sparse Tensor Core Hardware Architecture

NVIDIA Tensor Core [45]–[47] is a vital hardware architecture to accelerate matrix multiplication in modern deep learning. As shown in Fig. 3(a), Tensor Core computes matrix multiplication in the form of inner products, and we have simplified some details for ease of illustration. NVIDIA introduced Sparse Tensor Core (STC) in Ampere and later architectures [45], [46] to efficiently support N:M sparsity (2:4 sparsity in particular). As shown in Fig. 3(b), STC achieves up to 2× performance improvement by only adding minimal hardware overhead (*i.e.*, the multiplexer). N:M sparsity has become one of the most critical factors for NVIDIA GPU to provide higher computing performance [45].

III. Efficient Sparse Training with Transposable Block-wise N:M Sparsity

This section introduces the TBS pattern and systematically analyzes the representation space of the existing N:M sparsity patterns (Sec. III-A). Then, we introduce end-to-end sparse training and sparsification methods with TBS pattern on modern DL models (Sec. III-B).

A. Transposable Block-wise N:M Sparsity Pattern

1) Overview of TBS: Although N:M sparsity patterns have become a promising method to improve hardware utilization, the impact on model accuracy still cannot be ignored. Existing N:M sparsity patterns primarily explore flexibility in the reduction dimension and neglect freedom in other dimensions, which fundamentally contributes to their accuracy issues.

We propose a novel transposable block-wise N:M sparsity (TBS) pattern, which has a larger representation space (Fig. 4(a)). Our TBS pattern explores sparsity by both block and dimension based on the following two observations: First, because different parts of a layer show different sparse distributions, the block-wise sparsity has finer granularity than the tile-wise and row-wise. Concretely, we split the sparse matrix into blocks of size $M \times M$. Each block is still the N:M sparsity pattern, but each block can select a different N. Here, N is a factor of M (*e.g.*, $M = 8, N \in 0, 1, 2, 4, 8$). Second, each

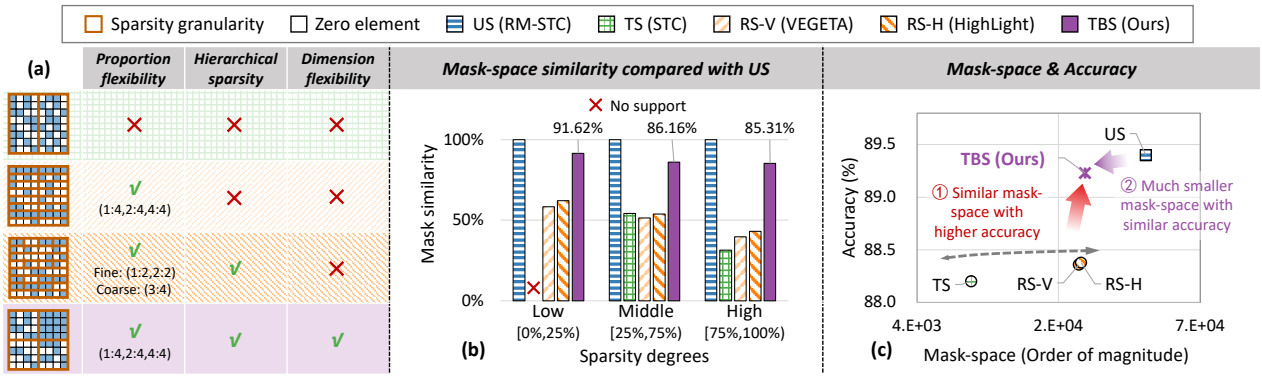


Fig. 4. (a) Comparison between existing N:M sparsity patterns. (b) The TBS pattern is much more similar to the US than other N:M sparsity patterns (Results from ResNet-50 model). (c) The relationship between model accuracy and mask-space on the ResNet and BERT models.

block can select non-zero elements not only in the reduction dimension, but also in the independent dimension, improving the representation space and model accuracy.

2) Representation Space Analysis: We introduce a new measure, called Mask-Space (MS), which is the concept of the representation space [2]. We define MS as follows: For a matrix of a certain size, the number of possible masks for a given sparsity pattern with the same sparsity granularity. We selected the following representative N:M sparsity patterns:

- *Tile-wise N:M sparsity (TS) pattern.* It means each tile with M elements has at most N non-zeros [42]. There are some variants of TS with more constraints [11], [25].
- *Row-wise N:M sparsity (RS) pattern.* VEGETA [27] (*RS-V*) allows different N to be used between different rows. HighLight [70] (*RS-H*) utilizes a hierarchical sparsity ratio to achieve multiple sparsity degrees.

Here, we assume that the matrix size is $X \times Y$ (Y is the size of the reduction dimension), and C_p^q denotes the combination $\frac{p!}{q!(p-q)!}$. M is the sparsity granularity, $k = \log_2 M$. The MS calculation expressions for the above N:M sparsity pattern are as follows:

$$MS_{TS} = \sum_{i=0}^{i \leqslant k} C_M^{2^i \frac{X \cdot Y}{M}} \tag{1}$$

$$MS_{RS-V} = \left[\sum_{i=0}^{i \leqslant k} C_M^{2^i \frac{Y}{M}} \right]^X \tag{2}$$

$$MS_{RS-H} = \sum_{i=M}^{i<2 \cdot M} \left[\left(C_i^M \times C_{\frac{M}{2}}^{M} \right)^{\frac{X \cdot Y}{i \cdot M}} + 2 \cdot \left(C_i^M \right)^{\frac{X \cdot Y}{i \cdot M}} \right] \tag{3}$$

$$MS_{TBS} = \left[\sum_{i=0}^{i \leqslant k} 2 \cdot C_M^{2^i M} \right]^{\frac{X \cdot Y}{M^2}} \tag{4}$$

[2]We refer to Mask-Diversity (MD) proposed in NM-T [25], which only considers the possible number of masks with the same sparsity. So, the MD is unable to accurately describe the difference of sparsity patterns at different sparsity degrees.

As shown in Fig. 4(b), we also show the mask similarity of different N:M sparsity patterns with US. We find that the TBS has 85.31%∼91.62% mask similarity with the US, far exceeding other N:M sparsity patterns. We further analyze the relationship between the MS and the model accuracy. As shown in Fig. 4(c), we choose the typical $X = Y$ and $M = 8$ to compare MS between different N:M sparsity patterns. First, compared with the existing N:M sparsity patterns, TBS achieves higher accuracy under a similar MS. TBS capitalizes on the potential offered by dimensional flexibility and significantly improves model accuracy, approaching that of the US pattern. Second, compared with the US pattern, TBS achieves similar accuracy in a much smaller MS.

B. Efficient Sparse Training Method

1) End-to-end Training Flow: Unlike existing pruning methods fine-tuning from pre-trained models, we train a sparse model from scratch. We find that applying the N:M sparsity pattern with fixed dimensions leads to losing much information. It is because not all non-zero elements are evenly distributed along a certain dimension, but in all dimensions. That is why our TBS pattern can retain more important elements and improve the model's accuracy.

Our key idea is to use the learnable mask to process the original weights and obtain the masked weights, and these weights are as close as possible after training. In each epoch, the main difference between sparse training and dense training lies in both forward pass and backward propagation. In the forward pass, we first obtain the threshold on the entire weight according to the target sparsity. Then, we divide the entire weight into several blocks and generate a number of masks that conform to TBS and are closest to the unstructured mask. In the backward propagation, the model will calculate the gradient of the input activation values under the guidance of the TBS mask.

2) Sparsification of TBS: Essentially, our approach centers on identifying the N:M pruning pattern that closely aligns with the unstructured pruning pattern, comprising three key steps. The pseudo-code of the approach above is shown in

Algorithm 1: TBS Sparsification

Input:
- $Model$: the dense model to be pruned.
- t_s: the target sparsity degree.
- M: the size of a sparse block.
- $N_{candidate} = \{N_1, N_2, \cdots, N_K\}$: candidate numbers of non-zero elements for each sparse block.

Output: the TBS pruning pattern.

/* Step 1: Unstructured Pruning */

Prune $Model$ to sparsity s_t in an unstructured manner

Divide weight matrices into $M \times M$ patches $\mathcal{P}$

for p *in* $\mathcal{P}$ **do**

  /* Step 2: Determine N */

  Get the binary unstructured pruning pattern $\mathbb{D}_{p,un}$

  Calculate the sparsity degree s_p of $\mathbb{D}_{p,un}$

  $N_p = \arg\min_{N_i} |N_i/M - s_p|$

  /* Step 3: Determine Pruning Direction */

  Get pruning pattern $\mathbb{D}_{p,1}$ in the reduction dimension by retaining elements with top-N_p absolute values in each block

  Get TBS pattern $\mathbb{D}_{p,2}$ in the independent dimension

  $\mathbb{D}_p = \arg\min_{\mathbb{D}_{p,nm} \in \{\mathbb{D}_{p,1}, \mathbb{D}_{p,2}\}} \mathcal{L}_1(\mathbb{D}_{p,nm}, \mathbb{D}_{p,un})$

Return the TBS pruning pattern.

Alg. 1. First, we apply unstructured pruning with the target sparsity degree t_s to acquire the unstructured pruning pattern $\mathbb{D}_{un}$. Second, we divide the weight matrices of each layer into $M \times M$ patches P and determine the optimal number of non-zero elements N_p from $N_{candidate}$ for each patch p. This is achieved by minimizing the sparsity difference between N:M pruning and unstructured pruning, ensuring the overall sparsity meets the predetermined target. Third, we select the pruning direction (reduction or independent dimension) and get the N:M pruning pattern $\mathbb{D}_p$. Specifically, we retain the elements with the top-N_p absolute values for each block. Calculations are carried out in both directions, resulting in two different pruning patterns $\mathbb{D}_{p,1}$ and $\mathbb{D}_{p,2}$. The pattern closest to the unstructured pruning pattern $\mathbb{D}_{p,un}$ in terms of $\mathcal{L}_1$ distance is chosen.

Note: It is important to clarify that *the sparsity pattern is orthogonal to the pruning criteria* [1]. Typical pruning criteria include not only magnitude [17], [19], [84], but also gradient [59], [72], Hessian [12], [32], and so on [33]. As verified in our experiments (Table II), TBS outperforms other N:M sparsity patterns in terms of accuracy across different pruning criteria.

IV. Overview of Transposable Block-wise N:M Sparse Tensor Core Architecture

Tensor Core is a vital hardware unit on NVIDIA GPUs that accelerates matrix multiplication computations. Since its specific hardware architectural details are not publicly available, we refer to the Tensor Core architecture in related work [24], [42], [78], as shown in Fig. 5(a). To support the TBS pattern, TB-STC is modified based on the original Tensor Core architecture with lightweight hardware units (Fig. 5(b)), including multiple Diverse Vector PEs, an adaptive codec unit, and a matrix $\mathbb{B}$ distribution (MBD) unit. The codec unit processes the corresponding matrix and sends the matrix' values and indices to DVPEs and the MBD unit, respectively. The MBD unit then selects and rearranges the matrix $\mathbb{B}$ elements according to the sparse indices. The final result is written back to the register files or on-chip memory via the codec unit.

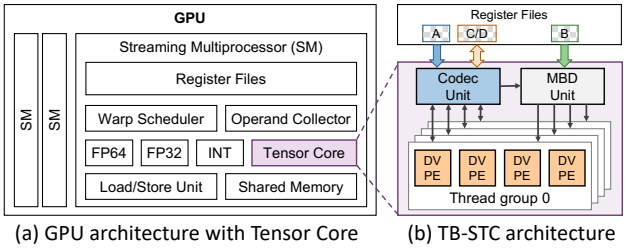
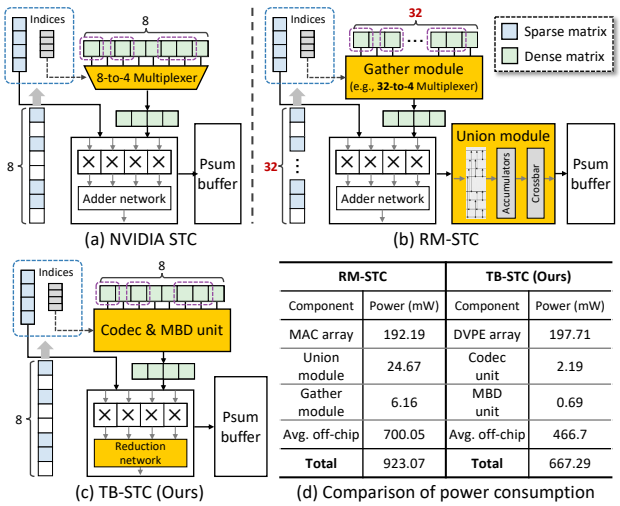


Fig. 5. (a) Modern NVIDIA GPU architecture with Tensor Core [24], [42], [78]. (b) TB-STC hardware architecture, which is modified based on Tensor Core, mainly includes a diverse VPE (DVPE) array, a codec unit, and a Matrix $\mathbb{B}$ Distribution (MBD) unit.



RM-STC		TB-STC (Ours)	
Component	Power (mW)	Component	Power (mW)
MAC array	192.19	DVPE array	197.71
Union module	24.67	Codec unit	2.19
Gather module	6.16	MBD unit	0.69
Avg. off-chip	700.05	Avg. off-chip	466.7
Total	923.07	**Total**	667.29

Fig. 6. Datapath comparison of (a) NVIDIA STC, (b) RM-STC, and (c) TB-STC. The yellow components represent the added modules to support sparsity. (d) The comparison of power consumption between RM-STC and TB-STC.

As shown in Fig. 6(a)-(c), we compare the differences in datapaths on a PE among NVIDIA STC, RM-STC, and TB-STC. STC adds multiplexers on the basis of TC to support the TS pattern, whose additional overhead is very small. RM-STC adds complex hardware modules (including gather and union modules) to support the US pattern, whose irregularity greatly burdens the hardware (Fig. 6(d)). TB-STC efficiently supports a more flexible TBS pattern by introducing acceptable hardware modules. When integrated on the A100 GPU, it only requires 1.57% additional area, which is less than RM-STC (about 1.8%).

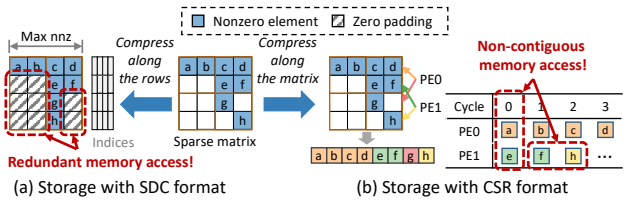


Fig. 7. The existing sparse storage formats face low bandwidth utilization when expressing TBS. (a) The single-dimensional compression (SDC) format compresses rows aligned, resulting in a large redundant memory access overhead. (b) The widely used CSR format has little redundant data but faces the problem of non-continuous memory access.

In Sec. V, we propose an adaptive codec architecture to improve bandwidth utilization. It includes an efficient storage format and a codec unit to implement on-chip low-overhead format conversion. In Sec. VI, we propose an I/O-aware configurable architecture with a hierarchical sparsity-aware scheduling strategy to enhance computation utilization. We exploit the inherent fine-grained balance characteristic of TBS to achieve workload balancing.

V. Adaptive Codec Architecture with Dual-dimensional Compression Format

Existing sparse storage formats face low bandwidth utilization because they cannot effectively express our proposed TBS pattern, which has different dimensions (*e.g.*, row/column-wise) for different sparse blocks. As shown in Fig. 7(a), the SDC format compresses the sparse matrix according to the maximum number of nonzero elements in each row through padding invalid elements (*e.g.*, zero) to ensure regular memory access. However, the irregular distribution of TBS leads to a large amount of redundant memory access overhead (>61.54% on average). As shown in Fig. 7(b), the CSR format only stores non-zero elements and corresponding sparse indices with little redundant memory access overhead. However, the CSR format faces the problem of non-continuous memory access, and this irregular memory access behavior will reduce memory bandwidth utilization (<38.2%).

To efficiently support the TBS pattern, we propose an *adaptive codec architecture* to achieve on-the-fly format conversion to take advantage of both the storage format and the computation format (Sec. V-B). Based on this idea, we first propose a *dual-dimensional compression (DDC)* method (Sec. V-A), the key idea is to *represent the sparsity dimension and sparsity degree through a hierarchical approach*.

A. Dual-dimension Compression Format

DDC format stores the sparse matrix in a block-wise fashion, including inter-block and intra-block storage (Fig. 8(a)). For inter-block storage, we record the information of each matrix block using an `Info.` table, whose size is the number of matrix blocks × 16 bits. We use the 1-bit `Sparsity dim.` to denote the actual N:M sparsity dimension of the matrix block (*i.e.*, row/column-wise), and the 3-bit `Sparsity ratio` to denote the N in the sparsity ratio (*i.e.*, N:M). The 12-bit `Element offset` represents the storage address

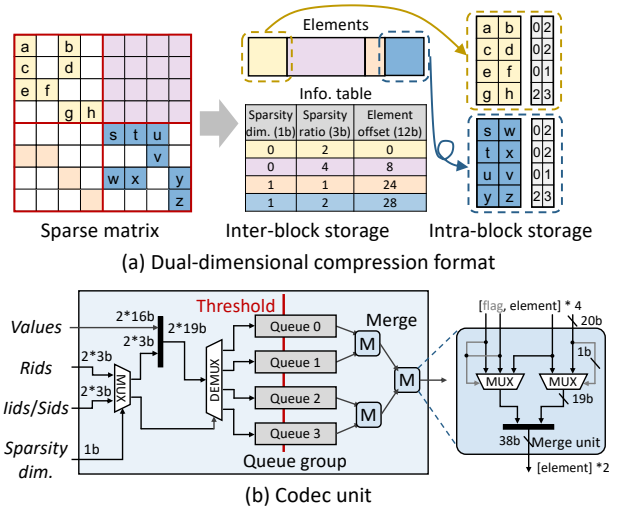


Fig. 8. (a) Storage format design for TBS. We divide the sparse matrix storage into two parts: inter-block storage and intra-block storage. (b) The codec unit implements the conversion between storage and computation formats for the sparse matrix.

offset of the first element in the matrix block. For intra-block storage, we store the sparse matrix blocks according to the sparsity dimension. For example, Fig. 8(a) shows the row-wise and column-wise 2:4 sparse matrix block in yellow and blue, respectively. Therefore, we store them in a compressed format according to their respective sparsity dimension.

B. Adaptive Codec Unit

We design an adaptive codec unit for the PE array to take advantage of both the storage and computation formats, as shown in Fig. 8(b). The codec unit is to implement the conversion between the storage and computation format. The codec unit mainly consists of a group of queues, a merger network, and several multiplexers. The queue group stores the input elements according to the indices and then outputs the processed elements when their size exceeds a given threshold. The merger network handles the conflicts during the output process, and merges the remaining elements at the final timestep.

Fig. 9(a) shows the N:M sparsity with reduction dimensions, which requires no format conversion. Fig. 9(b) illustrates the difference between the storage and computation format for the N:M sparsity with independent dimensions:

- The storage format: compress the non-zero values along the independent dimension (*i.e.*, column-wise) to minimize the storage overhead.
- The computation format: compress the non-zero values along the reduction dimension (*i.e.*, row-wise) to maximize memory access efficiency.

Fig. 9(c) further shows an illustrative example of the format conversion (from storage to computation) for 2:4 sparsity with independent dimension in Fig. 9(b). Firstly, the codec unit utilizes the *indices of reduction dimension* (`Rid`) and

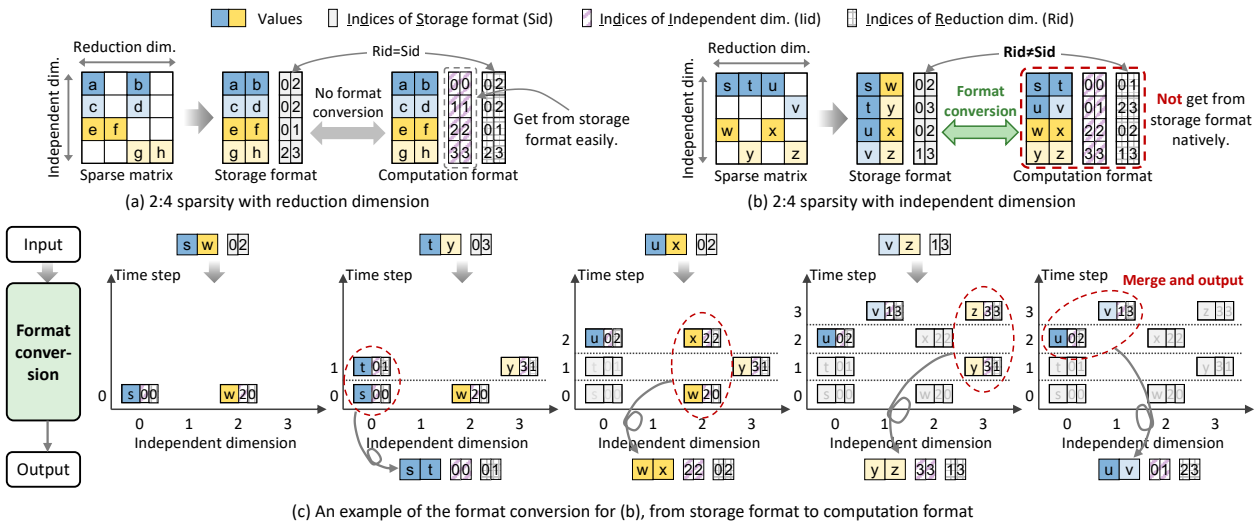


Fig. 9. Format conversion approach for TBS. (a) The N:M sparsity with reduction dimension does not need format conversion. (b) The N:M sparsity with independent dimension needs format conversion. (c) There is an example of the format conversion approach using the sparse matrix of (b). We show the conversion from storage format to computation format in detail.

the *indices of independent dimension* (`Iid`) to identify the position of each value in the original matrix block. Then, the codec unit takes two elements in the storage format as input (including the value and its `Sid`) at each timestep. The format conversion is mainly implemented by a queue group that stores the input elements according to the `Sid`. Then, the elements in the computation format output from the queue when its size exceeds a threshold (here is 2). Thus, the format conversion outputs "s&t", "w&x", and "y&z" at the timestep 1, 2, and 3, respectively. In the last timestep, the codec unit combines the remaining elements and outputs them to the PE array.

We evaluate the bandwidth utilization under different numbers of DVPEs, with the 64 GB/s peak off-chip memory bandwidth. With the strong support of the adaptive codec architecture, we achieve an average improvement of 1.47× in memory bandwidth utilization compared to other methods.

VI. I/O-aware Configurable Architecture with Hierarchical Sparsity-Aware Scheduling

To solve the challenge of low computing utilization caused by workload imbalance, we propose an *I/O-aware configurable architecture* based on the TBS for the *hierarchical sparsity-aware scheduling* strategy. We exploit the inherent characteristics of TBS with the dataflow-hardware co-design from inter-block and intra-block, respectively.

A. I/O-aware Configurable Architecture

1) Configurable Reduction Network Architecture: Vector-based PE (VPE) [3] is the basic component of Tensor Core [42], using the SIMD fashion to drive more data with one instruction for highly parallel computation. Fig. 10(a) shows the architecture of Diverse VPE (DVPE). Each reduction node (denoted as R in the figure) provides configurable functions

[3]Also called as Dot Product Unit [78].

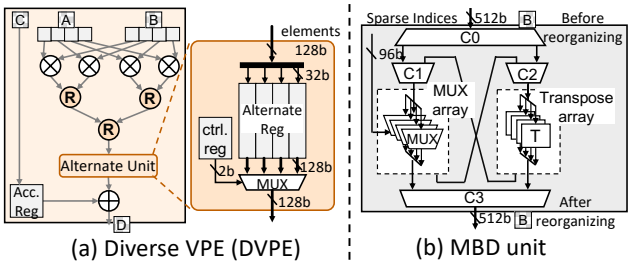


Fig. 10. (a) The alternate unit of DVPE balances the number of output elements by buffering. (b) The MBD unit consists of the MUX array and transpose array to efficiently index valid elements in matrix $\mathbb{B}$.

of accumulation or transmission to support different ratios of N:M sparsity. However, it still requires a large bandwidth in terms of output. For example, assuming that each multiplier in the DVPE outputs the final result (*i.e.*, all reduction nodes perform the transmit function), the DVPE's output requires 4 × bandwidth and adders to handle the accumulation. Our key idea is to *take advantage of the correlation between different matrix blocks and buffer the remaining elements, which will merge with the output result next time.* Based on this idea, we design the alternate unit in DVPE, which can effectively alleviate the impact of workload imbalance on the output.

2) Configurable Input Selector: The Matrix $\mathbb{B}$ Distribution (MBD) Unit supports both row-major and column-major storage formats by configurable MUX array and transpose array (Fig. 10(b)). The MUX array selects valid elements from matrix $\mathbb{B}$ under the sparse indices of matrix $\mathbb{A}$. The transpose array transforms the order of rows and columns for the matrix tile to facilitate DVPE computation. Multiplexers C0, C1, and C2 control the execution order of the MUX array and the transpose array according to the storage type. Finally, C3 outputs the reorganized data from matrix $\mathbb{B}$.

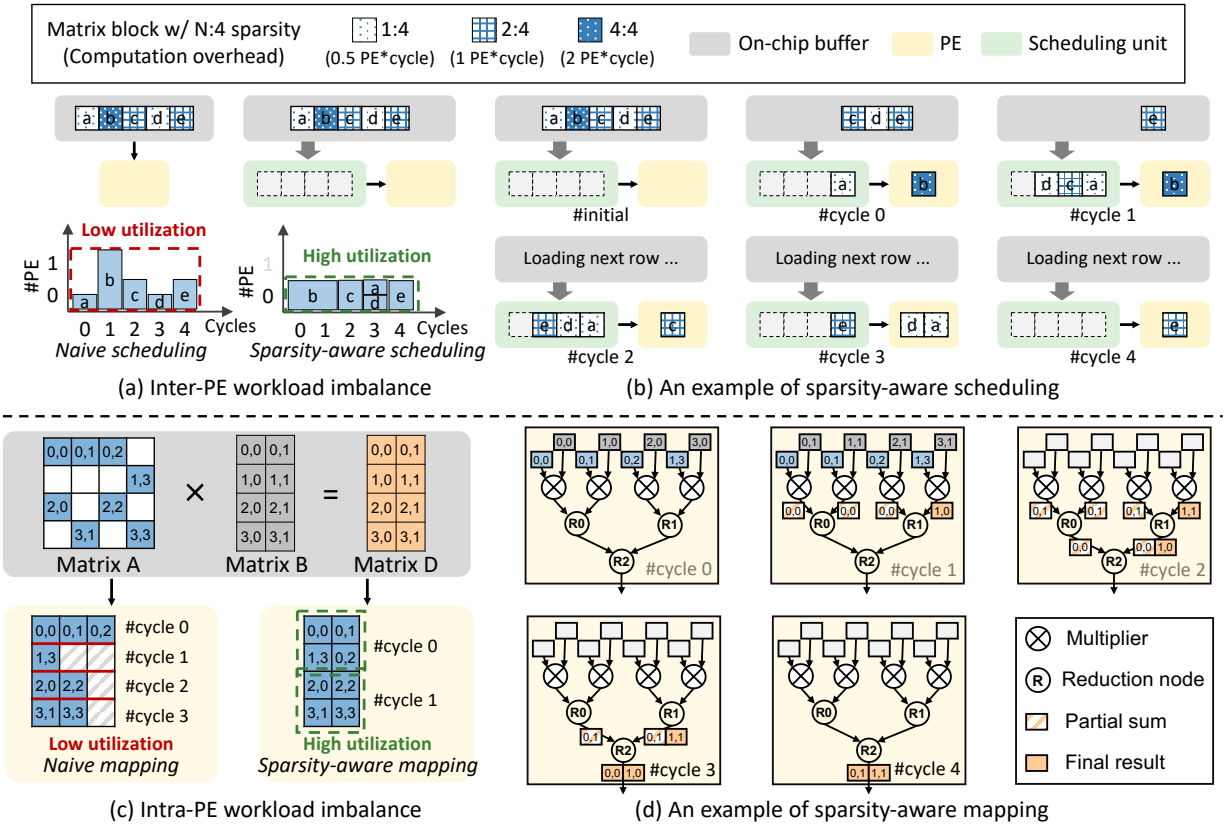


Fig. 11. (a) For the inter-PE workload imbalance problem, we propose an inter-block sparsity-aware scheduling method and use (b) an example to explain it. (c) For the intra-PE workload imbalance problem, we propose an intra-block sparsity-aware mapping method and use (d) as an example to explain it.

B. Hierarchical Sparsity-Aware Scheduling

1) Inter-block Scheduling: We propose an inter-block sparsity-aware scheduling method, which alleviates the problem of low PE utilization, especially when there is a significant difference in sparsity degrees between matrix blocks. Our key idea is *to use locality to balance the computational cost between consecutive matrix blocks to reduce the overall computational cost.* As shown in Fig. 11(a), if we directly map the computational workload (*i.e.*, matrix blocks a $\sim$ e) to PEs, we need a total of 10 PE$\times$cycles of computational overhead, and the PE utilization is only 50%. Therefore, we design a scheduling unit between the on-chip buffer and PEs, which sends the workload (*i.e.*, matrix block) to the PE according to the current workload sparsity degree. Compared with the naive scheduling method, the inter-block sparsity-aware scheduling method only needs 5 PE$\times$cycles of computational overhead.

As shown in Fig. 11(b), we illustrate the inter-block sparsity-aware scheduling process. The scheduling unit loads at most two matrix blocks from the on-chip buffer per cycle and then decides which matrix block to send to the PE based on the workload information. In cycle 0, the scheduling unit stores a, and the PE starts computing b. In cycle 1, since matrix block b needs two cycles to finish, we store both c and d in the scheduling unit. In cycle 2, the scheduling unit sends the matrix block c to the PE and stores the e. In cycle 3, the scheduling unit merges and sends the matrix blocks a and d to the PE. In cycle 4, the PE finishes the computation of the last matrix block e.

2) Intra-block Scheduling: We propose an inter-block sparsity-aware mapping method, which avoids the intra-PE workload imbalance caused by the independent dimensions N:M sparsity. Our key idea is *to exploit the balance property for the total number of non-zero elements in the matrix block (i.e., the total number is always an integer multiple of M).* As shown in Fig. 11(c), for naive mapping, the PE needs to take four pipeline cycles and has severe under-utilization. Therefore, we combine the four consecutive elements located in different rows and map them to PE together. Specifically, since the second row of the sparse matrix has only one element, we compute it together with the last element of the first row to achieve workload balance. The output of different elements is processed by the reduction node and alternate unit inside PE. The intra-block sparsity-aware mapping method can achieve higher PE utilization than the naive mapping method.

As shown in Fig. 11(d), we show that DVPE executes the matrix block computation with independent dimension N:M sparsity, omitting some non-essential details. In cycle

0, DVPE reads the concatenated sparse matrices $\mathbb{A}$ and the corresponding elements of $\mathbb{B}$ onto the input registers. Then, DVPE completes the corresponding multiplication computation in cycle 1, and loads the data for the next timestep. In cycle 2, the reduction node R0 performs the accumulation function to add the partial sum $\mathbb{D}(0,0)$, and the R1 performs the transmit function to by-pass the partial sum $\mathbb{D}(0,0)$ and final result partial sum $\mathbb{D}(1,0)$. In cycle 3, the R2 outputs the final result $\mathbb{D}(0,0)$ and $\mathbb{D}(1,0)$, the former result is accumulate from two partial sum $\mathbb{D}(0,0)$. In cycle 4, the DVPE outputs the remaining results in a pipelined manner.

VII. Evaluation

A. Experiment Setup

1) Evaluation methodology: The TB-STC architecture mainly includes a codec unit, an MBD unit, and 8 DVPE arrays. Each DVPE array has 2×8 DVPEs. Each DVPE consists of 8 FP16 multipliers, an alternate unit, and the corresponding reduction nodes. The MBD unit mainly consists of 16 8-to-1 multiplexers and four 8-to-8 transpose units.

We perform Register Transfer Level (RTL) implementation for each hardware unit and evaluate the area and power overhead. We use Sparseloop [71] to evaluate the power consumption of the DVPE arrays and memory access modules. Sparseloop is a widely used tool for analyzing and modeling the power consumption of sparse accelerators. Since Sparseloop cannot model the codec unit, we use Synopsys Design Compiler [60] to evaluate the power consumption of the codec unit and the area overhead of all logic units. We use the Ramulator [28] and DRAMPower [5] to get the cycle-level evaluation and energy results of DRAM. We use CACTI 7 [43] to obtain the area overhead of the on-chip buffer. For uniform comparison, we scale them to 7nm process technology according to [53], [58], as in previous studies [24], [39], [49]. In addition, we design a cycle-level performance simulator to model the hardware behavior and evaluate execution cycles. For a fair way, we model and evaluate the overhead in the same way for all baselines. We refer to the configuration of these works and keep the same peak performance, on-chip memory capacity, and 64GB/s off-chip memory bandwidth.

2) Baselines: To compare the capabilities of different N:M sparsity patterns, we select three baseline categories and their representative works as our knowledge.

- **Tile-wise N:M sparsity (TS).** It means each tile with M elements having at most N non-zero elements. NVIDIA Sparse Tensor Core (STC) [42] supports 2:4 sparsity with minimal hardware overhead. We also select the Tensor core with dense manner denoted as TC.
- **Row-wise N:M sparsity (RS).** It explores the fine granularity N:M sparsity on different rows. The typical work includes VEGETA [27] and HighLight [70]. HighLight utilizes a hierarchical sparsity ratio to achieve multiple sparsity degrees.
- **Unstructured sparsity (US).** To illustrate the superiority of the TBS pattern, we also make a comparison with unstructured sparsity acceleration. RM-STC [24] is the SOTA work to accelerate training and inference on Tensor Core, exploiting unstructured sparsity.

TABLE I
Comparison of algorithm accuracy on ResNet and BERT.

Models & datasets	ResNet-50	ResNet-18	BERT		Average
	Cifar10	ImageNet	sst-2	mrpc	
Sparsity degree[1]	75%	75%	50%	50%	-
Dense	95.04	89.08	92.32	84.04	90.12
US	94.93	88.15	91.43	83.09	89.40 (-0.00)
TS	94.32	86.37	90.25	81.86	88.20 (-1.20)
RS-V	94.32	86.89	90.37	81.84	88.36 (-1.04)
RS-H	94.79	86.61	90.48	81.62	88.38 (-1.02)
TBS (Ours)	94.91	87.53	91.38	83.09	**89.23 (-0.17)**

[1] TS employs 4:8 sparsity, so its sparsity degree is 50%.

TABLE II
Comparison of algorithm accuracy on OPT-6.7B and Llama2-7B.

Sparsity pattern	OPT-6.7B		Llama2-7B		Average
	Wanda	SparseGPT	Wanda	SparseGPT	
Dense	64.39	64.39	70.15	70.15	67.72
US (50%)	60.21	62.22	67.00	66.80	64.06 (-0.00)
TS (4:8)	56.32	59.54	63.50	63.93	60.82 (-3.24)
RS-V	57.78	59.89	63.61	64.45	61.43 (-2.63)
RS-H	57.81	59.86	63.68	64.57	61.48 (-2.58)
TBS (Ours)	59.68	61.82	65.23	66.88	**63.40 (-0.66)**

3) Algorithm Setup: As described in Sec. III-B, we set $M = 8$, $N_{candidate} = \{0, 1, 2, 4, 8\}$ in this experiment. We evaluate different sparsity patterns on CNN, transformer, and LLMs. For those models that need to be retrained, we use a test dataset independent of the training dataset to evaluate their model accuracy.

CNN models. To evaluate the effectiveness of our proposed method on CNNs, we prune the classical ResNet-50 and ResNet-18 [21] models on the Cifar-10 [29] and ImageNet [30] datasets, respectively. All layers are pruned except the stem layer and the final fully-connected layer.

Transformer model. We conduct experiments using BERT-base [9] across different tasks in the GLUE benchmark [67], namely MRPC [10] and SST-2 [55], with the goal of assessing the effectiveness of our proposed method.

Large Language Models. We also conduct LLM experiments using OPT-6.7B [79] and Llama2-7B [41], [63] across four commonsense reasoning benchmarks, namely Piqa [2], HellaSwag [76], WinoGrande [52], and Arc-e [7]. The LLM's accuracy is obtained by averaging the accuracy across the four datasets mentioned above.

B. Evaluation of Accuracy

1) Accuracy with retraining: For ResNet and BERT models, we retrain the models on different datasets (Table. I). To fairly compare the differences between sparsity patterns, we apply US, TS, RS-V, RS-H, and TBS to the training process

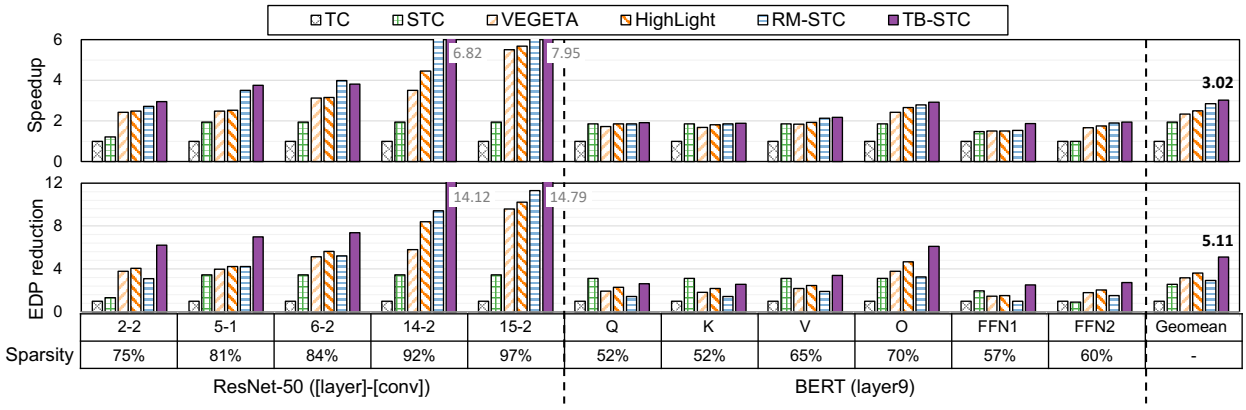


Fig. 12. The comparison of layer-wise speedup and normalized EDP across different sparsity degrees on ResNet-50 and BERT models.

with the same epochs. The accuracy of the TBS pattern is 0.85%∼1.03% higher than other N:M sparsity patterns, and it is also very close to the US (only 0.17% gap).

2) Accuracy with one-shot pruning: For the OPT-6.7B and Llama2-7B models, we evaluate their average accuracy in a one-shot pruning manner due to the large training overhead. We apply different patterns on two typical one-shot pruning methods: Wanda [59] and SparseGPT [12]. Under the same sparsity degree (50%), TBS can improve the average accuracy by 2.58% compared with TS (Table. II). Moreover, TBS narrows the accuracy gap between the SS and US pattern from 2.58%-3.24% down to 0.66%. We need to point out that the sparsity pattern and pruning criteria are orthogonal to each other, and using TBS on more efficient training methods [1], [16], [35], [40] is promising to further improve the accuracy.

C. Evaluation of Hardware Architecture

1) Layer-wise Speedup and EDP: As shown in Fig. 12, we first evaluate the results under different sparsity degrees on the typical layers of the ResNet-50 and BERT models. TB-STC achieves 1.55×, 1.29×, 1.21×, and 1.06× speedup compared with STC, VEGETA, HighLight, and RM-STC, respectively. TB-STC improves the EDP by 1.41× compared with the state-of-the-art structured sparsity work (*i.e.*, HighLight), thanks to a more efficient hardware architecture design and a more advanced sparsity pattern. Compared with the unstructured sparsity work (*i.e.*, RM-STC), TB-STC gains 1.75× EDP improvement, although their speedup is very similar (only 1.06×). This is because unstructured sparsity introduces non-negligible hardware overhead, resulting in reduced energy efficiency. In general, TB-STC outperforms existing works in computing tasks with a wide range of sparsity degrees.

2) End-to-end Speedup and EDP: As shown in Fig. 13, we further evaluate the end-to-end results on three representative models, including ResNet-50, BERT, and OPT-6.7B. Different from the layer-wise evaluation, which evaluates different works at the same sparsity degrees (except for STC, which is 4:8 sparsity). The end-to-end evaluation keeps the same accuracy for all works. Therefore, compared with other structured

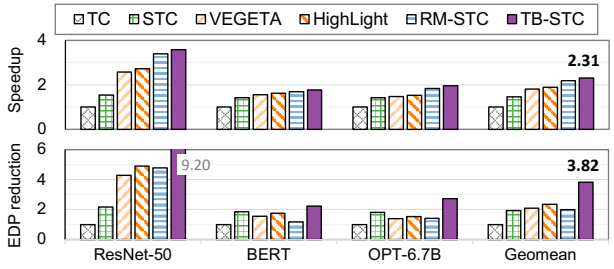


Fig. 13. The comparison of end-to-end speedup and normalized EDP for the inference on ResNet-50, BERT, and OPT-6.7B models.

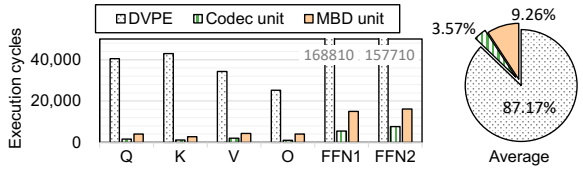


Fig. 14. Execution cycle breakdown.

sparsity works, TB-STC can utilize the potential benefits of a higher sparsity degree. In general, TB-STC improves the speedup by 1.22× and 1.06× and the EDP by 1.62× and 1.92× compared with HighLight and RM-STC.

3) Breakdown of Execution Cycle: As shown in Fig. 14, We select several typical layers from the 9th layer of the BERT model to analyze the execution cycle. The format conversion (implemented on the codec unit) can be hidden within the entire execution pipeline, thus having minimal impact on the execution. The results indicate that the overhead of format conversion accounts for only an average of 3.57% of the overall execution cycle.

4) Hardware Overhead: Finally, we evaluate the hardware overhead and power consumption of TB-STC. Table III shows the detailed area and power breakdown under 1GHz synthesized frequency. Our design has a total hardware area overhead of 1.47 mm^2 and consumes 200.59mW. The DVPE array is

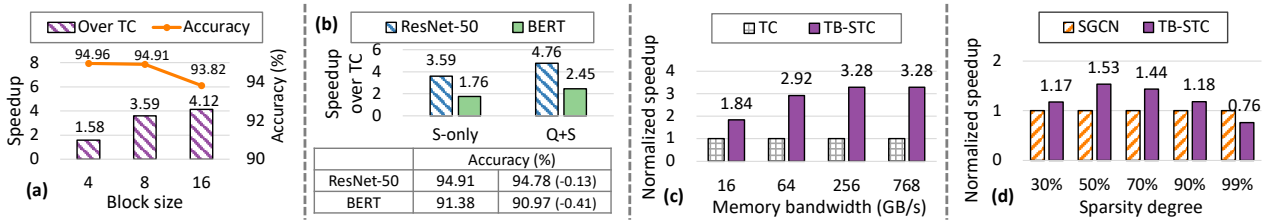


Fig. 15. (a) The effect of block size on the speedup and model accuracy. (b) The effects of quantization on the TBS pruned model. The effect of (c) memory bandwidth and (d) sparsity degree on the performance of TB-STC.

TABLE III
AREA AND POWER BREAKDOWN OF TB-STC.

Component	Area (mm^2)	Breakdown	Power (mW)	Breakdown
DVPE Array	1.43	97.28%	197.71	98.57%
Codec Unit	0.03	2.04%	2.19	1.09%
MBD Unit	0.01	0.68%	0.69	0.34%
Total	1.47	100.00%	200.59	100.00%

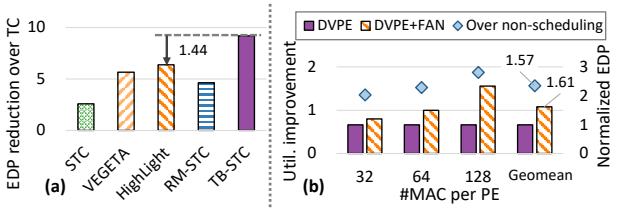


Fig. 16. The ablation study of (a) adaptive codec architecture and (b) I/O-aware configurable architecture.

the main part of the hardware overhead. Compared to GPU tensor cores, TB-STC adds a reduction network (total of 0.08 mm^2 area including alternate unit) within the DVPE array, codec units, and MBD units. TB-STC equals 1/108 of the tensor cores on an NVIDIA A100 GPU [45]. Therefore, TB-STC with equal proportion introduces 12.96 mm^2 additional area overhead (0.12×108=12.96 mm^2), only 1.57% for the 826 mm^2 die area of NVIDIA A100 GPU.

D. Evaluation of Sensitivity

1) Effect of Block Size: As shown in Fig. 15(a), we test the effect of different block sizes on speedup and model accuracy on the ResNet-50 model. We find that as the block size increases, the growth of the speedup result tends to become flat, which is because the granularity of the block size is sufficient for the hardware and can no longer provide more gains. However, the accuracy significantly decreases (94.91% → 93.82%) as the block size increases. Therefore, we select a block size of 8 to take into account these factors.

2) Effect of Quantization: Quantization is another common method for model acceleration, so we further consider the impact of quantization on TB-STC. As shown in Fig. 15(b), we adopt the weight 8-bit quantization method for TBS pruned models (ResNet-50 and BERT). We find that after using quantization (denoted as "Q+S"), TB-STC can achieve further acceleration with 1.33× and 1.39×, and the negative impact on accuracy is almost negligible (accuracy loss 0.13% and 0.41% for ResNet-50 and BERT).

3) Effect of Memory Bandwidth: As shown in Fig. 15(c), we analyze the normalized speedup effect of TB-STC under different memory bandwidths. Our analysis results show that for a memory bandwidth setting of 64GB/s, TB-STC is still limited by memory access when handling tasks with higher sparsity. As memory bandwidth increases, TB-STC can achieve further acceleration. When the bandwidth exceeds 256GB/s, TB-STC no longer accelerates further because it is limited by computation.

4) Effect of Sparsity Degree: As shown in Fig. 15(d), we further evaluate the effect of different sparsity degrees on the performance of TB-STC. We choose the current SOTA SGCN [73] as the baseline, effectively accelerating graph neural networks with high sparsity. For the cases of high sparsity degrees (*e.g.*, 95%), SGCN can perform better and outperform TB-STC. This is because SGCN is specifically optimized for high sparsity, such as maintaining a high ratio of memory bandwidth (256GB/s) to computing performance and complex processing for sparse patterns. TB-STC still has better performance (average 1.32×) for the cases of sparsity degrees ranging from 30% to 90%, which is also the sparsity range of most deep learning models (including CNNs and LLMs [77]).

E. Ablation Studies

1) Adaptive Codec Architecture: To verify the validity of the adaptive codec architecture design, we deploy the TBS pruned ResNet-50 model on different hardware architectures, respectively. As shown in Fig. 16(a), we find that even with the advanced TBS pattern, the performance of other hardware architectures still has a gap over 1.44× with TB-STC. This is because other hardware architectures do not efficiently support the dimensional flexibility existing in TBS, which indicates that the adaptive codec architecture design of TB-STC is very important for the contribution to the final performance.

2) I/O-aware Configurable Architecture: We compare two types of methods, including non-scheduling optimization and adopting the scheduling strategy supported by the existing hardware architecture. As shown in Fig. 16(b), compared with the non-scheduling method, we achieve an average of 1.57× computation utilization improvement due to the full exploitation of the TBS pattern. In addition, the workload

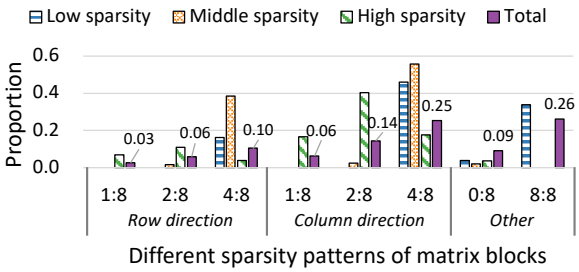


Fig. 17. The distribution of the sparsity pattern at the block level, based on the results of the pruned ResNet-50 model.

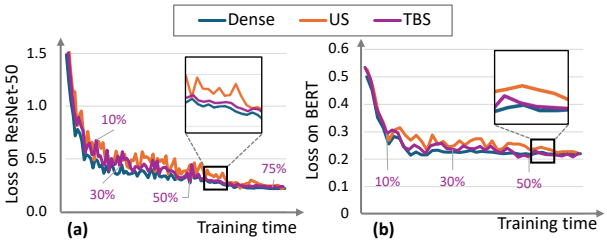


Fig. 18. The loss of different training methods.

balancing designs in existing sparse accelerators [49], [57], [78] do not adequately address the potential execution efficiency challenges of TBS. For instance, SIGMA [49] utilizes the bitmap information of sparse matrices and designs a fine-grained Forwarding Adder Network (FAN) at the element level. FAN suffers from severe hardware inefficiency because the design of FAN does not take advantage of the two-level equalization (inter-block and intra-block) characteristic of TBS. The normalized EDP of "DVPE+FAN" is on average 1.61× larger than our DVPE.

F. Other Discussion

1) Distribution of Sparsity Pattern: In order to further illustrate the reason for the improvement in model accuracy, we analyze the distribution of sparse blocks in the TBS pruned ResNet-50 model. As shown in Fig. 17, we separately select three typical layers with low, medium, and high sparsity degrees, as well as the average distribution (Total) of all layers in the whole model. We find that the distribution of the sparsity pattern at the block level is correlated with the sparsity degree. From the average result, the proportion of the sparse blocks in the row direction is 18.7%, the proportion of the sparse blocks in the column direction is 46.0%, and the proportion of other sparse blocks is 35.3%. This further illustrates that the existing N:M sparse methods are not enough to only explore the sparsity in a single dimension.

2) The convergence situations of different sparse patterns during training: As shown in Fig. 18, we train the ResNet-50 and BERT models using three methods, namely dense training, US training, and TBS training, respectively. The ResNet-50 model is retrained on the Cifar-10 dataset, and BERT is further trained based on the pre-trained model on the mrpc dataset. We also mark the sparsity variation of TBS training in Fig. 18. Although TBS training requires more training time than dense training, but it can still achieve almost the same loss. In addition, since TB-STC can accelerate part of TBS training, the training time for TBS is shorter than that of the US. This is because the search space of the US is larger and thus requires more training overhead.

VIII. Related Works

A. Deep Learning Acceleration

Most work exploits the sparsity of attention by pre-computing. For example, SpAtten [68] approximates sparse attention matrices by low-precision data. DOTA [51] uses low-rank methods to detect connection relationships between inputs at runtime. CTA [69] and ELSA [18] approximate calculation using hashing, but this approximation method may hurt the accuracy in high-sparsity cases. There are also some works to reduce the computational overhead of LLM by quantization. For example, Mokey [74] and Olive [15] address the problem of outliers caused by quantization through pre-processing and hardware design.

B. GPU Kernel of SpMM

Shfl-BW [23] and nmSPARSE [36] attach new pruning algorithms to restrict the sparse patterns and utilize Tensor Core to accelerate SpMM kernel and DNNs with a relatively small accuracy loss. CuSPARSELt [44] leverages the Sparse Tensor Core in NVIDIA Ampere architecture [6], [42], but only supports 2:4 weight sparsity which restricts the flexibility for SpMM. cuSPARSE [48] provided by Nvidia has different algorithms of SpMM and SDDM implementation. ASpT [22] devises an adaptive tiling strategy, which individually processes dense tiling blocks and sparse tiling blocks, to enhance SpMM and SDDMM kernel performance.

IX. Conclusion

In this paper, we propose **TB-STC**, a Transposable Block-wise Structured Sparse Tensor Core that is the first to explore N:M sparsity in both reduction and independent dimensions. First, we propose a novel TBS pattern with an efficient end-to-end sparse training method. Furthermore, TB-STC introduces an adaptive codec architecture for on-the-fly storage format conversion with a higher bandwidth utilization (1.47×), and implements an I/O-aware configurable architecture for sparsity-aware scheduling with a better computational utilization (1.57×). Compared with existing work, TB-STC improves the Energy-Delay Product (EDP) by an average of 3.82× across different sparse workloads, and offers an enhanced accuracy-EDP Pareto frontier for various DL models.

Acknowledgements

This work was sponsored by the National Natural Science Foundation of China (No. 62104128, U21B2031, 62325405), Shanghai Rising-Star Program (No. 24QB2706200), Tsinghua EE Xilinx AI Research Fund, Beijing National Research Center for Information Science and Technology (No. BNR2024TD03001). We thank the insightful discussion with colleagues at Infinigence-AI.

REFERENCES

[1] A. R. Bambhaniya, A. Yazdanbakhsh, S. Subramanian, S.-C. Kao, S. Agrawal, U. Evci, and T. Krishna, "Progressive gradient flow for robust n: M sparsity training in transformers," *arXiv preprint arXiv:2402.04744*, 2024.

[2] Y. Bisk, R. Zellers, J. Gao, Y. Choi *et al.*, "Piqa: Reasoning about physical commonsense in natural language," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 7432–7439.

[3] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson, "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks," in *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, 2009, pp. 233–244.

[4] S. Cao, C. Zhang, Z. Yao, W. Xiao, L. Nie, D. Zhan, Y. Liu, M. Wu, and L. Zhang, "Efficient and effective sparse lstm on fpga with bank-balanced sparsity," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 63–72.

[5] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, "Drampower: Open-source dram power & energy estimation tool," *URL: http://www. drampower. info*, vol. 22, 2012.

[6] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "Nvidia a100 tensor core gpu: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[7] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[10] B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *Third international workshop on paraphrasing (IWP2005)*, 2005.

[11] C. Fang, W. Sun, A. Zhou, and Z. Wang, "Efficient n: M sparse dnn training using algorithm, architecture, and dataflow co-design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[12] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," in *International Conference on Machine Learning*. PMLR, 2023, pp. 10 323–10 337.

[13] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "Ai and memory wall," *IEEE Micro*, 2024.

[14] M. A. Gordon, K. Duh, and N. Andrews, "Compressing bert: Studying the effects of weight pruning on transfer learning," *ACL 2020*, p. 143, 2020.

[15] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[16] S. Guo, F. Wu, L. Zhang, X. Zheng, S. Zhang, F. Chao, Y. Shi, and R. Ji, "Ebft: Effective and block-wise fine-tuning for sparse llms," *arXiv preprint arXiv:2402.12419*, 2024.

[17] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," *Advances in neural information processing systems*, vol. 29, 2016.

[18] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 692–705.

[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[22] C. Hong, A. Sukumaran-Rajam, I. Nisa, K. Singh, and P. Sadayappan, "Adaptive sparse tiling for sparse matrix multiplication," in *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, 2019, pp. 300–314.

[23] G. Huang, H. Li, M. Qin, F. Sun, Y. Ding, and Y. Xie, "Shfl-bw: accelerating deep neural network inference with tensor-core aware weight pruning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1153–1158.

[24] G. Huang, Z. Wang, P.-A. Tsai, C. Zhang, Y. Ding, and Y. Xie, "Rm-stc: Row-merge dataflow inspired gpu sparse tensor core for energy-efficient sparse acceleration," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 338–352.

[25] I. Hubara, B. Chmiel, M. Island, R. Banner, J. Naor, and D. Soudry, "Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks," *Advances in neural information processing systems*, vol. 34, pp. 21 099–21 111, 2021.

[26] A. K. Jaiswal, H. Ma, T. Chen, Y. Ding, and Z. Wang, "Training your sparse neural network better with any mask," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9833–9844.

[27] G. Jeong, S. Damani, A. R. Bambhaniya, E. Qin, C. J. Hughes, S. Subramoney, H. Kim, and T. Krishna, "Vegeta: Vertically-integrated extensions for sparse/dense gemm tile acceleration on cpus," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 259–272.

[28] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[29] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[30] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, no. 2, 2012.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[32] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989.

[33] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv preprint arXiv:1810.02340*, 2018.

[34] J. Li, J. Xu, S. Huang, Y. Chen, W. Li, J. Liu, Y. Lian, J. Pan, L. Ding, H. Zhou *et al.*, "Large language model inference acceleration: A comprehensive hardware perspective," *arXiv preprint arXiv:2410.04466*, 2024.

[35] Y. Li, L. Niu, X. Zhang, K. Liu, J. Zhu, and Z. Kang, "E-sparse: Boosting the large language model inference through entropy-based n: M sparsity," *arXiv preprint arXiv:2310.15929*, 2023.

[36] B. Lin, N. Zheng, L. Wang, S. Cao, L. Ma, Q. Zhang, Y. Zhu, T. Cao, J. Xue, Y. Yang *et al.*, "Efficient gpu kernels for n: M-sparse weights in deep learning," *Proceedings of Machine Learning and Systems*, vol. 5, 2023.

[37] J. Liu, G. Dai, H. Xia, L. Guo, X. Shi, J. Xu, H. Yang, and Y. Wang, "Tstc: Two-level sparsity tensor core enabling both algorithm flexibility and hardware efficiency," in *2023 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2023.

[38] J. Liu, Z. Xu, R. Shi, R. C. Cheung, and H. K. So, "Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers," *arXiv preprint arXiv:2005.06870*, 2020.

[39] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 977–991.

[40] Y. Lu, S. Agrawal, S. Subramanian, O. Rybakov, C. De Sa, and A. Yazdanbakhsh, "Step: Learning n: M structured sparsity masks from scratch with precondition," in *International Conference on Machine Learning*. PMLR, 2023, pp. 22 812–22 824.

[41] meta llama. (2023) Llama-2-7b-hf. [Online]. Available: https://huggingface.co/meta-llama/Llama-2-7b-hf

[42] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," *arXiv preprint arXiv:2104.08378*, 2021.

[43] A. S. Naveen Muralimanohar and V. Srinivas, "Hewlettpackard/cacti," https://github.com/HewlettPackard/cacti Accessed May 22, 2023.

[44] NVIDIA, "cusparselt: A high-performance cuda library for sparse matrix-matrix multiplication," https://developer.nvidia.com/cudnn.

[45] NVIDIA, "Nvidia a100 tensor core gpu architecture," https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

[46] NVIDIA, "Nvidia h100 tensor core gpu architecture," https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper.

[47] NVIDIA, "Nvidia tesla v100 gpu architecture," https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf Accessed Jul 29, 2023.

[48] I. NVIDIA, "The api reference guide for cusparse, the cuda sparse matrix library." https://docs.nvidia.com/cuda/cusparse/index.html Accessed May 22, 2023.

[49] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 58–70.

[50] Y. Qin, Y. Wang, D. Deng, Z. Zhao, X. Yang, L. Liu, S. Wei, Y. Hu, and S. Yin, "Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–14.

[51] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, "Dota: detect and omit weak attentions for scalable transformer acceleration," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 14–26.

[52] K. Sakaguchi, R. L. Bras *et al.*, "Winogrande: An adversarial winograd schema challenge at scale," *Communications of the ACM*, 2021.

[53] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[54] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[55] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[56] L. Song, Y. Chi, A. Sohrabizadeh, Y.-k. Choi, J. Lau, and J. Cong, "Sextans: A streaming accelerator for general-purpose sparse-matrix dense-matrix multiplication," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2022, pp. 65–77.

[57] N. Srivastava, H. Jin, J. Liu, D. Albonesi, and Z. Zhang, "Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 766–780.

[58] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180 nm to 7 nm," *Integration*, vol. 58, pp. 74–81, 2017.

[59] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, "A simple and effective pruning approach for large language models," *arXiv preprint arXiv:2306.11695*, 2023.

[60] I. Synopsys, "Design compiler (synopsys.com)," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html Accessed May 22, 2023.

[61] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[62] V. Thangarasa, A. Gupta, W. Marshall, T. Li, K. Leong, D. DeCoste, S. Lie, and S. Saxena, "Spdf: Sparse pre-training and dense fine-tuning for large language models," in *Uncertainty in Artificial Intelligence*. PMLR, 2023, pp. 2134–2146.

[63] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[64] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[65] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[66] P. Villalobos, J. Sevilla, T. Besiroglu, L. Heim, A. Ho, and M. Hobbhahn, "Machine learning model sizes and the parameter gap," *arXiv preprint arXiv:2207.02852*, 2022.

[67] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[68] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 97–110.

[69] H. Wang, H. Xu, Y. Wang, and Y. Han, "Cta: Hardware-software co-design for compressed token attention mechanism," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 429–441.

[70] Y. N. Wu, P.-A. Tsai, S. Muralidharan, A. Parashar, V. Sze, and J. S. Emer, "Highlight: Efficient and flexible dnn acceleration with hierarchical structured sparsity," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2023.

[71] Y. N. Wu, P.-A. Tsai, A. Parashar, V. Sze, and J. S. Emer, "Sparseloop: An analytical approach to sparse tensor accelerator modeling," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1377–1395.

[72] S.-K. Yeom, P. Seegerer, S. Lapuschkin, A. Binder, S. Wiedemann, K.-R. Müller, and W. Samek, "Pruning by explaining: A novel criterion for deep neural network pruning," *Pattern Recognition*, vol. 115, p. 107899, 2021.

[73] M. Yoo, J. Song, J. Lee, N. Kim, Y. Kim, and J. Lee, "Sgcn: Exploiting compressed-sparse features in deep graph convolutional network accelerators," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 1–14.

[74] A. H. Zadeh, M. Mahmoud, A. Abdelhadi, and A. Moshovos, "Mokey: enabling narrow fixed-point inference for out-of-the-box floating-point transformer models," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 888–901.

[75] O. Zafrir, A. Larey, G. Boudoukh, H. Shen, and M. Wasserblat, "Prune once for all: Sparse pre-trained language models," *arXiv preprint arXiv:2111.05754*, 2021.

[76] R. Zellers, A. Holtzman *et al.*, "Hellaswag: Can a machine really finish your sentence?" *arXiv preprint arXiv:1905.07830*, 2019.

[77] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang *et al.*, "Flightllm: Efficient large language model inference with a complete mapping flow on fpgas," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2024, pp. 223–234.

[78] C. Zhang, Y. Wang, Z. Xie, C. Guo, Y. Liu, J. Leng, G. Sun, Z. Ji, R. Wang, Y. Xie *et al.*, "Dstc: Dual-side sparsity tensor core for dnns acceleration on modern gpu architectures," *IEEE Transactions on Computers*, 2024.

[79] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[80] Y. Zhang, L. Zhao, M. Lin, Y. Sun, Y. Yao, X. Han, J. Tanner, S. Liu, and R. Ji, "Dynamic sparse no training: Training-free fine-tuning for sparse llms," *arXiv preprint arXiv:2310.08915*, 2023.

[81] K. Zhong, Z. Zhu, G. Dai, H. Wang, X. Yang, H. Zhang, J. Si, Q. Mao, S. Zeng, K. Hong *et al.*, "Feasta: A flexible and efficient accelerator for sparse tensor algebra in machine learning," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 349–366.

[82] A. Zhou, Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun, and H. Li, "Learning n: m fine-grained structured sparse neural networks from scratch," *arXiv preprint arXiv:2102.04010*, 2021.

[83] M. Zhu, T. Zhang, Z. Gu, and Y. Xie, "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*, 2019, pp. 359–371.

[84] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.