








DSTC: Dual-Side Sparse Tensor Core for DNNs Acceleration on Modern GPU Architectures

Chen Zhang , Yang Wang , Zhiqiang Xie , Cong Guo , Yunxin Liu , *Senior Member, IEEE*,
Jingwen Leng , *Member, IEEE*, Zhigang Ji , Yuan Xie , and Ru Huang

Abstract—Leveraging sparsity in deep neural network (DNN) models holds significant promise for accelerating model inference. However, current GPUs can only harness sparsity in model weights, leaving activations unutilized due to their dynamic and unpredictable nature, which poses a considerable challenge for exploitation. In our research, we introduce a novel architectural approach aimed at effectively leveraging dual-side sparsity, encompassing both weight and activation sparsity. Our methodology involves a systematic examination of previous sparsity-related architectures, and culminating in the proposal of an uncharted paradigm that combines outer-product computation primitive and bitmap-based encoding format. Our approach showcases feasibility through minimal modifications to existing production-scale inner-product-based Tensor Cores. We introduce a set of innovative ISA extensions and carefully co-design matrix-matrix multiplication and convolution algorithms, the two predominant computation patterns in contemporary DNN models, to exploit our novel dual-side sparse Tensor Core. Our evaluation demonstrates the efficacy of our design, unlocking the full potential of dual-side DNN sparsity and delivering performance enhancements of up to an order of magnitude while incurring only modest hardware overhead.

Index Terms—Neural networks, graphics processing units, general sparse matrix-matrix multiplication, sparse convolution, model pruning.

I. INTRODUCTION

THE widespread deployment of deep learning has spurred the need to accommodate billions of daily inference

queries in data centers [12]. Many AI applications are subject to stringent service level agreements, demanding high-scale, low-latency, and energy-efficient model execution. Consequently, model compression and sparsification have assumed paramount importance as optimization strategies aimed at reducing parameters, minimizing arithmetic operations, and enhancing computational and energy efficiency. These optimizations span various hardware platforms, including ASICs [8], [15], [29], [42], GPUs [10], [40], and FPGAs [5], [21].

To harness the potential for accelerated sparse neural networks, GPU vendors have introduced architectural support. Notably, the introduction of the sparse Tensor Core [22] is a novel approach to exploit weight sparsity within DNN models. The latest NVIDIA Ampere and Hopper architecture [24], [25] introduces a redesigned sparse Tensor Core with a fixed 50% weight pruning target, achieving improved accuracy and performance trade-offs [5], [40].

In addition to weight sparsity, DNN models also exhibit another form of sparsity referred to as *activation* sparsity, introduced by activation functions [1]. Activation sparsity is prevalent in activation feature maps used in both computer vision [4] and natural language processing [7] tasks. Numerous prior works have reported high activation sparsity levels, ranging from 50% to 98% [29]. However, the current sparse Tensor Core primarily focuses on weight sparsity and does not effectively leverage activation sparsity. Effectively harnessing activation sparsity remains an open and challenging research problem due to its dynamic nature, which varies with input and cannot be pre-determined or controlled through pruning methods.

While previous efforts have addressed dual-side sparsity in ASIC designs, these solutions are not directly applicable to GPUs. Given the broad utility of GPUs, it is imperative to support both sparse general matrix-matrix multiplication (SpGEMM) and sparse convolution (SpCONV). These two kernels are pivotal to contemporary DNN models, spanning convolutional neural networks (CNNs) [14] and transformer-based models [38]. However, current ASIC designs primarily focus on either SpGEMM [28], [35], [42] or SpCONV [8], [15], [29]. In this study, our objective is to accelerate both dual-side SpCONV and SpGEMM on the Tensor Core architecture.

The primary challenge in enabling dual-side SpCONV and SpGEMM on Tensor Cores lies in handling the unpredictable and randomly distributed non-zero elements within the input tensors. The dot product unit, the fundamental computational unit in Tensor Core hardware, is responsible for conducting

Received 2 January 2024; accepted 14 August 2024. Date of publication 8 October 2024; date of current version 20 January 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62032001, Grant 62072297, and Grant 62222210, in part by the 111 Project (B18001), and in part by the Research Grants Council of HKSAR under Grant 16213824. Recommended for acceptance by R. Sriram. (Corresponding author: Chen Zhang.)

Chen Zhang, Cong Guo, and Zhigang Ji are with Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: chen-zhang.sjtu@sjtu.edu.cn; guocong@sjtu.edu.cn; zhigangji@sjtu.edu.cn).

Yang Wang is with Microsoft Research, Beijing 100080, China (e-mail: yangwang5@microsoft.com).

Zhiqiang Xie is with Stanford University, Stanford, CA 94305 USA (e-mail: xiezqh@cs.stanford.edu).

Yunxin Liu is with Tsinghua University, Beijing 100084, China (e-mail: liuyunxin@air.tsinghua.edu.cn).

Jingwen Leng is with Shanghai Jiao Tong University, Shanghai 200240, China, and also with Shanghai Qi Zhi Institute, Shanghai 2719, China (e-mail: leng-jw@sjtu.edu.cn).

Yuan Xie is with Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong (e-mail: yuanxie@ust.hk).

Ru Huang is with Peking University, Beijing 100871, China (e-mail: ruhuang@pku.edu.cn).

Digital Object Identifier 10.1109/TC.2024.3475814

TABLE I
TECHNICAL DIFFERENCES TO RELATED WORK

	Inner-product	Outer-product	Misc
CSR	[15], [43]	[28], [29], [42]	[35]
Bitmap	[8]	Our work	-

vector-vector inner products. While Sparse Tensor Core [22], [24] effectively addresses the irregularity of weight sparsity by implementing a structural pruning scheme, which enforces a consistent 50% weight sparsity to balance the workload and exploit parallelism within the dot-product unit. This approach is not directly applicable to SpGEMM because activation sparsity in SpGEMM is input-dependent and cannot be predetermined through pruning techniques. In contrast, previous ASIC designs have taken two distinct approaches to leverage dual-side sparsity, as summarized in Table I. SparTen and Extensor [8], [15], for instance, accelerate inner-product computations by designing specialized hardware for the inner join process, which identifies non-zero elements by matching positions in two sparse vectors and accessing those elements. However, this process's computational complexity can be as high as $O(n^2)$ (where n represents the vector length of the inner join), resulting in substantial overhead, including complex prefix sum hardware and explicit barriers. For example, SpArch [42] uses an array with 16 floating point multipliers for SpGEMM, and requires specialized Merge Tree and matrix read/write hardware, which occupies the 98.4% die area ($\approx 28\text{mm}^2$ @ 40nm). If we were to scale the design to 110592 FP multipliers in A100 [24], the estimated area cost would be prohibitively expensive (i.e., 193536mm^2). Conversely, other efforts such as OuterSPACE [28] and SpArch [42] focus on accelerating SpGEMM with an outer-product approach but do not accommodate SpCONV efficiently, leading to significant performance overhead when directly converting SpCONV to SpGEMM. Furthermore, their target matrix densities, ranging from 6×10^{-3} to 5×10^{-5} , become inefficient for mainstream Deep Neural Network (DNN) models, which typically fall within the density range of 5×10^{-1} to 1×10^{-2} . Similarly, SCNN [29] primarily considers SpCONV but lacks support for SpGEMM.

It's also challenging to accelerate SpCONV. GPUs usually transform a CONV operator into a GEMM operator via the im2col method. Sparse Tensor Core [22], [24], [43] utilizes weight sparsity but maintains dense input, requiring only dense im2col. However, exploiting dual-side sparsity demands *sparse* im2col for convolution, resulting in irregular memory access patterns. Moreover, vendor-supplied DNN acceleration library cuDNN [6] optimizes *implicit im2col* to fuse address generation into matrix multiplication, typically offering the best performance. Yet, applying implicit im2col to sparse input tensors proves notably more challenging due to randomly distributed non-zero elements. In our findings, a naive implementation of implicit sparse im2col can be $10\times$ to $100\times$ slower than its dense counterpart.

To address these challenges, we conducted a thorough analysis of the computation patterns for sparse im2col and SpGEMM while exploring various approaches. Our investigation led us to

conclude that a bitmap-based encoding format is well-suited for efficient sparse im2col acceleration, while outer-product methods are highly effective for capitalizing on SpGEMM's potential with Tensor Core. Consequently, we introduced a bitmap-based sparse im2col algorithm for SpCONV and an outer-product-based dual-side sparse Tensor Core architecture for SpGEMM. Furthermore, we proposed an outer-product-friendly sparse im2col method and a bitmap-based outer-product SpGEMM algorithm to enhance the synergy between these techniques. Through these innovations, we achieved an efficient design for implicit sparse im2col in SpCONV acceleration. Our experiments, conducted on Accel-Sim with the A100 architecture, showcased remarkable acceleration for both SpCONV and SpGEMM on the proposed dual-side sparse Tensor Core architecture, delivering up to a tenfold speedup compared to state-of-the-art baselines while imposing minimal hardware overhead.

The key technical contributions of this work are as follows:

- We propose a novel method that combines outer product and bitmap encoding to accelerate SpGEMM (Section III) and SpCONV (Section IV). Building upon the bitmap encoding, we introduce a novel methodology known as TIA. TIA leverages the inherent shift-invariant characteristics of the sparsity pattern within the OTC's output to effectively mitigate irregularities in the sparse accumulation process, a unique challenge posed by the outer-product operation.
- We demonstrate the compatibility of our method with existing GPU architectures through a targeted set of modifications that enable the use of dual-side sparsity with Tensor Cores. The design accommodates a range of data types and tensor shape configurations, catering to the varied precision demands of AI computations. Moreover, we introduce instruction set extensions that enhance our capability to exploit established high-performance libraries, which are further discussed in Section V.
- Through comprehensive evaluations, our dual-side sparse Tensor Core achieves remarkable speedup improvements, with performance gains of $1.25\text{-}7.49\times$ for SpCONV and $2.63\text{-}8.45\times$ for SpGEMM compared to state-of-the-art methods. Remarkably, these enhancements come with a minimal 1.56% area overhead, as detailed in Section VI.

II. BACKGROUND AND RELATED WORK

A. Opportunities of Sparsity in DNNs

Weight sparsity has garnered extensive attention across diverse domains, including computer vision and natural language processing tasks [9], [39]. These studies have consistently showcased the achievement of high sparsity levels through various pruning techniques. While reducing the number of weight parameters can result in storage savings, it often falls short of delivering substantial acceleration in inference due to fragmented and irregular patterns [9], [20]. Some researchers have proposed pruning methods optimized for hardware implementations, leading to practical gains in speedup [43]. Notably, NVIDIA's latest Ampere GPU introduces Sparse Tensor Cores,

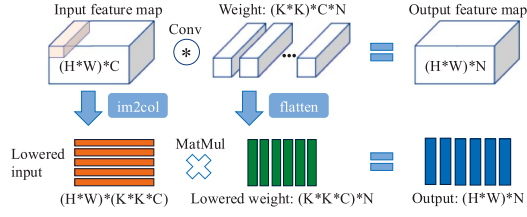


Fig. 1. The im2col-based transformation of CONV to GEMM.

embracing fine-grained structural pruning [5], [22], [24], marking a significant milestone in GPU design.

Activation sparsity naturally occurs in CNN and transformer layers, followed by activation functions [4]. Unlike weight sparsity, activation sparsity dynamically changes with input data and is highly unstructured. Previous works [4], [29] demonstrate that activation sparsity can be as high as 45% to 98%. Some researchers [32] accelerate activation sparsity on GPUs by adding blocked masks, but it requires external knowledge and is not generic. However, to the best of our knowledge, no previous work has demonstrated meaningful speedup by exploiting activation sparsity on GPU.

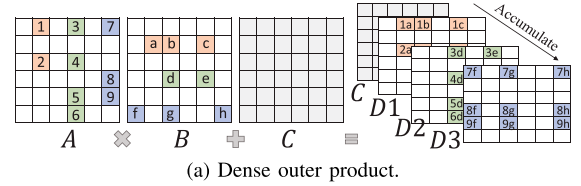
B. Computation Kernels

Deep neural networks consist of multiple interconnected layers, encompassing both linear and non-linear functions. Matrix multiplication and convolution operations stand out as the primary computational kernels within these networks, accounting for a substantial proportion of both the model's parameters and computational workloads [41].

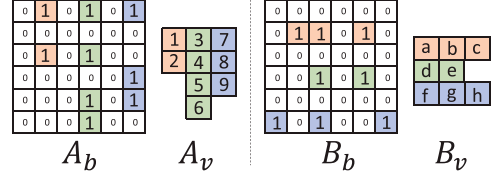
Matrix multiplication (GEMM) is the key computational kernel in NLP models, like RNNs [17] and transformers [7], [38]. Dense GEMM is one of the fundamental computation primitives provided by GPU, which has been under continuous optimization. Especially, Tensor Core, as the specialized hardware, has recently been deployed in GPU to boost GEMM performance by an order of magnitude. *Convolution* plays a vital role in CNNs [14], handling feature extraction and spatial filtering in images or feature maps. It dominates computational workloads in CNNs, exceeding 90% [41].

To leverage tensor core's matrix-multiply capabilities, state-of-the-art DNN acceleration libraries like cuDNN often transform convolution via the im2col function, which restructures input feature maps into lowered feature maps, aligning each row with a 2D sliding window in the input. Im2col on weight parameters straightforwardly flattens $K \times K \times C$ kernels. Weight-sparse architecture treats im2col as "dense im2col," but dual-side sparse architecture faces the challenge of "sparse im2col," which has not been fully discussed in previous work.

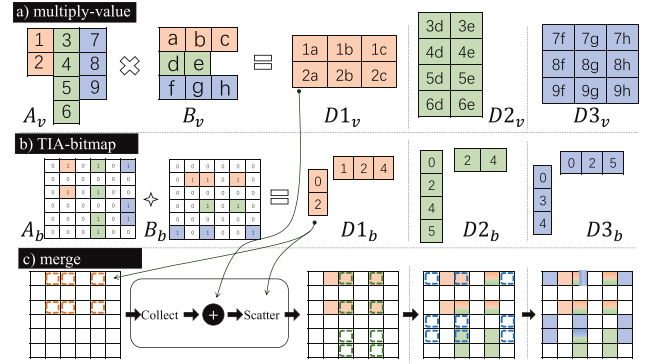
The naïve approach, called *explicit im2col*, separates im2col and GEMM operations but increases global memory usage ($K \times K$ times) due to overlapped sliding window data duplication. Modern DNN libraries, like cuDNN, opt for "implicit im2col" for better input data utilization. This method retains the original feature map layout in global memory and employs an address



(a) Dense outer product.



(b) Encoded bitmaps of Matrix A and B.



(c) Procedures of the proposed SpGEMM.

Fig. 2. Our proposed bitmap-based outer-product SpGEMM.

conversion scheme for on-chip cache-based im2col. By avoiding physical data duplication in global memory, implicit im2col is a widely adopted state-of-the-art technique for accelerating convolution with GEMM operators, addressing the memory inefficiencies of explicit im2col.

III. BITMAP-BASED SPGEMM

We propose an outer-product-based algorithm to accelerate SpGEMM using the bitmap-based sparse encoding format.

A. Overview

To harness the advantages of dual-side sparsity, we introduce an efficient SpGEMM algorithm based on the outer-product approach. A basic step in this method involves computing the cross-product between a column of the matrix \mathbb{A} and a row of the matrix \mathbb{B} , resulting in a partial matrix denoted as $\langle M, N \rangle = \langle M, 1 \rangle \times \langle 1, N \rangle$. This process is illustrated as $\mathbb{D}13$ in Fig. 2(a). To derive the final output, these partial results must be accumulated in conjunction with the bias matrix \mathbb{C} .

Our approach achieves efficient outer-product computations by employing a bitmap representation, as depicted in Fig. 2(b). Each input matrix is encoded using a two-tuple representation comprising a bitmap (e.g., \mathbb{A}_b) and a collection of non-zero values (e.g., \mathbb{A}_v). In the bitmap, '1's indicate the positions of non-zero values, while '0's correspond to zeros. To facilitate the

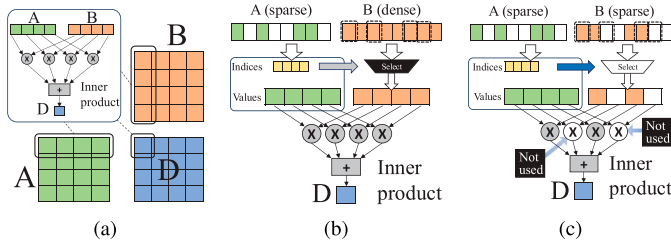


Fig. 3. (a) $4 \times 4 \times 4$ matrix multiplication primitive used in inner-product-based Tensor Core [27]; (b) A sparse inner-product unit in A100 [22], [24]; (c) Dual-side sparsity unit based on inner-product.

outer-product operation, matrix \mathbb{A}_v is encoded in column-major order, while \mathbb{B}_v is encoded in row-major order.

The proposed SpGEMM algorithm contains three fundamental operations on the bitmap-encoded matrices, denoted as *multiply-value*, *TIA-bitmap*, and *merge*, as illustrated in Fig. 2(c). The *multiply-value* operation calculates the cross-product (e.g., $\mathbb{D}1_v$) for each vector-vector pair of \mathbb{A}_v and \mathbb{B}_v . This outer-product approach inherently eliminates the need for an explicit inner-join process, resulting in a more regular multiplication process. The *TIA-bitmap* operation determines the index addresses (e.g., $\mathbb{D}1_b$) of the sparse non-zero elements of \mathbb{D} . Lastly, the *merge* operation combines values (e.g., $\mathbb{D}1_v$) and indexes (e.g., $\mathbb{D}1_b$) to accumulate across multiple iterations, transitioning from $\mathbb{E}1$ to $\mathbb{E}3$ and so on. Despite the irregular and sparse nature of the resulting partial matrices, the choice to handle single-side irregular accumulation proves advantageous, as it offers computational efficiency compared to addressing dual-side sparse multiplication. It's important to note that sorting the irregular pattern for output sparsity incurs a complexity of $O(2n)$, which is significantly smaller than the $O(n^2)$ complexity associated with addressing dual-side irregularity in sparse inner product.

In the subsequent section, we provide a detailed analysis of the challenges encountered when accelerating SpGEMM using inner-product-based Tensor Cores, and introduce a SpGEMM algorithm tailored for outer-product Tensor Cores.

B. SpGEMM in a Warp

1) *Problems of Inner-Product Tensor Core*: In the A100 architecture [24], each Streaming Multiprocessor (SM) is equipped with four tensor cores. These tensor cores are individually managed by distinct warps and have the capability to perform an $8 \times 4 \times 8$ dense matrix multiplication. The core computational unit within a tensor core comprises an 8-element parallel vector-vector dot product unit, responsible for carrying out the multiplication and accumulation of a row from matrix A and a column from matrix B. To illustrate this process more clearly, we have provided a simplified $4 \times 4 \times 4$ depiction in Fig. 3(a). In the context of single-sided sparse matrix multiplication, the dot product operation requires precise selection and access to non-zero elements situated at corresponding positions within the dense vector.

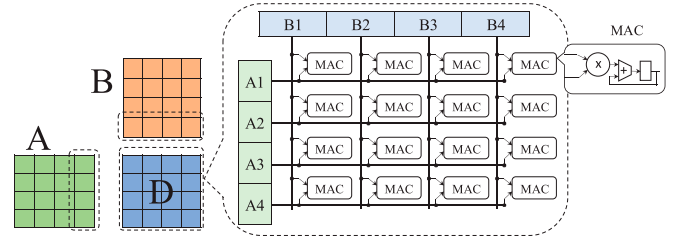


Fig. 4. Outer-product matrix multiplication ($4 \times 4 \times 4$ primitive) and basic unit of a tensor core.

However, the presence of irregularities due to the sparsity inherent in sparse models can lead to resource underutilization and, consequently, a detrimental impact on achievable performance. To address this issue, the sparse tensor core, as outlined in [22], [24], employs a structural pruning technique, conducting a 2-out-of-4 pruning within each partitioned sub-vector. This approach ensures a consistent 50% weight sparsity, effectively balancing the workload and harnessing parallelism within the dot-product unit, as illustrated in Fig. 3(b).

Nonetheless, this pruning method proves to be inefficient for dual-sided sparse matrix multiplication. This inefficiency arises because activation sparsity is input-dependent, and the number of non-zeros to be jointly matched becomes unpredictable. This unpredictability makes it challenging to fully exploit dot product parallelism, as shown in Fig. 3(c). While some prior ASIC designs [8], [15], [29] have proposed dedicated hardware solutions to tackle this issue, their methods often involve complex prefix sum hardware, costly shuffling registers, or explicit barriers. These solutions introduce substantial overheads and necessitate significant modifications to the Tensor Cores.

2) *Outer-Product Tensor Core (OTC)*: Our design incorporates an Outer-Product-Based Tensor Core (OTC), as is exemplified by the $4 \times 4 \times 4$ primitive shown in Fig. 4. In this approach, matrix multiplication is executed by aggregating multiple partial output matrices, such as \mathbb{D} , sized $M \times N$. Each partial output is computed through a cross-product operation involving a column-vector (sized $M \times 1$) from matrix \mathbb{A} and a row-vector (sized $1 \times N$) from matrix \mathbb{B} . One of the inherent advantages of the OTC design is its innate ability to avoid the inner-join process and eliminate irregular sparse matching. This is achieved because there is no reduction involved between two operands. Consequently, the computation order can be rearranged into a more regular pattern without adversely affecting the final results. As illustrated in Fig. 5, outer-product-based solutions facilitate the alignment of all non-zero elements in each column of matrix \mathbb{A} to the upper side and all non-zero elements in each row of matrix \mathbb{B} to the left, forming two dense vectors. The outer-product multiplication performed on these condensed inputs results in condensed matrix multiplication. After this, non-zero elements are concentrated, allowing tensor cores to execute fewer instructions to complete a matrix multiplication. Consequently, this approach can lead to significant speedup compared to the original dense matrix multiplication.

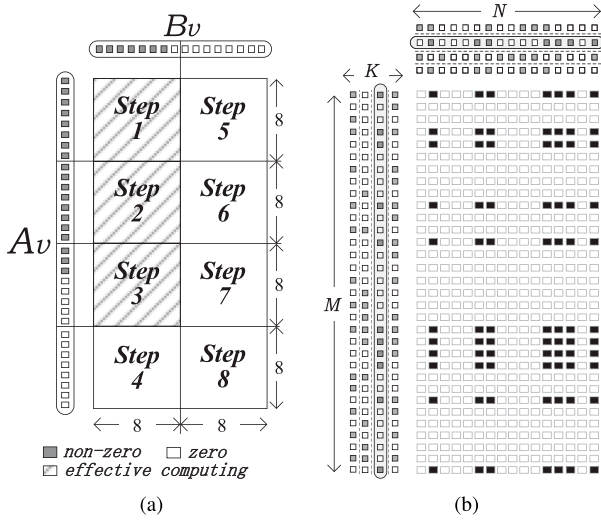


Fig. 5. (a) Outer-product with two sparse inputs ($M = 32, N = 16, K = 4$); (b) Leveraging dual-side sparsity by condensing the inputs into shorter but dense vectors.

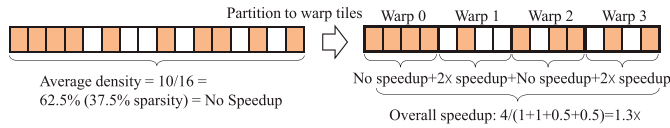


Fig. 6. Speedup on global matrix.

3) Warp-Level Outer-Product SpGEMM: We introduce an efficient warp-level SpGEMM approach utilizing Outer-Product-Based Tensor Cores (OTCs). Fig. 5(b) provides an illustrative example where our warp-level SpGEMM demonstrates significant speedup when applied to OTCs with sparse inputs. The figure illustrates the computation of a $32 \times 16 \times K$ warp tile using the outer-product methodology.

OTCs perform computations in cycles. For an $8 \times 8 \times 1$ OTC, it takes 8 steps to complete a full $32 \times 16 \times 1$ outer-product or 16 steps for a $32 \times 32 \times 1$ computation. In scenarios involving sparse inputs, such as those represented by A_v and B_v , which contain fewer non-zero elements in each column and row, we can achieve substantial speedup by skipping certain OTC steps.

In Fig. 5(b), it can be observed that the column vector from A_v contains 19 non-zero elements within 32 elements, while the row vector from B_v contains 7 non-zero elements within 16 elements. Consequently, 5 out of the 8 OTC steps consist entirely of zero elements and can be skipped, theoretically resulting in a speedup of $\frac{8}{3} = 2.67x$. The extent of OTC step skipping is contingent on the sparsity levels of the input vectors, which are specified as $\langle 0\%, 25\%, 50\%, 75\% \rangle$ on the A_v side and $\langle 0\%, 50\% \rangle$ on the B_v side in this particular case. To maintain OTC's 8×8 tile dimension, zero padding is applied to the inputs when necessary.

Discussion: While the potential for acceleration is often limited by a predefined set of sparsity ratios, such as $\langle 0\%, 50\% \rangle$ for B_v , our approach operates at the global matrix level, tran-

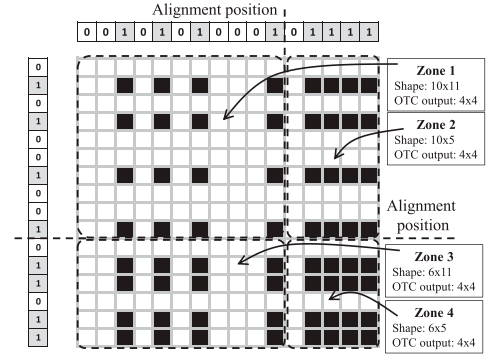


Fig. 7. An example to show the irregular pattern in OTC's sparse output matrix.

scending this constraint. In Fig. 6, we provide an illustrative example where a row within the global matrix exhibits a 37.5% sparsity. Traditionally, this condition might not result in any speedup, assuming the sole benefit from 50% sparsity. However, our method enables us to achieve an approximate $1.3x$ speedup by considering warp tiling at the global matrix level. Given the typical uneven distribution of non-zero values across the global matrix, specific warps, such as warp 1 and 3, can still capitalize on the speedup advantages offered by our SpGEMM approach.

4) Merge: The merge operation plays a crucial role in accumulating partial results, such as E_1 , E_2 , and E_3 in Fig. 2(c), and writing them back to the output buffer of the tensor cores. However, the accumulation of partial results from the outer product poses a unique challenge due to its irregular and unpredictable nature. This irregularity can be attributed to two key factors. First, the positions of the non-zero elements are inherently irregular and cannot be pre-determined. Second, the number of non-zero elements in each input vector, such as A 's column or B 's row, remains unpredictable.

Fig. 7 provides a detailed visualization of this irregularity. Consider a warp with dimensions $\langle M = 16, N = 16, K = 1 \rangle$, performing matrix multiplication with an OTC sized at 4×4 . It takes 4 OTC steps to complete the calculation of all non-zero outputs, with each step generating a 4×4 square non-zero matrix. Due to sparsity, the physical locations of these non-zero elements in the final 16×16 matrix are distributed irregularly. In Fig. 7, these irregularly distributed zones are labeled as zones 0 to 3, and the alignment position indicates the location of the last bit, representing the boundary of each OTC output tile.

Determining the alignment position is challenging because it must match the throughput of OTC computation. Addresses for the accumulation data in matrix D buffer must be prepared before conducting the accumulation process. To address this challenge, we introduce a dedicated hardware unit called the Tile Index Aligner (TIA) designed to identify alignment positions in parallel. This architecture incorporates multiple parallel sorters and parallel barrel shifters [11], as depicted in Fig. 8. The 4-way sorter, which includes a crossbar and leading-one detectors (LOD), can sort a 4-element vector and generate condensed addresses in parallel. The 4-to-8 (8-to-16) merger, built on barrel shifters, concatenates two vectors into a longer

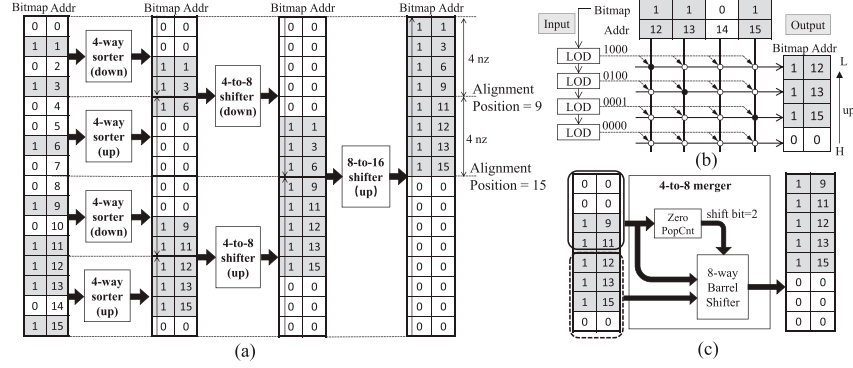


Fig. 8. (a) Tile index aligner (TIA) unit finds the alignment position for OTC's output matrix. (b) The 4-way sorter. (c) The 4-to-8 shifter.

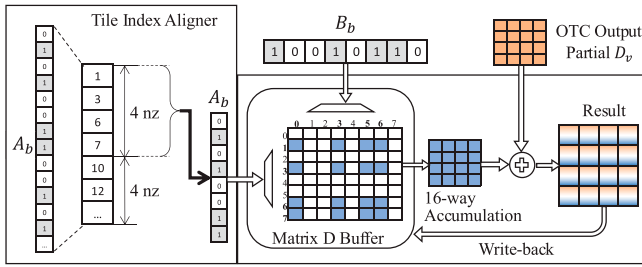


Fig. 9. Integration of TIA and accumulation buffer in OTC.

one with no leading zeros. This architecture ensures that the accumulation buffer has all the necessary addresses for each OTC output (up to 64×64) in a timely manner.

To integrate the TIA unit with the OTC for the *merge* operation, we incorporate it into the output matrix buffer of the Tensor Cores, as shown in Fig. 9. On one side, the TIA unit facilitates parallel accumulations that can match the processing throughput of the OTC. For an 8×8 OTC, we equip the accumulation buffer with 64-way parallel accumulators. On the other side, we design a lightweight operand collector to handle memory access conflicts. Further details regarding the hardware design and evaluation can be found in Sections V and VI.

Discussion: Thanks to the inherent characteristics of the outer product, the positions of its output matrix elements exhibit a shift-invariant structure. For instance, the sparsity patterns between non-zero rows (and columns) remain exactly the same, as demonstrated by the sparsity pattern in the product output shown in Fig. 5(a). Consequently, deriving addresses to index the non-zero elements can be achieved with *linear* complexity, relying on the input vector bitmaps, such as A_b and B_b in Fig. 2(b). As there are two inputs involved, the complexity remains in $O(2n)$. Notably, this complexity is significantly lower than the $O(n^2)$ complexity associated with addressing the dual-side irregularity in sparse inner product computations.

C. SpGEMM on the Device

The primary challenge in deploying SpGEMM across the entire device stems from poor output data reuse. When ex-

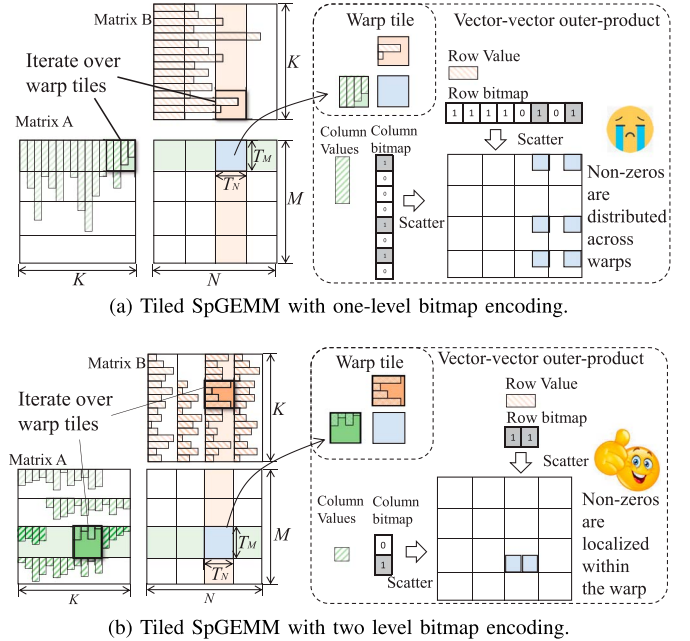


Fig. 10. Data-locality aware sparse representation.

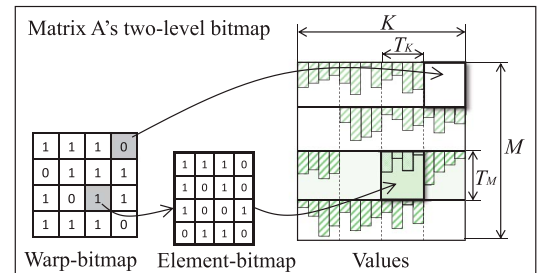


Fig. 11. Two-level bitmap encoding format.

cutting large matrix multiplications, the outer products yield a significant number of partial matrices. In the case of SpGEMM, where non-zero elements are distributed randomly across these partial matrices, it results in a vast addressing space. Often, this addressing space surpasses the local buffer capacity of a warp,

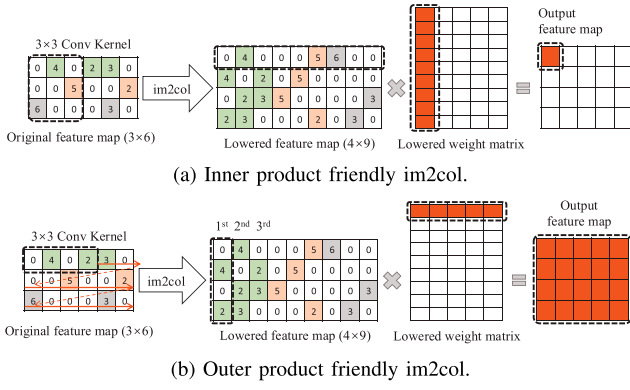


Fig. 12. Outer product friendly im2col on dense matrix.

resulting in fragmented global memory accesses, as depicted in Fig. 10(a).

To address this issue, we propose a hierarchical bitmap encoding format tailored to align with the GPU’s tiling scheme for SpGEMM, as illustrated in Fig. 10(b). This two-level bitmap encoding format consists of three distinct tuples, visualized in Fig. 11. The first-level bitmap encodes individual partitioned matrix tiles. Within this element-bitmap, each ‘1’ or ‘0’ signifies elemental non-zeros or zeros within the warp tile. Importantly, this element-bitmap demonstrates remarkable efficiency because it confines the positions of non-zero elements within the output partial matrix to this tile. Consequently, these elements can be accommodated in the Tensor Core’s high-speed local buffer, eliminating the need for external memory access. The second-level bitmap, known as the warp-bitmap, employs ‘1’ or ‘0’ to denote the entire tile. ‘0’s indicate an empty tile, while ‘1’s indicate a non-empty one. Warps corresponding to ‘0’ warp-bits can be skipped, signifying that either of the two input tiles contains only zeros.

IV. DUAL-SIDE SPARSE CONVOLUTION

GPUs typically process dense convolution by converting it into a GEMM operation using the im2col function. The main role of im2col is to restructure the data layout of input feature maps, making them suitable for GEMM processing. However, an inadequately designed im2col can negatively impact the efficiency of data reuse in matrix multiplication. To mitigate the space and time overheads associated with traditional im2col operations, we have developed a novel “implicit” sparse im2col algorithm. This innovative method effectively reorganizes data within registers, thereby bypassing the need for explicit im2col processes, yet still preserving the advantages of optimized data arrangement.

A. Outer-Product Friendly Im2col

In Fig. 12(a), we present a visual representation of the im2col operation applied to a 3×6 feature map with a 3×3 convolution kernel. Im2col essentially reorganizes all the elements within the sliding 3×3 window into a single row within the “lowered” feature map. This process aligns seamlessly with

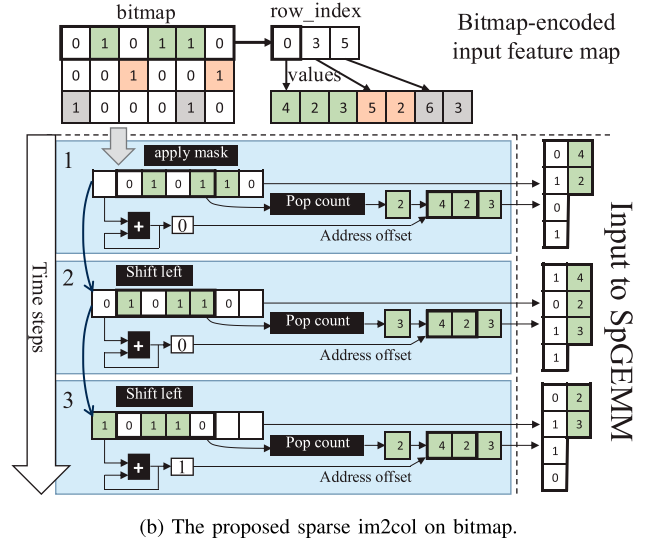
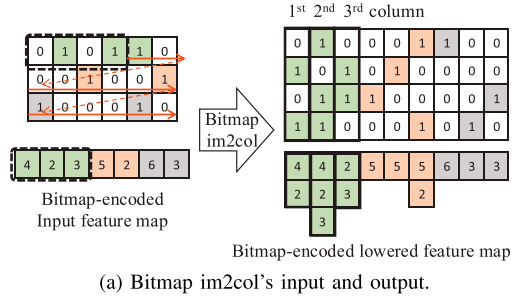


Fig. 13. The im2col on a bitmap-encoded feature map.

inner-product operations because, at each step, one row corresponds to the computation involving multiplication and accumulation in inner-products.

On the other hand, for outer-product operations, a column of data is required at each step, a functionality that im2col isn’t inherently designed to provide from the fully “lowered” feature map. Consequently, we introduce an outer-product-friendly im2col method, as illustrated in Fig. 12(b). In this arrangement, the first three columns, for example, correspond to data from the first row of the original feature map. These three columns share common data elements among themselves. By employing a zig-zag pattern to scan the feature map with a 1×4 window, we construct a column-major “lowered” feature map, which serves as the input for GEMM operations.

B. Bitmap-Based Sparse Im2col

Similar to dense *implicit im2col*, our *sparse implicit im2col* preserves the bitmap-encoded sparse feature maps in global memory while reorganizing the data layout in registers. Matrix B is essentially a bitmap-encoded representation of the flattened sparse weight matrix.

Bitmap encoding proves to be highly efficient for sparse im2col operations because it inherits the structural information from a dense matrix. Consequently, a bitmap-based encoding format can be leveraged to perform im2col operations on the

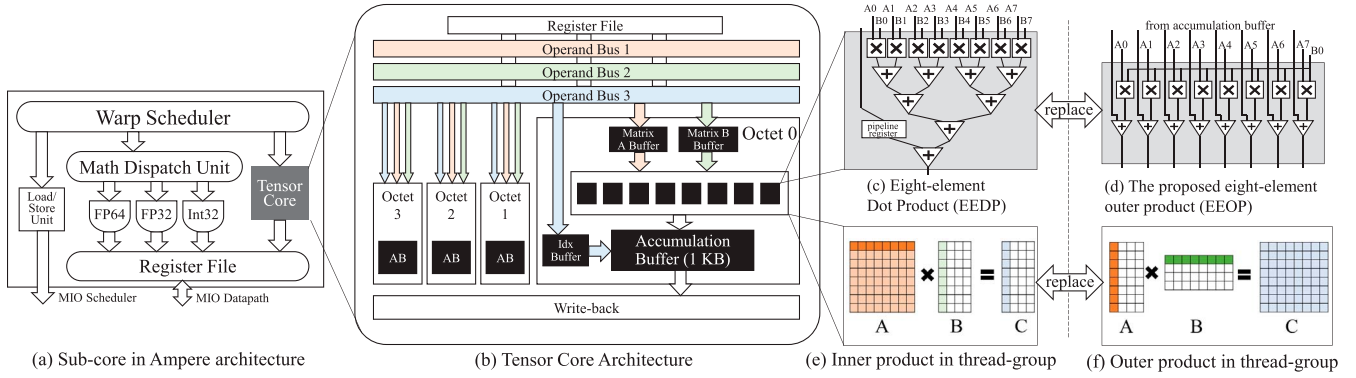


Fig. 14. Our modification to the Tensor Core includes the accumulation buffer in (b) and the replacement of EEDP Unit (c) with EEOP (d). The thread-group level inner-product (e) is replaced with outer-product (f).

bitmap itself, employing a technique akin to dense im2col. Subsequently, we employ this lowered bitmap as a mask to retrieve the corresponding non-zero values. For a detailed illustration of our approach, refer to Fig. 13. In this example, we take a 3×6 feature map convolved by a 3×3 kernel, resulting in a 4×9 lowered feature map as shown in Fig. 13(a).

- S0 Initially, obtain the original feature map in bitmap encoding.
- S1 Take the first bitmap row and corresponding non-zeros.
- S2 For the first column, employ a mask on the bitmap row. For others, perform a leftward bit shift, pop out the leftmost bit.
- S3 Accumulate the shifted-out bits. Use it as the address offset for accessing the non-zero values.
- S4 Apply *population count* to determine the number of non-zeros within the mask. Output the value vector.

Our approach is efficient for two reasons. Firstly, all necessary operations are of low cost and can be executed in register files. Secondly, the resulting data is already in a condensed format, enabling direct integration into outer-product SpGEMM via register reads.

V. OUTER PRODUCT SPARSE TENSOR CORE

In this section, we introduce the micro-architecture extensions to support our bitmap-based SpGEMM and SpCONV.

A. Outer-Product Tensor Core (OTC)

We modify Tensor Core hardware from inner-product to outer-product for *dense matrix multiplication* because it is a pre-requisite for our SpGEMM algorithm.

1) *Hardware Modification*: Tensor Cores, specialized hardware for matrix multiplication, have been integrated into NVIDIA's GPGPU architecture since the Volta era [27], significantly boosting the acceleration of machine learning tasks. Subsequent architectural iterations in Ampere and Hopper [24], [25] have continued to enhance their performance. Fig. 14(a) provides an overview of a Sub-Core within Ampere's streaming processor (SM). Each Tensor Core can accomplish a $8 \times 4 \times 8$ dense matrix multiplication within a single cycle [31]. In

the A100 GPU, a total of 432 Tensor Cores are thoughtfully distributed across 108 SMs, with each Sub-Core housing one Tensor Core and each SM containing 4 Sub-Cores. This configuration results in an impressive peak performance of 312 TFLOPS, operating at a clock frequency of 1410 MHz.

Fig. 14 delves into the detailed architecture of the Tensor Core within a Sub-Core. Each Tensor Core boasts 32 inner-product units, with each unit capable of executing an eight-element dot-product (EEDP). This arrangement translates to a total computing power of 256 multiply-accumulate operations per cycle within a single Tensor Core. Fig. 14(c) illustrates the EEDP structure, which concurrently multiplies and accumulates two eight-element vectors from matrices A and B. These 32 EEDPs are organized into four 'Octets,' with each Octet responsible for performing an $8 \times 1 \times 8$ matrix multiplication. The four Octets collaboratively execute an $8 \times 4 \times 8$ matrix multiplication, as exemplified in Fig. 14(e). Within a warp comprising 32 threads, data movement between register files and the memory hierarchy is efficiently managed.

We modify the above-mentioned inner-product Tensor Core to fit for *dense outer-product's* computation. In Fig. 14(d), we illustrate the modifications we made to the EEDP hardware. Our adaptation, known as the eight-element outer product (EEOP), operates by multiplying one element from A with eight elements from B in parallel and then accumulates the partial results using adders. Consequently, a group of eight EEOPs collaboratively executes an 8×8 outer-product operation, as demonstrated in Fig. 14(f). Similar to the previously described inner-product Tensor Core, when combined in sets of four Octets, they effectively perform an $8 \times 8 \times 4$ matrix multiplication.

2) *ISA Extensions for Dense Outer Product*: Each tensor core performs a $8 \times 4 \times 8$ dense matrix multiplication, constituting a machine-level HMMA instruction, either HMMA.16816 or HMMA.1688. This instruction computes an output block of dimensions 16×8 by employing a 16×16 (or 16×8) tile from matrix A and a 16×8 (or 8×8) tile from matrix B, as exemplified in Fig. 15(a). To compute an $8 \times 4 \times 16$ tile, two sets of HMMA instructions are employed. At the warp level, CUDA provides a WMMA API that leverages these HMMA instructions to perform a larger $16 \times 16 \times 16$ matrix operation in 16 cycles [25].

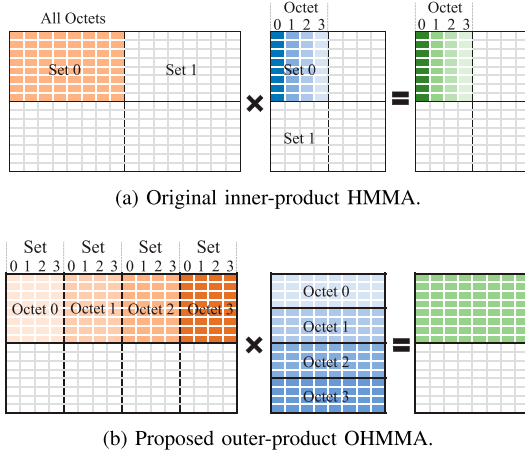


Fig. 15. The original HMMA instruction computes a $16 \times 8 \times 16$ matrix multiplication in four steps. We define OHMMA (outer-product HMMA) operation that computes the same tile with 2 steps and 4 sets of HMMA.884 micro-instructions.

```
HMMA.OHMMA.16816.F16.F16 {R9, R10}, {R1, R2, R3, R4},
                               {R5, R6}, {R7, R8};
HMMA.BITIA.3216.B32.B32 {R2, R3, R4, R5}, R1;
HMMA.BITIA.1616.B32.B32 {R2, R3}, R1;
```

Fig. 16. Extended OHMMA/BITIA instructions.

```
SPWMMA.MMA.SYNC.A_LAYOUT.B_LAYOUT.M32N16K16.set.f16.f16
{%RD0~%RD7}, {%RB0~%RB7}, {%RA0~%RA7}, {%RC0~%RC7};
```

Fig. 17. Our SpWMMA API.

Fig. 15(b) illustrates our OTC interface. Each OTC engages in an $8 \times 8 \times 4$ dense matrix multiplication through a vector-vector outer product approach. We have defined four sets of Outer-Product HMMA (OHMMA) instructions to complete a single $8 \times 8 \times 16$ matrix multiplication. In total, an OTC requires 8 cycles to execute an OHMMA.16816 instruction. Each cycle processes four pairs of 8×1 tiles from \mathbb{A}_v and 1×8 tiles from \mathbb{B}_v as input.

The Tile Index Aligner (TIA) plays a pivotal role in our bitmap-based SpGEMM. It is responsible for deriving aligned indexes for the accumulation process of each HMMA.884 instruction, as depicted in Fig. 9. We introduce a new instruction called BITIA, which takes the bitmap of the input matrix as input and generates aligned indexes for HMMA outputs. Additionally, it generates prediction bits that guide the execution of the OTC in sparse mode. Given that the warp tile size can reach $\langle m = 32, n = 32 \rangle$, we utilize 8 bits to store each index value. Consequently, four vector registers are employed to store the TIA output for a 32×16 FP16 matrix's bitmap, and two vector registers suffice for a 16×16 FP16 matrix's bitmap. OHMMA and BITIA instructions are defined in Fig. 16.

```
// R1: A[32x16]_bitmap, R2: B[16x16]_bitmap
HMMA.BITIA.3216.B32.B32 {R2, R3, R4, R5}, R1;
HMMA.BITIA.1616.B32.B32 {R6, R7}, R2;
// ...
@p0 HMMA.OHMMA.884.F16.F16 {R21, R22, R23, R24}, {R11, R12},
                               {R19, R20}, {R21, R22, R23, R24};
@p1 HMMA.OHMMA.884.F16.F16 {R25, R26, R27, R28}, {R13, R14},
                               {R19, R20}, {R25, R26, R27, R28};
// ...
@p7 HMMA.OHMMA.884.F16.F16 {R49, R50, R51, R52}, {R17, R18},
                               {R19, R20}, {R49, R50, R51, R52};
```

Fig. 18. SpWMMA compiled to machine-level instructions.

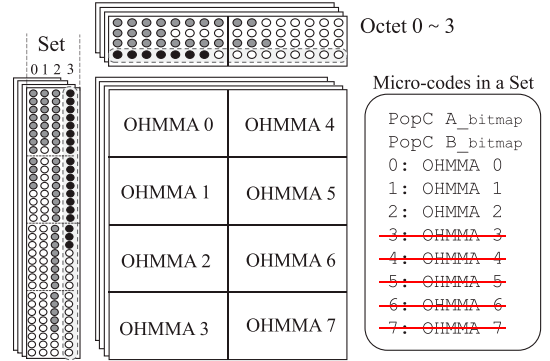


Fig. 19. The proposed SpWMMA includes 8 OHMMA.884 micro-instructions in dense mode, which is skipped on the sparse mode.

B. Dual-Side Sparse Tensor Core

We introduce two adaptations aimed at achieving speedup while leveraging the hardware and instruction extensions mentioned above. On the software front, we introduce SpWMMA, a warp-level API for dual-side SpGEMM that exploits the sparsity inherent in matrices A and B by dynamically skipping OHMMA instructions. On the hardware side, we propose the integration of an accumulation buffer, which efficiently aggregates partial results generated by the outer-product units.

1) *Warp-Level Interface*: We define a SpWMMA API that works on a warp-level matrix tile in Fig. 17. A SpWMMA breaks down into 4 Octets working concurrently. Each Octet includes 4 sets, and each set includes a $32 \times 32 \times 1$ outer product in Fig. 15. Since the machine-level OHMMA instruction computes an $8 \times 8 \times 1$ outer product within a warp. And each SpWMMA is compiled to 8 OHMMA instructions, as shown in Fig. 18.

For sparse inputs, both \mathbb{A}_v and \mathbb{B}_v exhibit a sparse pattern, leading to performance improvement through the strategic omission of OHMMA instructions using predication operations. Predication operations, a commonly employed technique in GPGPU computing, enable the skipping of specific instruction executions. In our approach, we utilize population count instructions (POPC), which are widely supported in GPGPU architectures for counting the number of "1" bits in binary numbers, to set predication bits. The count of "1" bits in the bitmaps of \mathbb{A}_v and \mathbb{B}_v serves as an indicator of the number of element-wise multiplications required in each row/column. As illustrated in Fig. 19, consider the computation for Set 3 as an example. We count the bitmaps of \mathbb{A}_v and \mathbb{B}_v to identify that

TABLE II
SUPPORTED DATA TYPE AND WARP SHAPE

A/B	C/D	OTC shape	Warp Shape	A/B-sparsity
32-b	32-b	m4n4k4	m16n16k16	$\langle 0\%, 25\%, 50\%, 75\% \rangle$
32+16-b 16+16-b	32-b	m8n4k4	m16n16k16	$\langle 0\%, 50\% \rangle$ or $\langle 0\%, 25\%, 50\%, 75\% \rangle$
16-b	16-b	m8n8k4	m32n16k16	
16+8-b 16+16-b	16-b	m8n8k4	m32n16k16	
8-b	16-b	m8n8k4	m32n16k16	
8-b	8-b	m16n8k4	m32n32k16	

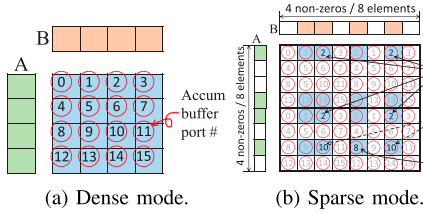


Fig. 20. Accum. buffer's access pattern.

the sparse multiplication necessitates 19 out of 7 multiplications in each row/column. In our design, each OHHMA instruction encompasses an 8×8 condensed sparse outer product multiplication. Consequently, we enable OHHMA0/1/2 by configuring predication bits while bypassing OHHMA3/4/5/6/7 for Set 3.

To meet the diverse precision requirements of modern AI applications, GPGPU architectures [24], [25], [27] have evolved to support multiple data types. This diversity in data type bit-length presents challenges for micro-architecture design due to the static nature of buffer sizes, such as cache line and vector register lengths, as well as the corresponding data paths required for data movement. To tackle these challenges, we have delineated a variety of warp-level sparse wmma instructions that accommodate an array of tensor shapes, thereby supporting major data types including FP-32/16/8, TF-32, and Int-32/16/8, detailed in Table II. These instructions are tailored to optimize the placement and movement of input and output tensors within the buffers and data paths of the OTCs.

2) *Accumulation Buffer*: The accumulation buffer has two modes, a *dense* mode, and a *sparse* mode. In dense mode, the accumulation buffer configures each read/write port directly connected to each output from EEOP units. In sparse mode, a large amount of partial matrix is generated (e.g., 32×16 FP16 for the warp-tile in SpWMMA). We extend the accumulation buffer to a multi-bank memory. Since the four Octets are working concurrently, we design four accumulation buffers private to each Octet, each with 1 KByte ($32 \times 16 \times 2$ Bytes). This setting also improves data locality and releases the pressure to memory accesses. After all four Octets finish their computation in the private buffer, their output matrix will be merged with element-wise accumulation and written back from OTC to warp register. Furthermore, the gather-accumulate-scatter method, discussed in Section III-B4, requires random access to multiple banks. We design an operand collector to schedule bank reads and writes to optimize the effective bandwidth.

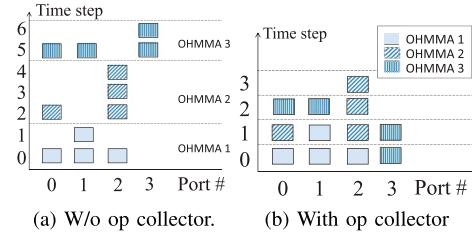


Fig. 21. Memory access schedule optimization.

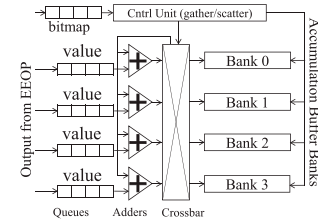


Fig. 22. Accumulation buffer design.

a) *Dense mode*: Fig. 20(a) shows an example of the EEOPs' outputs memory access to the accumulation buffer port. For simplicity, we use a 4×4 example. Since one OHHMA instruction is issued per cycle, the accumulation buffer uses 16 ports (e.g., the numbers in circles) for each EEOP output (e.g., elements in the blue matrix).

b) *Sparse mode*: Fig. 20(b) provides an illustration of the memory access pattern within the accumulation buffer during the execution of SpWMMA Instruction. We consider an $8 \times 8 \times 1$ warp tile with both input vectors having a 50% sparsity. Under these conditions, OHHMA continues to generate 16 outputs per cycle. However, accumulating these outputs can lead to numerous bank conflicts since they are distributed randomly across the partial matrix, as indicated in Fig. 20(b).

To address this challenge, we propose the integration of a small operand collector into our accumulation buffer, enhancing memory bandwidth utilization. The operand collector, a technique utilized in NVIDIA's GPU microarchitecture, enables the overlapping of source operand readings from register file banks among multiple instructions. As depicted in Fig. 21, this operand collector efficiently combines non-conflicting memory accesses from various instructions, resulting in a substantial increase in memory throughput. Fig. 22 provides an overall design overview of our accumulation buffer, incorporating the aforementioned operand collector. This design can seamlessly support both dense and sparse outer-product operations, with the sparse mode being automatically activated through the SpWMMA API.

VI. EVALUATION

We undertake an extensive set of experiments encompassing diverse micro-benchmarks and DNN models to comprehensively assess the efficacy of our software and hardware design. Our investigation delves into four primary aspects: 1) We examine the effectiveness of bitmap-based im2col in mitigating

TABLE III
DETAILS OF OUR EVALUATED SPARSE DNN MODEL

Models	Pruning Scheme	Dataset	Accuracy
VGG-16	AGP [44]	ImageNet	88.86% (top 5)
ResNet-18		ImageNet	86.46% (top 5)
Mask R-CNN		COCO	35.2 (AP)
BERT-base encoder	MP [16], [33]	SQuAD	83.3 (F1 score)
Llama	Wanda [37]	WikiText	6.15 (ppl)

the decoding overhead. 2) We gauge the extent to which our SpGEMM can enhance performance across various sparsity ratios. 3) We measure the acceleration achieved in diverse neural network layers. 4) We evaluate the extent of hardware overhead introduced in terms of hardware area. Our evaluation results demonstrate that our design yields substantial performance improvements with minimal hardware overhead.

A. Experimental Setup and Methodology

Simulation Platform We employ Accel-Sim [18], a cycle-accurate simulator built upon GPGPU-Sim [19]. This simulator offers a versatile front-end architecture, optimized cache and shared memory models, and notably enhances simulation precision compared to its predecessors. In our simulations, we model an A100 GPU [24]. To accommodate SpWMMA instructions, we integrate a cycle-accurate tensor core model, which aligns with our hardware design outlined in Section V. We expand the simulator’s front-end capabilities to support our instruction extensions, as depicted in Fig. 19.

Baselines We select CUTLASS [26] and cuDNN [6] for dense GEMM and Conv operations. CUTLASS is an open-source GEMM library known for high-performance levels. cuDNN [6] is a widely utilized vendor-optimized library for DNNs. In our comparisons for SpGEMM and SpCONV, we consider two baselines: the vendor-optimized sparse matrix library cuSparse [23] and Sparse Tensor Core [43] adapted to A100 with 75% sparsity. To ensure fair comparisons, our SpGEMM and SpCONV implementations adhere to the same loop tiling and software computation pipeline as CUTLASS [26].

DNN Models and Pruning We evaluate our algorithms using various types of DNN models, including 1) three widely-used CNN models: VGG-16 [34], ResNet-18 [14], and Mask R-CNN [13]; 2) two transformer models: BERT-base [7] encoder, a representative and well-known attention-based model; and a large language model (LLM), Llama [38], developed by Meta AI. Llama stands out as one of the leading LLM models, excelling in various NLP benchmarks. Notably, Llama comes in four versions, each with varying parameters, ranging from 7 billion to 65 billion. In this case, we use Llama-13B version.

We fine-tune and prune the CNN models with Automated Gradual Pruner (AGP) [44] on Distiller [45]. We use the fine-pruned BERT-base encoder model [16], [33] on the SQuAD task. Unlike CNN models, BERT encoder and RNN models usually have high sparsity on only weights but not feature maps. We prune Llama-13B model with Wanda’s methodology on Wikitext-2 [3] dataset, which investigates 50% weight sparsity

TABLE IV
NORMALIZED IM2COL TIME COMPARISON USING A TYPICAL CONVOLUTION LAYER FROM RESNET-18 (FEATURE MAP H/W=56, FILTER H/W=3, IN/OUT CHANNEL=128) UNDER DIFFERENT SPARSITY RATIOS

Sparsity (%)	0	25	50	75	99	99.9
Dense Im2col	1	1	1	1	1	1
CSR Im2col	101.3	67.1	45.2	14.5	4.7	1.2
Bitmap Im2col	8.31	6.87	4.73	2.5	1.5	1.1

patterns, both structured and unstructured, to uncover acceleration opportunities on Sparse Tensor Cores. For the purposes of demonstrating the advantages of our Outer-Product Tensor Core, we employed an unstructured pruning technique. Note that our work does not affect the model accuracy because we do not propose any new pruning algorithm. Table III summarizes the sparse model accuracy, which is consistent with previous pruning works. The detailed layer-wise activation and weight sparsity ratios are listed in Fig. 24.

B. Performance of Bitmap-Based Im2col

To assess the performance of our bitmap-based im2col, we conducted a comparative evaluation against both dense im2col and CSR-encoded im2col. CSR (Compressed Sparse Row) encoding is chosen for comparison as it represents one of the most commonly used sparse matrix encoding techniques. We implemented these three im2col algorithms using the PyTorch ATEN library [30] and utilized a typical convolution layer from ResNet-18 as the basis for comparison. The evaluation involved measuring the execution times of these algorithms and normalizing the results to the dense im2col case. We varied the feature-map sparsity from 0% to 99.9% and summarized the outcomes in Table IV.

The findings demonstrate a significant performance advantage of our bitmap-based im2col, particularly when compared to CSR-encoded im2col across various sparsity levels. Notably, our approach achieves an order-of-magnitude improvement in execution time when the sparsity ratio is below 50%. Only when the sparsity ratio reaches exceptionally high levels, such as 99.9%, does CSR-encoded im2col manage to attain a roughly comparable performance level to our bitmap-based im2col. This performance gap arises because CSR encoding introduces two additional data-dependent memory reads for each non-zero data access, while bitmap encoding efficiently compresses non-zero data offsets into bits, substantially reducing operational intensity in the im2col process.

C. Performance of SpGEMM

We conducted a performance evaluation of our SpGEMM, comparing it with CUTLASS, cuSparse, and Sparse Tensor Core [43]. Our assessment focused on the execution time of multiplying matrices sized $4096 \times 4090 \times 4090$ across various sparsity ratios. For cuSparse, we maintained a 99% sparsity in matrix B and varied matrix A’s sparsity from 90% to 99.9%, noting that cuSparse exhibited significant slowdowns when matrix

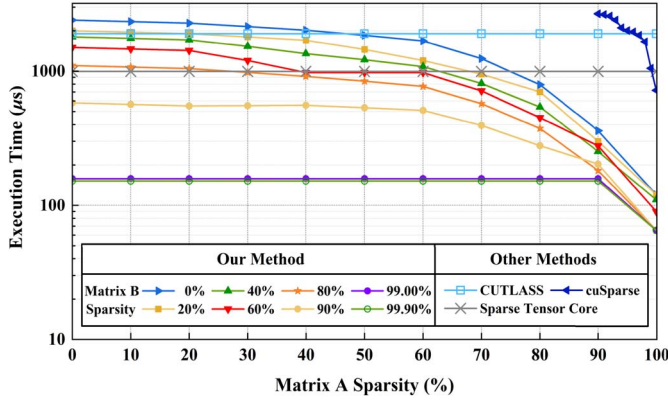


Fig. 23. Performance comparison of SpGEMM on the CUTLASS baseline. cuSparse only outperforms the baseline when the sparsity is large ($>95\%$). CSR-based Sparse Tensor Core cannot fully exploit dual-side sparsity. Our SpGEMM achieves a much higher speedup and also supports a very wide range of sparsity of matrices A and B.

A's sparsity was below 90%. The results, depicted in Fig. 23, led to some key insights.

Firstly, cuSparse appears unsuitable for accelerating sparse neural networks. Despite matrix B's high sparsity of 99%, cuSparse only outperformed CUTLASS (the dense case) when matrix A's sparsity exceeded 95%, indicating its limited utility in practical scenarios where extremely high sparsity is uncommon. Secondly, Sparse Tensor Core [43] consistently delivered around $1.9\times$ speedup over CUTLASS, attributable to its design focus on a fixed pruning ratio of 75%. However, this approach fails to capitalize on the potential sparsity in the other matrix, restricting its acceleration capability.

In contrast, our bitmap-based dual-side SpGEMM effectively leverages the sparsity in both matrices. For instance, with a 99% sparsity in matrix B, our approach reached a $12.0\times$ speedup over CUTLASS, even when matrix A had no sparsity. This speedup increased dramatically to $29.2\times$ when matrix A's sparsity was 99.9%, markedly outperforming cuSparse by $13.9\times$. Moreover, our SpGEMM surpassed CUTLASS even when matrix B's sparsity was 0, provided matrix A's sparsity exceeded approximately 25%. Thus, our SpGEMM achieves significant acceleration over dense cases across a broad spectrum of sparsity scenarios.

D. Performance of Real Neural-Network Inference

We assess the real neural network inference using the five aforementioned DNN models. For CNN models, we conduct performance comparisons across five scenarios: 1) *Dense Explicit*: dense GEMM (CUTLASS) with explicit im2col. 2) *Dense Implicit*: dense GEMM (cuDNN) with implicit im2col. 3) *Single Sparse Explicit*: Sparse Tensor Core [43] with explicit im2col. 4) *Single Sparse Implicit*: our SpCONV method, exploiting weight sparsity. 5) *Dual Sparse GEMM*: our dual-side sparsity method. For the Transformer-base models, which do not use im2col, we evaluate performance across three cases: 1) *Dense GEMM*: dense GEMM on CUTLASS. 2) *Single Sparse*

GEMM: Sparse Tensor Core [43]. 3) *Dual Sparse GEMM*: our dual-side sparsity method.

Fig. 24 shows the layer-wise and full-model speedup of the five DNNs. We select a set of representative layers for brevity because the rest layers have the same shape. For CNN models, we normalize the speedup against the *Dense Implicit* method, which outperforms *Dense Explicit* thanks to optimized im2col convolution operations. Additionally, *Single Sparse Explicit* [43] exhibits improved performance over *Dense Explicit* by leveraging weight matrix sparsity, though it doesn't consistently outperform *Dense Implicit*, showing speedup values ranging from $0.78\times$ to $1.74\times$ (with an average of $1.36\times$). Our approach, *Single Sparse Implicit*, capitalizes on weight matrix sparsity and bitmap-based implicit im2col, frequently outperforming *Dense Implicit*. It achieves an average speedup of $1.92\times$ (ranging from $0.63\times$ to $4.5\times$). Leveraging dual-sided sparsity and bitmap-based implicit im2col, our *Dual Sparse Implicit* method stands out by achieving a substantial speedup of $1.25\times$ to $7.49\times$ over *Dense Implicit*. On average, it reaches a speedup of $4.38\times$, surpassing *Single Sparse Explicit* [43] by $2.22\times$. Furthermore, Fig. 24 demonstrates that our method closely approaches the theoretical upper bound in certain CONV layers. Estimating the upper bound accurately remains challenging due to non-zero distribution dependencies. Smaller speedups observed in some layers, such as ResNet-18 layer 4-4, result from their limited sizes, where performance is constrained by data movement.

In the context of transformer-based NLP models, we benchmark speedup against *Dense GEMM*. Notably, *Single Sparse GEMM* [43] consistently outperforms *Dense GEMM* but exhibits a modest speedup, ranging from $1.10\times$ to $1.48\times$, with an average of $1.3\times$. However, when applied to BERT models, our approach demonstrates substantial superiority over *Single Sparse GEMM*, achieving a remarkable speedup ranging from $3.62\times$ to $8.45\times$. Our method's average speedup stands at $5.1\times$, which is $4.25\times$ higher than that of *Single Sparse GEMM*. This substantial improvement arises because Sparse Tensor Core [43] primarily accelerates SpGEMM with a rigid 75% sparsity limit. Conversely, our pruned BERT-base encoder model [33] boasts over 90% weight sparsity, a condition where our method thrives. Regarding Llama sparse pruning, we follow Wanda [37] 4:8 structured pruning approach for *Single Sparse GEMM*, allowing it to leverage A100's sparse tensor core primitive. Meanwhile, we've implemented *unstructured* pruning for *Dual Sparse GEMM* as our approach imposes little sparsity pattern constraints. Both methods achieve a 50% sparsity rate, but the unstructured pruning, with dual-side sparse GEMM acceleration, delivers higher model quality, boasting 6.15 vs. 7.40 perplexity (ppl). Recall the example in Fig. 6, demonstrating that our approach transcends fixed-ratio limits thanks to our innovative sparse tiling technique. Furthermore, it is worthwhile to mention that high-end GPUs face significant memory wall challenges when processing LLMs with low batch sizes, especially during the context generation stage [38]. As memory access time becomes the bottleneck, both traditional and newly proposed Sparse Tensor Cores exhibit similarly sub-optimal performance. Notably, the bitmap encoding introduced

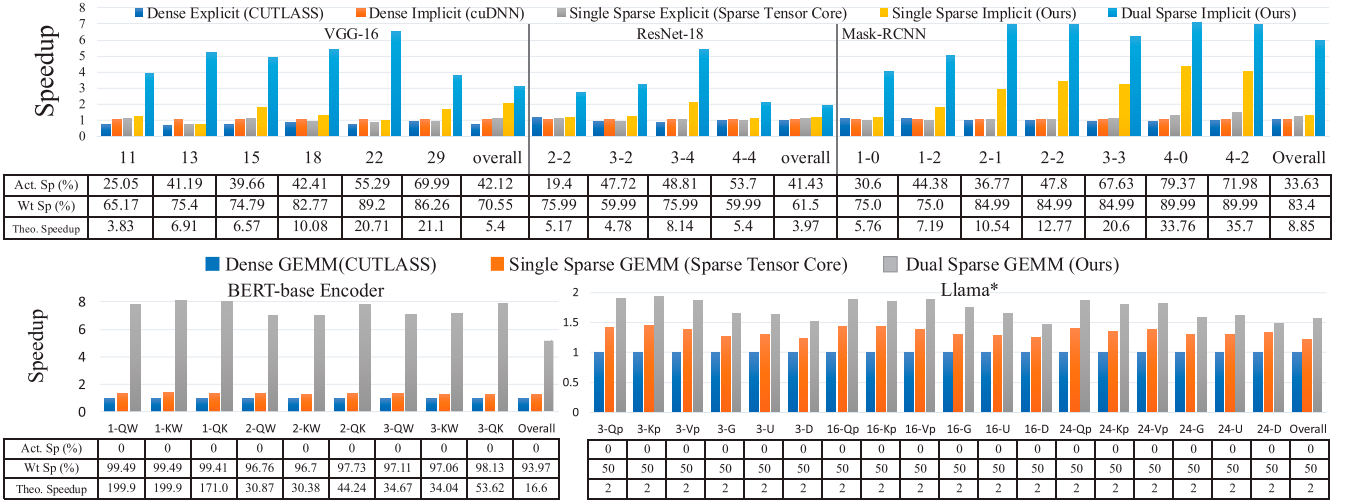


Fig. 24. Model-inference performance comparison for different layers in the five DNN models. Note that the theoretical speedup is a loose upper bound, e.g., $100\times$ speedup on 99% sparsity. *For Llama sparse pruning, Wanda [37] proposes three pruning schemes, 2:4 pruning, 4:8 pruning, and 50% unstructured pruning. We use 4:8 pruning for Single Sparse GEMM, and unstructured pruning on Dual Sparse GEMM. Among them, the unstructured pruning (with DSTC acceleration) has the best model performance on the perplexity metric (ppl).

TABLE V
AREA AND POWER OVERHEAD ESTIMATION

Module Name	Area Overhead (mm^2 , 12 nm)	Power Consum. (W, 12 nm)
Float Point Adders	0.252	4.72
Tie Index Aligner	0.809	0.22
Accumulation Operand Collector	1.51	0.46
Shared Accumulation Buffer	10.331	1.08
Total overhead on A100	12.902 (1.56%)	6.48 (2.59%)

in this study offers some advantages over traditional methods. For example, an 8-bit sparse weight, shaped $m \times n$, requires at least $2 \times (1/2 \times m \times n)$ bits for a 2:4 STC encoding [24], but only $1/8 \times m \times n$ bits for bitmap encoding, resulting in less overhead on reading the sparse matrix.

E. Hardware Overhead

Finally, we evaluate the hardware overhead and power consumption of shared buffers and queues using CACTI 7 [2] with 22 nm process technology and scale them to 12 nm [36]. We estimate Tile Index Aligner, Accumulation Operand Collector and Float Point Adders overheads and energy consumption in RTL implementation. As shown in Table V, our design introduces a total hardware overhead of 12.902 mm^2 , which is 1.56% of the whole A100 die area of 826 mm^2 , and it consumes an additional 6.48 W that is 2.59% of A100's 250 W TDP.

VII. CONCLUSION

In this paper, we demonstrate the potential for significant speedup in both SpGEMM and SpCONV operations on GPU Tensor Cores with minimal hardware extensions. Our approach hinges on the synergy between the outer product of matrix multiplication and bitmap-based sparse encoding, enabling the full utilization of dual-side sparsity to enhance the efficiency

of GEMM and implicit im2col. Importantly, our design is versatile, accommodating a wide range of sparsity ratios, and it outperforms state-of-the-art baselines by up to one order of magnitude, all while incurring minimal hardware overhead. These results pave the way for the next significant performance leap in future GPUs.

ACKNOWLEDGMENTS

The authors thank the anonymous reviews for their thoughtful comments and suggestions.

REFERENCES

- [1] A. F. Agarap, "Deep learning using rectified linear units (ReLU)," 2018, *arXiv:1803.08375*.
- [2] R. Balasubramanian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optimiz.*, vol. 14, no. 2, pp. 1–25, 2017.
- [3] J. Bradbury, S. Merity, C. Xiong, and R. Socher, "Quasi-recurrent neural networks," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2017, pp. 1–11.
- [4] S. Cao et al., "SeerNet: Predicting convolutional neural network feature-map sparsity through low-bit quantization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 11216–11225.
- [5] S. Cao et al., "Efficient and effective sparse lstm on FPGA with bank-balanced sparsity," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2019, pp. 63–72.
- [6] S. Chetler et al., "cuDNN: Efficient primitives for deep learning," 2014, *arXiv:1410.0759*.
- [7] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
- [8] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "SparTen: A sparse tensor accelerator for convolutional neural networks," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2019, pp. 151–165.
- [9] C. Guo et al., "Accelerating sparse DNN models without hardware-support via tile-wise sparsity," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2020, pp. 1–15.
- [10] C. Guo et al., "Balancing efficiency and flexibility for DNN acceleration via temporal GPU-systolic array integration," in *Proc. Des. Automat. Conf.*, 2020, pp. 1–6.

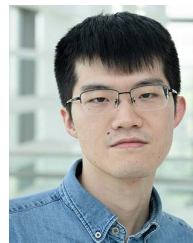
- [11] I. Hashmi and H. M. H. Babu, "An efficient design of a reversible barrel shifter," in *Proc. 23rd Int. Conf. VLSI Des.*, Piscataway, NJ, USA: IEEE Press, 2010, pp. 93–98.
- [12] K. Hazelwood et al., "Applied machine learning at Facebook: A datacenter infrastructure perspective," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 620–629.
- [13] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2961–2969.
- [14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
- [15] K. Hegde et al., "ExTensor: An accelerator for sparse tensor algebra," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2019, pp. 319–333.
- [16] "Huggingface," Accessed: Jan. 02, 2025. [Online]. Available: https://github.com/huggingface/block_movement_pruning#fine-pruned-models
- [17] N. Kalchbrenner et al., "Efficient neural audio synthesis," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 2410–2419.
- [18] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: an extensible simulation framework for validated GPU modeling," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 473–486.
- [19] J. Lew et al., "Analyzing machine learning workloads using a detailed GPU simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 151–152.
- [20] X. Liu, Y. Liu, B. Yin, H. Yang, Z. Luan, and D. Qian, "SpAMM: Optimizing large-scale sparse approximate matrix multiplication on sunway taihulight," *Frontiers Comput. Sci.*, vol. 17, no. 4, 2023, Art. no. 174104.
- [21] L. Lu and Y. Liang, "SpWA: An efficient sparse winograd convolutional neural networks accelerator on FPGAs," in *Proc. 55th ACM/ESDA/IEEE Des. Automat. Conf. (DAC)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 1–6.
- [22] A. Mishra et al., "Accelerating sparse deep neural networks," pp. 1–14, 2021, *arXiv preprint*.
- [23] M. Naumov, L. Chien, P. Vandermersch, and U. Kapasi, "Cuspars library," in *Proc. GPU Technol. Conf.*, 2010, pp. 1–93.
- [24] Nvidia, "Nvidia a100 tensor core architecture," NVIDIA, Tech. Rep., 2020.
- [25] Nvidia, "Nvidia h100 tensor core architecture," NVIDIA, Tech. Rep., 2022.
- [26] C. Nvidia, "Cutlass library," *NVIDIA Corporation, Santa Clara, California*, vol. 15, no. 27, 2008, Art. no. 31.
- [27] T. NVIDIA, "V100 GPU architecture. The world's most advanced data center GPU," Tech. Rep., NVIDIA, 2017, Art. no. 108.
- [28] S. Pal et al., "OuterSPACE: An outer product based sparse matrix multiplication accelerator," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 724–736.
- [29] A. Parashar et al., "SCNN: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 27–40, 2017.
- [30] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.
- [31] M. A. Raihan, N. Goli, and T. M. Aamodt, "Modeling deep learning accelerator enabled GPUs," in *Proc. Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 79–92.
- [32] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, "SBNNet: Sparse blocks network for fast inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8711–8720.
- [33] V. Sanh, T. Wolf, and A. M. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 20378–20389. [Online]. Available: <https://papers.nips.cc/paper/2020/file/ea15aaba768ae4a5993a8a4f4fa6e4-Paper.pdf>
- [34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations (ICLR)*, Y. Bengio and Y. LeCun, Eds., 2015, pp. 1–14.
- [35] N. Srivastava, H. Jin, J. Liu, D. Albonese, and Z. Zhang, "MatRaptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 766–780.
- [36] 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.
- [37] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, "A simple and effective pruning approach for large language models," 2023, *arXiv:2306.11695*.
- [38] H. Touvron et al., "Llama: Open and efficient foundation language models," pp. 1–27, 2023, *arXiv:2302.13971*.
- [39] H. Yang, S. Gui, Y. Zhu, and J. Liu, "Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit. (CVPR)*, 2020, pp. 2178–2188.
- [40] Z. Yao, S. Cao, W. Xiao, C. Zhang, and L. Nie, "Balanced sparsity for efficient DNN inference on GPU," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 5676–5683.
- [41] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 11, pp. 2072–2085, Nov. 2019.
- [42] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "SpArch: Efficient architecture for sparse matrix multiplication," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2020, pp. 261–274.
- [43] 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 *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2019, pp. 359–371.
- [44] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," in *Proc. 6th Int. Conf. Learn. Representations (ICLR)*, Vancouver, BC, Canada, 2018, pp. 1–10.
- [45] N. Zmora, G. Jacob, L. Zlotnik, B. Elharar, and G. Novik, "Neural network distiller: A Python package for DNN compression research," 2019, *arXiv:1910.12232*.



Chen Zhang received the Ph.D. degree in EECS from Peking University, Beijing, China, in 2017. He is an Assistant Professor with Shanghai Jiao Tong University. He served as a Senior Researcher with Microsoft Research Asia and a GPGPU Architect with Alibaba. His research interests include computer architectures and heterogeneous computing for cloud & edge AI systems. He received FPGA 2015 Best Paper Nomination, TCAD 2019 Donald O. Pederson Best Paper Award, MICRO 2022 Top-picks Honorable Mention.



Yang Wang received the graduate and Ph.D. degrees from the University of Electronic Science and Technology of China, in 2014 and 2022. He is a Researcher with Microsoft Research Asia. His research interests include computer architecture, efficient neural network inference, and data center networking.



Zhiqiang Xie is currently working toward the Ph.D. degree with Stanford University. His research endeavors are centered on enhancing the observability and efficiency of computer systems, with a specific emphasis on cloud computing and machine learning systems.

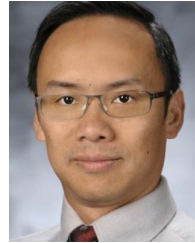


Cong Guo received the B.Sc. degree from Shenzhen University, China. He is currently working toward the Ph.D. degree in computer science with the Department of Computer Science and Engineering of Shanghai Jiao Tong University, China, under the supervision of Dr. Jingwen Leng. His research interests include computer architecture, high-performance computing, and AI accelerator design.



Paper Award.

Yunxin Liu (Senior Member, IEEE) received the B.S. degree from Shanghai Jiao Tong University (SJTU), the M.S. degree from Tsinghua University, and the Ph.D. degree from University of Science and Technology of China (USTC). He is a Guoqiang Professor with the Institute for AI Industry Research (AIR), Tsinghua University. His research interests include mobile computing and edge computing. He received MobiSys 2021 Best Paper Award, SenSys 2018 Best Paper Runner-up Award, MobiCom 2015 Best Demo Award, and PhoneSense 2011 Best



Society Edward J. McCluskey Technical Achievement Award (2021), and the IEEE CAS Society Industrial Pioneer Award (2023).

Yuan Xie received the B.S. degree in electronic engineering from Tsinghua University, and the M.S. and Ph.D. degrees in computer engineering from Princeton University. Currently, he is a Chair Professor with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology. He is a Fellow of ACM, and a Fellow of AAAS. He has a rich industry experience with IBM, AMD, and Alibaba Group. He is a recipient of many awards, including the NSF CAREER Award (2006), the IEEE Computer



Jingwen Leng (Member, IEEE) is a Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include intelligent computer system design for the artificial intelligence, with the focus on performance, energy efficiency, and reliability. His work has received best paper award or nomination at venues/conferences including IEEE Micro Top Picks, DAC, and PACT.



Ru Huang received the Ph.D. degree in microelectronics from Peking University, Beijing, China, in 1997. She is a Professor with Peking University. She is also serving as the President of Southeast University, Nanjing, China, since 2022. She is an Elected Academician of the Chinese Academy of Science, an Elected Member of TWAS Fellow. Her research interests include nano-scaled CMOS devices, ultra-low-power new devices, new device for neuromorphic computing, emerging memory technology, and device variability/reliability.



non-CMOS devices, DTCO, and emerging technologies for applications such as hardware security and new-paradigm computing.

Zhigang Ji received the B.Eng. degree in electrical engineering from Tsinghua University, in 2003, the M.Eng. degree in microelectronics from Peking University, in 2006, and the Ph.D. degree in microelectronics from Liverpool John Moores University, in 2010. In 2020, he joined Shanghai Jiaotong University, where he currently holds the position as a Professor in Nanoelectronics and the Director of LEMON Lab. He has authored or co-authored over 200 scientific papers, including IEDM and VLSI. His research interests include nanoscale CMOS and