CPSAA: Accelerating Sparse Attention Using Crossbar-Based Processing-In-Memory Architecture

Huize Li, Member, IEEE, Hai Jin, Fellow, IEEE, Long Zheng, Member, IEEE, Xiaofei Liao, Member, IEEE, Yu Huang, Member, IEEE, Cong Liu, Student Member, IEEE, Jiahong Xu, Student Member, IEEE, Zhuohui Duan, Member, IEEE, Dan Chen, Student Member, IEEE, and Chuangyi Gui, Student Member, IEEE

Abstract—The attention-based neural network attracts great interest due to its excellent accuracy enhancement. However, the attention mechanism requires huge computational efforts to process unnecessary calculations, significantly limiting the system’s performance. To reduce the unnecessary calculations, researchers propose sparse attention to convert some dense-dense matrices multiplication (DDMM) operations to sampled dense–dense matrix multiplication (SDDMM) and sparse matrix multiplication (SpMM) operations. However, current sparse attention solutions introduce massive off-chip random memory access since the sparse attention matrix is generally unstructured. We propose CPSAA, a novel crossbar-based processing-in-memory (PIM)-featured sparse attention accelerator to eliminate off-chip data transmissions. 1) We present a novel attention calculation mode to balance the crossbar writing and crossbar processing latency. 2) We design a novel PIM-based sparsity pruning architecture to eliminate the pruning phase’s off-chip data transfers. 3) Finally, we present novel crossbar-based SDDMM and SpMM methods to process unstructured sparse attention matrices by coupling two types of crossbar arrays. Experimental results show that CPSAA has an average of 89.6×, 32.2×, 17.8×, 3.39×, and 3.84× performance improvement and 755.6×, 55.3×, 21.3×, 5.7×, and 4.9× energy-saving when compare with GPU, field programmable gate array (FPGA).

Index Terms—Attention mechanism, domain-specific accelerator, processing-in-memory, resistive random access memory (ReRAM).

I. INTRODUCTION

ATTENTION-BASED neural network shows accuracy leaps in machine learning applications, e. g., natural language processing (NLP) [10] and computer vision [8]. Different from the commonly used convolutional neural network (CNN) or recurrent neural network (RNN) models, Transformer [30] adopts a pure attention-based neural network to better identify the dependencies between tokens of the input sequence. Following this design, Transformer and its variants achieve great accuracy improvement in NLP tasks [10], such as machine translation [30], question answering [6], etc. Attention is also widely used in computer vision tasks [8], including image classification [1], object detection [18], etc.

The vanilla attention mechanism [30] is usually implemented as dense-dense matrices multiplication (DDMM) and softmax operations. By computing an attention score matrix, the attention mechanism can pay attention to these relevant token pairs. There is overwhelming computation pressure in processing these irrelevant token pairs, leading to intolerable execution time [7]. Researchers propose sparse attention by adding a sparsity pruning phase before the attention calculation to reduce irrelevant calculations [7], [19], since most tokens in the input sequence are unrelated to the current query. There are two types of sparse attention designs, i.e., software-based and software-hardware co-design methods [19]. Software-based methods [29], [38] aim to propose various optimization algorithms to reduce computational overhead by increasing sparsity. Software-hardware co-design solutions accelerate sparse attention by taking advantage of high-parallelism hardware, such as field programmable gate array (FPGA) [16], [40] and application specific integrated circuit (ASIC) [7], [19], [32].

The above solutions only achieve limited speedups since both the sparsity pruning and attention calculation phases involve many off-chip data transfers. Emerging crossbar-based architectures are promising to solve the off-chip data transmission problem, such as resistive random access memory (ReRAM) and ReRAM-based content addressable memory (ReCAM) [12]. ReCAM is suitable for high-parallel comparison, the core operation of content-based similarity search in the attention mechanism. ReRAM is ideal for vector-matrix multiplication (VMM) operation, which has superior performance handling the DDMM operations of attention-based neural network. Utilizing the in-situ processing ability of ReRAM arrays, there emerge ReRAM-based processing-in-memory (PIM)-featured solutions to accelerate traditional neural network [27] and the dense attention mechanism [11], [36]. However, these PIM-based solutions can hardly extend to accelerate the sparse attention for the following reasons.

1) The ReRAM array’s write overhead cannot be ignored as it is in ReRAM-based CNN and RNN accelerators. Solving the ReRAM write overhead is urgent because...
many matrices in the attention mechanism cannot be reused and need to be written in runtime.  

2) The sparse attention involves the sparsity pruning phase, which is not considered by current dense attention accelerators. Using the current software-based pruning algorithm can promote the PIM-based attention accelerator. However, the software-based pruning methods have poor performance because they need to load all input matrices from the off-chip memory to the processor. Moreover, the sparsity of the attention mechanism is pretty unstructured, which will introduce lots of off-chip random memory access to the attention calculation phase.  

3) Finally, the sparse attention involves the sampled dense–dense matrix multiplication (SDDMM) and sparse matrix multiplication (SpMM) operations. Direct application of current ReRAM-based sparse methods [17], [28] to ReRAM-based SDDMM and SpMM operations will achieve inferior performance (for details to see Sections IV-C and IV-D).  

Given this landscape, we propose CPSAA, a novel Crossbar-based PIM-featured Sparse Attention Accelerator.  

1) We design the attention calculation mode to increase the parallelism of CPSAA dataflow while hiding ReRAM writing overhead.  

2) We design a novel PIM-based sparsity pruning architecture to support our calculation mode while removing the off-chip data transmissions.  

3) Finally, we couple ReRAM and ReCAM arrays and propose new PIM-based SDDMM and SpMM methods to utilize the unstructured sparsity.  

The main contributions of this article are as follows.  

1) We propose CPSAA, a PIM-based sparse attention accelerator, to speed up both the pruning and attention calculation phases of the neural network inference.  

2) We design the pruning phase as a novel PIM architecture to eliminate off-chip data transfers. The attention calculation architecture involves novel ReRAM-based SDDMM and SpMM methods to increase the parallelism of sparse attention.  

3) We evaluate and compare CPSAA with state-of-the-art attention mechanism solutions. The experimental results show that CPSAA has performance improvement in speedups and energy-saving.

II. BACKGROUND AND MOTIVATION

A. Sparse Attention

The vanilla attention mechanism maps a query matrix \( Q \) and a key-value matrix pair \( K-V \) to an output matrix \( Z \). Fig. 1(a) depicts the calculation procedure of the dense attention mechanism. First, the query (\( Q \)), key (\( K \)), and value (\( V \)) matrices are obtained by multiplying the embedded input sequence \( X \) with the corresponding weight matrices, \( W_Q, W_K, \) and \( W_V \). Next, the score matrix \( S \) is calculated by multiplying \( Q \) with \( K^T \). Then, the \( S \) matrix is normalized with a rowwise softmax function \( Softmax(S) \). Finally, \( S \) is multiplied by the \( V \) matrix to get the output \( Z \).

As Fig. 1(a) shows, \( S \) reveals the relevance between the tokens of \( Q \) and \( K \) matrices. Researchers find that most tokens in \( Q \) are irrelevant to \( K \), making the \( S \) matrix inherently sparse [7]. The above fact makes it possible to utilize the sparsity of \( S \) to avoid computing these irrelevant token pairs [7]. Thus, people propose sparse attention, which adopts a sparsity pruning phase to save computational resources of the attention calculation phase

\[
G[i, j] = \text{binarize}(\tilde{S}[i, j], \theta) = \begin{cases} 
1, & \text{if } \tilde{S}[i, j] \geq \theta \\
0, & \text{otherwise}.
\end{cases}
\]  

SANGER [19] proposes a prediction-based pruning method and achieves high sparsity. The idea of their pruning method is to calculate the approximate score matrix \( \tilde{S} \) using a low-precision DDM operation, \( \tilde{S} = \text{Softmax}(QU^{-1}(QU(Q) \cdot QU(K^T)))/\sqrt{d} \), where \( d \) is the dimension of the input embeddings, and \( QU(x) = \text{round}(\gamma x) \) is a quantization operator that maps input buffers to low-bit by a scaling factor \( \gamma \) and a rounding bit-shift. \( QU^{-1}(\cdot) \) is the corresponding dequant operator that transforms low-bit outputs back into high precision. Then, a binarization procedure described in (1) will convert the \( \tilde{S} \) matrix to a binary mask matrix \( G \), where \( \theta \) is the threshold of this binarization function. Because the sparsity of the mask matrix is similar to the score matrix \( S \), SANGER can convert the DDM operation \( S = QU^{-1}(QU(Q) \cdot QU(K^T)) \) to SDDMM operation. As Fig. 1(b) shows, SDDMM operation generates the sparse score matrix \( S \) with three input matrices (two dense matrices \( Q, K \), and one sparse mask matrix), which can utilize the mask matrix to avoid calculating the zero-value (white cells) of the \( S \) matrix. Using the sparse \( S \) matrix, people can convert the DDM operation \( Z = S \cdot V \) to the SpMM operation. Fig. 1(b) shows the visualization of the SpMM operation, which is a VMM operation between a sparse matrix \( S \) and a dense matrix \( V \).

B. ReRAM and ReCAM Basics

Fig. 2(a) shows one ReRAM cell, which is composed of a top electrode, a metal oxide layer, and a bottom electrode. Following the Kirchhoff’s current law, ReRAM crossbar array can process VMM operation efficiently [33], denoted as \( I_n = \sum_{m=0}^{N} V_m \times G(m,n) \), as Fig. 2(b) shows. Utilizing this high-parallel in-situ VMM operation, several ReRAM-based neural network accelerators [27] and graph processing accelerators [22, 28, 42] have been proposed.  
The ReCAM array architecture is shown in Fig. 2(c), which is composed of lots of two transistors and two memristors (two
transistors and two memristors (2T2R)) ReCAM bit-cells [12]. One ReCAM bit-cell contains a couple of ReRAM cells. One ReCAM array contains a Key register, a ReCAM cells array, drivers (DRV), and tag registers (TAG). ReCAM arrays can perform vector-scalar comparisons in parallel [12]. The TAG will latch the “1” signal if one row matches with the Key register.

C. Related Work

Table I lists the recent studies of the attention mechanism. Different from SANGER’s classification, we divide these works into three categories, i.e., software-based, software-hardware co-design, and the new adding hardware-based. The software-based methods can be further divided into static and dynamic sparsity patterns. The static sparsity pattern can only get coarse-grained sparsity for data dependent [2], [38]. The dynamic sparsity pattern can achieve higher-sparsity conditioned on individual input samples during runtime, but their unstructured sparsity increases the random memory access overhead [3], [41].

Software-hardware co-design methods adopt more efficient hardware, such as FPGAs and ASICs. FTRANS [16] is an efficient acceleration framework for transformer-based NLP. Zhang et al. proposed a novel pruning method and design the FPGA-based sparse attention accelerator [40]. A³ [7] can accelerate the attention mechanism with algorithmic approximation and hardware specialization. SpAtten [32] leverages token sparsity, head sparsity, and quantization opportunities to reduce redundant computation and random access. SANGER [19] presents a prediction-based pruning algorithm to reduce unnecessary calculations while designing the “splitting and packing” algorithm to reduce the massive random memory access overhead. DOTA [23] proposes a dynamic weak connections detector to avoid unnecessary attention calculation and promote sparse attention.

Although these co-design solutions can relieve the random memory access by designing hardware-specific algorithms, many off-chip data transmissions exist because of their separate memory and processor architecture. Using SANGER as an example, the splitting and packing algorithm can reduce random memory access by converting the unstructured sparsity to fine-grained structured sparsity. However, the benefit of the fine-grained structured sparsity comes at the cost of the dynamically configured control signals, which are highly complex to scheduling processing elements and memory access in runtime. In addition, SANGER introduces off-chip random access and data dependency to their sparsity pruning phase. Thus, no current accelerators can solve the off-chip random memory access problem elegantly.

We also list current hardware-based PIM-featured dense attention accelerators in Table I. These solutions use high-parallel ReRAM arrays to significantly reduce the latency of DDMM operations [11], [36]. However, since the sparsity of the attention matrices is not considered, current PIM-based attention accelerators need to calculate all five DDMM operation steps, wasting lots of computational resources on irrelevant tokens. Therefore, how to design a PIM-based sparse attention accelerator and reducing unnecessary calculations become the key to achieve further accelerations.

D. Motivation

We Find that Current Sparse Attention Accelerators Have Many Off-Chip Data Transmissions: We design experiments on two sparse attention accelerators, i.e., SANGER and DOTA, to confirm this statement. Fig. 3 presents the operations ratio of SANGER [19] and DOTA [23] running five real-world datasets (details to see Section V). The ratio of the operation comes from breaking down the response time to the matrix generation (MA-GE) and the attention mechanism calculation (AT-CA). To assess the impact of off-chip memory access, we further break down the MA-GE and AT-CA to the processor execution time (MA-GE-P and AT-CA-P) and the memory

---

### TABLE I

**CURRENT ATTENTION MECHANISM SOLUTIONS**

| Features                  | Hardware-based | Software-hardware co-design | Software-based |
|---------------------------|----------------|-----------------------------|----------------|
| Spar. pattern             | dense          | static                      | static         |
| Pruning pl.               | dynamic        | dynamic                     | dynamic        |
| Attention pl.             | -              | CPU/GPU                     | CPU/GPU        |
| Sparsity visualization    | unstructured   | coarse-grain structured     | coarse-grain structured |
| Regularity                | -              | coarse-grain structured     | fine-grain structured |
| Speedup                   | high           | low                         | medium         |

**Table II**

**CPSAA: ACCELERATING SPARSE ATTENTION**

---

**Fig. 2.** (a) One ReRAM cell, (b) ReRAM crossbar architecture, and (c) ReCAM array architecture.
access time (MA-GE-M and AT-CA-M). The memory access time is obtained by subtracting the average kernel runtime from the total execution time.

First, the Overhead of the Mask Generation and Attention Calculation Phases Both Cannot Be Ignored: Fig. 3 shows the MA-GE takes an average of 17.9% (14.3% in DOTA) response time, while the AT-CA takes 82.1% (85.7% in DOTA). The calculation overhead of the pruning phase is smaller than the attention calculation since the pruning phase of SANGER and DOTA uses low-precision computing.

Second, Most of the Overhead of the Mask Generation Phase Comes From the Off-Chip Memory Access: The MA-GE-M takes an average of 94.6% (92.7% in DOTA) response time while the MA-GE-P takes 5.4% (7.3% in DOTA). The reason is that the MA-GE of SANGER (DOTA) needs to access and from the off-chip memory, which involves many off-chip data transfers and takes lots of response time.

Third, the Attention Calculation Also Spends a Lot of Time on Off-Chip Data Transfers: The AT-CA-M takes an average of 71.2% (63.5% in DOTA) response time while the AT-CA-P takes 28.8% (36.5% in DOTA). That is, because all the input matrices, , , and , need to be sent from the off-chip memory to the ASIC-based processor. These off-chip data transmissions greatly increase access latency and further hurts performance. The massive off-chip data transmissions motivate us to design a novel sparse attention accelerator to simultaneously accelerate MA-GE and AT-CA with pretty low-off-chip random access.

III. CALCULATION MODE

In conventional ReRAM-based CNN and RNN accelerators, the write overhead of their weight matrices can be ignored because these matrices can be reused for different inputs. Unfortunately, the pretty high-ReRAM write overhead of the and matrices in the attention mechanism cannot be ignored because these matrices vary for different input sequences. Even worse, the write operation works serial with the VMM operation and greatly increases the latency, just as the calculation mode of ReBERT [11] shown in Fig. 4(a). ReBERT uses the same calculation mode as SANGER, which has pretty high-VMM parallelism since it can get , , and in serial. However, the VMM operation has to sequentially get the quantized (i.e., mask matrix). ReTransformer should get first; then use the quantized to get the quantized ; finally, to get quantized . Therefore, the generation of (DDMM operation) becomes the critical path of the mask generation.

1) Data Dependency: As Fig. 4(b) shows, ReTransformer has to sequentially get matrix with high-data dependency, while , , and matrices could be parallel computed in ReBERT.

2) Poor Performance When Extending to Mask Generation: Things get worse in sparse attention because there is a mask generation phase. Suppose ReTransformer wants to get the quantized (i.e., mask matrix). ReTransformer should get first; then use the quantized to get the quantized . Hence, the generation of Q (DDMM operation) becomes the critical path of the mask generation.

3) Data Dependency Between the Mask Generation and Attention Calculation: The mask generation of ReTransformer must wait for its attention calculation (i.e., data dependency). Because ReTransformer must generate and matrices to get the mask matrix. Therefore, the mask generation of ReTransformer cannot work concurrently with its attention calculation, i.e., ReTransformer does not see the potential of parallel execution between mask generation and attention calculation

\[
S = Q \cdot K^T = (X \cdot W_Q) \cdot (X \cdot W_K)^T \quad \text{(2)}
\]

\[
S = (X \cdot W_Q) \cdot \left( W_K^T \cdot X^T \right) = X \cdot \left( W_Q \cdot W_K^T \right) \cdot X^T. \quad \text{(3)}
\]

We develop a novel calculation mode to solve the above challenges, as Fig. 4(c) shows. We find that the above three challenges are the victims of the data dependency of . To unbind the data dependency of , we precalculate the weight matrix as performing , and we prestore and in ReRAM arrays. The calculation of (2) can be converted to (3) due to the combination law of matrix multiplication. Based on (3), we can get the score matrix by performing two VMM operations, i.e., .
IV. CPSAA

A. Overview

Fig. 5 shows the overview architecture of CPSAA, which contains several Tiles, and each Tile has two main parts, i.e., the peripheral components (red-dotted rectangle B) and ReRAM arrays. The peripheral components (PC) contain a controller (CTRL), several Buffers, Quant and De-Quant Units (QU and DQU), one softmax unit (SU), one Binarization Unit (BU), and two ReCAM arrays worked as Scheduler. The ReCAM arrays are classified as read-only array (ROA) and write-enable arrays (WEA). The parameters obtained from pretraining are evenly distributed to the read-only arrays in each tile of CPSAA. The tiles assigned to the same attention head communicate with each other through on-chip bus. Because real-world applications usually need multiple attention layers working together, we also show the input matrix from the previous layer and the output matrix to the next layer, which is the off-chip data transmission managed by the data transfer controller (DTC A). We use the circled color numbers to show the CPSAA dataflow, which is corresponded to the numbers in Fig. 7. The functions of all components are as follows.

CTRL: The controller will generate control signals for all components in one Tile.

Buffers: The input buffer (IB) will store the input embeddings, i.e., X matrices. The crossbar buffer (CB) will store the matrices generated at running time. The address information table (AIT) will record the address information of matrices in the ReRAM arrays, which will be useful when we need to know the location of a specific submatrix.

QU and DQU: The Quant Unit will quantize the matrix to a low-precision representation (via QU(-) function), and the Dequant Unit will convert the low-precision representation back to high precision (via QU(-1) function). The architecture of the QU is shown in Fig. 6(a), and the DQU is designed in the inverse process.

SU and BU: The Softmax Unit will perform the softmax function, and the detailed architecture of the SU is the same as A3 [7], shown in Fig. 6(b). The input of SU is matrix S, which is calculated by multiple crossbar arrays in one tile. While crossbar arrays calculate matrix S in different tiles, CPSAA will gather these results to generate S. The Binarization Unit will convert a given matrix to the “01” matrix, which is performed by a binarization comparator.

ReCAM Scheduler: The ReCAM arrays will store the mask matrices, which will be used to generate control signals for the ReRAM-based VMM operations.

ROA: The read-only arrays will preserve those matrices that can be reused for different inputs, i.e., QU(W5), W5, and Wv. The ROA contains several array groups (AG), and we set a read-only mark for the ROA.

WEA: The write-enable arrays will store these matrices that are generated in the runtime, i.e., QU(XT), XT, and V.

AG: The Arrays Group (B) is the basic storage and computing units, which contains one shift and add unit (S+A), one input register (IR), one output register (OR), one analog-digital-converter (ADC), and several ReRAM arrays.

B. Dataflow

The color-circled numbers in Fig. 7 show two CPSAA visualization dataflows, i.e., the pruning phase’s dataflow (blue-circled numbers) and attention calculation dataflow (yellow-circled numbers). To illustrate the dataflow more clearly, we divide the CPSAA dataflows into four steps, as Fig. 7 shows. Step1 is a PIM-based pruning method to generate the mask matrix. Step2 contains the calculation of $M = X \cdot W_5$, $V = X \cdot W_v$, and the write of the matrix $X^T$. (Noting that Step1 and Step2 can run in parallel since we remove the datapath dependency of $Q$ between these two steps). Step3 contains the SDDMM operation $S = M \cdot X^T$ and the write of matrix V. Step4 performs the SpMM operation $Z = S \cdot V$. It is worth noting that Step2-Step4 is the detailed version of the computational mode in Fig. 4(c).

Pruning Dataflow: The pruning algorithm of SANGER needs to use the full-precision Q and K matrices to generate the mask matrix, which makes the pruning phase wait for the calculation of these intermediate matrices Q and K. To avoid the waiting of the pruning phase (i.e., Step1 waiting for Step2), we fine-tune the mask generating algorithm as (4), where $Bi(\cdot)$ is the binarization function and $Soft(\cdot)$ is the softmax function, and the $QU(X)$, $QU(W_5)$, and $QU(X^T)$ denotes the low-precision $X$, $W_5$, and $X^T$. Our new pruning algorithm can use the input embedding matrix and the weight matrix to calculate the mask matrix, which does not need to use the results of Step2, i.e., $M$ or $V$.  

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
As Step1 in Fig. 7 shows, we prestore the low-precision $QU(W_S)$ in ROA. When an input matrix $X$ arrives at CPSAA, the system will send $X$ to the QU and get $QU(X)$ (1). Then, $QU(X^T)$ will be written to WEA (3'), and a VMM operation will be performed between $QU(X)$ and $QU(W_S)$ to get $QU(M)$ (3). Next, the system will perform a VMM operation between $QU(M)$ and $QU(X^T)$ to generate $QU(S)$ (3). After that, $QU(S)$ will be sent to the DQU, SU, and BU to get the mask matrix (4). Finally, the mask matrix will be written to a ReCAM array (5). The sparsity of the mask is similar to the matrix $S$, so we can use the mask matrix to convert the DDMM operation $S = M \cdot X^T$ to the SDDMM operation.

Attention Calculation Dataflow: As shown in Fig. 7, the weight matrices $W_S$ and $W_V$ are prestored in ROA. At the beginning, CPSAA receives and stores an input embedding matrix $X$ to the Input Buffer (1). Then, the following three phases will be performed in parallel: 1) CPSAA will write $X^T$ to WEA (3'); 2) CPSAA will send $X$ to the IR via AIT to perform the DDMM operation $M = X \cdot W_S$ (2.3); and 3) CPSAA will perform the DDMM operation $V = X \cdot W_V$ to generate matrix $V$ (2.3). After that, CPSAA can use mask matrix to perform the SDDMM operation $S = M \cdot X^T$ to get the score matrix $S$ while writing $V$ to WEA (4). After CPSAA performs the softmax function on $S$ (5), CPSAA will use the mask matrix to perform a SpMM operation $Z = S \cdot V$ to get the final result (6.7).

C. SDDMM Method

CPSAA can use the mask matrix to convert the DDMM operation to SDDMM operation. However, current SDDMM solutions, such as DOTA [23] and SANGER [19], can hardly extend to ReRAM-based sparse attention for the following reasons.

1) Vectorwise Unmatched With Matrixwise: DOTA and SANGER adopt vectorwise parallelism to utilize the spatial locality and reduce the random memory access, i.e., dividing the matrix into single vectors for processing. ReRAM naturally has matrixwise parallelism (spatial locality failure). Thus, it is not feasible to map a vectorwise approach to matrixwise hardware.

2) Less Flexible Matrix Storage in ReRAM: The input matrix $Q$ and $K$ can both be reordered and scheduled in DOTA. However, CPSAA can only reorder and schedule $Q$ because the $K$ matrix must be fixed in the crossbar arrays. The failure of scheduling $K$ makes the overall scheduling methods of DOTA and SANGER fail in ReRAM.

3) Difficult to Remove All Random Memory Access: The SDDMM scheduling methods of DOTA and SANGER cannot eliminate the random memory access because they cannot access every nonzero value without access zero-values. Therefore, they can never achieve their ideal performance when extending to ReRAM. Three main challenges led to no available ReRAM-based SDDMM method, i.e., vectors binding, fixed $K$, and random memory access.

Here, we use the $4 \times 128$ $Q$ and $K$ matrices as the example to explain our SDDMM acceleration method. As Fig. 8(b) shows, the elements of the $Q$ matrix are marked from "a" to "p," where $a$ is a row vector with 32 32-bit numbers. The most popular ReRAM-based mapping strategy (used in ReBERT and ReTransformer) of $K$ uses 32 arrays to store each bit of 32-bit $K^T$ (one array for one bit $K^T$). This mapping binds all vectors of $K^T$ and makes it impossible to
utilize the vectorwise parallelism. So we map all bits of one vector into the same ReRAM array as Fig. 8(c) shows. Each ReRAM array has 32 rows and 32 columns, and each ReRAM array stores 32 32-bit numbers, with each row storing one number. We further adopt a ReCAM scheduler to generate the control signals for SDDMM operation, which can eliminate the random memory access of $Q$ as Fig. 8(a) shows. Although $K^T$ is fixed in ReRAM arrays as shown in Fig. 8(e), our new mapping strategy can work well with the ReCAM scheduler to achieve a better-scheduling results versus DOTA. Next, we will introduce how our ReCAM scheduler controls the SDDMM operation.

**ReRAM-ReCAM Coupling:** We assume that the $4 \times 4$ matrix shown in Fig. 8(a) is the sparse mask matrix stored in a ReCAM array. In the beginning, the ReCAM array will perform the rowwise searching row-by-row as the colored arrows in Fig. 8(a) shows. We are taking the second row (green arrow line) in Fig. 8(a) as an example. The matching result of the second row indicates that the second row of $S$ has two zero elements (white cells), so we need to avoid the calculation of these zero-value cells. To locate the address of nonzero elements, the coordinates $\langle \alpha, \beta \rangle$ of these matched “1” cells will be sent to the CTRL. Then, the CTRL will utilize the row coordinate $\alpha$ to find the corresponding row of $Q$ as the green arrow in Fig. 8(b) shows. At the same time, the CTRL will utilize the column coordinate $\beta$ to find the corresponding ReRAM arrays in Fig. 8(e). As these green arrow lines in Fig. 8(e) show, the row indicated by $\alpha$ will be sent to the IR of these arrays indicated by $\beta$. The searching of the ReCAM array will iterate four times, and the other three iterations are marked with red arrow lines in Fig. 8(a). When all iterations are finished, the $Q$ matrix will be distributed to the IR of the ReRAM arrays, like these red and purple rectangles in Fig. 8(e) show. Finally, all arrays can perform the VMM operations using the topmost vector in their IR until all vectors are processed. To visually represent the advantage of our new approach, Fig. 8(d) uses the same color to present the elements which can be parallel calculated. We can see that our new method can finish the calculation of a $4 \times 4$ $S$ matrix in two cycles.

**Effect of Sparsity:** The sparsity of Fig. 8(a) is 0.5, so our novel method can reduce the latency from four cycles to two cycles. Note that the sparsity of the attention mechanism is around 0.1, so our approach can save up to $10 \times$ latency in real-world applications. In summary, our new method can finish the calculation of SDDMM with fewer latency iterations and higher parallelism. Further, our method can efficiently utilize the unstructured sparsity without complex scheduler and control signals.

**D. SpMM Method**

**Current Problems:** Many ReRAM-based SpMM solutions [17], [28] have been proposed. We classify them into two categories while extending to ReRAM-based sparse attention. One chooses to store sparse $S^T$ in ReRAM while using dense $V^T$ as the input. Another chooses to store the dense $V$ in ReRAM while using sparse $S$ as the input. A heavy wait-for-write latency occurs if we choose the first one. That is, because we cannot perform $Z = S \cdot V = (V^T \cdot S^T)^T$ until we finish the write of $S^T$ (the write of $S^T$ will become the critical datapath). Fig. 9 shows the visualization limitation of the second solution. As set in BERT [4] and A$^3$ [7], $S$ is a $320 \times 320$ sparse matrix and $V$ is a $320 \times 64$ dense matrix. With the dense matrix $V$ stored in the ReRAM arrays [Fig. 9(b)], people can perform the VMM operation row-by-row with the sparse matrix $S$ as the input [Fig. 9(a)]. Because many elements of $S$ are zero-value, current solutions choose to set zero signals for these zero values. However, these solutions will cause the array wasting problem, as Fig. 9(b) shows. Taking the first row of $S$ (marked in red frame rectangle) as an example, many rows (rows other than red) in ReRAM arrays keep idle. Therefore, the parallelism and the runtime memory utilization of ReRAM is quite low. In addition, the SpMM method in Fig. 9 can only save energy but cannot save execution cycles.

**Detailed Designs:** The sparsity of matrix $S$ is the same as the mask matrix, so it is possible to rearrange the storage of the $V$ matrix using the mask matrix. Here, we also use a $320 \times 320$ sparse $S$ matrix and a $320 \times 64$ dense $V$ matrix as an example to illustrate the details. As Fig. 10(a) shows, the mask matrix is stored in a ReCAM array (blue cells are 1 and white cells are “0”). First, the ReCAM array will perform a rowwise searching operation (red arrow line), which can find the coordinates $\langle \alpha, \beta \rangle$ of these matched 1 values (1). Then, the coordinates of these matched cells will be sent to...
F. Application-Level Designs

CPSAA is designed to eliminate the massive off-chip data transmissions in the current sparse attention accelerators. However, real-world NLP applications usually involve several attention layers working with full-connection (FC) layers, such as BERT [4]. The encoder is the basic computation unit of BERT, which contains one attention layer and several FC layers. We configure one CPSAA chip with several ReRAM-based FC layers to make up one encoder, and several encoders work together to make up the BERT-based solution.

We perform a pretraining and fine-tuning model to get the weight matrices, \( W_Q, W_K, \) and \( W_V \), for a BERT-based text classification task [4]. Then, we precalculate \( W_S = W_Q \cdot W_K^T \) and \( QU(W_S) \). We prestore the \( W_S, W_K, \) and \( QU(W_S) \) matrices in some ROAs. Note that all the above procedures are preprocessing because these matrices can be reused for different input sequences. In the beginning, an input matrix \( X \) arrives at the input buffer (IB).

1) The \( X \) matrix will be processed by the CPSAA with no off-chip memory access to generate the \( Z \) matrix.
2) The \( Z \) matrix will be sent to the FC layer to generate the result of this encoder, which is processed following the ReRAM-based dot-product method proposed in ISAAC [27].
3) Finally, the result of the previous encoder will be sent as the next encoder’s input until the final inference results are obtained.

V. METHODOLOGY

CPSAA Configurations: We employ a Python cycle-accurate simulator to model the CPSAA attention accelerator. We use the 1000GB/s on-chip interconnect (OCI) for the on-chip transfer bandwidth [9]. ReRAM’s energy consumption is obtained with 7pJ per bit as in [35]. As configured in ISAAC [27], the “cycle” in CPSAA means the time of ADC processing 32 column signals, i.e., 25 ns.

The CPSAA configurations are shown in Table II. We configure CPSAA with 64 tiles, and each tile includes one group of peripheral components, 11 ROAs, and 56 WEAAs. We use two 512 \( \times \) 512 ReCAM arrays as the Scheduler to store the mask matrices, while we use 32 \( \times \) 32 crossbar as the ReRAM array to perform the VMM operation. The crossbar configurations used in CPSAA are designed under the 32 nm process with 533 MHz clock frequency [21]. Based TaoX ReRAM cells from [21], we conduct SPICE simulation for the area and power of crossbar configuration. We use CACTI 6.5 [20] in 32nm technology to evaluate the power and area of all registers. To obtain parameters of other peripheral components, we use Cadence-simulator [13] for S/H, S+A in a crossbar. The power, area, and latency of a 2-bit precision DAC and a 2.0 mW 8b 1.0 GS/s ADC in 32-nm CMOS are obtained from [14] and [26], respectively. The crossbars are read and written in a row parallel manner, and the SET/RESET latency is 1.52/2.11 ns for single-level cell (SLC) [34]. The area and energy of SU, BU, QU, DQU, and CTRLs are established by...
SPICE circuits, too. The write endurance of the ReRAM cell can be alleviated greatly. For instance, considering up to 10^{12} ReRAM write endurance [39], CPSAA can achieve hundreds of millions (10^8) of inferences when processing billions of tokens documents (10^8 times writing).

**Benchmarks:** We use three typical attention-based NN models and get the weight matrices by pretraining, i.e., BERT-base [4], GPT-2 [24], and BART [15]. BERT-base (12 encoders) utilizes the encoder in Google Transformer to solve many NLP tasks. GPT-2 (12 decoders) uses the decoder of Transformer to process various NLP tasks. BART (six encoders and six decoders) is an optimized version of the Google Transformer. Our evaluation datasets include eight text classification tasks from the general language understanding evaluation (GLUE) benchmark [31] (CoLA, SST-2, MRPC, STS-B, QQP, MNLI, WNLI, RTE) and Stanford question answering dataset (SQuAD) V2.0 [25].

As configured in Transformer [30], BERT [4], A^3 [7], and SANGER [19], we set the model dimension d_{model} = 512 and d_{k} = d_{Q} = 64 for all workloads. In addition, model dimension (d) is not likely to vary widely since a choice of too large d can lead to a decrease in model accuracy [37]. To fit the memory space of CPSAA, we divide all datasets into some little batches, and each batch has 320 embeddings, as set in BERT [4] and A^3 [7]. Embeddings in the same batch can be parallel processed by CPSAA without off-chip data transmission, and embeddings in different batches are processed in serial with small off-chip data transfers. We use the giga operations per second (GOPS) as the performance metric (throughput) and the GOPS/W as the energy metric (energy efficiency).

**Data Overflow Prevention:** The precision of input and weight matrices are set to 32-bit. The learning rate for GLUE and SQuAD v2.0 datasets are 2e-5 and 3e-5, and the number of fine-tuning epochs is set to 3 and 2, respectively. The attention mechanism involves lots of matrix multiplication calculations, so preventing data overflow is as important as ensuring accuracy. We extract the exponential bit (EB) of the whole array to make the fraction bit (FB) remaining 32-bit fixed-point as in [5]. Thus, all VMM operations in CPSAA can be performed between 32-bit fixed-point numbers. After we get the results of the VMM operation between the FB, we can multiply the EB by the FB and get the final results.

**Comparison Platforms:** CPSAA is compared with state-of-the-art platforms as follows. We choose GPU as the NVIDIA TITAN RTX GPU@1770 MHz, 576 Tensor cores, 4608 CUDA cores, 24 GB memory, 672 GB/s memory bandwidth, and 280 Watt TDP. We choose Bigbird as the attention algorithm for the GPU [38]. We measure power consumption via nvidia-smi and execution latency using CUDA Event. We measure the performance of GPUs using PyTorch with cuBLAS 11.2. We choose the FPGA-based software-hardware co-design accelerator proposed in [40] as the FPGA platform for comparison. The configurations are also following [40]. We choose the configurations of SANGER as the ASIC-based attention accelerator for comparison [19]. We choose two architectures, ReBERT [11] and ReTransformer [36], as the PIM-featured attention accelerator for comparison. The configurations of crossbars used in ReBERT and ReTransformer are the same as CPSAA.

**VI. EVALUATION**

**A. Performance and Energy Efficiency**

Fig. 11(a) and (b) present the average speedup and energy efficiency of various models.
This work proposes a new calculation mode to hide the ReRAM write overhead while improving parallelism by removing data dependency. Therefore, we also design experiments to evaluate the effectiveness of our calculation mode, as shown in Fig. 14. We design a dense-version of CPSAA, called CPDAA, to eliminate the acceleration effect of the sparsity. Fig. 14 shows the execution time and energy consumption of ReBERT, ReTransformer, and CPDAA, which is normalized to the execution time of CPDAA. Therefore, CPSAA achieves 3.39× × and 3.84× × performance improvement against the GPU- and FPGA-based platforms. ReBERT and ReTransformer have an average of 83.7 GOPS/W and 97.1 GOPS/W energy efficiency, respectively. Thus, CPSAA has an average of 5.7× and 4.9× × energy saving compared with ReBERT and ReTransformer. We also analyze the reasons for these results as follows. ReBERT and ReTransformer are both PIM-based platforms, and they can achieve performance improvement against the GPU- and FPGA-based platforms since they do not contain the off-chip data transfers. Compared with ReBERT and ReTransformer, CPSAA designs efficient ReRAM-based SDDMM and SpMM methods, which can save lots of execution time by avoiding these unnecessary VMM operations. Moreover, the novel attention calculation mode of CPSAA can hide the pretty high-ReRAM write overhead while maintaining the VMM parallelism. Although CPSAA introduces an extra sparsity pruning phase, it can work parallel with the attention calculation and not introduce extra latency.

### B. Effectiveness of Calculation Mode

This work proposes a new calculation mode to hide the ReRAM write overhead while improving parallelism by removing data dependency. Therefore, we also design experiments to evaluate the effectiveness of our calculation mode, as shown in Fig. 14. We design a dense-version of CPSAA, called CPDAA, to eliminate the acceleration effect of the sparsity. Fig. 14 shows the execution time and energy consumption of ReBERT, ReTransformer, and CPDAA, which is normalized to the execution time of CPDAA. Therefore, CPSAA achieves 3.39× and 3.84× performance improvement against the GPU- and FPGA-based platforms. ReBERT and ReTransformer have an average of 83.7 GOPS/W and 97.1 GOPS/W energy efficiency, respectively. Thus, CPSAA has an average of 5.7× and 4.9× energy saving compared with ReBERT and ReTransformer. We also analyze the reasons for these results as follows. ReBERT and ReTransformer are both PIM-based platforms, and they can achieve performance improvement against the GPU- and FPGA-based platforms since they do not contain the off-chip data transfers. Compared with ReBERT and ReTransformer, CPSAA designs efficient ReRAM-based SDDMM and SpMM methods, which can save lots of execution time by avoiding these unnecessary VMM operations. Moreover, the novel attention calculation mode of CPSAA can hide the pretty high-ReRAM write overhead while maintaining the VMM parallelism. Although CPSAA introduces an extra sparsity pruning phase, it can work parallel with the attention calculation and not introduce extra latency.

![Fig. 12. Execution time normalized to CPSAA.](image)

![Fig. 13. Consumed energy normalized to CPSAA.](image)

![Fig. 14. Compared CPDAA with ReBERT and ReTransformer, which is normalized to CPDAA.](image)

![Fig. 15. Compared CPDAA with ReBERT and ReTransformer, which is normalized to ReTransformer.](image)
1.64× execution time against CPDAA, respectively. As for the energy consumption, ReBERT and ReTransformer consume 1.30× and 1.21× than CPDAA. To further reveal the reasons for these results, we design more experiments as follows.

We further design two metrics to compare the calculation mode of CPDAA with ReBERT and ReTransformer, i.e., the execution time of waiting for write (CPDAA-W4W) and the number of arrays for parallel VMM operation (CPDAA-P). The experimental results in Fig. 15 are all normalized to ReTransformer (ReTran-W4W and ReTran-P). ReBERT and CPDAA take 1.94× and 1.48× execution time on write operations compared with ReTransformer. In contrast, the VMM parallelism of ReBERT and CPDAA is 2.88× and 2.03× compared with ReTransformer. ReTransformer has the minimal write latency but the worst-VMM parallelism because it is designed to reduce the write overhead as much as possible. However, they turn the VMM operations that could be executed in parallel into strictly serial execution, reducing the wait time for write operations while increasing the wait time for the previous VMM operations. ReBERT has the maximal write overhead but the best-VMM parallelism because they use a write-then-calculate computational mode, which maximizes the VMM execution efficiency but takes longer to wait for write. CPDAA takes a tradeoff between ReBERT and ReTransformer, which reduces unnecessary write overhead while preserving necessary VMM parallelism.

In the experiments shown in Fig. 15, ReTransformer optimizes the write operations, but the performance of which is worse than ReBERT for the following reason. Taking into account the non-negligible write overhead, this work designs all ReRAM-based platforms to use a single-level cell (SLC) with lower-write latency and lower-write energy consumption. If a multilevel cell (MLC) with higher-write overhead is used, the advantage of ReTransformer will be revealed.

C. Acceleration of Pruning Architecture

CPSAA proposes a novel ReRAM-based PIM-featured pruning method to generate the mask matrix. Therefore, we also develop experiments to evaluate the effectiveness of our novel pruning architecture. As Fig. 16 shows, we choose five indicators to reveal the advantages of our novel method against the state-of-the-art pruning method proposed in SANGER. They are the execution time of the pruning phase (Pruning-T), the execution time of the attention calculation phase (Attention-T), the number of pruning phase’s VMM operations (VMM-N), the CTRL runtime scheduling time (CTRL-T), and the accuracy comparison (Accuracy).

The pruning phase in SANGER takes 85.1× execution time compared with our novel pruning method. That is, because our novel method eliminates the off-chip data transfers and greatly improves the computational parallelism when generating mask matrices, which can save lots of execution times. Moreover, the Attention-T of SANGER is 18.7× of CPSAA, which comes from two aspects.

1) CPSAA eliminates massive off-chip data transfers compared with SANGER.
2) CPSAA can leverage the high-parallel VMM operations of the ReRAM arrays, which can greatly reduce the latency of the VMM operations.

Fig. 16 also shows that CPSAA can save around 16.37× VMM operations when generating the mask matrix. CPSAA can generate the mask matrix directly using the input matrices, while SANGER needs lots of VMM operations to generate some intermediate matrices $Q$ and $K$ first.

Experimental results show that SANGER takes 11.4× execution time on CTRL-T when compared with CPSAA. That is, because SANGER’s splitting and packing algorithm has complex scheduling control signals for sparse $S$ and pretty high-processing elements reconfiguration overhead. CPSAA can use ReCAM Scheduler to reduce the control signals and eliminate the reconfiguration overhead by designing a ReRAM-ReCAM coupling SDDMM and SpMM methods. CPSAA also adopts a new equation using the low-precision weight matrix $W_3$ and quantified $M$ to generate the mask matrix. Therefore, it is necessary to evaluate the accuracy of our pruning method. Fig. 16 shows that CPSAA loses less than 0.2% of accuracy compared to SANGER.

D. Novel SDDMM and SpMM Designs

Compared to current PIM-based dense attention accelerators ReBERT and ReTransformer, CPSAA designs two novel ReRAM-based methods to accelerate the SDDMM and SpMM operations, respectively. Therefore, we also design experiments to reveal the effectiveness of the novel SDDMM and SpMM methods. As shown in Fig. 17, the metrics of this evaluation are the execution time and the consumed energy. The baseline is the execution time of the DDM operations (DDMM-T) and the consumed energy of the DDM operations (DDMM-E) in ReBERT. All experimental results are normalized to DDMM-T (100) and DDMM-E (100). The latency of the SDDMM and the SpMM methods are 17.5% and 0.54% of the DDMM method, respectively. The reason for the speedups of the SDDMM operation is as follows. The SDDMM approach can minimize these unnecessary VMM calculations and greatly reduce the latency by using ReCAM Scheduler to guide the generation of control signals. Our new SpMM method can significantly reduce the number of idle rows in VMM operations, greatly increasing the parallelism and reducing the latency. Fig. 17 also shows that the consumed
cannot get datasets with multiplied data volumes in real-world of input embeddings on the WNLI dataset. Because we

E. Scalability

low-SpMM latency. CPSAA design principle of using space cost in exchange for overhead. These experimental results are consistent with the

298

SpMM method. For all five datasets, our SpMM method has

memory utilization, throughput, and data replication of our
to 1. SpMM-M, SpMM-T, and SpMM-R refer to the runtime
utilization, throughput, and data replication are normalized
with different encoder layers. We design experiments to reveal the sequence lengths scal-
ability. We configure six sequence lengths, i.e., 128, 256, 512,
1024, 2048, and 4096. Fig. 20 shows the average throughput
of running all datasets. The three von Neumann accelerators,
GPU, FPGA, and SANGER, show the same trend of sequence
length scalability. The CPU-, FPGA- and SANGER-based
platforms decline noticeably when the number of encoder
layers increases. Unlike the trend in the GPU platform, the throughput of CPSAA remains stable. That is,
because more encoder layers will introduce more computa-
tional tasks, generate more intermediate matrices, and require
more memory space. Therefore, the GPU-based platform has
more random memory access while taking more execution
time. But in the PIM-based platform, such as CPSAA, more
memory space means more computational resources, and the
throughput of CPSAA will not increase as the number of
encoder layers increases.

We design experiments to reveal the sequence lengths scal-
ability. We configure six sequence lengths, i.e., 128, 256, 512,
1024, 2048, and 4096. Fig. 20 shows the average throughput
of running all datasets. The three von Neumann accelerators,
GPU, FPGA, and SANGER, show the same trend of sequence
length scalability. The CPU-, FPGA- and SANGER-based
accelerators can maintain stable throughput when the sequence
lengths ≤512. When the sequence lengths are greater than 512,
the throughput of the above platforms decreases significantly
as the sequence lengths increase. That is, because the increased
sequence length generates a large amount of intermediate data,
which exacerbates the random access and off-chip transfers of the von Neumann accelerators, thereby reducing
system efficiency. The above results are consistent with the
time complexity analysis in Section IV-E. The throughput
of the PIM-based accelerators, ReBERT, ReTransformer, and
CPSAA, do not vary significantly with sequence lengths. That
is, because the PIM-based accelerators can on-chip process all
the intermediate data without transferring them to the main
processor.

VII. CONCLUSION

We investigate current sparse attention accelerators and find
the reasons for their limited speedups. This work presents
a crossbar-based PIM-featured sparse attention accelerator,
CPSAA, to tackle current problems. We design a novel calculation mode of attention mechanism to hide the write overhead while maintaining the VMM parallelism. We also present a new PIM-based pruning method to eliminate the off-chip memory access when generating the mask matrix. To solve the challenges that all current ReRAM-based attention accelerators can hardly efficiently extend to sparse attention, we design novel ReRAM-based SDDMM and SpMM methods. Finally, we design experiments to evaluate the effectiveness of all these core designs mentioned in this work.

REFERENCES

[1] C.-F. R. Chen, Q. Fan, and R. Panda, “Crossvit: Cross-attention multi-scale vision transformer for image classification,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2021, pp. 357–366.

[2] R. Child, S. Gray, A. Radford, and I. Sutskever, “Generating long sequences with sparse transformers,” 2019, arXiv:1904.10599.

[3] B. Cui, Y. Li, M. Chen, and Z. Zhang, “Fine-tune BERT with sparse self-attention mechanism,” in Proc. Conf. Empir. Methods Nat. Lang. Process. Int. Joint Conf. Nat. Lang. Process. (EMNLP-IJCNLP), Hong Kong, China, 2019, pp. 3548–3553.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” 2018, arXiv:1810.04805.

[5] B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek, “Enabling scientific computing on memristor accelerators,” in Proc. ACM/IEEE Int. Symp. Comput. Archit. (ISCA), 2018, pp. 367–382.

[6] G. Geigle, N. Reimers, A. Rücklé, and I. Gurevych, “TWEAC: Transformer with extendable QA agent classifiers,” 2021, arXiv:2104.07081.

[7] T. J. Ham et al., “Α*: Accelerating attention mechanisms in neural networks with approximation,” in Proc. IEEE Int. Symp. High Perf. Comput. Archit. (HPCA), 2020, pp. 328–341.

[8] K. Han et al., “A survey on vision transformer,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 45, no. 1, pp. 87–110, Jan. 2023.

[9] N. P. Jouppi et al., “Ten lessons from three generations shaped google’s TPU: Industrial product,” in Proc. ACM/IEEE Int. Symp. Comput. Archit. (ISCA), 2021, pp. 1–14.

[10] K. S. Kalyan, A. Rajasekharan, and S. Sangeetha, “Ammus: A survey of transformer-based pretrained models in natural language processing,” 2021, arXiv:2108.05542.

[11] M. Kang, H. Shin, and L.-S. Kim, “A framework for accelerating transformer-based language model on ReRAM-based architecture,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 41, no. 9, pp. 3026–3039, Sep. 2022.

[12] R. Kaplan, L. Yavits, R. Ginosar, and U. Weiser, “A resistive CAM work,” in Proc. Int. Symp. Low Power Electron. Design, New York, NY, USA, 2020, pp. 175–180.

[13] J. Lin, Z. Zhu, Y. Wang, and Y. Xie, “Learning the sparsity for ReRAM: Mapping and pruning sparse neural network for ReRAM-based accelerator,” in Proc. Asia South Pacific Design Autom. Conf., New York, NY, USA, 2019, pp. 639–644.

[14] Z. Liu et al., “Swin transformer: Hierarchical vision transformer using shifted windows,” 2021, arXiv:2103.14030.

[15] L. Lu et al., “Sanger: A co-design framework for enabling sparse attention reconfigurable architecture,” in Proc. Annu. IEEE/ACM Int. Symp. Microarchit., New York, NY, USA, 2021, pp. 977–991.

[16] N. Muralimohan, R. Balasubramonian, and N. P. Jouppi, Cacti 6.0: A Tool to Model Large Caches, HP Lab., Palo Alto, CA, USA, 2009, pp. 1–25.

[17] D. Niu, C. Xu, N. Muralimohan, N. P. Jouppi, and Y. Xie, “Design of cross-point metal-oxide ReRAM emphasizing reliability and cost,” in Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD), 2013, pp. 17–23.

[18] L. Qian, “Graph processing and machine learning architectures with emerging memory technologies: A survey,” Sci. China Inf. Sci., vol. 64, no. 6, 2021, Art. no. 160401.

[19] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, “DOTA: Detect and omit weak attentions for scalable transformer acceleration,” in Proc. ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst., 2022, pp. 14–26.

[20] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” OpenAI Blog, vol. 1, no. 8, p. 9, 2019.

[21] P. Rajpurkar, R. Jia, and P. Liang, “Know what you don’t know: Unanswerable questions for SQuAD,” 2018, arXiv:1806.03822.

[22] S. Saberi, R. Lotti, K. Maﬁnezhad, and W. A. Serdijn, “Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation ADCs,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 58, no. 8, pp. 1736–1748, Aug. 2011.

[23] A. Shafiee et al., “ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” in Proc. Int. Symp. Comput. Archit. (ISCA), 2016, pp. 14–26.

[24] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, “GraphPR: Accelerating graph processing using ReRAM,” in Proc. IEEE Int. Symp. High Perform. Comput. Archit., 2018, pp. 531–543.

[25] Y. Tay, D. Bahri, L. Yang, D. Metzler, and D.-C. Juan, “Sparse sinkhorn attention,” in Proc. Int. Conf. Mach. Learn., 2020, pp. 9438–9447.

[26] A. Vaswani et al., “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 5998–6008.

[27] 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,” 2018, arXiv:1804.07461.

[28] H. Wang, Z. Zhang, and S. Han, “SpAtten: Efﬁcient sparse attention architecture with cascade token and head pruning,” in Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA), 2021, pp. 97–110.

[29] W. Wen, Y. Zhang, and J. Yang, “Wear leveling for crossbar resistive memory,” in Proc. ACM/ESDA/IEEE Design Autom. Conf. (DAC), 2018, pp. 1–6.

[30] C. Xu, D. Niu, N. Muralimohan, N. P. Jouppi, and Y. Xie, “Understanding the trade-offs in multi-level cell ReRAM memory design,” in Proc. ACM/EDAC/IEEE Design Autom. Conf., 2013, pp. 1–6.

[31] M. Yanet et al., “HyGCN: A GCN accelerator with hybrid architecture,” in Proc. IEEE Int. Symp. High Perf. Comput. Archit. (HPCA), 2020, pp. 15–29.

[32] X. Yang, B. Yan, H. Li, and Y. Chen, “ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration,” in Proc. Int. Conf. Comput.-Aided Design (ICCAD), 2020, pp. 1–9.

[33] Z. Yin and Y. Shen, “On the dimensionality of word embedding,” in Proc. Adv. Neural Inf Process. Syst. (NeurIPS), 2018, pp. 1–12.

[34] M. Zahner et al., “Big Bird: Transformers for longer sequences,” in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2020, pp. 17283–17297.

[35] F. Zahoore, T. Z. A. Zulkifli, and F. A. Khayand, “Resistive random access memory (RRAM): An overview of materials, switching mechanism, performance, multilevel cell (MLC) storage, modeling, and applications,” Nanoscale Res. Lett., vol. 15, no. 1, pp. 1–6, 2020.

[36] X. Zhang, Y. Wu, P. Zhou, X. Tang, and J. Hu, “Algorithm-hardware co-design of attention mechanism on FPGA devices,” ACM Trans. Embed. Comput. Syst., vol. 20, no. 5s, pp. 1–24, 2021.

[37] G. Zhao, J. Lin, Z. Zhang, X. Ren, Q. Su, and X. Sun, “Explicit sparse transformer: Concentrated attention through explicit selection,” 2019, arXiv:1912.11637.

[38] L. Zheng et al., “Sparsa: An energy-efficient ReRAM-based accelerator for sparse graph analytics applications,” in Proc. IEEE Int. Parallel Distrib. Process. Symp., 2020, pp. 696–707.
Huize Li (Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2022. His current research interests include computer architecture, emerging nonvolatile memory, and processing in memory.

Hai Jin (Fellow, IEEE) received the Ph.D. degree in computer engineering from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1994. He is a Chair Professor of Computer Science and Engineering with HUST. He was with The University of Hong Kong, Hong Kong, from 1998 to 2000, and as a Visiting Scholar with the University of Southern California, Los Angeles, CA, USA, from 1999 to 2000. He has coauthored more than 20 books and published over 900 research papers. His research interests include computer architecture, parallel and distributed computing, big data processing, data storage, and system security. Dr. Jin was awarded a German Academic Exchange Service Fellowship to visit the Technical University of Chemnitz, Germany, in 1996. He was awarded the Excellent Youth Award from the National Science Foundation of China in 2001. He is a Fellow of CCF and a Life Member of the ACM.

Long Zheng (Member, IEEE) received the Ph.D. degree in computer engineering from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2016. He is currently an Associate Professor with the School of Computer Science and Technology, HUST. His current research interests include program analysis, runtime systems, and configurable computer architecture with a particular focus on graph processing.

Xiaofei Liao (Member, IEEE) received the Ph.D. degree in computer science and engineering from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2005. He is currently a Professor with the School of Computer Science and Technology, HUST. His research interests are in the areas of system software, P2P system, cluster computing, and streaming services. Prof. Liao has served as a reviewer for many conferences and journal papers. He is a member of the IEEE Computer Society.

Huize Li (Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2022. His current research interests include computer architecture, emerging nonvolatile memory, and processing in memory.

Cong Liu (Student Member, IEEE) received the bachelor’s degree from Dalian Maritime University, Dalian, China, in 2018. She is currently pursuing the Ph.D. degree with the Huazhong University of Science and Technology, Wuhan, China. Her research interests include in-memory computing and ReRAM-based accelerator.

Jiahong Xu (Student Member, IEEE) received the B.S. degree from the School of Control and Computer Engineering, North China Electric Power University, Beijing, China, in 2018. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. His research interests mainly include nonvolatile memory and in-memory computing.

Zhuohui Duan (Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2022. He is currently a Postdoctoral Fellow with the School of Computer Science and Technology, Huazhong University of Science and Technology. His research interests are in hybrid memory and distributed memory pool. Dr. Duan has presented papers in several conferences, such as ASPLOS, SC, DAC, ICDCS, ICCD, and DATE.

Dan Chen (Student Member, IEEE) received the B.S. degree from the North China Electric Power University, Beijing, China, in 2018. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. His research interests focus on processing-in-memory and graph neural network.

Chuangyi Gui (Student Member, IEEE) received the B.E. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2017, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Technology. His current research interests include graph processing and graph mining systems.

Yu Huang (Member, IEEE) received the Ph.D. degree from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2022. He is currently a Postdoctoral Fellow with the School of Computer Science and Technology, HUST. His research interests focus on processing-in-memory architecture and graph processing.