Exploring Compute-in-Memory Architecture Granularity for Structured Pruning of Neural Networks

Fan-Hsuan Meng, Xinxin Wang, *Graduate Student Member, IEEE*, Ziyu Wang, Eric Yeu-Jer Lee, and Wei D. Lu, *Fellow, IEEE*

***Abstract*— Compute-in-Memory (CIM) implemented with Resistive-Random-Access-Memory (RRAM) crossbars is a promising approach for Deep Neural Network (DNN) acceleration. As the DNN size continues to grow, the finite on-chip weight storage has become a challenge for CIM implementations. Pruning can reduce network size, but unstructured pruning is not compatible with CIM, while structured pruning leads to higher neural network accuracy drop. In this work we systematically evaluate how structured pruning can be efficiently implemented in CIM systems. We show that by utilizing the inherent computational granularity in CIM operations, fine-grained structured pruning can be supported with improved accuracy and minimal hardware cost. We discuss the hardware implementation in a practical system and the expected performance in terms of accuracy, energy and effective throughput. With the proposed approach, compression ratio up to 11.1 (i.e. 9% weights remaining) can be achieved with only 0.6% accuracy drop with minimal hardware overhead in the hardware design.**

***Index Terms*— Neural network, pruning, memristor, crossbar, compute-in-memory.**

I. Introduction

DEEP neural networks have been widely adopted for Artificial Intelligence (AI) applications [1], [2], [3], [4]. CIM-based DNN accelerators offer advantages over conventional architectures by eliminating data movements of weights between the memory unit and the computation unit [5]. However, this approach requires all weights need to be stored on-chip, which becomes increasingly challenging as the size of state-of-the-art models grow exponentially over time [1], [2], [6], [7]. Pruning techniques can reduce the storage requirement by removing less important weights [8], [9], [10] but highly unstructured sparse operations cannot be efficiently computed in CIM structures. The ability to efficiently support large DNNs through pruning would significantly expand the appeal of CIM-based DNN accelerators.

One promising method of leveraging sparsity is to prepare the sparse neural network with the constrains of the targeted hardware, either through training with predefined sparsity pattern [11], [12], [13], or structured pruning co-designed with the CIM architecture. In this work, we systematically examined how CIM architecture properties can be leveraged to allow effective pruning and mapping of the network.

We identified opportunities for utilizing the computation parallelism within one vector-vector multiplication that rises from weight precision mapping on multiple devices, and separate signed and unsigned weight mapping. This allows us to decrease the effective granularity of the sparse neural network. In addition, we examined the time-multiplexing and scheduling of ADC computation within CIM architectures to achieve even finer-grained structured pruning. We demonstrated that with these techniques applied, minimal neural network accuracy lost can be achieved at high compression ratio.

Manuscript received 1 May 2022; revised 13 August 2022; accepted 15 September 2022. Date of publication 7 December 2022; date of current version 19 December 2022. This work was supported in part by the National Science Foundation under Award CCF-1900675 and in part by the Semiconductor Research Corporation (SRC) and Defense Advanced Research Projects Agency (DARPA) through the Applications Driving Architectures (ADA) Research Center. This article was recommended by Guest Editor S. Yu. *(Corresponding author: Wei D. Lu.)*

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: wluee@umich.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2022.3227471.

Digital Object Identifier 10.1109/JETCAS.2022.3227471

II. Background

A. Compute-in-Memory

The majority of DNN computation is made up of multiply-and-accumulate (MAC) operations in vector and matrix forms, i.e., Vector-Matrix-Multiplication (VMM) or Matrix-Matrix-Multiplication (MMM). This property of DNN makes CIM a promising approach by keeping weights stationary in memory and performing VMM in place in a highly parallel fashion. Many CIM architectures based on different memory technologies such as RRAM and PCM have been proposed for DNN acceleration [5], [14], [15], [16].

Below we examine DNN operation in CIM systems and opportunities to implement effective pruning techniques. A typical CIM operation of a CIM-based Processing Element (PE) is illustrated in Fig. 1. DNN operation in CIM consists of three main parts. The first is the off-line mapping of the model. Before computing, the flattened weights should be stored on the crossbar follwing a mapping schedule [15]. For inference-only applications, the weights remained unchanged in the run time, therefore this process would only be performed once. To increase inference efficiency, the weights and activations in a DNN is usually quantized for efficient inference. It has

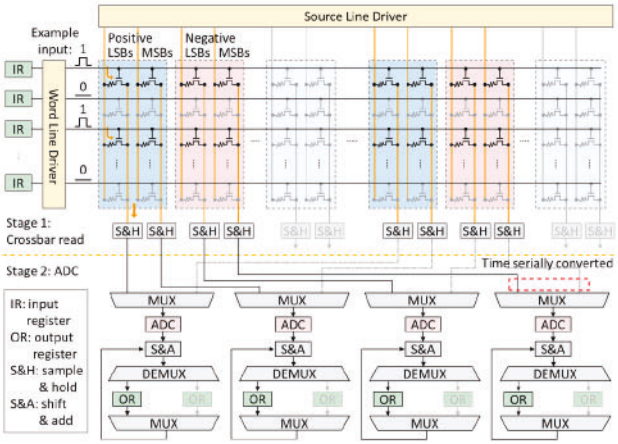

Fig. 1. A representative crossbar-based CIM PE design, showing the inputs, outputs, ADC sharing, and the input/output pipeline stages.

been demonstrated that 8-bit weights and activations can significantly reduce computation cost and memory size without too much DNN performance degradation [17], [18], [19], [20]. However, it is currently unrealistic to store an 8-bit value in one RRAM cell, and a practical way is to use multiple cells to represent different bits of the value.

The second step is the application of inputs to the crossbar array. When computing the VMM, the vector should be encoded as input voltages (amplitude or pulse width) that will be applied to the wordlines (WLs) of the crossbar. The current collected at the end of the bitlines (BLs) will then be the accumulated, representing the encoded input activation multiplied with the conductance that represent the encoded weights. Bit-serial input voltage encoding is a common practice, that is, an 8-bit activation will be represented as 8 consecutive input pulses. Unlike a digital MAC engine, in a crossbar implementation the input is shared by WLs and is expected to generate outputs at all BLs, which puts clear constraints on model mapping, MAC operations and pruning techniques that it supports.

The third step is the conversion of the analog outputs of the crossbar to digital values to be sent to the rest of the system. This is typically achieved via Analog-to-Digital Converters (ADCs). Due to the large size of ADCs, it is impractical to design PE with one (high-resolution) ADC for each RRAM column, and typically several columns will share one ADC instead. In this case, the analog outputs at the bitline will first be latched with a Sample-and-Hold (S&H) unit and sequentially applied to the shared ADC. Afterwards, Shift-and-Add (S&A) units are used to accumulate the partial outputs from the different weight bits and inputs bits to produce the final VMM results (Fig. 1).

B. Sparse Neural Networks

State-of-the-art DNN models are usually very large in size with large number of weights [1], [2], [3], [4], [21], making them expensive to fit on a CIM chip. Additionally, as the models evolve rapidly, it becomes difficult for a CIM chip with fixed on-chip memory to be "future proof", limiting their application potential.

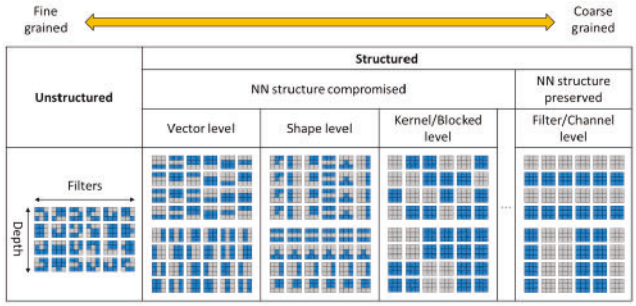

Fig. 2. Various types of pruning methods. In general, finer pruning granularity leads to better accuracy after pruning.

DNN pruning has been widely studied and some techniques have been shown to be effective in reducing the model size without significant DNN performance drop [8], [9], [10]. Different types of pruning schemes reported are illustrated in Fig. 2. Simply pruning away individual connections that are least significant will result in an unstructured sparse DNN model. This type of model has very fine granularity and can usually effectively maintain the model accuracy post pruning [8]. In contrast, structured pruning prunes away groups of weights that are defined by certain grouping rules. The grouping rules usually reflect the targeted hardware for the model to be computed on. For example, vector-wise pruning prunes away vectors in the weight kernel and is targeted to run on hardware accelerators with parallel vector computations [22]. Another example of structured pruning is shape-wise pruning, where the same pixel locations in different weight kernels are grouped and pruned together, therefore, the shape of the sparse kernel would have the same shape across kernels after pruning [23], [24]. Pruning with larger blocks in kernel level have also been demonstrated [25], [26], [27]. A unique case of structured pruning is filter/channel pruning, which prunes away entire filters (output channels). After filter/channel pruning, the DNN model structure is preserved, the pruned model will simply be a smaller DNN model instead of a sparse DNN [28], [29], [30].

Typically, the finer the group granularity during pruning, the better the post-prune model performance in terms of accuracy. However, fine granularity pruning typically leads to highly unstructured models that increases computation overhead due to sparsity indexing and makes parallel processing difficult. Therefore, balancing the trade-off between model accuracy and hardware overhead becomes an important task, particularly for accelerators such as CIM systems that have very clear hardware limitations.

C. Pruning Neural Networks for CIM

Prior works on pruning DNNs in CIM falls into two main categories, mapping schemes and structured pruning, as illustrated in Fig. 3.

The mapping approach takes an unstructured pruned model and implements various technique to map the weights on crossbars to achieve highest crossbar utilization. A simple

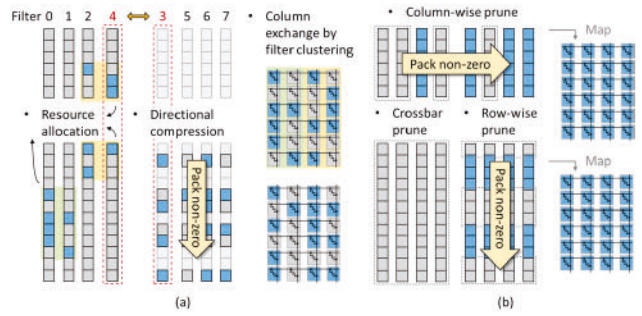

Fig. 3. Implementation of sparse neural networks in crossbar based CIM architecture with (a) mapping schemes and (b) structured pruning schemes.

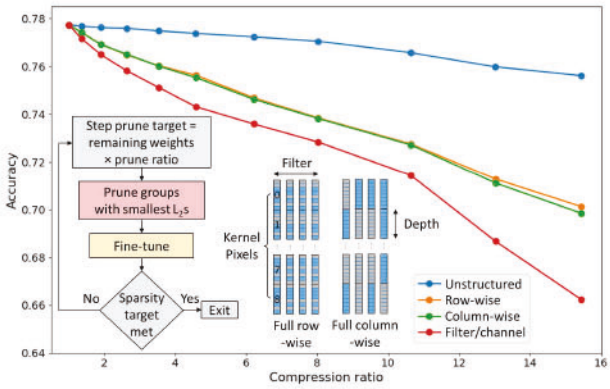

Fig. 4. Post pruning accuracy at various pruning steps for unstructured, row-wise, column-wise, and filter/channel pruning.

way of mapping irregular weights on crossbars is to take the sparsified matrix and compress it in a chosen direction by squeezing out the pruned weights in a structured manner [31], [32]. For example, SNrram splits one sparse matrix into dense sub-matrices and implements a resource allocation scheme to re-assemble the sub-matrices with the same shape into different parts of a crossbar [33]. An indexing register unit is then used to correctly accumulate the partial results. [34] proposed a mapping scheme based on k-means clustering, which shuffles the columns in the weight matrix and eliminates all-zero crossbars. However, mapping of unstructured weights to a structured format will inevitably result in underutilization of weight storage, especially when the sparsity level is high (i.e. high compression ratio after pruning). Encoding and decoding the indices for the unstructured inputs to the corresponding rows of the crossbar will also add significant hardware and latency overhead.

In CIM-constrained structured pruning, weights in a 2-Dimensional (2D) flattened matrix can be grouped in two possible directions, row-wise and column-wise, which aligns with shape-wise pruning in Fig. 2. Row-wise pruning prunes away the same pixels of multiple filters, while column-wise pruning prunes away the same pixels of multiple input channels in the channel-first mapping implementation [35], [36], [37]. In general, the groups could also be 2D, utilizing both directions. In an extreme case, [35] prunes entire crossbars, therefore requires no hardware change in the CIM unit.

In addition, [38] proposed an automatic bit-pruning technique to achieve structured bit-sparsity in neural networks.

In general, structured pruning achieves higher crossbar utilization than mapping of unstructured-pruned DNN, therefore would provide better hardware performance and will be the focus of this study.

III. Fine-Grained CIM Constrained Structured Pruning

The practical goal of pruning is to produce a compact model that can be efficiently mapped, while minimizing accuracy drop and hardware overhead. Here we systemically analyze the trade-offs between prune grouping type, sparsity, and model accuracy to find practices that will lead to near-optimal design points for CIM accelerators.

A. Pruning Method

We will use the popular network Wide ResNet (WRN) 16×8 [39] as an example, but the key characteristics and findings should be valid for other DNN models. To maintain a high model accuracy at high compression level, multiple pruning steps should be taken to gradually prune the model to a desired compression ratio (number of weights after pruning/number of weights of the original model). Structured pruning is achieved by grouping multiple weights together with a predefined scheme. In the following discussion, we will refer to the group of weights to be pruned as a prune group (PG). We evaluate the importance of each PG with their L_2 value. At each prune step, a fixed ratio of existing groups with the smallest L_2 will be pruned, followed by fine-tuning of the weights through re-training to increase the post prune model accuracy [23]. This process is then repeated until the desired compression ratio is obtained, e.g. 0.9 with only 10% of weights remaining. The inserted flow chart in Fig. 4 shows the iterative prune-and-fine-tune process.

In Fig. 4, we compare four types of pruning schemes on WRN16×8 trained with the CIFAR-100 dataset [40]. Row-wise structured pruning is done by grouping the weights at the same location in all the filters of each layer. Column-wise pruning is done by grouping all the pixels in different channels in the same group. Filter/channel prune is done by pruning an entire filter with the smallest L_2, therefore, the entire channel of the next layer.

B. Pruning and NN Model Performance Degradation

Different structured pruning schemes can be viewed as pruning with different PG designs. To evaluate the effect of PG design on model accuracy, a collection of different pruning schemes with different PG sizes and aspect ratios are selected. Fig. 5(a) shows an example of the 2D weight sparsity maps of one layer in WRN, after pruning with the same PG size of 64 but different aspect ratios, where the aspect ratio is defined as the width of the PG divided by the height of the PG. The accuracy results shown in Fig. 5(b) are averaged over five pruning experiments for each case. It can be observed that with large PG size the aspect ratio can slightly affect the results,

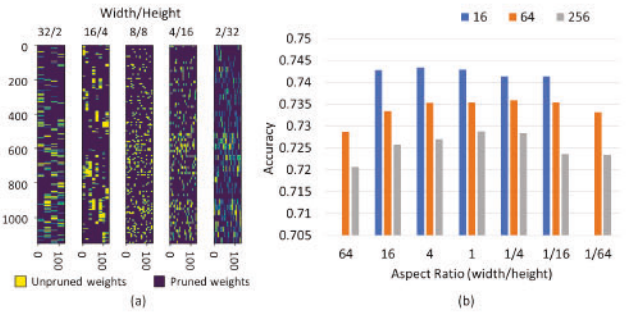

Fig. 5. Structured pruning experiment results for different PG sizes and aspect ratios, showing (a) weights of one WRN layer with fixed PG size and different aspect ratios, and (b) post-pruning accuracy of models with different PG sizes and different aspect ratios.

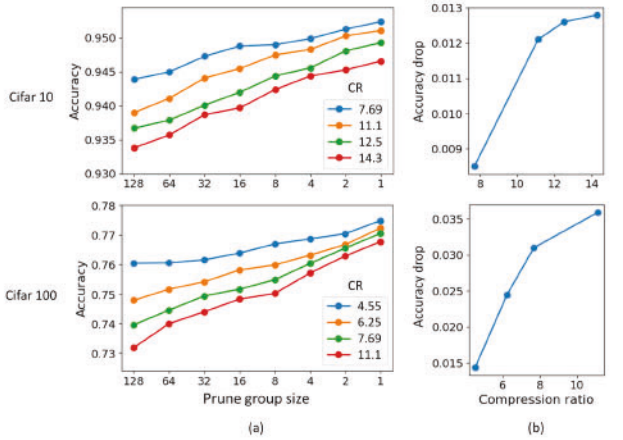

Fig. 6. (a) Accuracy for models pruned with different PG sizes and different target compression levels for WRN trained on CIFAR-10 and CIFAR-100. (b) Accuracy degradation between unstructured pruning (equivalent to group size=1) and PG size 128 for different compression levels.

i.e. pruning with very narrow or wide PGs tends to suffer more accuracy lost. However, for smaller group sizes, which is more relevant since they tend to lead to less accuracy drop, the results for all aspect ratios between 16:1 to 1:16 are very similar, suggesting the aspect ratio, i.e, the pruning direction and shape, is not an important factor. On the other hand, the PG size has a much more significant effect on the model accuracy, with smaller PG size leading to higher accuracy after pruning. As a result, in the following discussion we will focus on techniques based on row-wise pruning, which could be directly supported by the CIM hardware in a finer granularity.

Fig. 6 shows the post pruning accuracy of row-wise pruning under different PG sizes and target compression levels for CIFAR-100 and CIFAR-10 [40]. Again it could be observed that the PG size largely affects the post pruning model accuracy. In addition, the effect of PG size is more significant at higher compression levels, i.e. when more weights need to be removed. These results show that the most important design point for preserving high model performance after pruning is to increase the pruning granularity, particularly if high compression ratios are desired.

IV. Computational Granularity for Fine-Grained Pruning

In row-wise pruning, the parallel computational nature across different crossbar columns is preserved and the output would be correctly accumulated. This limits the granularity during pruning to the whole crossbar row for crossbar-based CIM implementations, which can cause significant accuracy drop. To solve such problem, we will exploit the inherit computational granularity in CIM to improve the pruning granularity that leads to improved model accuracy and on hardware performance (latency and throughput).

A. Fine-Grain Row-Wise Sparsity in Crossbars

As illustrated in Fig. 1, the run-time CIM operation consists of two steps, analog compute, and analog-to-digital conversion. Since the ADC circuits are usually large, it would be impractical to provide one ADC at each column [15]. Therefore, sharing schemes between crossbar columns is usually implemented. In particular, ADC sharing means that only a subset of columns will be computed and converted at a given time, offering opportunities for finer grain row-pruning without additional latency overhead. We call this technique sub-grouping within a crossbar. Additionally, since normally multiple devices are used to store one weight value, additional granularity can be leveraged in the CIM based implementation.

In the following discussions, we will refer to the group of columns in a crossbar who can operate on the same input as the computational group (CG). A CG thus forms the smallest unit for structured pruning in the CIM system, and equals to PG if a weight is stored in one device. Without pruning, the CG equals to the total number of columns in a crossbar. However, due to ADC sharing, the shared columns need to be computed serially (Fig. 1), so CG can be made smaller than the total number of columns. Ideally, all available ADCs should still be utilized to improve the parallelism, so the smallest CG size equals to the number of ADCs per crossbar. Below that, there will be ADCs that cannot be utilized. Therefore, in theory, smaller number of ADCs per PE (e.g. more ADC sharing through time-multiplexing) will create opportunities for finer grained computation. Since ADCs are typically large, more ADC sharing should also reduce the area of the PE. On the other hand, latency can significantly suffer if one ADC is shared between too many columns. These different factors have to be balanced in the PE design.

Below we consider how the number of ADC sharing and CG size will affect latency when designing a fine-grained computation process. To maximize hardware utilization and reduce latency, the crossbar read and ADC conversion processes should be pipelined. After a set of inputs is applied to the WL of the crossbar, it would take a couple cycles for the signals to stabilize and be latched with the S&H units [41]. In the next pipeline stage, the ADCs will convert the signals serially among the shared columns.

A crossbar read and ADC conversion pipeline for the case of no sub-grouping, (i.e. CG = number of columns) is shown in Fig. 7(a). Typically, the ADC operates at higher frequency than the system frequency, but since one ADC needs to convert outputs from multiple columns, the ADC stage is normally the pipeline bottleneck. The cycles it takes to for the ADC(s) to

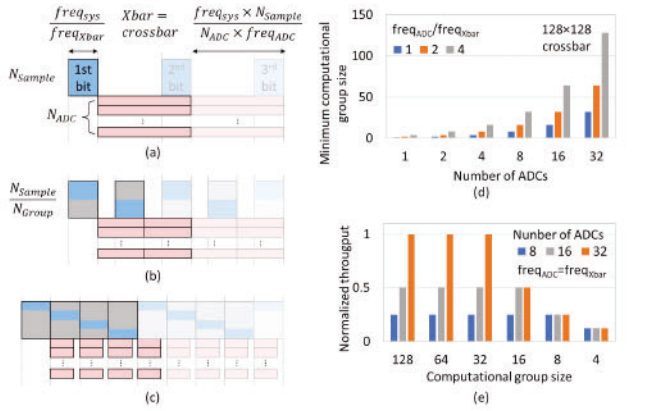

Fig. 7. Illustration of the crossbar read and ADC pipeline. (a) Case with a single ADC assigned to the crossbar with with a single CG, (b) two CGs, and (c) four CGs where the crossbar read stage can stall the pipeline following Eq. 1. (d) Minimum CG size without performance degradation for different number of ADCs and different ADC frequency; and (e) normalized throughput of different CG size and number of ADCs, with ADCs operating at the same frequency as the crossbar read frequency.

convert all the required samples would be,

$$\frac{freq_{sys} \times N_{sample}}{freq_{ADC} \times N_{ADC}} \tag{1}$$

where $freq_{sys}$ is the system operating frequency, $freq_{ADC}$ is the sampling rate of the ADC(s). N_{sample} is the number of analog signals to be converted, and N_{ADC} is the number of ADCs available for the conversion stage. Due to ADC sharing, the ADC needs to serially operate those cycles to convert all the data stored in the S&H units.

To employ sub-grouping within a crossbar, we turn off the Source Line (SL) drivers for selected columns, so that only a sub-group (i.e. CG) of columns receive input at a given time. As a result, the inputs equation 1 will be scaled by the number of sub-groups since only a smaller accumulated data need to be serially processed. However, as long as the ADC conversion stage is still dominating, the overall latency is not affected, as the total number of serially converted data remain the same, while latency due to the increase in inputs is hidden by the pipeline, as shown in Fig. 7(b). As a result, we can achieve smaller CG thus finer pruning granularity without latency cost. However, When the sub-group size becomes too small such that the crossbar read become the bottleneck, as shown in Fig. 7(c), the pipeline is stalled, and latency will be increased.

To illustrate the two constrains, consider a typical CIM case with 128 × 128 crossbars, 50 MHz system frequency. Typically, ADC sampling rate is higher than the crossbar read frequency. As shown in Fig. 7(d), the fewer the number of ADC assigned to each crossbar, the smaller the CG size can be achieved without throughput degradation. Reducing the ADC frequency would also result in smaller CG sizes. When the ADC frequency is the same as the crossbar read frequency, the theoretical minimum CG size, which equals to the number of ADCs per PE, can be obtained without latency overhead. When the CG is further reduced below the number of ADCs, throughput will be degrated, as shown in Fig. 7(e). In this example with 16 ADCs per crossbar, the smallest CG size is thus 16. This matched ADC design with RRAM read frequency has also been demonstrated to achieve good performance and cost tradeoff [41], [42], [43].

Another point to note is the factor that multiple RRAM cells are needed to store one weight value. In the case of N-bit RRAM cell resolution and 8-bit weights, one weight will require $\frac{8}{N}$ RRAM cells to store the value. This means that the equivalent VMM dimension would be the number of columns in a crossbar divided by $\frac{8}{N}$. In addition, in CIM, two columns are used to map the positive and negative weights separately, and their outputs are separately accumulate too. Since the positive and negative weights share the same input, the actual PG equals to $CG \div (2 \times \frac{8}{N})$. Therefore in an example of 128 columns in a crossbar, 8 bit weights, 2 bits per cell, positive and negative weights mapped on same crossbar, the width of the VMM would be 16, which is 8 times smaller than the crossbar width, providing another factor to support finer-grained sparsity in row-wise pruning.

B. Fine-Grained Sparsity Mapping and Indexing

In this section we discuss the implementation of the fine-grained row-wise pruned NN in CIM. To support sparsified weights, the packed weights should be stored along with a set of sparsity index. A straightforward indexing scheme is the direct indexing, where one bit is assigned to the vector to represent if the value is zero or not, as illustrated in Fig. 8(a). Another indexing method, step indexing, shown in Fig. 8(b), is done by storing the steps between each nonzero value. If the number of bits is not enough to represent a step, fake non-zero values need to be padded in between for proper sparse indexing. The step index scheme has been demonstrated to provide better performance in terms of area and power [44], in addition, this indexing scheme will result in a fixed sparsity index size regardless of the sparsity of the weights. This way the additional storage overhead for the index could be fully utilized. The overhead will also be minimal compared to the weights stored. For example, at 10 compression ratio (i.e. 10% weights remaining after pruning) and 8-bit weights, the index would only be 10

To match the correct input activations to the crossbar inputs, the full input activation vectors need to be fetched and then the corresponding values need to be selected using the weight sparsity index and fed to the proper rows. In the example above, each set of input vectors will take 8 VMM to consume, as shown in the pipeline stages in Fig. 8(c).

C. Input Buffering

A typical tiled architecture with hierarchical computational units of Compute Unit (CU)/PE is shown in Fig. 9(a) [41]. In this architecture, one CU contains multiple crossbar-based PEs. The PEs in the same CU share the same fetch and distribute logic such as finite state machines (FSMs) which are used to prepare inputs for the VMM computation and keep track of the computation progress. The input activations are buffered in an activation buffer and can be shared by multiple CUs. If a filter is mapped across different PEs and CUs, the output of the PEs are the partial sums (Psum) of the final output and should be accumulated. Psum accumulators

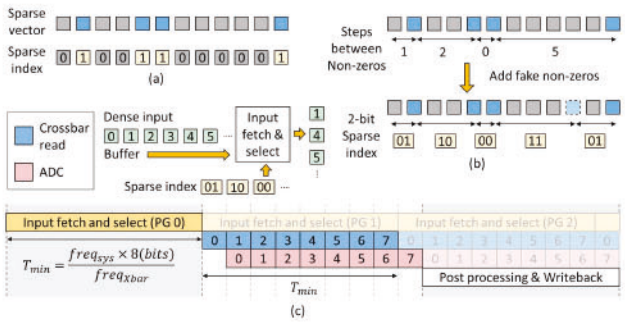

Fig. 8. (a) Illustration of direct indexing scheme, and (b) step indexing scheme. (c) Pipeline stages of input fetch, crossbar read, and ADC conversion stages.

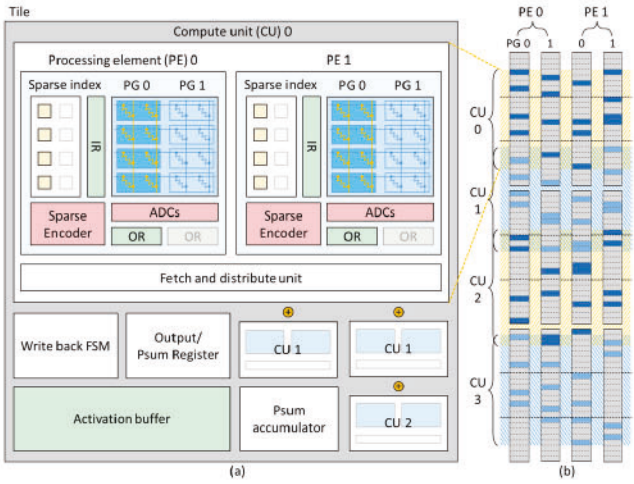

Fig. 9. (a) Illustration a tiled architecture with hierarchical computational units of Compute Unit (CU)/Processing Element (PE) that supports multiple PG within one PE. (b) A filter mapping example of 4 PGs across 2 PEs.

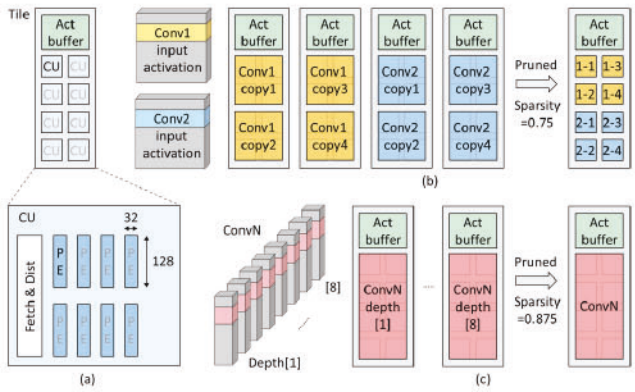

Fig. 10. (a) CU hierarchy and dimension. Weight and activation mapping illustration for (b) shallow layers with wider spatial dimensions and smaller depth, and (c) deeper layers with smaller spatial dimensions and larger depth.

and output/psum registers shared within one CU are used to accumulate the final output and a writeback unit is used to write the output to the designated CU.

A flattened weight mapping example of 4 PGs across 2 PEs is shown in Fig. 9(b). Because each CG is pruned individually, therefore the input activation they correspond to will not align, therefore, the related input required by each CUs will likely overlap. The FSM in the fetch unit needs to be designed to be more flexible to account for the arbitrary input activation range. To make sure the correct inputs are assigned to corresponding PE, the fetch unit will broadcast the inputs in the dense form to all the PEs, and each PE will encode the inputs with the weight sparsity index stored within the PE.

To map sparse NN on the CIM-based accelerator, the activation buffering should also be considered. More layers would be mapped to the same CU for a pruned model, potentially resulting in larger required activation buffer size. However, in actual practice, the increment in required buffer size is not extreme. Fig. 10(a) shows a tile/CU/PE hierarchy where there are 8 CUs in one tile and 8 PEs in one CU, each PE does one 128×8 VMM. These parameters are optimized based on convolution dimensions of typical NN models. Typically, in shallower layers, the input activation would have wider spatial dimensions and smaller depth, while the input activations of deeper layers have narrower spatial dimensions and larger depth. The MAC/weight ratio is typically larger in shallow layers too, therefore, to balance the operation of the entire NN computation, the weights of the shallower layers will often be duplicated to increase throughput, as shown in Fig. 10(b). Therefore, even with a 4-times compression ratio, number of layers mapped to the same CU will be less than 4 times. In addition, to maintain high accuracy after pruning, the shallower layers are often not pruned too aggressively. For deeper layers, the required buffer size for the dense case is lower, because the weights are mapped onto multiple CUs split by the depth, as shown in Fig. 10(c). After, pruning, the entire activation might be mapped to the same CU, but it would not require a large buffer size.

D. ADC Design Points for Pruned Neural Networks

For a crossbar with R_{Xbar} rows, b_{in} bit input voltage, and b_{cell} bit weights per cell, the number of bits at each column output, b_{out} would be,

$$b_{out} = log_2(R_{Xbar}) + b_{in} + b_{cell} \quad (2)$$

To preserve all the bits in the output, the ADC resolution should theoretically be the same as b_{out}. However, in a typical NN inference hardware, it has been demonstrated that 8-bit activation quantization is effective for high NN performance application [17], [18], [19]. Therefore, the ADC resolution could be designed to be lower than b_{out} resulting in some lost in output precision, without having too much NN model performance degradation. Since the power and area usually scales exponentially with ADC bits in many ADC types [45], a few bits of difference could result in a large performance difference.

However, high compression ratio could result in changes in weight distributions after crossbar mapping. After the pruning process, the lower valued weights are removed and the remaining higher valued weights are packed into a dense format and mapped on the crossbar. Therefore, the outputs

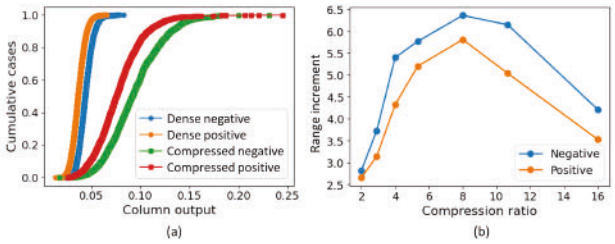

Fig. 11. (a) Column output distribution of one layer for the original model and the compressed model at 11.1 compression ratio. (b) the range increased in the compressed model comparing to the original model at various compression levels.

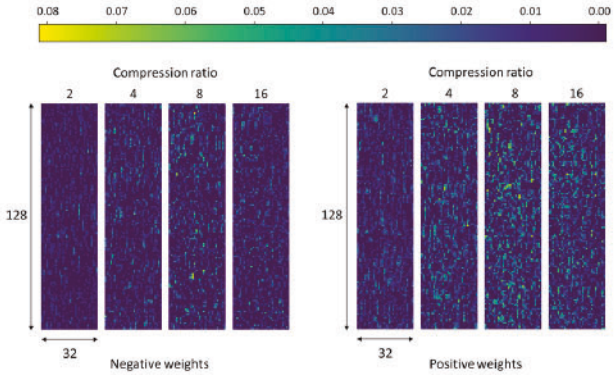

Fig. 12. Mapped weights on a single 128×128 crossbar with 4 cells representing one weight value.

at each crossbar column would have a wider distribution. Fig. 11(a) shows the output distribution of a WRN layer under 1000 test input images. The positive and negative weights are mapped separately on different crossbars for convenience and both the positive and negative output of the compressed model (compression ratio = 11.1) has wider distribution than the distribution of the original model. To quantify the changes, we define the output range increase as the compressed output range divided by original output range. It can be observed in Fig. 11(b) that the range increases more at higher compression ratio and then lowered at compression level of 8. From the weights mapped on one crossbar shown in Fig. 12, the weight values are largest at 8 compression. It is likely that at very high compression ratio, during the fine-tuning, some weights need to be lowered to account for lover strength connections.

E. Performance Evaluation

We then examine the performance of the fine-grain pruned models on CIM systems. We consider a realistic case with crossbars and ADCs both operating at 50 MHz frequency, 128 × 128 crossbar size, and 16 ADCs per crossbar.

A simulator is built to evaluate performance of the CIM-based architecture. The performance of the digital components is evaluated using parameters extracted from Verilog simulation with TSMC 22 nm process. The ADC parameters are extracted from an in-house low power ADC, designed specifically for NN acceleration [46]. RRAM parameters are projected from [47].

This work focuses on leveraging the inherent granularity in typical CIM architectures during the pruning process.

System	
Frequency	500 MHz
Technology	22 nm
Crossbar	
Frequency	50 MHz
Size	128×128
ADC	
Frequency	50 MHz
Count	16
Resolution	8 bit

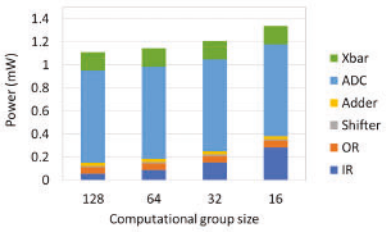

Fig. 13. Power consumption of a single PE for different CG sizes.

Computational group size	16	Prune group size	2	Bit per RRAM cell	2

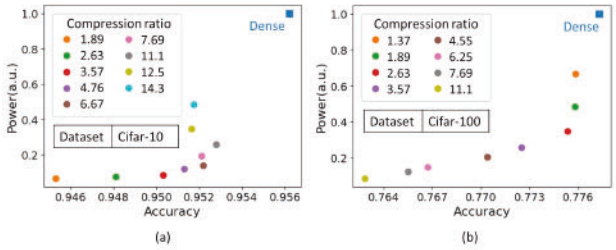

Fig. 14. Accuracy and power of the original and pruned WRN 16×8 for different compression ratios trained using the proposed technique, for (a) CIFAR-10 and (b) CIFAR-100.

Non-ideal RRAM characteristics are not a focus of the study and was not included for simplicity. We note that effects of non-ideal device and circuit behaviors have been extensively discussed in prior publications [48], [49], and may be mitigated during training [50]. To account for hardware utilization in terms of power and throughput, the NN model is manually mapped on the architecture and the performance is scaled based on the hardware utilization.

Fig. 13 shows the CIM power profile at different CG sizes. As discussed earlier, the smallest CG size that can be supported without lowered ADC utilization is 16. The power overhead per crossbar for CG size of 16 comparing to CG size of 128 is 20%, which is mostly due to the additional activation read and write stages. Fig. 14 examines the trade-off of power profile (for running the whole model) and NN accuracy post pruning, It can be observed in Fig. 14(a) that for CIFAR-10, the structurally pruned model consumes only 13% of the power, with only 0.6% accuracy drop compared with implementing the original model. For CIFAR-100, 0.8% accuracy drop can lead to 78% power reduction using the proposed technique, as shown in Fig. 14(b).

In Fig. 15, we consider the original model and the pruned models mapped to an accelerator of the same size. Since the pruned models have fewer weights, higher equivalent throughput can be obtained as shown in Fig. 15(a). The latency in Fig. 15(b) is also largely improved in the pruned models.

F. Discussion

To further analyze the area overhead for supporting sparse NN computation, we conducted area estimation using projected area numbers from literature, i.e. ISAAC [41] and SRE [32], along with area reported from Verilog simulation. The area overhead comes from two parts, the sparse encoder within

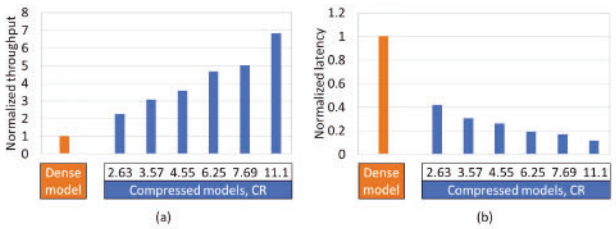

Fig. 15. Normalized (a) throughput and (b) latency of the original WRN 16×8 and pruned NN at different compression levels using the proposed technique.

each PE, and the registers storing the sparse indexes. The first do not grow with finer graduality, while the second increase linearly with the number of prune groups supported in each PE. Our analysis showed that the area overhead for supporting a single prune group, i.e., prune group size of 16 is 3.2% for a single PE, and 8.45% for maximum prune group count of 8 per PE.

Fundamentally, this work aims to identify the smallest parallel computational unit to implement structurally pruned models. Therefore, the concept could be applied to other types of CIM, for example, CIM concepts based on different types of memory, or digital CIM [51], [52]. Digital CIM architectures reported typically map different bits of a weight in different memory cells. In addition, signed and unsigned weights are also mapped separately. Therefore, these characteristics could also be used to support fine-grained sparse NN implementation as discussed earlier. However, since ADCs are not used, digital CIM may or may not share computing units among the bitlines, so this level of granularity based on shared computing units might not be utilized. Using [51] as an example, in this digital CIM implementation with 64 columns per memory macro, 4 cells are used to map a 4-bit value, and positive and negative weights are mapped separately, resulting in a total of 8 cells for a single 4-bit weight. A prune group size of 8 could thus be achieved without shared computing units among bitlines.

The fine-grained pruning technique can be applied to other DNN models on other datasets to preserve the accuracy at high compression ratio. For example, we analyzed the effects with ResNet-18 on ImageNet. We compare fine-grained pruning with prune group size of 2, and standard row-wise pruning with prune group size of 32 (128 columns and 4 cell per weight value) at compression ratio of 10.4. With the proposed fine-grained pruning, we were able to reduce the accuracy drop to 1.2% accuracy as compared to 3.9% accuracy drop for the standard row-wise pruning technique.

V. Conclusion

In this work, we demonstrated the importance of granularity in maintaining high accuracy after pruning. Since ADC takes up large areas comparing to crossbar arrays, typical CIM design implements ADC sharing. By taking advantage of this characteristic along with the fact that multiple cells are needed to store a single weight value, finer grained row-wise pruning could be supported with minimum computational and hardware overhead. Our analysis show the fine-grained structured pruning allows low-cost indexing of the inputs and minimizes input buffer size increase. The proposed approach supports aggressive pruning (over 10 compression ratio) with minimal accuracy drop, and significant throughput and latency improvements compared with the original model.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[3] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. pattern Recognit.*, Jun. 2015, pp. 1–9.
[4] J. Gu et al., "Recent advances in convolutional neural networks," *Pattern Recognit.*, vol. 77, pp. 354–377, May 2018.
[5] M. A. Zidan, J. P. Strachan, and W. D. Lu, "The future of electronics based on memristive systems," *Nature Electron.*, vol. 1, no. 1, pp. 22–29, Jan. 2018.
[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.
[7] I. Goodfellow et al., "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014, pp. 139–144.
[8] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 2, 1989, pp. 1–8.
[9] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–9.
[10] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016, *arXiv:1611.06440*.
[11] A. Ardakani, C. Condo, and W. J. Gross, "Sparsely-connected neural networks: Towards efficient VLSI implementation of deep neural networks," 2016, *arXiv:1611.01427*.
[12] S. Alford, R. Robinett, L. Milechin, and J. Kepner, "Pruned and structurally sparse neural networks," in *Proc. IEEE MIT Undergraduate Res. Technol. Conf. (URTC)*, Oct. 2018, pp. 1–4.
[13] J. Kepner and R. Robinett, "RadiX-Net: Structured sparse matrices for deep neural networks," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2019, pp. 268–274.
[14] P. Yao et al., "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, 2020.
[15] X. Wang et al., "TAICHI: A tiled architecture for in-memory computing and heterogeneous integration," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 12, pp. 559–563, Feb. 2021.
[16] A. Sebastian, M. Le Gallo, and E. Eleftheriou, "Computational phase-change memory: Beyond von Neumann computing," *J. Phys. D, Appl. Phys.*, vol. 52, no. 44, Oct. 2019, Art. no. 443002.
[17] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.
[18] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.
[19] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.
[20] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*.
[21] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 12, pp. 6999–7019, Dec. 2021.
[22] H. Mao et al., "Exploring the regularity of sparse structure in convolutional neural networks," 2017, *arXiv:1705.08922*.
[23] V. Lebedev and V. Lempitsky, "Fast ConvNets using group-wise brain damage," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2554–2564.
[24] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 1–9.

[25] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger, "CondenseNet: An efficient DenseNet using learned group convolutions," 2017, *arXiv:1711.09224*.
[26] D. T. Vooturi, D. Mudigere, and S. Avancha, "Hierarchical block sparse neural networks," 2018, *arXiv:1808.03420*.
[27] D. T. Vooturi, G. Varma, and K. Kothapalli, "Dynamic block sparse reparameterization of convolutional neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 1–8.
[28] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1389–1397.
[29] Z. Liu et al., "MetaPruning: Meta learning for automatic neural network channel pruning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Oct. 2019, pp. 3296–3305.
[30] V. Radu et al., "Performance aware convolutional neural network channel pruning for embedded GPUs," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Nov. 2019, pp. 24–34.
[31] H. Ji, L. Song, L. Jiang, H. Li, and Y. Chen, "ReCom: An efficient resistive accelerator for compressed deep neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 237–240.
[32] T.-H. Yang et al., "Sparse ReRAM engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proc. 46th Int. Symp. Comput. Archit.*, vol. 2019, pp. 236–249.
[33] P. Wang, Y. Ji, C. Hong, Y. Lyu, D. Wang, and Y. Xie, "SNrram: An efficient sparse neural network computation architecture based on resistive random-access memory," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.
[34] 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. 24th Asia South Pacific Design Autom. Conf.*, Jan. 2019, pp. 639–644.
[35] L. Liang et al., "Crossbar-aware neural network pruning," *IEEE Access*, vol. 6, pp. 58324–58337, 2018.
[36] C. Chu et al., "PIM-prune: Fine-grain DCNN pruning for crossbar-based process-in-memory architecture," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.
[37] J. Meng, L. Yang, X. Peng, S. Yu, D. Fan, and J.-S. Seo, "Structured pruning of RRAM crossbars for efficient in-memory computing acceleration of deep neural networks," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 5, pp. 1576–1580, May 2021.
[38] S. Qu, B. Li, Y. Wang, and L. Zhang, "ASBP: Automatic structured bit-pruning for RRAM-based NN accelerator," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 745–750.
[39] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016, *arXiv:1605.07146*.
[40] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," Univ. Toronto, Tech. Rep., 2009.
[41] A. Shafiee et al., "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.
[42] J. M. Correll et al., "A fully integrated reprogrammable CMOS-RRAM compute-in-memory coprocessor for neuromorphic applications," *IEEE J. Explor. Solid-State Comput. Devices Circuits*, vol. 6, pp. 36–44, 2020.
[43] J. Chen, J. Li, Y. Li, and X. Miao, "Multiply accumulate operations in memristor crossbar arrays for analog computing," *J. Semiconductors*, vol. 42, no. 1, Jan. 2021, Art. no. 013104.
[44] S. Zhang et al., "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.
[45] B. Murmann. (2011). *ADC Performance Survey 1997–2011*. [Online]. Available: http://www.stanford.edu/ murmann/adcsurvey.html
[46] J. M. Correll et al., "An 8-bit 20.7 TOPS/W multilevel cell ReRAM-based compute engine," in *Proc. IEEE Symp. VLSI Technol. Circuits (VLSI Technology Circuits)*, Jun. 2022, pp. 264–265.
[47] Q. Wang, X. Wang, S. H. Lee, F.-H. Meng, and W. D. Lu, "A deep neural network accelerator based on tiled RRAM architecture," in *IEDM Tech. Dig.*, Dec. 2019, pp. 4–14.
[48] C.-C. Chang et al., "Device quantization policy in variation-aware in-memory computing design," *Sci. Rep.*, vol. 12, no. 1, pp. 1–12, 2022.
[49] G. Krishnan et al., "Robust RRAM-based in-memory computing in light of model stability," in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Mar. 2021, pp. 1–5.
[50] Q. Wang, Y. Park, and W. D. Lu, "Device variation effects on neural network inference accuracy in analog in-memory computing systems," *Adv. Intell. Syst.*, vol. 4, no. 8, Aug. 2022, Art. no. 2100199.
[51] Y.-D. Chih et al., "16.4 An 89TOPS/W and 16.3TOPS/mm^2 all-digital SRAM-based full-precision compute-in memory macro in 22 nm for machine-learning edge applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 252–254.
[52] J.-M. Hung et al., "A four-megabit compute-in-memory macro with eight-bit precision based on CMOS and resistive random-access memory for AI edge devices," *Nature Electron.*, vol. 4, no. 12, pp. 921–930, 2021.

Fan-Hsuan Meng received the B.S. degree in electrical engineering and the M.S. degree in electronics engineering from National Tsing Hua University, Hsinchu, China, in 2014 and 2016, respectively. She is currently pursuing the Ph.D. degree with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA. She worked as a Process Integration Engineer in 7 nm FinFET devices at TSMC, Hsinchu, from 2016 to 2017. She is currently working on system level optimization for in-memory computing-based neural network accelerators. Her research interests include memristive devices and its application for neuromorphic computing.

Xinxin Wang (Graduate Student Member, IEEE) received the B.S. degree in microelectronics science and engineering from Peking University in 2018. She is currently pursuing the Ph.D. degree with the Electrical Engineering and Computer Science Department, University of Michigan. Her research interests include hardware neural network accelerator design based on resistive random access memory (RRAM) crossbar arrays.

Ziyu Wang received the B.E. degree from Tsinghua University, Beijing, China, in 2019, and the M.S. degree in electrical and computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 2021, where he is currently pursuing the Ph.D. degree with the Department of Electrical Engineering and Computer Science. His research interests focus on vulnerability analysis of emerging analog in-memory computing accelerator for deep neural networks and designing secure and reliable in-memory computing systems.

Eric Yeu-Jer Lee received the B.S. degree from National Tsing Hua University, Taiwan, in 2017, and the M.S. degree from National Yang Ming Chiao Tung University, Taiwan, in 2020. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI, USA. His current research interests include asynchronous event-based recognition systems, spiking neuron networks, and RRAM-based in memory computing chip design.

Wei D. Lu (Fellow, IEEE) received the B.S. degree in physics from Tsinghua University, Beijing, China, in 1996, and the Ph.D. degree in physics from Rice University, Houston, TX, USA, in 2003. From 2003 to 2005, he was a Post-Doctoral Research Fellow at Harvard University, Cambridge, MA, USA. He joined as a Faculty Member of the University of Michigan in 2005, where he is currently a Professor with the Electrical Engineering and Computer Science Department. His research interests include resistive-random access memory (RRAM), memristor-based logic circuits, neuromorphic computing systems, aggressively scaled transistor devices, and electrical transport in lowdimensional systems. He was a recipient of the NSF Career Award.