# SoBS-X: Squeeze-Out Bit Sparsity for ReRAM-Crossbar-Based Neural Network Accelerator

Fangxin Liu, Zongwu Wang, Yongbiao Chen, Zhezhi He, Tao Yang, Xiaoyao Liang, and Li Jiang†, *Member, IEEE*

***Abstract*—Resistive Random-Access-Memory (ReRAM) crossbar is a promising technique for deep neural network (DNN) accelerators, thanks to its in-memory and in-situ analog computing abilities for Vector-Matrix Multiplication-and-Accumulations (VMMs). However, it is challenging for crossbar architecture to exploit the sparsity in DNNs. It is inevitably complex and costly to exploit fine-grained sparsity due to the limitation of the tightly-coupled crossbar structure.**

**As a countermeasure, we develop a novel ReRAM-based DNN accelerator, named Sparse-Multiplication-Engine (SME), based on a hardware and software co-design framework. First, we orchestrate the bit-sparse pattern to increase the density of bit-sparsity based on existing quantization methods. Such quantized weights can be nicely generated using the Alternating Direction Method of Multipliers (ADMM) optimization during the DNN fine-tuning, which can exactly enforce bit patterns in weights. Second, we propose a novel weight mapping mechanism to slice the bits of the weight across crossbars and splice the activation results in peripheral circuits. This mechanism can decouple the tightly-coupled crossbar structure and cumulate the sparsity in the crossbar. Finally, a superior squeeze-out scheme empties the crossbars mapped with highly-sparse non-zeros from the previous two steps. We design the SME architecture and discuss its use for other quantization methods and different ReRAM cell technologies. We further propose a workload grouping algorithm and a pipeline to achieve workload balance among crossbar-rows that concurrently execute multiply-accumulate operations to optimize the system latency. Putting all together, with the optimized model, compared with prior state-of-the-art designs, the SME shrinks the use of crossbars up to $8.7\times$ and $2.1\times$ using ResNet-50 and MobileNet-v2, respectively, and achieve average $3.1\times$ speed up with no or little accuracy loss on ImageNet.**

***Index Terms*—ReRAM, sparsity, neural network, accelerator**

## I. Introduction

RESISTIVE Random-Access-Memory (ReRAM) crossbar emerges as a promising solution to accelerate the inference of Deep Neural-networks (DNNs) [1]–[7]. One of the reasons is that ReRAM accelerators adopt an in-situ scheme that fastens the weights of DNN on ReRAM crossbars, greatly reducing the massive cost of data movement in tradition Von-Neumann structures. In addition, ReRAM crossbar can perform highly-parallel Vector-Matrix Multiplication (VMM) by sharing inputs in rows and gathering currents in column using Kirchhoff's laws [1], [2], [8]–[11]. However, with the development of surprisingly large DNN models such as GPT-3 [12] with 175B parameters, the increasing requirement of ReRAM crossbars to accommodate enormous DNN weights become the main hurdle of this technology.

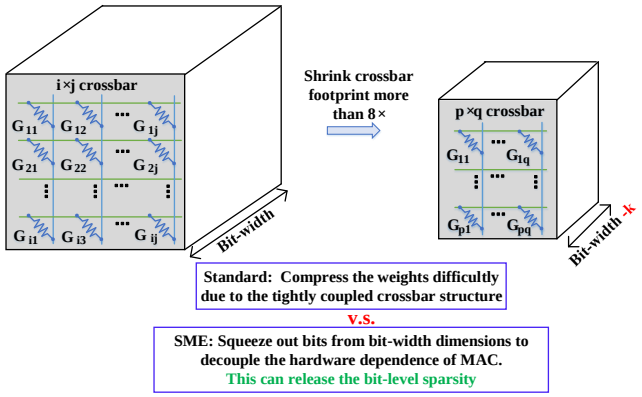


Fig. 1: Concept illustration of SME with the squeeze-out scheme. SME exploits the bit-level sparsity to achieve better energy efficiency and performance by proposing software and hardware mechanisms.

One of the reasons that contributes to the vast demand of ReRAM crossbar is the tightly coupled layout of the weight operands on ReRAM cells required by the highly-parallel VMM calculation, which makes it difficult to exploit sparsity in DNNs. Extensive works propose weight sparsification methods to reduce the number of ReRAM crossbars. Hardware-independent pruning algorithms, such as filter-/channel-wise sparsity, can derive a "smaller" dense weight matrices and directly map them to crossbars [13]–[16]. These algorithms, however, are too coarse-grained and lead to limited sparsity utilization. Finer-grained sparsification methods adopt the software-hardware co-optimization fashion [17]–[19]. These works first map the weights to crossbars and then prune the whole crossbar-columns or crossbar-rows. The weight matrix change asks for extra peripheral circuits to coordinate the shape of input and output feature maps. Moreover, DNNs have to be retrained, which is not always feasible in a real-world scenario [8].

The second source of the vast requirement of ReRAM crossbars derives from the small bit-width, i.e., 1-3 bits [3], [6], [15], [16], [20], that a ReRAM cell can stably store due to process imperfections and limitation. Thus, quantization is necessary to reduce the bit-width of the weight. Most

†Li Jiang is the corresponding author

F. Liu, Z. Wang, Y. Chen, Z. He, T. Yang and X. Liang are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240 CHN (e-mail: {liufnagxin, wangzongwu, chenyongbiao0319, zhezhi.he, yt594584152}@sjtu.edu.cn and liangxy@cs.sjtu.edu.cn).

L. Jiang is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240 CHN, also with the MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University, Shanghai, 200240 CHN, and also with Shanghai Qi Zhi Institute, Shanghai, 200232 CHN. (e-mail: ljiang_cs@sjtu.edu.cn).

existing ReRAM-crossbar accelerators quantize the weight in 8-bits and decompose the 8 bits to 8 cells, respectively. The resulting massive *bit-level sparsity* is difficult to exploit. The sparse ReRAM-crossbar architecture SRE [21] attempts to exploit fine-grained sparsity derived by swapping the crossbar-columns and crossbar-rows. While exploiting the bit-level sparsity, this design also suffers from the extra hardware overhead from complex control, costly indexing and routing, which almost offset the ear efficiency derived from the reduced crossbars.

The fundamental constraint limiting the exploitation of the sparsity is that *the data mapping and the VMM computation are tightly coupled with the crossbar structure*, denoted as *structural-coupling problem*. To solve this problem, in this paper, we exploit the bit-level sparsity to improve the area- and energy-efficiency of ReRAM-crossbar based DNN accelerators. We propose an algorithm-hardware co-design framework called SME including a novel weight mapping schemes and a data path to squeeze out the bit-wise sparsity. SME can apply to many quantization methods, and it is training-free and orthogonal to existing pruning methods.

Such an SME-based framework is inspired by our observation that a large proportion of bit-level sparsity actually exists and is regularized when the DNN is mapped onto the crossbars with the SME mapping scheme. By taking advantage of our proposed squeeze-out scheme and SME architecture, we can decouple the weight layout and calculation result to release the sparsity and significantly shrink the used crossbar footprint via squeezing out the fewer magnitude bits, as depicted in Fig. 1. Our design principle is to maximize the exploitation of bit-level sparsity with minimal hardware overhead. Therefore, we enforce that all weights mapped onto the crossbar have a certain pattern of bit distributions, resulting in bit sparsity that is more favorable for the squeeze-out scheme. Motivated by the powerful Alternating Direction Method of Multipliers (ADMM) optimization, which can enforce patterns in the fine-tuning while maintaining high accuracy [15], [22], we exploit the opportunity of algorithm and hardware co-design. Specifically, we exploit ADMM and the constraint of bit distribution to fine-tune the DNN model such that the non-sparse bits in the weights mapped to the crossbar are constrained to be at several consecutive positions. Moreover, the non-ideal effect of ReRAM-based crossbar leads to an inevitable constraint on the number of crossbar-rows for concurrent execution of MAC operations. Therefore, we further design a latency-matching pipeline in conjunction with this inherent constraint to offset the latency induced by the algorithm without adding the additional circuit. The contributions are summarized as follows:

- We propose a bit-wise sparse pattern and an inter-crossbar bit-slicing scheme to accumulate the 0-bits to the same crossbars. This can increase the sparsity in the encoded bits and decouple the hardware dependence of MAC to release the bit-level sparsity.
- We propose a squeeze-out scheme that empties highly sparse crossbars by sacrificing a limited amount of least-significant bits. This realizes approximately irregular sparsity exploitation with the regular structure to maximize efficiency and throughput.
- We design a latency-matching pipeline in conjunction with the inherent constraint of crossbars to optimize the system latency, which is further formulated as an optimization problem and solved by a novel workload grouping algorithm.
- We design a hardware architecture of SME with a limited 2 Kb overhead. Extensive experiments on various datasets show that the proposed SME reduces up to $8.7\times$ and $2.1\times$ crossbars for ResNet-50 and MobileNet-v2, respectively, and achieves average $3.1\times$ speed up compared with the state-of-the-art methods.

## II. Background and Motivation

### A. ReRAM-based Sparse NN Accelerators

The *Structural-coupling problem* manifests itself as the inability to freely skip the multiplication of zero operands because weight-bits in the same crossbar-row share the same input, and the current derived by multiplication in cells are accumulated in the same crossbar-column [7]. In Fig. 3(a), suppose each weight has 4-bit and is partitioned into four cells. If a single cell containing 0-bit is removed, other cells can not fill their position since they are from a different row or column. Moving weight-bits across crossbar-rows/columns leads to wrong MAC results without modifying peripheral circuits, as shown in Fig. 3(b).

Structural pruning methods avoid this problem by pruning the weights in a granularity that the whole crossbar-column (or -row) can be removed at the cost of extra peripheral circuits [17]–[19], [24]. The extra peripheral circuits, including input-fetching and output-alignment modules, process the raw feature maps into the pruned weight matrices mapped on the crossbars. We break down the peripherals' area overhead and find that PIM-Prune [19] needs 4KB index storage to skip fetching the unnecessary activation (multiplied by zero weight) for ResNet-50. To achieve fine-grained pruning, excessive peripheral circuits will be introduced to route modified matrices. In contrast, a coarse-grained pruning will inevitably modify weight values and thus requires finetuning to retain accuracy.

The weight matrix after structural pruning still contain many 0 bits. As a result, enormous ReRAM-cells, denoted as sparse cells, are mapped with 0-bits and do not contribute to the final result. SRE [21] breaks up crossbars into smaller parts, namely Operating Units (OUs), to exploit finer-grained sparsity. SRE utilize the sparsity of empty OU rows and columns, which is easier to find than larger empty crossbar rows and columns. Thus, they can utilize more sparsity that are impossible to exploit before. However, SRE dramatically increases the peripheral circuit's overhead to accumulate the correct result and introduces 778KB index storage for ResNet-50. There is a dilemma between the sparse utilization and the OU size: shrinking the size of OUs can exploit finer-grained sparsity but significantly increase the index overhead and the routing overhead of control circuits (see the extra routing in Fig. 3(b)).

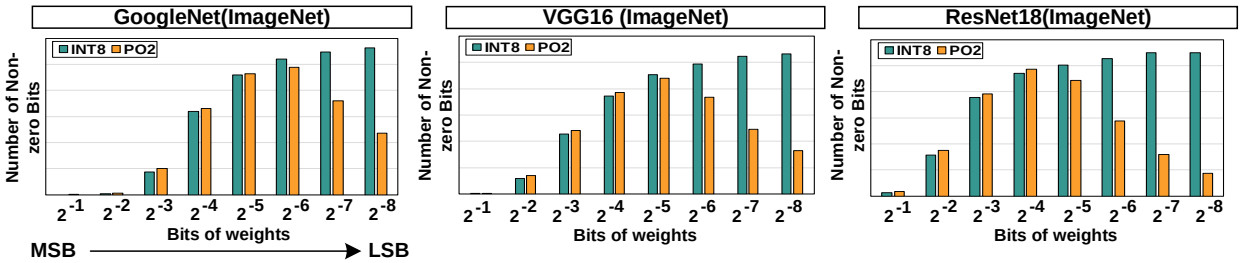


Fig. 2: Comparison of the INT8 and PO2 quantization in terms of the bit-level sparsity of weights in the quantized DNNs on ImageNet [23].

### B. Weight Quantization for ReRAM-crossbar

The ReRAM cell is programmed into multiple conductance levels to represent a value, e.g., a 4-bit unsigned value requires $2^4$ different levels. The ReRAM cell's process limitations [21], [25] constrain the bit-width of the value a ReRAM cell can store. Consequently, a weight is conventionally segmented into multiple subwords and each subword is deployed on a ReRAM cell. This segmentation results in high crossbar costs. For example, ResNet-18 with 32-bit weights consumes more than $20,000$ crossbars of $128 \times 128$ size [20]. Thus, quantization is compulsory to reduce the bit-width of DNNs's weight.

On the one hand, some quantization methods only reduce the variety of weights to shorten the encoding bit-widths of weights. Quantization using weight clustering [26] and sharing [27], [28] strive to reduce the number of values that represent weights, but these values are still floating numbers, which can not shrink the crossbar amount.

On the other hand, some quantization methods can actually reduce the bit-width of weights on crossbars. Uniform quantization like INT8 quantization [29] is thus a well-accepted quantization method for ReRAM-crossbar [4]. The resulting integer values can be well aligned and mapped to ReRAM-crossbar. Adaptive quantization methods, like HAQ [26], uniformly quantize weights using different bit-widths to optimize both the memory occupation and accuracy. Some other quantization methods such as POWER-OF-2 based quantization (PO2) [8], [28], [30] quantize values in the form of exponents. However, they may incur complex finetuning and they also need to align the weights to the most and least significant bit of the whole crossbar, which means that the bit-width on crossbars will be larger than their encoding bit-width.

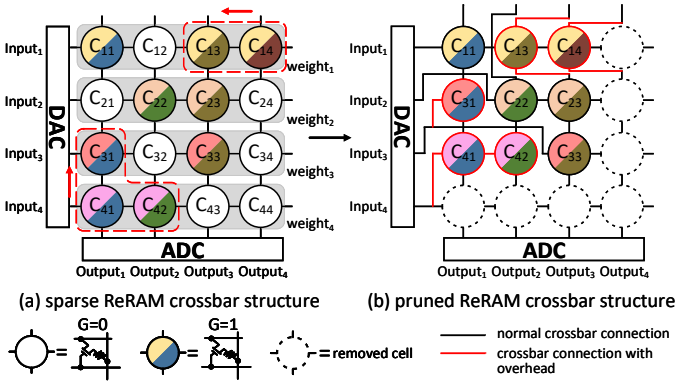


Fig. 3: Crossbar row/column exchange and structural-coupling problem. The same color of the upper/lower semicircle indicates that the cells share the same input, and the cells' results are accumulated in the same bit-line, respectively.

### C. Motivation and Key Insights for Bit-Wise Sparsity Exploitation

**Motivation:** A certain number of ReRAM crossbar-based DNN accelerator designs are built in prior works. Nevertheless, those designs are still subject to several weaknesses:

- Excessive ReRAM crossbar footprints and extra peripheral circuits that are needed to accommodate large NNs [1], [3];
- Mapping the weights onto the ReRAM-based crossbar inevitably generates a large amount of sparsity, which was difficult to exploit in previous works [19], [25];
- The inherent non-ideal effects [21], [31] of current summation and IR drop that constrain the number of crossbar-rows for concurrent execution of MACs, thus, hampering the throughput.

As shown in Fig. 2, we can see a high degree of sparsity in the quantized weights, especially in the three Most Significant Bits (MSB). In PO2 quantization, the Least Significant Bits (LSBs) also contain much sparsity. However, such bit-wise sparsity can hardly be exploited by ReRAM-crossbar accelerators due to the structural-coupling problem. The key is to decouple the crossbar structure, which is described in Section III.

**Key Insights:** As illustrated in Fig. 4, the SME contains six steps, consisting of four algorithm parts with customized circuits, and two hardware parts needed to finish the whole process.

- **User- and hardware-friendly model compression method.** We conduct a software and hardware co-design first to bring up a user- and hardware-friendly model and training-free framework, which squeeze-out the bit-level sparsity of the target DNN, in an attempt to enhance the crossbar utilization and energy efficiency.
- **Making use of the unstructured sparsity to increase the energy efficiency.** The bit-level sparsity, which is regarded as unstructured and hard to use in previous works [19], [21], gives us an opportunity to decouple the tightly-coupled structure. We reorganize it into a crossbar-friendly configuration, which largely increases our sparsity on the crossbar.
- **Offsetting cleverly the latency constraints caused by non-ideal effects and sparsity exploitation to optimize the performance.** In practical design, the number of

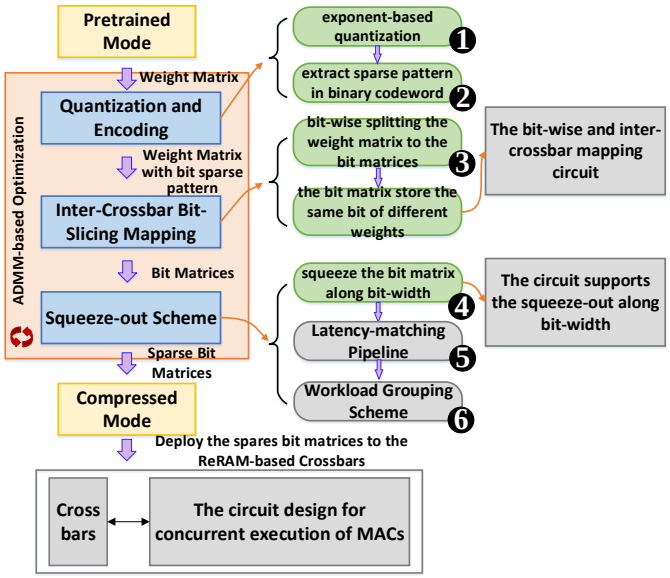


Fig. 4: The procedure of Sparse-Multiplication-Engine. We first quantize the weight matrix to present them in binary with the bit-sparse pattern. Then, the weights are sliced onto several bit matrices by our mapping scheme. After that, we squeeze out the inter- and intra- bit matrix sparsity by row squeezing while minimizing the value bias generated on weights. Finally, we design a latency-matching pipeline to optimize the system throughput, which can be formulated as an optimization problem and solved by a novel workload grouping algorithm.

ReRAM-based crossbar-rows for concurrent execution of MACs is usually constrained [32], allowing us to mitigate the latency induced by the sparsity exploitation. We design a latency-matching pipeline scheme that combines the inherent latency constraint and algorithm-induced latency constraints, enabling the two to cancel each other out for better performance.

## III. SME Algorithm

This section describes the whole software procedure to decouple the crossbar structure by novel weight mapping algorithms. The following sections describe these three steps: quantization, inter-crossbar bit slicing, and squeezing, as shown in Fig. 5.

### A. Quantization and Encoding Scheme

We assume single-level ReRAM cell as an example throughout this paper for simplicity. We quantize and encode the weights by extending the additive PO2 quantization[1] [28], [33] that represent the weight with the sum of several power-of-twos so that we can increase the bit-level sparsity in crossbars while retaining the values well to avoid fine-tuning. We map each $N_q$-bit weight onto $N_q$ cells, $b_{1:N_q}$:

$$w^q = \sum_{i=1}^{N_q} b_i 2^{-i}, \quad b_i \in \{0, 1\} \tag{1}$$

[1]Other quantization methods, such as adaptive quantization [8], [26] can also apply to SME, which is discussed in Section V-C.

In Step ❶, we *quantize the weights into sum of power-of-twos, whose exponent are among $S$ consequtive integers.* The above quantization can be derived by rounding the APT quantization result as follow:

$$w^q = \sum_{i=k}^{\min\{N_q, k+S-1\}} b_i 2^{-i}, \; k \in \{1, 2, \ldots, N_q\} \tag{2}$$

The maximum absolute value Eq. 2 can represent is $|w^q| \leq 1-2^{-S}$. Consequently, we scale all the weights into that range using a simple shift operation in the architecture (described in Section IV). Compared with the INT8 quantization, this

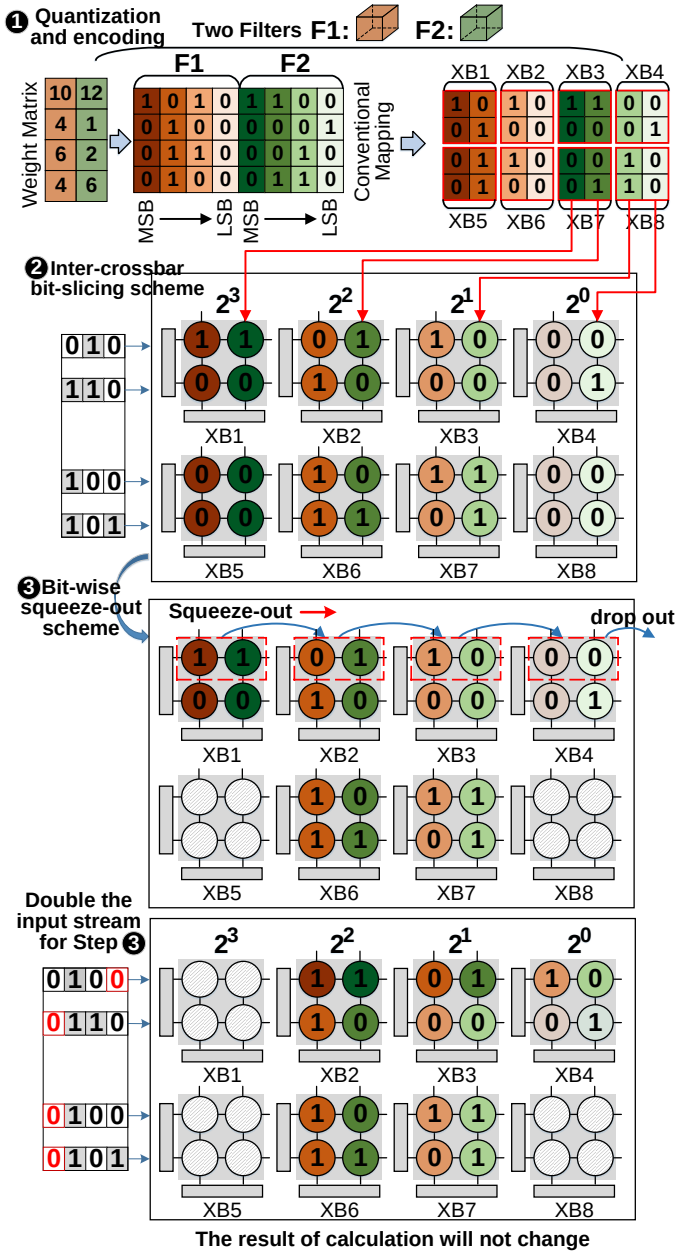


Fig. 5: The overview of proposed algorithm framework is composed of three parts. ①: the quantization and encoding scheme generates massive structured bit-level sparsity while retaining precision. ②: bit-wise and inter-crossbar mapping that utilize different level of sparsity in different bits. ③: squeeze-out scheme that decouples cell site and output result.

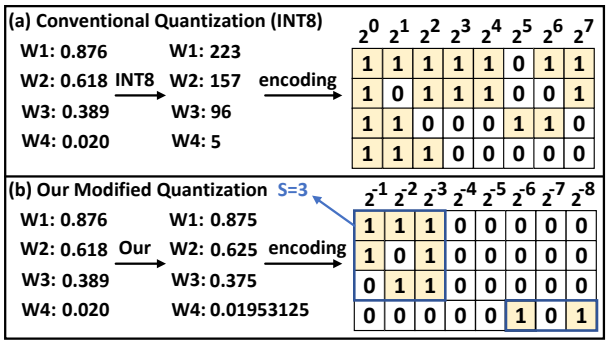


Fig. 6: The sparsity generated by INT8 and modified APT quantization whose '1' bits are constrained in consecutive 3 positions (marked in blue frame).

quantization method can increase and accumulate the bit-level sparsity in a codeword, as shown in Fig 6.

### B. Inter-crossbar Bit-slicing Scheme

To decouple the crossbar structure, we propose the **inter-crossbar bit-slicing** scheme. The key idea of *bit-slicing* is to map the same bit of quantized weights into the same bit crossbar, as shown in Step ❷. In this mapping process, a $W \times H$ weight matrix quantized with $N_q$ bits is sliced into $N_q$ bit-sliced matrices of size $W \times H$. Then, each bit-sliced matrix is further partitioned and mapped to ReRAM crossbars with size $xw \times xh$.

For example, in step ❷ of Fig 5, the bit matrix consisting of two filters is first bit-sliced into four bit matrices, $XB_{1,5}$ for the most significant bit, $XB_{2,6}$ for the second one and after that $XB_{3,7}$ and $XB_{4,8}$. Specifically, the first weight 1010 in $F1$ is sliced and mapped onto the top-left cells of $XB_{1-4}$. MSBs in filter $F1$ and $F2$ are mapped to crossbar $XB_1$ and $XB_5$. The above mapping scheme can aggregate the sparsity in a crossbar (e.g., $XB_5$ and $XB_8$), so that these empty crossbars can be saved by the mechanism of light-weight index [19], [21], [34].

It is worth noting that this bit-slicing mapping scheme requires minor modification on the peripherals (refer to Section IV-B), and demands the same amount of peripheral circuits, such as ADCs, shifters, and adders, as conventional mapping method.

### C. Bit-wise Squeeze-out Scheme

In the previous section, we aggregated a large amount of sparsity by the bit-slicing scheme so that some of the crossbars become empty and can be saved directly, but there are still crossbars that are very sparse and cannot be saved directly. We shrink the sparsity from the full crossbar to a smaller granularity, rows, since all the cells in the same crossbar-row share the same input. Fortunately, our SME approach make sure that the first few most-significant bit matrices are highly sparse. Fig. 7 shows there are less than 10% non-empty rows in the most significant bit matrix on average. Based on the above observation, we propose a clever squeeze-out scheme that circumvents the structure couple problem. The essence is the row swapping among the crossbar group, without introducing either overhead or large accuracy loss.

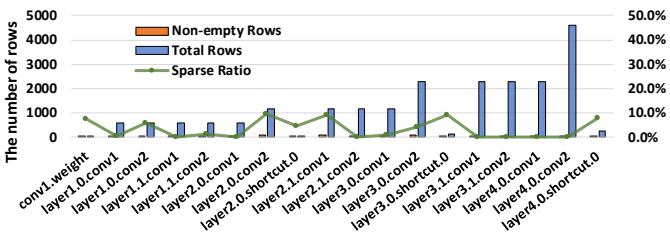


Fig. 7: Non-empty row statistics of crossbars stored MSB of the ResNet-18.

In Step ❸, we squeeze the crossbar-rows containing non-zeros in preceding XBs to the subsequent XBs until these rows in tailing XBs are dropped out. For example, the first crossbar-row in $XB_1$ is remapped to $XB_2$, whose first crossbar-row is shifted to $XB_3$. And the LSB crossbar $XB_4$ drops its first crossbar-row. Based on step ❶, releasing these crossbars will not lead to loss on network's accuracy. A corresponding operation on the input of these rows is performed. This step does not introduce extra indices for accumulating the output.

The following observations inspire the bit-wise squeeze-out scheme: the first few bit matrices are too sparse to compose a single crossbar. According to our bit-wise sparse pattern, '1's will only appear in successive positions after its most significant bit, which means that for the weights whose first few bits are '1', their last few bits must be '0'. We can empty crossbars that store the MSB by squeezing-out these bits without changing the actual quantized weight.

After squeeze-out by one bit, the corresponding non-empty contents in $XB_i^j$ are moved to $XB_i^{j+1}$ ($j \in \{1, 2, \cdots, N_q\}$) and these contents in the last bit crossbar $XB_i^{N_q}$ are abandoned. According to Eq. (2), the value in that row is approximately halved with 1-bit squeezed. To ensure the invariance of the calculation results, we propose a scheme to double the input (refer to Step ❸ in Fig. 5). Note that we can even perform this step iteratively to squeeze-out multiple bits.

In this paper, we use the same style of input as in ISAAC [1]. The input is converted into bit-serial voltage and the number of cycles required is equal to the bit-width of the input. If we squeeze out with $x$-bit, we delay the input of these rows for $x$ clocks, which implemented by Fig. 9 Ⓑ.

Assume that the weights and inputs are both 4-bit and we perform squeeze-out scheme for 1-bit. For example, in Fig. 5 ❷, the first row in $XB_1$ is non-empty. Starting from $XB_1$, the crossbar's non-empty rows are placed at the same position in the latter crossbar, and the rows of the last crossbar ($XB_4$ in Fig. 5) are dropped out. After that, the crossbar $XB_1$ storing MSB (i.e., $2^{-1}$) can be saved, and the bit-width of weights is changed from 4- to 3-bit. The bits representing $2^{-1}$ in $XB_1$ is moved to $XB_2$ and represent $2^{-2}$, which means the part of weight shrinks by half (i.e., $W_1$ changes from $10 = 1010_{(2)}$ to $5 = 0101_{(2)}$). Thus, we shift the input of the first row one bit to the left ($input \times 2$), which acts on the first row in each remaining crossbar (i.e., $XB_2 \sim XB_4$) to make up for the changes in weight, i.e. $I_1 \times W_1 = (I_1 \times 2) \times (W_1/2)$. This process equals to delaying the input one cycle, which

means that the cycle of input changes from 4 to 5. As a result, the total amount of computation changes from $4 \times H \times W \times 4$ to $5 \times H \times W \times 3$, the computation has been reduced.

### D. Latency Analysis

In the proposed architecture, different configurations of the crossbar and our bit-wise squeeze-out scheme (mentioned in Section III-C) will directly influence the computing latency and the hardware overhead. The main design parameters of PEs include the design choices of the degree of squeeze-out and the crossbar size. This subsection will introduce the analysis of design choices and hardware performance.

Assume the crossbar size is $K \times K$, $M_w$ is the bit-width of weights, and $M_a$ is the bit-width of inputs. We map $K \times K$ weights in $M_w$ crossbars and feed the input in through the DACs serially 1-bits at a time. The MAC operation is finished in $M_a$ cycles. The squeeze-out operation begins with inter-crossbar bit-slicing scheme. Here, we index the sliced crossbars from MSB to LSB with $XB_m$, and $m = [1; M_w]$, such that $m = 1$ and $m = M_w$ are corresponding to MSB and LSB, respectively. $D$ is the degree of being squeezed, representing the number of iterations in the squeeze-out scheme. Specifically, the squeeze-out degree increases by 1 with squeezing 1-bit along the direction of bit-width. We take Fig. 5 as an instance to intuitively illustrate this process, $D = 1$ denotes the sliced crossbars change from $XB_1 \sim XB_8$ to $XB_2 \sim XB_8$.

Those kernels mapped in sliced crossbars could be duplicated in different computation units and could take multiple input data to generate independent outputs simultaneously. In this case, we need to duplicate $K \times M_w \times M_a$ kernels could speed up without perform squeeze-out scheme. If we further squeeze weights with $D$ bits, resulting in a reduction in the number of crossbars by $D$. However, the cycles it takes for MAC operation to complete increase from $M_a$ to $M_a + D$. The rows are partitioned into two parts according to the performed squeeze and the none performed, denoted as $A$ and $B$ respectively. Then we can calculate the number of duplicates needed to achieve the speedup. Only if the number of duplicates needed after the execution of squeeze is smaller than that of unexecuted ones, the optimization will have a performance benefit. Its formulated expression can be mathematically described as:

$$\begin{aligned} &\big((M_a + D) \times A + M_a \times B\big) \times (M_w - K) \\ &= KM_wM_a - KM_wD + DAM_w - D^2A \leq KM_wM_a \quad (3) \\ &\text{s.t. } A + B = K \end{aligned}$$

Without loss of generality, in this paper, we take $K = 128$, $M_a = 8$ and $M_a = 8$. If we squeeze 1-bit, i.e., set $D = 1$, then the overall performance gain increases as long as the squeezed rows are less than 146 (i.e., $A \leq 146$). As described in Fig. 2, almost all the layers in ResNet-18 satisfy this condition. According to Eq. (3), we can find that: (1) the larger the degree of being squeezed $D$, the size of crossbar $K$ and input bit-width $M_a$ are, the more feasible the scheme is; (2) when the weight bit-width $M_w$ is getting smaller, the scheme becomes more feasible accordingly (i.e., $A$ is larger).

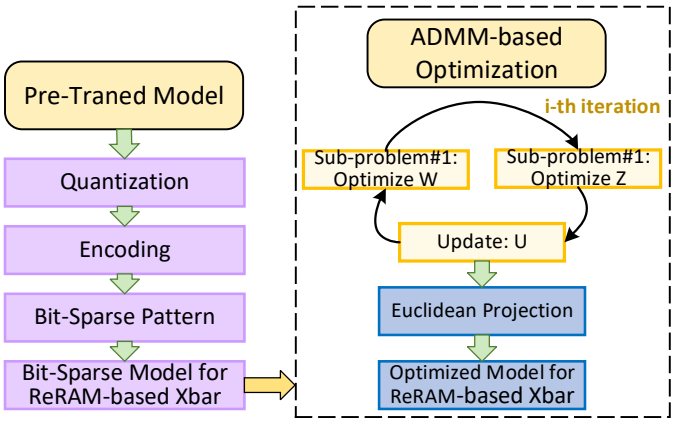


Fig. 8: Process of ADMM-based quantization optimization.

### E. ADMM-based Quantization Optimization

ADMM optimization is an advanced optimization technique that works by decomposing the original optimization problem into two subproblems, which are then solved in separate iterations. Therefore, we incorporate ADMM-based optimization with the Bit-Sparse Pattern to regularize the weights into the same bit distribution. Eventually, the non-zero bits in the weights are regularized in a set of consecutive bits. Specifically, we incorporate the requirement for the quantization pattern into ADMM optimization to achieve and optimize the desired quantization pattern and ensure the classification accuracy of the model through the fine-tuning process. Consider a DNN with $L$ layer (i.e., convolutional/fully-connected layers), given the input $\boldsymbol{x}$ and corresponding label $\boldsymbol{t}$, the DNN inference loss can be described as:

$$\mathcal{L}\big(f(\boldsymbol{x}, \{\mathbf{W}_l^{\text{fp}}\}_{l=1}^{L}); \boldsymbol{t}\big) \quad (4)$$

where $\mathbf{W}_l^{\text{fp}} \in \mathbb{R}$ denotes the full precision (i.e., FP32) weights of $l$-th layer. To quantize the weights, our loss function can be expressed as:

$$\begin{aligned} &\min_{Q(\mathbf{W}^{\text{fp}})} \mathcal{L}\big(\boldsymbol{x}, \{\mathbf{W}_l^{\text{Q}}\}_{l=1}^{L}); \boldsymbol{t}\big) - \mathcal{L}\big(\boldsymbol{x}, \{\mathbf{W}_l^{\text{fp}}\}_{l=1}^{L}); \boldsymbol{t}\big) \\ &\text{subject to} \quad \mathbf{W}_l^{\text{Q}} \in \mathbf{Q}_l,\ l = 1, 2, 3, \ldots, L. \end{aligned} \quad (5)$$

where $\mathbf{W}_l^{\text{Q}}$ is the quantized weights from $l$-th layer, and $\mathbf{Q}_l$ is the constraint set of bit-sparse pattern quantization.

For the bit-sparse pattern quantization constraint, the set $\mathbf{Q}_l$ indicating the weights in the $l$-th layer are quantized into the sum of power-of-twos, whose exponents satisfy the pattern, that is, the number of PO2 summed is less than or equal to $S_i$, and the exponential distribution is consecutive. $S_i$ is predefined hyperparameters. The quantization pattern depends on the requirements of model classification accuracy and the reduction in crossbar resources. As shown in Fig. 8, the overall ADMM-based optimization process is an iterative fine-tuning process, where $Z$ is an auxiliary variable, and $U$ is the dual variable. This optimization is similar to the one in [15], [19], [22].

## IV. SME Architecture

This section describes the SME accelerator architecture that can execute the optimized DNN models generated by the SME optimization framework.

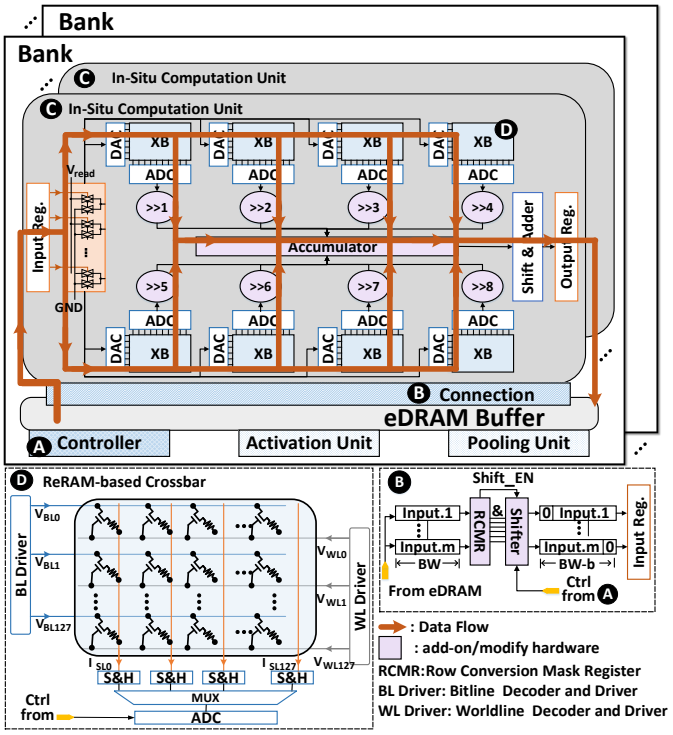


Fig. 9: Architecture overview and data path of SME.

### A. Architecture Overview

We present the overview of SME architecture, aiming at inference in edge devices. As shown in Fig. 9, each bank consists of three parts, all of which are connected to the shared bus: 1) the controller decodes instructions and provides control signals to all the peripheral circuits; 2) in-situ Computation Units (CU) is the core computing and storage unit; 3) the shared blocks contain the activation unit, pooling unit, and eDRAM buffer for storing activations (i.e., intermediate computing results). The SME add-on hardware implements the computation function of matching our algorithm, including simple modifications to the existing crossbar peripheral circuits, which is easier to manufacture than integrate complex logic into the chip.

### B. Module and DataFlow

**Controller.** Controller in Fig. 9 A provides control signals to all the peripheral circuits and drive the finite state machines that steer the inputs and outputs correctly after every cycle based on the technique configurations.

**In-situ Computation Unit.** Fig. 9 C shows the CU that is composed of crossbars and peripheral circuits. It includes DAC/ADCs for data conversion, accumulator and shifters, which sum up the partial sums of crossbars, and shift-adder unit for aggregating the partial results of input cycles.

First, the input is in the form of the bit sequence, and then each cycle enters respectively 8 crossbars to the same in-situ CU to carry out MAC operation, and each gets the output currents. After output current generated on SL, first, latch the computation results by S&H circuits, and then sent 128 analog voltages to ADC to convert them into digital signals through a MUX. In each cycle, 8 crossbars are sampled in parallel, and a calculation result of different bits of the same weights is obtained through the corresponding shifter which is connected to each crossbar. Then, these results are sent to the accumulator to generate the complete calculation results of the weight. Finally, the results are transmitted into the output register for updating the outputs by shifting and adding. After traversing the 128 samplings, the next cycle calculation is repeated.

**ReRAM-based crossbar.** Fig. 9 D shows the ReRAM-based crossbar with $128 \times 128$ size, which perform parallel MAC operations. We adopt the SLC as the ReRAM cell since SLC is more reliable against process variation compare to the MLC counterpart. The ReRAM array is implemented with a one-transistor–one-memristor (1T1R) cell structure, in which one resistance cell with a transistor to control the write current to facilitate more precise writes to ReRAM cells. Moreover, the peripheral circuits, including the word-line driver, sample-and-hold blocks, multiplexer (MUX) and ADCs. The output analog signals are transmitted to ADC via MUX to control analog-to-digital conversion.

**Buffer Connection**, which supports the squeeze-out strategy. Fig. 9 B shows the communication between eDRAM Buffer and input register. The RCMR is used for fetching inputs from buffer to the register, which needs to convert the bit-width of input enable increase input cycle caused by the squeeze-out strategy (refer to Section III-C). Data from the buffer pass through the RCMR and the controller determines whether the squeeze-out scheme needs to be performed at the current layer, and if it does, the inputs bit-width will be extended to $(8+x)$-bit (we initially default the weights were quantized to 8-bit). The shifter follows the RCMR, '1' means to shift $x$-bit to the left, while '0' indicates padding $x$ zeros in front of MSB. After that, data transmit into the input register.

**Data Flow.** To adapt the sparse multiplication caused by SME, we change the data flow (the red arrow in the Fig. 9).

**First**, we fetch the data from the eDRAM buffer to the input register. The data pass through the RCMR, and the controller judges whether the squeeze-out scheme has been performed in the current layer. If it has, the inputs will be processed by Fig. 9 B. The processed data is transmitted into the input register.

**Second**, input data from the input register is fed to the crossbar for calculation. To reduce the hardware overhead, we reserve the sparsity of rows, resulting in the crossbars representing different bits sharing the same input. Only one group MUX is needed for bit-line. Therefore, the inputs are passed through the MUX array to the DACs to convert the digital inputs into analog voltage signals. Then, the input voltage is shared by ReRAM cells in the same row (bit-line) of the crossbar arrays along the horizontal direction to perform multiplication using Ohm's law. The current generated by the input voltage passing through the cell is accumulated in the vertical direction of the crossbar by using Kirchhoff's law. The bit-level partial sums (psums) are generated as the output

of each column. Therefore, the inputs are passed through the MUX array to the DACs on the crossbars representing different bits for converting the digital inputs into analog voltage signals. Then, the input voltage is shared by ReRAM bit cells in the same row (bit-line) of the crossbar arrays along the horizontal direction to perform multiplication using Ohm's law. The current generated by the input voltage passing through the cell is accumulated in the vertical direction of the crossbar by using Kirchhoff's law, and the psums are generated as the output of each column.

**Third**, the output analog currents enter the peripheral circuits. The TIA and S&H module is turned on, which is connected to the columns in parallel to convert the accumulated current to voltage. Then, the voltage signal is sent to the MUX, which is converted into the digital signal (psums) from the shared ADC at the MUX.

The psums pass through the CSM register, and pad the output with zeros to the correct position. Then, the psums of each column are shifted so that the crossbars' output represents different bits is converted to the calculation result of the corresponding bit (i.e., the calculation result of a part of the weight). After that, the shifted psums of the same column with crossbars representing different bit is aggregated in the accumulator to generate the calculated result of complete weight for the cycle. Finally, accumulated psums generated by each input cycle are calculated in the shift-add-unit and get the final output stored in the output register.

### C. Data interaction between banks

Most of the layers of the neural networks can not be mapped on a single CU, it is inevitable to incur data communication between CUs. After the input register (IR) results of each CU are obtained, the control unit in Fig. 9 accesses the IR data of the same filter in different XBs (contains positive XBs and negative XBs) and then merges in the global shifter&add. Thus, the final output is obtained. After that, activation and pooling operations are accomplished with the same of ISAAC [1].

### D. Latency-matching Pipeline Scheme

We analyze the constraint of the squeeze-out scheme in the ReRAM-based crossbar and employ a novel pipeline scheme to make the SME more efficient.

**Latency Constraint:** The IR drop [21], [25] caused by wire resistance is another main constraint that may cause the crossbar latency, that is, the input voltage disperses along the wire, which causes considerable voltage drop on the target cell and affects the performance and reliability. As a result, the more crossbar-rows for concurrent execution of MACs, the larger current will be on the input wire, which leads to a more severe voltage drop and more significant bias in the final calculation. Therefore, the non-ideal effect constrains the number of crossbar-rows for concurrent execution of MACs (i.e., computation parallelism), resulting in the increased latency of the ReRAM-based accelerator (**Constraint 1**). Due to the inherent structure of ReRAM-based crossbars, the multiplication results of the activated rows are accumulated

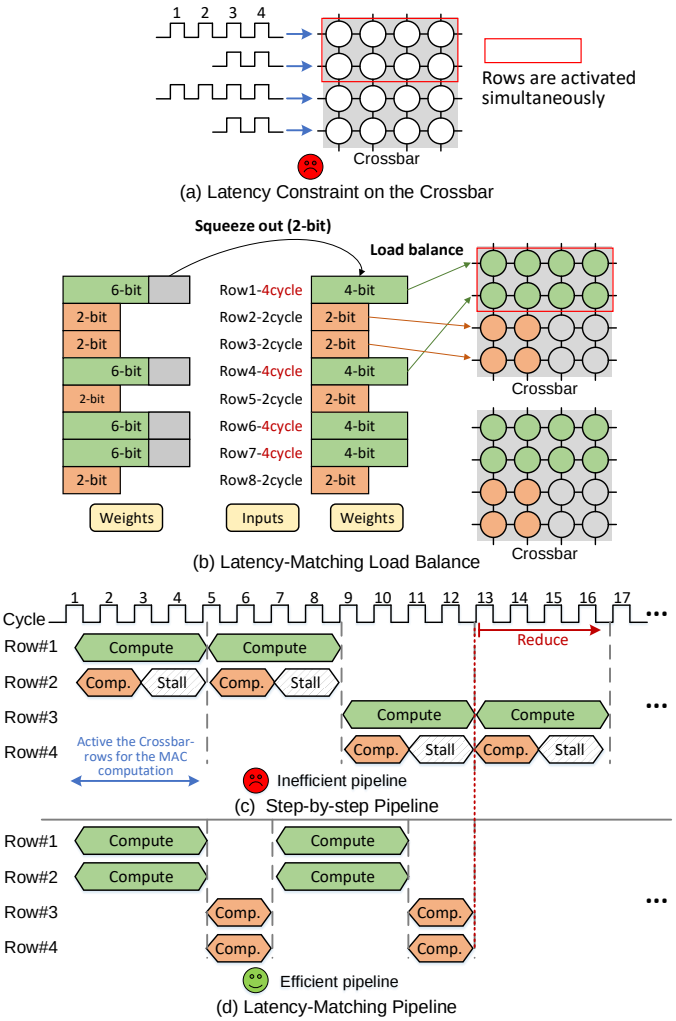


Fig. 10: Constraints in ReRAM crossbars and pipeline optimization. (a) Computing latency constraint on the crossbar. (b) Latency-aware load balance scheme. (c) Step-by-step pipeline on the crossbar without optimizing. (d) Proposed latency-matching pipeline on the crossbar.

together on each bitline (column). As described in Fig. 10(a), for correct computation, we need to ensure that all rows in the same crossbar wait for the binary voltage levels of inputs to be fully entered into the crossbar before switching to the next set of inputs. Given a DNN, after performing the squeeze-out scheme, we need to delay the input cycle to keep the computation correct, increasing the bit-width of the input (**Constraint 2**).

We assume that inputs to the ReRAM-based crossbar arrays are provided as multiple sequential bits, with one bit per cycle fed into the crossbar. For 2-bit fixed-point inputs and 6-bit fixed-point weights, as shown in Fig. 10(b), suppose we optimize the DNN model with squeeze-out by 2-bit, the bit-width of the weights in partial crossbar-rows is squeezed from 6-bit to 4-bit, while the sequence length of the corresponding inputs is expanded by 2-bit accordingly. In that case, the MAC computation of some rows will be stalled because one crossbar needs to wait for the result of activated crossbar-rows per cycle, which is subject to the limitation of calculating the

**Algorithm 1** Workload Grouping Scheme

**Data**: the squeezed weight matrix $\{\mathbf{W}_l^S\}_{l=1}^L$ of a model.
**Result**: The crossbar-rows selected to compute in the cycle $t$: $\{\mathbf{R}_t^l\}_{t=1}^T$

```
Initiate the minimal workload set as NSS = ∅ and the set of selected rows R_t^l = ∅;
Initiate K_I = the bit-width of input before the model executes the squeeze-out scheme;
for rid = 0 to #Rows do
    N_i: the number of input cycles in (i.e., assigned workload) the i-th crossbar-row;
    if N_i > K_I then
        SCORE_i = N_i − K_I;
    else
        Add crossbar-row number into the set NSS;
    end if
end for
maxWorkload = the rid with largest SCORE;
R_t = R_t ∪ {maxWorkload};
Remove rid maxWorkload from SCORE;
```

slowest row. The pipeline bubble between two MAC computations (inputs switching) is the computation delay generated by the squeeze-out scheme during the row activation phase. As illustrated in Fig. 10(c), if we active the crossbar-rows of the MAC operation phase sequentially as soon as it is available, the unbalanced bit-width of inputs causes much idle time. In that case, we need to wait for the most time-consuming row to be executed at each activated crossbar-rows. After two computation cycles, we can switch the inputs of the crossbar and feed them to the crossbar for the next step computation. However, this also poses an inefficient pipeline due to the latency constraint of ReRAM crossbars. Therefore, we propose a latency-matching pipeline, as shown in Fig. 10 (b) and (d). This scheme cleverly combines Constraint 1 and Constraint 2, that is, the inherent constraint of the crossbar and the squeeze-out scheme introduced constraint such that the two constraints offset each other.

### *E. Workload Grouping Scheme*

The bit-wise squeeze-out scheme proposed in Section III-C poses constraints on input processing order. In this section, we propose a workload grouping scheme to determine which rows can be activated concurrently in each cycle. In this algorithm, we aim to minimize the workload variance between the activated crossbar-rows that concurrently execute MAC operations.

**Problem Definition:** With the constraint generated in Section III-C, we define the problem as follows. Given an optimized DNN model by our SME algorithm, we can select a set of crossbar-rows to activate and compute in one cycle which satisfies:

1) Selected rows of different crossbars to be simultaneously activated should have the same or similar workload.

2) Crossbar-rows with the same or similar workload should be located in the same activated batch of the crossbar.

To optimize latency, we want sets assigned to the same group to have a more similar workload. Workload balance indicates the preference to assign sets with a more similar number of squeezing bits. Given $N$ rows $r_1, r_2, \ldots, r_N$, we assign them into $G$ groups, and each group has the similar workloads. We describe the above problem in an optimization problem as follows.

$$\begin{aligned} \textbf{minimize} \quad & F(\mathbf{R}) \\ \textbf{s.t.} \quad & F(\mathbf{R}) = \max_{1\le g\le G}\{|\bigcup_{i=1}^{N} r_{\{g,i\}}\ |\} \\ & F(\mathbf{R}) = \min\{WL(r_i) - WL(r_j)\} \\ & \forall\ 1 \le r_i, r_j \le G, r_i \ne r_j \end{aligned} \tag{6}$$

where stands for the maximum number of differences in load among the groups under the assignment $\mathbf{R}$, $G$ indicates the number of assigned groups. $WL(\cdot)$ denotes the workloads of the row. Here, we adopt a heuristic algorithm similar to Algorithm 1 to solve it.

With the workload grouping scheme above, we get groups of crossbar-rows that can be activated in parallel for concurrently executing MAC operations. To support these rows to be selectively activated usually requires introducing a significant routing overhead. Therefore, we allow crossbars to activate rows selectively by a binary firing vector. The firing vector enables the rows of the crossbars that need to be accumulated to complete the MAC operation. With the firing vector, the crossbar-rows will be selected by the multiplexers and be controlled by the central controller. The central controller then activates the crossbar-rows of the corresponding workload group, sends the corresponding inputs to these rows, and accumulates the results of the crossbars. Since our workload grouping is based on the result of the squeeze-out scheme, groups of workload can be obtained offline to generate a firing vector (bitmap identifying the rows with matches to be activated for MAC operations). In such a way, the crossbars have the capability to perform the MAC operation selectively.

## V. EXPERIMENTS

### *A. Experimental Setup*

TABLE I: Parameters of the CU in SME Architecture.

| In-situ Computation Unit Hardware Configuration (1GHz, 32nm process, 128 banks per chip, 12 CUs per bank) | | | | |
|---|---|---|---|---|
| **Components** | **Param** | **Spec** | **Area** ($mm^2 \times 10^{-3}$) | **Power (mW)** |
| MUX Array | number | 8 | 4.23 | 0.56 |
| S&H | number | $8 \times 128$ | 0.03125 | 0.0011 |
| ADC | resolution | 6-bit | 4.7 | 5.14 |
| | number | 8 | | |
| | frequency | 1.2GSps | | |
| DAC | resolution | 1-bit | 0.34 | 4.25 |
| | number | $8 \times 128$ | | |
| Memristor Array | size | $128 \times 128$ | 2.03 | 1.3 |
| | bits-per-cell | 1 | | |
| | number | 8 | | |

Table I summarizes the parameters and their power/area values of each CU in our SME. We redesign the data path

since we add some peripheral circuits to integrate the techniques. Moreover, the energy consumption and area overhead of memories, including eDRAM buffer, input/output register, and registers stored mask (i.e., RCM Register), are calculated with CACTI [35] based on the 32-nm CMOS process. For ADC and DAC, the model from [36] is used. The memristors adapted SLC-based ReRAM devices with a resistance range of $10K\Omega \sim 100K\Omega$ and crossbar size set as $128 \times 128$. We model the power and area for ReRAM devices using the results from [37]. We modify NVSim [38] with these models to estimate time, area and energy consumption.

The goal of SME is to improve the throughput and efficiency of the hardware accelerator. We have taken the following factors into consideration when selecting the benchmarks. They should be sufficiently representative and diverse enough to cover a range of hardware performance and overhead caused by compression effect, ranging from small scale dataset to big scale dataset. Therefore, we evaluate our work on classical image classification task, using several representative DNNs (VGG-16 [39], ResNet-18/50 [40], MobileNet-v2 [41]) on CIFAR-10 [42] and ImageNet ILSVRC-2012 [23]. We compare the accuracy and exploited sparsity with the PIM-Prune [19], SmartExchange [34], Pattern-based [22], and P-RM [43]. We compare with the performance of PIM system with ISAAC [1], PIM-Prune [19], SRE [21]. We use the ISAAC as the baseline [1]. For results that are not available, we reproduce their experiments and report the results. Our method can also be combined with other sparsity utilization solutions, such as SRE, PIM-Prune, etc. We implement our SME algorithm framework in the Pytorch framework [44].

### B. Results and Analysis

*1) Accuracy and Sparsity:* Tab. II first shows the NN accuracy for the networks on CIFAR-10 and ImageNet datasets. We can observe that our SME and other solutions are orthogonal and can further improve the effect when combined with the exciting methods with negligible accuracy loss, even on the large-scale dataset. Specifically, for ResNet-50 on the ImageNet, SME combined with PIM-Prune achieves $91.23\%$ sparse rate with $0.6$ accuracy loss compared to SmartExchange ($58.6\%$ sparse rate with $2.07\%$ accuracy loss) and PIM-Prune ($71.91\%$ sparse rate with $1.10\%$ accuracy loss) on ImageNet. For the more compact MobileNet-v2 that has a much smaller model size, it is difficult to exploit the weight sparsity directly. In contrast, our scheme focuses on the bits, which achieves higher accuracy ($71.57\%$) while ensuring the exploited sparsity ($78.74\%$) by squeezing the more significant bits into the positions of the less significant ones.

*2) Energy- and Area-Efficiency:* Fig. 11 shows the energy- and area-efficiency of different accelerators for the four networks. We normalize the energy-efficiency to that of the model without any compression. On average, for ImageNet, SME improves energy efficiency by $2.3\times$ and area efficiency by $6.1\times$ on ResNet-18/50 compared to PIM-Prune and SRE. Even on MobileNet-v2, our method is still superior to the existing methods. The reason is that it is challenging for pruning-based methods to compress networks for large-scale datasets or compact networks with acceptable accuracy. However, massive bit-level sparsities always exist and can be used by our SME. Even on MobileNet-v2, our method also improves the area efficiency by $2.7\times$. The energy and area reduction against PIM-Prune and SRE is mainly due to the reduced crossbar resources and less overhead. Compared to PIM-Prune and SRE, SME improves energy efficiency by $2.3\times$ and area efficiency by $6.1\times$ on average. From the plot, we see that SME achieves higher energy efficiency with the workload grouping scheme and latency-matching pipeline scheme, but it will slightly degrade the area efficiency. This is because SME-WG optimizes the dynamic energy consumption of the system by balancing the workload for the activated crossbar-

TABLE II: Inference accuracy on CIFAR-10 and ImageNet datasets.

| Model | Ori.Acc (%) | Method | Acc.(%) | Spars.(%) |
|---|---|---|---|---|
| CIFAR-10 | | | | |
| VGG-16 | 93.66 | SmartExchange | 92.87 | 92.80 |
| | 93.7 | P-RM | 93.2 | 98.0 |
| | 93.7 | PIM-Prune | 93.23 | 93.10 |
| | **93.7** | **this work** | **93.6** | **84.15** |
| | **93.7** | **this work+PIM-Prune** | **93.18** | **97.11** |
| ResNet-18 | 94.58 | SmartExchange | 94.54 | 91.30 |
| | 94.14 | P-RM | 93.22 | 98.3 |
| | 94.14 | PIM-Prune | 93.84 | 91.71 |
| | **94.14** | **this work** | **94.19** | **83.19** |
| | **94.14** | **this work+PIM-Prune** | **93.85** | **96.97** |
| ImageNet ILSVRC-2012 | | | | |
| VGG-16 | 71.18 | SmartExchange | 70.97 | 87.7 |
| | 74.5 | Pattern-Based | 73.6 | 92.3 |
| | 71.59 | PIM-Prune | 70.02 | 72.35 |
| | 71.59 | **this work** | **71.37** | **69.13** |
| | 71.59 | **this work+PIM-Prune** | **70.91** | **91.27** |
| ResNet-50 | 76.13 | SmartExchange | 74.06 | 58.60 |
| | 76.13 | Pattern-Based | 75.6 | 85.3 |
| | 76.13 | PIM-Prune | 74.91 | 71.91 |
| | **76.13** | **this work** | **76.03** | **67.35** |
| | **76.13** | **this work+PIM-Prune** | **75.46** | **91.23** |
| MobileNet-v2 | 72.19 | SmartExchange | 70.16 | 79.79 |
| | 71.88 | Pattern-Based | 70.47 | 88.21 |
| | 71.88 | PIM-Prune | 70.11 | 77.13 |
| | **71.88** | **this work** | **71.57** | **78.74** |
| | **71.88** | **this work+PIM-Prune** | **71.02** | **84.51** |

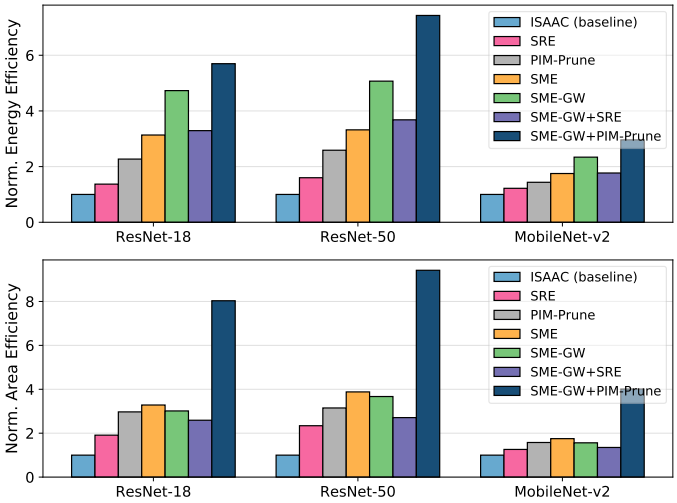


Fig. 11: Normalized energy- and area-efficiency on various neural networks. SME-WG denotes SME combined with workload grouping scheme and latency-matching pipeline scheme.

rows that concurrently execute MAC operations and pipelining each operation with high efficiency. Supporting these policies inevitably introduces some routing procedures. However, these overheads are minimal thanks to the fact that our workload grouping scheme is executed offline (i.e., pre-determined based on the squeeze-out scheme).

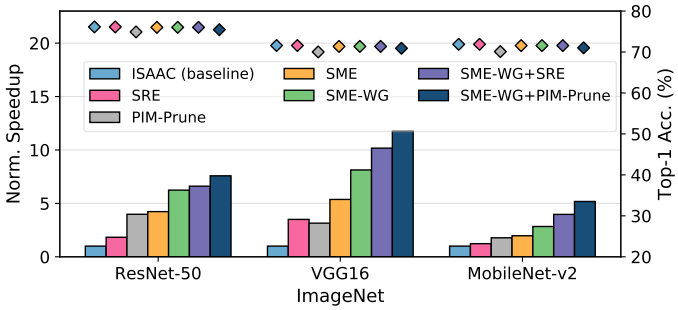


Fig. 12: Comparison of speedup with different ReRAM-based scheme for three networks, which normalized to ISAAC. SME-WG denotes SME combined with workload grouping scheme and latency-matching pipeline scheme.

*3) Performance Analysis:* Fig. 12 shows total execution cycles on ReRAM-based designs for three networks, which are all normalized to the baseline ReRAM-based accelerator ISAAC that does not exploit any sparsity. With the SME, the performance increases from $1.97\times$ to $4.23\times$. For example, regarding ResNet-50, compared to SRE and PIM-Prune, SME achieves nearly $2.3\times$ and $1.1\times$ performance improvement, respectively. This is because SME reduces more crossbar resources while weight-duplication [45] is introduced to maximize memory utilization for improving performance. On the one hand, with our pipeline design, SME-WG achieves significant performance gains, ranging from $2.93\times$ to $6.74\times$. Still taking ResNet-50 as an example, SME-WG can achieve $1.87\times$ performance improvement compared to SME without the pipeline and workload grouping scheme thanks to the workload balance of concurrent execution of MACs on the crossbars in our scheme. On the other hand, previous pruning and sparsity exploration techniques are orthogonal to our scheme and can be combined with our approach to further improve performance. For instance, SME-WG combined with PIM-Prune achieves $1.68\times$ performance improvement compared to stand-alone SME-WG. The performance speedup against stand-alone SME-WG is mainly due to crossbar-aware pruning on the bits resulting in higher bit sparsity on the crossbar. Note that the speedup brought by the combination of schemes (i.e., SME-WG with PIM-Prune) is at the cost of about $0.9\%$ average accuracy loss for the benchmark like ImageNet.

Fig. 12 also shows the accuracy of various ReRAM-based schemes for three networks. For ImageNet, SME shows a significant $1.31\%$ average accuracy improvement over PIM-Prune. We can find that the accuracy of SME-WG is always equivalent to SME because our pipeline design and workload grouping algorithm do not affect the correctness of computation.

*4) Varied squeeze-out schemes with crossbar resource:* Fig. 13 and Fig. 14 report the evolution of accuracy and

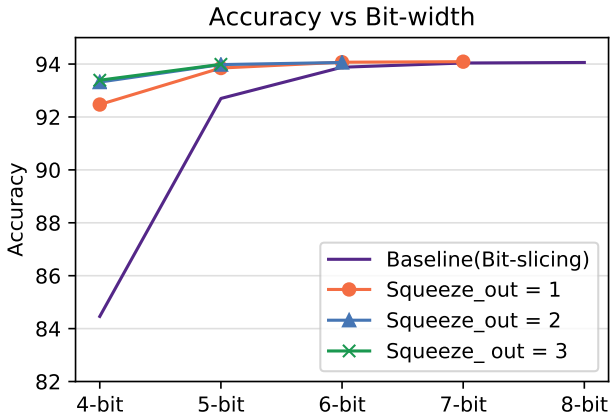


Fig. 13: Comparison of NN accuracy with our different scheme on ResNet-18. Baseline is quantized with INT8 method.

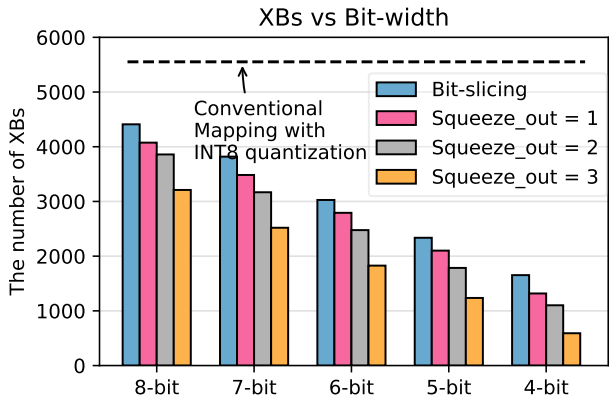


Fig. 14: The number of crossbars required varies as our different schemes on ResNet-18. Baseline is quantized with INT8 method.

crossbar resource with the different squeeze-out schemes, respectively. We respectively compare the accuracy and necessary crossbar resources of squeezing 1, 2, 3 bits. We use the squeeze-out scheme to reduce the number of cells representing weights far better than directly reducing the cells because the MSBs are more critical than the LSBs. If we can reduce the error caused by the MSBs, the overall error can be effectively decreased [20].

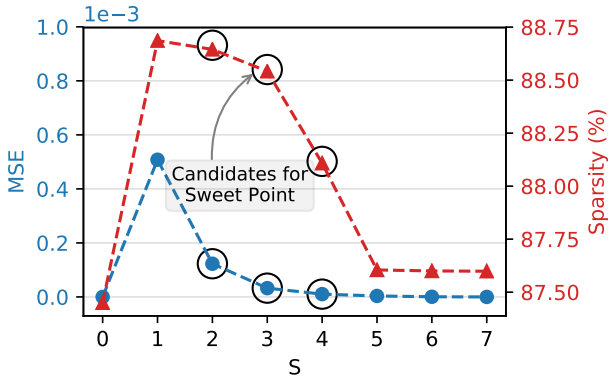


Fig. 15: Trade-off between quantization error and bit-level sparsity along with the change of the consecutive region's size containing '1.' w.r.t. $S$.

*5) Sweet-spot for the size of consecutive region containing*

*'1':* As our discussion in section III-A, Fig. 15 shows the trade-off between the sparsity and quantization error caused by different number $S$ of consecutive '1'. We use mean square error (MSE) to measure the loss caused by quantization, defined as the absolute difference between the exact and the approximate weights [8]. In Fig. 16, we find that if we set $S = 2$, the overall sparsity of the network began to decrease, while $S = 4$, the overall error of the model, is almost zero. However, we combine with the overall sparsity and the bit-level sparse distribution. We find that $S = 3$, SME achieves an optimal point for ResNet-18.

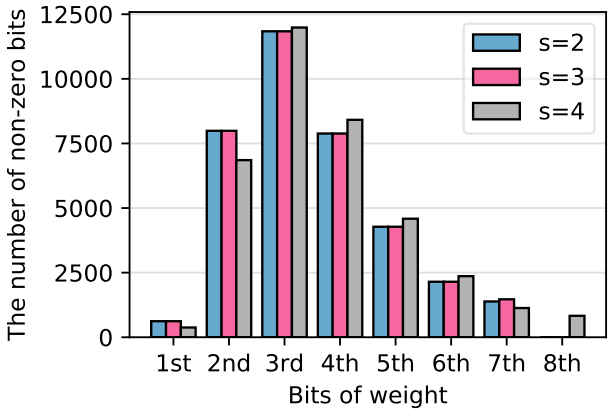


Fig. 16: The sparsity of bit-level under different $S$ in the candidate set for sweet point.

*6) Overhead Analysis:* Fig. 17 shows the storage overhead of different networks. On average, SME achieves $84.6\%$ and $98.1\%$ register overhead reduction compared to PIM-Prune and SRE with only quantization and bit-slicing scheme; it achieves $77.8\%$ and $96.8\%$ register overhead reduction compared to PIM-Prune and SRE further combined with the squeeze-out scheme. However, the significant reduction in overhead benefits from two parts: (1) our squeeze-out scheme solves the index's problem for aligning the output by processing the input. (2) we retain crossbars if the crossbar cannot be released, so the index is continuous. Specifically, for ResNet-50, compared to SRE, which requires the 778KB register for the index, our SME consumes nearly $100\%$ (i.e., without input index) and $95.6\%$ less storage of input index and output index. However, the significant reduction in overhead benefits from our algorithm framework design – the padded output is continuous by our SME since we aim to reduce the crossbar resources.

## C. Design Exploration

In this section, we discover our method can support the network with intra-layer mixed-precision [26], and also support MLC-based crossbar but perform better on SLC. We also discuss the scalability of our scheme in terms of the adopted quantization method.

*1) Support intra-layer mixed-precision:* Fig. 18 shows the crossbar ($128 \times 128$) consumption of ResNet-18 quantized by intra-layer mixed-precision under conventional mapping and our SME method. The pie chart of the Fig. 18 also shows the

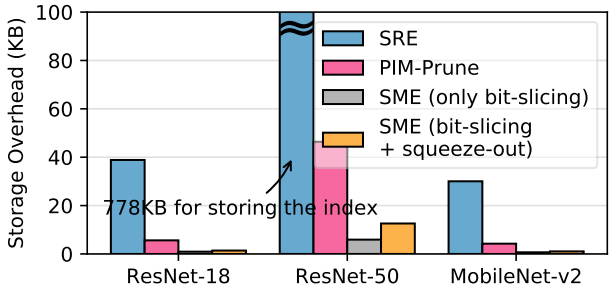


Fig. 17: Comparison of storage overhead with different networks.

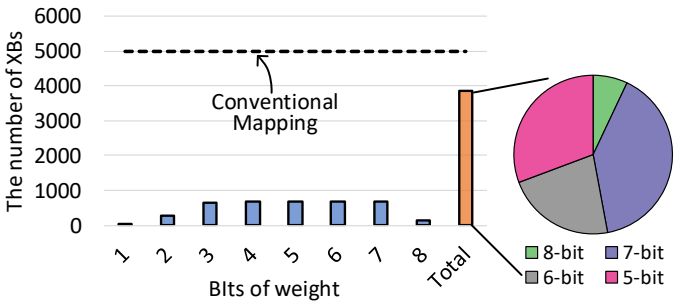


Fig. 18: The number of crossbar for ResNet-18 with mixed-precision.

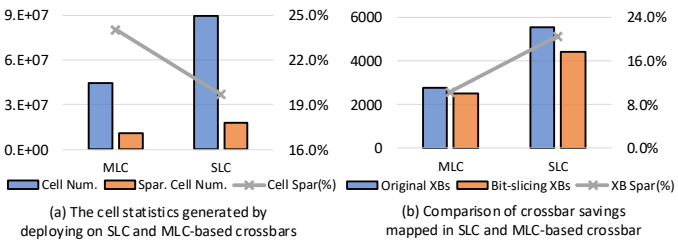


Fig. 19: Comparison of ResNet-18 deployed on SLC/MLC-based cell.

mixed-precision contains 5 to 8-bit in layers of ResNet-18. The conventional mapping approach cannot take advantage of mixed-precision benefits due to structural-coupling. The weights within a filter are mapped to crossbars, and the maximum bit-width of weights determines the number of cells required per weight. So there are massive sparse cells. In contrast, the SME slices each bit of the weight into different crossbars, aggregating the bits' sparsity. In such a way, we can achieve decoupling of the crossbar structure, reducing over $1,000$ crossbars than the conventional mapping method. This is because the squeeze-out scheme integrates weight quantization and network sparsification. In the process of squeeze-out, we selectively discard the low significant bits and swap them with more significant bits at the same locations. This not only achieves the approximate irregular sparsity utilization on ReRAM-based crossbar but also achieves the effect of mixed-precision quantization, which maximizes the utility of compression.

*2) Support MLC-based crossbar:* MLC suffers from more variation and reliability problems than SLC since the SLC technology is more mature and stable than MLC. Therefore, this paper adopts the SLC-based ReRAM crossbar. Here, we also explore the scalability of our proposed scheme on the MLC-based ReRAM crossbar. Fig. 19 shows that our bit-

slicing scheme is also applicable to MLC-based crossbar. The number of sparse cells significantly decreases when the network is mapped onto the MLC-based crossbar. Thus, our bit-slicing scheme's benefit is also affected, as we reduce approximately $11\%$ crossbars compared to the conventional mapping scheme. Besides, we can use the squeeze-out scheme to save resources further, and to squeeze one MLC-based cell is equivalent to squeezing 2-bit on the SLC-based crossbar.

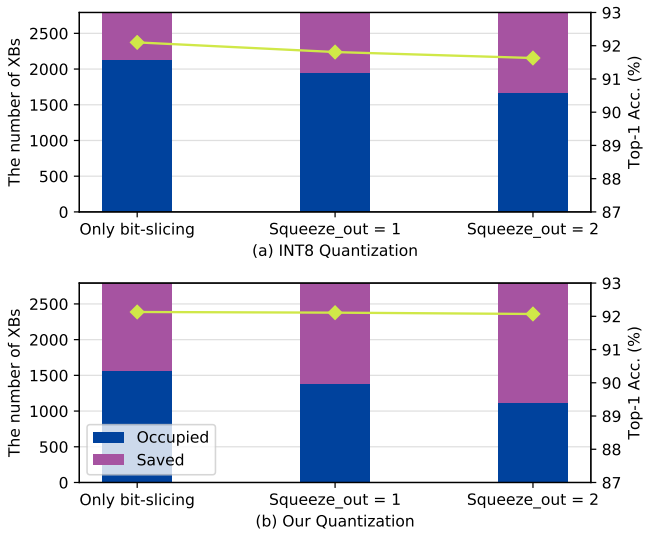


Fig. 20: Analyzation of the scalability in terms of the adopted quantization method. (a) The number of crossbar and accuracy under (a) general quantization method (i.e., INT8) and (b) our quantization for ResNet-20 on CIFAR-10.

*3) Support general quantization method:* To further analyze the scalability provided by the proposed scheme, Fig. 20 studies the impact of crossbar resources for storing the weights and NN accuracy on ResNet-20 considering the adopted quantization method. We sweep squeezed bits from 0 (i.e., only bit-slicing scheme) to 2. Generally, more saved crossbars (purple bars) mean our scheme is better and allows for better performance, but it will degrade the NN accuracy. However, the model processed by our scheme provides a distribution of bit sparsity consistent with our scheme benefits from our sparse pattern and ADMM-based optimization method. In such a way, our scheme achieves higher accuracy and more crossbars are saved compared to INT8. From this plot, we find that even with the most popular quantization method, INT8, SME achieves a reduction of $1124$ crossbars with less than $0.5\%$ accuracy loss for compact ResNet-20, which is remarkable and demonstrates the scalability of our scheme.

## VI. Conclusion

Bit-level sparsity cannot be utilized, leading to the limited performance of NN inference. We propose **SME**, an algorithm-hardware co-design framework that decouples the hardware dependence of multiplication to release the sparse cells in the crossbars for higher energy-/area-efficient inference of NNs. We design the architecture to efficiently support our algorithm through well-designed crossbars with the peripheral circuit. In addition to the architecture design, we develop a latency-matching pipeline scheme and workload grouping algorithm for SME to balance workload and exploit the inherent constraints among crossbars. Putting all together, our evaluation shows that the proposed SME outperforms other similar solutions in terms of energy, performance, and accuracy.

## Acknowledgment

This work was partially supported by the National Natural Science Foundation of China (Grant No. 61834006 and 62102257) and the National Key Research and Development Program of China (2018YFB1403400). We thank Wu Wen Jun Honorary Doctoral Scholarship, AI Institute, Shanghai Jiao Tong University.

## References

[1] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[2] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[3] T. Chou, W. Tang, J. Botimer, and Z. Zhang, "Cascade: Connecting rrams to extend analog dataflow in an end-to-end in-memory processing paradigm," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 114–125.

[4] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, 2020.

[5] R. Guo, Z. Yue, X. Si, T. Hu, H. Li, L. Tang, Y. Wang, L. Liu, M.-F. Chang, Q. Li *et al.*, "15.4 a 5.99-to-691.1 tops/w tensor-train in-memory-computing processor using bit-level-sparsity-based optimization and variable-precision quantization," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 242–244.

[6] N. Challapalle, S. Rampalli, L. Song, N. Chandramoorthy, K. Swaminathan, J. Sampson, Y. Chen, and V. Narayanan, "Gaas-x: Graph analytics accelerator supporting sparse data representation using crossbar architectures," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 433–445.

[7] F. Liu, W. Zhao, Y. Zhao, Z. Wang, T. Yang, Z. He, N. Jing, X. Liang, and L. Jiang, "Sme: Reram-based sparse-multiplication-engine to squeeze-out bit sparsity of neural network," in *Proceedings of the 39th IEEE International Conference on Computer Design*, 2021.

[8] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[9] F. Liu, W. Zhao *et al.*, "Im3a: Boosting deep neural network efficiency via in-memory addressing-assisted acceleration," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, 2021.

[10] X. Yang, B. Yan, H. Li, and Y. Chen, "Retransformer: Reram-based processing-in-memory architecture for transformer acceleration," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[11] Y. Ji, Y. Zhang, X. Xie, S. Li, P. Wang, X. Hu, Y. Zhang, and Y. Xie, "Fpsa: A full system stack solution for reconfigurable reram-based nn accelerator architecture," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 733–747.

[12] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[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 *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 639–644. [Online]. Available: https://doi.org/10.1145/3287624.3287715

[14] L. Liang, L. Deng, Y. Zeng, X. Hu, Y. Ji, X. Ma, G. Li, and Y. Xie, "Crossbar-aware neural network pruning," *IEEE Access*, vol. 6, pp. 58 324–58 337, 2018.

[15] G. Yuan, P. Behnam, Z. Li, A. Shafiee, S. Lin, X. Ma, H. Liu, X. Qian, M. N. Bojnordi, Y. Wang *et al.*, "Forms: Fine-grained polarized reram-based in-situ computation for mixed-signal dnn accelerator," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[16] F. Liu, W. Zhao, Z. He *et al.*, "Bit-transformer: Transforming bit-level sparsity into higher preformance in reram-based accelerator," in *Proceedings of the 40th International Conference on Computer-Aided Design*, 2021.

[17] 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 *DAC*. IEEE, 2018, pp. 1–6.

[18] H. Ji, L. Song, L. Jiang, H. Li, and Y. Chen, "Recom: An efficient resistive accelerator for compressed deep neural networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 237–240.

[19] C. Chu, Y. Wang, Y. Zhao, X. Ma, S. Ye, Y. Hong, X. Liang, Y. Han, and L. Jiang, "Pim-prune: fine-grain dcnn pruning for crossbar-based process-in-memory architecture," in *DAC*. IEEE, 2020, pp. 1–6.

[20] Y. Cai, T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Low bit-width convolutional neural network on rram," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 7, pp. 1414–1427, 2019.

[21] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 236–249.

[22] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 907–922.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[24] L. Deng, L. Liang, G. Wang, L. Chang, X. Hu, X. Ma, L. Liu, J. Pei, G. Li, and Y. Xie, "Semimap: A semi-folded convolution mapping for speed-overhead balance on crossbars," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 1, pp. 117–130, 2018.

[25] F. Liu, W. Zhao, Z. Wang, Y. Zhao, T. Yang, Y. Chen, and L. Jiang, "Ivq: In-memory acceleration of dnn inference exploiting varied quantization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[26] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[27] H. Yang, L. Duan, Y. Chen, and H. Li, "Bsq: Exploring bit-level sparsity for mixed-precision neural network quantization," in *9th International Conference on Learning Representations (ICLR)*, 2021.

[28] F. Liu, W. Zhao, Z. He *et al.*, "Improving neural network efficiency via post-training quantization with adaptive floating-point," in *International Conference on Computer Vision (ICCV)*, 2021.

[29] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[30] S.-E. Chang, Y. Li, M. Sun, R. Shi, H. K.-H. So, X. Qian, Y. Wang, and X. Lin, "Mix and match: A novel fpga-centric deep neural network quantization framework," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 208–220.

[31] X. Chen, J. Jiang, J. Zhu, and C.-Y. Tsui, "A high-throughput and energy-efficient rram-based convolutional neural network using data encoding and dynamic quantization," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 123–128.

[32] W.-H. Chen, K.-X. Li, W.-Y. Lin, K.-H. Hsu, P.-Y. Li, C.-H. Yang, C.-X. Xue, E.-Y. Yang, Y.-K. Chen, Y.-S. Chang *et al.*, "A 65nm 1mb nonvolatile computing-in-memory reram macro with sub-16ns multiply-and-accumulate for binary dnn ai edge processors," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 494–496.

[33] Y. Li, X. Dong, and W. Wang, "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks," in *8th International Conference on Learning Representations (ICLR)*, 2020.

[34] Y. Zhao, X. Chen, Y. Wang, C. Li, H. You, Y. Fu, Y. Xie, Z. Wang, and Y. Lin, "Smartexchange: Trading higher-cost memory storage/access for lower-cost computation," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 954–967.

[35] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[36] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," *ASPLOS*, vol. 53, no. 2, pp. 1–14, 2018.

[37] M.-F. Chang, J.-J. Wu, T.-F. Chien, Y.-C. Liu, T.-C. Yang, W.-C. Shen, Y.-C. King, C.-J. Lin, K.-F. Lin, Y.-D. Chih *et al.*, "19.4 embedded 1mb reram in 28nm cmos with 0.27-to-1v read using swing-sample-and-couple sense amplifier and self-boost-write-termination scheme," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 332–333.

[38] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[39] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[41] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

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

[43] X. Ma, G. Yuan, S. Lin, Z. Li, H. Sun, and Y. Wang, "Resnet can be pruned 60×: Introducing network purification and unused path removal (p-rm) after weight pruning," in *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2019, pp. 1–2.

[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[45] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "Dnn+ neurosim v2. 0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.