Journal of Systems Architecture 153 (2024) 103192

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

CRPIM: An efficient compute-reuse scheme for ReRAM-based Processing-in-Memory DNN accelerators

Shihao Hong [a], Yeh-Ching Chung [b,*]

[a] School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, 518172, Guangdong, China
[b] School of Data Science, The Chinese University of Hong Kong, Shenzhen, 518172, Guangdong, China

ARTICLE INFO

Keywords:
Resistive random access memory (reRAM)
Deep neural network accelerator
Repetition reuse

ABSTRACT

Resistive random access memory (ReRAM) is a promising technology for AI Processing-in-Memory (PIM) hardware because of its compatibility with CMOS, small footprint, and ability to complete matrix–vector multiplication workloads inside the memory device itself. However, redundant computations are brought on by duplicate weights and inputs when an MVM has to be split into smaller-granularity sequential sub-works in the real world. Recent studies have proposed repetition-pruning to address this issue, but the buffer allocation strategy for enhancing buffer device utilization remains understudied. In preliminary experiments observing input patterns of neural layers with different datasets, the similarity of repetition allows us to transfer the buffer allocation strategy obtained from a small dataset to the computation with a large dataset. Hence, this paper proposes a practical compute-reuse mechanism for ReRAM-based PIM, called CRPIM, which replaces repetitive computations with buffering and reading. Moreover, the subsequent buffer allocation problem is resolved at both inter-layer and intra-layer levels. Our experimental results demonstrate that CRPIM significantly reduces ReRAM cells and execution time while maintaining adequate buffer and energy overhead.

1. Introduction

Non-volatile memory (NVM) is prominent for addressing the "Memory Wall" problem and has been widely applied to Processing-in-Memory (PIM). Resistive Random Access Memory (ReRAM) stands out among PIM devices due to its compact size and lower energy consumption. ReRAM crossbar arrays (XBAs) perform analog computation of MVM with high parallelism. According to Kirchhoff's law, currents representing the computation results are generated simultaneously in a ReRAM matrix, reducing the computational complexity from $O(n^2)$ to $O(1)$. Memristor crossbar-based PIM has been adopted in many fields, such as image recognition [1], graph processing [2,3], signal processing [4], DNA alignment [5,6], and wave simulation [7].

Though many ReRAM-based DNN accelerators (e.g. [8,9]) have been proposed, the real-world implementation confronts challenges related to both temporal and spatial dimensions, affecting the device's resilience.

(1) **ReRAM memory Space.** Matrix–vector multiplication (MVM) on ReRAM-based DNN accelerators relies on a foundation of massive XBAs. For example, to support the full precision network of ResNet-50 [10], more than 49,800 single-bit 128 × 128 XBAs are required, yet recent advanced models are much larger.

(2) **Time consumption.** Various technological constraints and practical non-idealities of ReRAM-based accelerators have been studied [11–13]. To ensure that the variation in calculation accuracy is within a tolerable limit, one solution is to make MVM computations be performed at a finer level known as an Operation Unit (OU) [14,15]. However, performing MVM computation at the OU level necessitates breaking down a single MVM operation into OU-level sub-MVM operations (OU-MVMs), which increases the overall computation time.

Various sparsity-aware methods have been devised to reduce the requisite ReRAM cells. Structured specification methods generate group-wise sparse models by nullifying all weights within specific groups. Auto-Prune [16] learns a column-wise sparse model that is friendly to pruning ReRAM cells. Approaches such as ReCom [17], SNrram [18], and SRE [19] integrate structural sparsity and eliminate trivial columns or XBAs. They reduce the proportion of ReRAM cells that store zeros, resulting in more compact accelerators. For models that lack inherent structural sparsity, contemporary research explores weight and activation redundancy to condense XBAs. The MVM computation is decomposed into smaller granularity, e.g., XBA-level, column-level, and block-level. Repetition-aware methods remember the repeated patterns

* Corresponding author.
E-mail addresses: shihaohong@link.cuhk.edu.cn (S. Hong), ychung@cuhk.edu.cn (Y.-C. Chung).

https://doi.org/10.1016/j.sysarc.2024.103192
Received 13 January 2024; Received in revised form 19 April 2024; Accepted 27 May 2024
Available online 31 May 2024

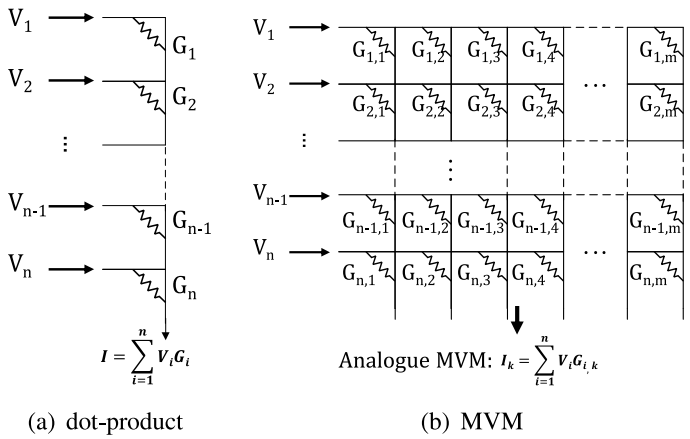


Fig. 1. Illustration of ReRAM-based (a) Dot-product of two vectors, (b) Matrix–Vector Multiplication.

Table 1
Normalized input cycles and unit output sizes of the layers with the highest (top1 and top2) and lowest (low1 and low2) cycle costs across four models.

Model	Normalized input cycles/output size			
	top1	top2	low1	low2
ResNet-50	785-1	785-4	1-500	3-576
AlexNet	101-1	10-13.2	1-1624.9	1-722.2
VGG-16	1407-1	1407-21.3	1-59468	1-9709
GoogLeNet	785-1.3	257-3.1	1-333	3-216

of weights and inputs and bypass redundant computations, thereby reducing time overhead. RePIM [20] suggests using indexing tables to share computational outcomes among repetitive weights and inputs. PattPIM [21] treats OU blocks as weight patterns and records repeated instances in a table to decrease redundant storage and computations.

However, these studies rarely discuss the implementation and performance aspects of the input-sharing scheme. Assume patterns are input vectors for an OU block. The sharing process needs to record unique pattern vectors and their associated outcomes, termed pattern results. Storing all pattern results incurs a spatial complexity of $O(\frac{2^b log l}{l} HW)$. Here, H and W denote the height and width of the weight matrix, while l represents the length of an input pattern vector. To perform full sharing (all pattern results are buffered), it consumes more memory for the indexing table than the weight matrix. In current studies, little correlation has been utilized between the overhead of neural layers and the buffering strategy of repetitive patterns. Table 1 shows the summary of input cycles and output size of the layers with the highest 2 and lowest 2 cycle costs across models. The cost of layers shows an imbalance distribution. The most time-consuming layers produce a rather small unit output, inspiring the buffer strategy leaning towards the layers with fewer storage costs but higher time-saving benefits. In other words, the buffer allocation strategy should take into account the inter-layer imbalance in computation demand.

In this paper, we propose CRPIM, a practical compute-reuse mechanism exploring the repetitiveness of weights and inputs. CRPIM reduces a considerable amount of needed ReRAM cells and shrinks the computation time by buffering frequently used OU-MVM results. This study also balances the memory overhead other than ReRAM devices. Our contributions are as follows:

(1) We divide the input vectors into OU-inputs and analyze the frequency distribution of input patterns. The preliminary experiments reveal that certain input patterns are frequently repeated while changing datasets, allowing us to replace the repetitive computations with buffering and reading. We then define the intra-layer and inter-layer buffer allocation problems subject to the constraint of limited buffer size. We show that the latter problem is a variation of the bounded knapsack problem, and a dynamic programming approach is proposed.

(2) We propose a Compute-Reuse mechanism for weight patterns to convert part of the computational work with repetitive weight patterns into buffering and reading. Based on Input Compute-Reuse (ICR) and Weight Compute-Reuse (WCR), we propose the CRPIM and introduce the architecture design.

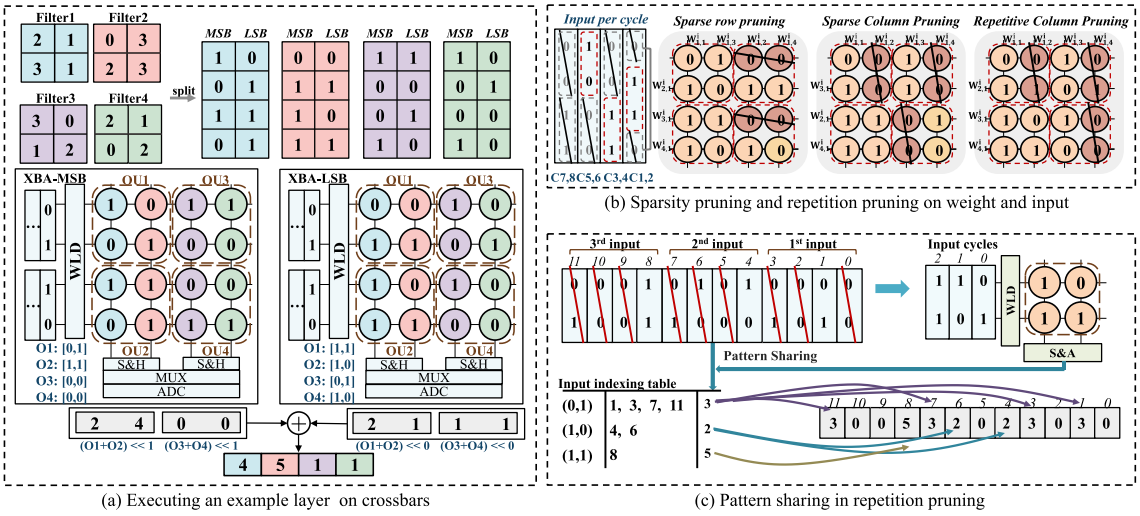
(3) We evaluate the performance of CRPIM with several popular DNN models on the Imagenet dataset. The results show that though the storage overhead of CRPIM is linear to weight size, it achieves up to 2.63× speedups and an average of 37.9% ReRAM compression rate compared to the recent ReRAM-based accelerators. Its energy consumption is on par with SRE for most models.

2. Preliminaries and motivation

2.1. ReRAM crossbar-based DNN accelerator

ReRAM, a non-volatile memory device, owns the advantages of low power, compact size, and compatibility with complementary metal–oxide–semiconductor (CMOS) technology. ReRAM crossbar arrays (XBAs) arrange ReRAM cells as a matrix, which can perform dot-product operations and store data simultaneously. As depicted in Fig. 1(a), when ReRAM cells are configured in a 1D array, voltages function as inputs, and each cell conductance represents a weight value. The resulting currents converge to yield the analog outcome of the dot product between two vectors, i.e., the sum of the currents, denoted by $I = \sum V_i \cdot G_i$. Notably, the generation and summation of currents are completed within a single cycle. By arranging ReRAM cells in a 2D array, as shown in Fig. 1(b), summed currents in all columns reach the ends simultaneously. Consequently, the computational complexity of matrix–vector multiplication (MVM) operations is reduced from $O(n^2)$ to $O(1)$. Because MVMs account for the bulk of computational demands in neural networks, many DNN accelerators utilize XBAs as MVM processing units.

ISAAC [9] is a pipeline-aware yet over-idealized CNN accelerator based on ReRAM that ignores the influence of non-ideal electrical properties on computation. DL-RSIM [14] simulates the error rate of practical ReRAM-based DNN accelerators, illustrating the necessity of splitting MVM into multiple operation units (OUs). SRE [19] employs OUs in accelerator design, limiting the number of activated word-lines and bitlines to the OU size. Some studies use single-level-cell (SLC) ReRAM devices to improve error resilience. Bit-Transformer [22] and SoBS-X [23] suggest decomposing an N-bit fixed-point weight matrix into N bit-matrices and mapping 1-bit values with identical



(a) Executing an example layer on crossbars

(b) Sparsity pruning and repetition pruning on weight and input

(c) Pattern sharing in repetition pruning

Fig. 2. Examples of splitting convolutional computations to OU computations and weight/input compression.

bit-positions onto the same XBA. Fig. 2(a) presents an OU-based convolutional computation using bit-split ReRAM XBAs. The 2-bit filter weights are mapped onto two XBAs, XBA-MSB and XBA-LSB, which store the most significant bits and least significant bits, respectively. An XBA is further logically divided into a collection of OUs in computation. In each cycle, only one OU of wordlines and bitlines is activated. Subsequently, Analog-to-Digital Converters (ADCs) obtain the accumulated currents from sample-and-hold ($S\&H$) circuits and convert them into digital data. In Fig. 2(a), O1 and O2 are accumulated, then O3 and O4 are accumulated. All XBAs perform computations simultaneously, whose results are shifted and added to form the MVM output. However, ReRAM device overhead is a concern for DNN models. For the full-precision ResNet-50 model, which comprises 25.6 million parameters, accommodating all weight filters within (128×128) SLC crossbar arrays necessitates in excess of 49,800 ReRAM crossbar arrays. Additionally, executing MVMs in serial OU blocks diminishes the degree of parallelism.

2.2. XBA pruning

Sparsity pruning is a common technique to reduce the XBA demand for ReRAM-based accelerators. To perform XBA-grained computations, PIM-Prune [24] iteratively removes less critical columns and rows from a coarse-grained unit block, which is larger than the XBA, until the pruned block can fit into an XBA. For OU-grained computations, a large MVM is partitioned into multiple OU-MVMs. SRE [19] reduces ReRAM cells by rearranging the rows and columns of the XBA and removing zero or near-zero OU columns/rows. For instance, the first XBA in Fig. 2(b) results from swapping the 2nd and 3rd columns of XB-LSB in Fig. 2(a), while the second XBA is derived from interchanging its 2nd and 3rd rows. The all-zero OU rows or columns are grouped together and skipped.

Repetition sharing is another widely explored strategy to reduce computational redundancy in large MVMs. RePIM [20] performs computations only on unique OU-columns to save repetitive computations. The third XBA in Fig. 2(b) shows that the repeated [0,1] on the first OU-row are skipped so that the number of OU-MVM is reduced from two to one.

2.3. Input activation pruning

In addition to weights, input activations are also highly compressible due to their sparsity and repetitions. In the field of ASIC design, Bit-Serial Cache [25] introduces a cache-assisted Processing Engine (PE) architecture to buffer the partial results produced by repeated input activations. In the ReRAM-based accelerator area, SRE avoids wasting clock cycles on inactive inputs by using dynamic OU formation and skipping zeros and repetitive inputs. RePIM [20] shares computation results among repetitive inputs to reduce redundant computations. Fig. 2(b) shows an example that SRE compresses an MVM from 16 OU operations to 6 by forming OUs dynamically. In the example, four newly formed all-zero vectors are skipped, and the earlier input vector [1,1] is reused for one time. Fig. 2(c) shows the repetitive input-sharing method in RePIM, where only the unique input patterns are kept for OU-MVM. The 12 MVM operations are replaced by 3 unique MVMs and 7 data-sharing operations. Repetition pruning in SRE and RePIM greatly reduces the computations with redundant input. However, given that the MVM result of each unique input pattern occupies $O(\frac{logb}{l}HW)$ of space, full sharing – where all pattern results are stored in buffers – results in a spatial complexity of $O(\frac{logb}{l}HW2^l)$, with l equal to H_{OU}. The storage overhead becomes untenable, particularly for neural layers with large feature maps. Moreover, the objective of achieving full sharing exacerbates this challenge. Therefore, it is crucial to investigate methods for maximizing the pruning effect with limited buffer resources.

2.4. Motivation

Previously mentioned weight or input pruning methods have done little to explore the concentrated distribution of inputs. We conduct the first pre-experiment to investigate the distribution of input patterns. Let OU be a (8×8) block and define input patterns as elements within the power set of an H_{OU}-bit binary vector space, where H_{OU} is the height of an OU block. We unfold the input feature map into multiple cycles of 1-bit input vectors and further divide them into multiple OU-rows. We take 1024 images as the batch size for inference and randomly select an OU-row to observe the OU-inputs composition for all layers. Fig. 3 shows the composition of OU-inputs for Alexnet [26], GoogleNet [27], ResNet-50 [10], and VGG-16 [28]. OU-inputs exhibit over 36% sparsity in three of four models. The remaining non-zero OU-inputs are distributed unevenly across the 255 patterns, with at least 22% concentrated in 32 patterns. Such concentrated characteristics shared by the four models provide the potential for OU-input pruning.

We further conduct pre-experiments to determine whether the frequent input patterns are still prevalent when changing the dataset. In other words, the pre-experiment explores whether the centralized distribution of input patterns overlaps across different datasets. Two non-overlapping datasets are independently and randomly selected

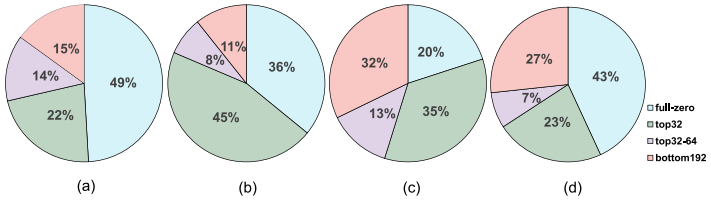


Fig. 3. Composition of OU-inputs in (a) Alexnet, (b) GoogLeNet, (c)ResNet-50, (d) VGG-16.

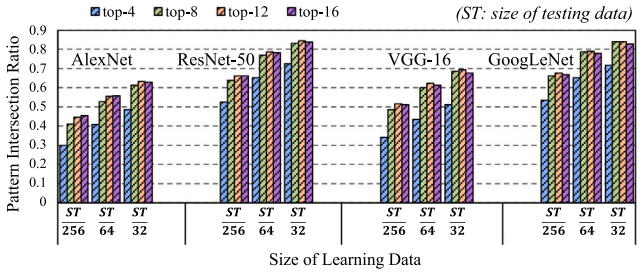


Fig. 4. Intersection Ratio of Input Patterns with different top-k and learning data sizes.

from the ImageNet dataset, named learning and testing data. We collect the k (k ranging from 4 to 16) most frequent input patterns from the two datasets. The intersection ratio, i.e., the overlap of the two top-k pattern sets as a proportion of k is calculated for all layers. We maintain a fixed testing data size (tsize) and adjust the learning data size (*lsize*) from $\frac{tsize}{256}$ to $\frac{tsize}{32}$. Fig. 4 illustrates the impact on the average pattern intersection ratio for varying lsize and k values. In the context of limited learning data, specifically when the learning size (lsize) is set to $\frac{tsize}{256}$, it can be noted that the top four patterns continue to demonstrate an intersection ratio exceeding 30% for all models. For each model, a consistent increase in the ratio is observed as lsize expands, with the value of k remaining fixed. However, as k grows, the intersection ratio tends to reach the maximum when k approaches 12. This finding inspires us to derive a buffer strategy from the learning dataset that is also applicable to the testing dataset, allowing for buffering a small fraction of computation results to eliminate the majority of repetitive computations.

3. Compute-reuse scheme

In this section, we present the overview of the compute-reuse scheme and show the details of ICR scheme for input patterns and the WCR scheme for weight patterns.

3.1. Overview of compute-reuse scheme

Based on the above observations, we propose a buffering-based compute-reuse framework named CRPIM, where we divide the process of computing MVMs into two modules: Input Compute-Reuse (ICR) scheme and Weight Compute-Reuse (WCR) scheme. As depicted in Fig. 5, input vectors initially go through the ICR part, where an offline-learned pattern buffer strategy categorizes the vectors into buffered and unbuffered patterns. For buffered patterns, the results are directly retrieved from buffers, bypassing the need to conduct computation. Unbuffered patterns, on the other hand, are processed by the WCR scheme. In WCR, input vectors first compute MVM with all weight patterns, and the pattern results are then used to construct the complete MVM outputs. The detailed discussion regarding the strategy of pattern buffer allocation and WCR scheme will be introduced in Sections 3.2 and 3.3.

3.2. Compute-reuse for input patterns

We first utilize the time and space demand among neural layers and consider the proportional strategy. Fig. 6 illustrates the size distribution of input and output data among neural layers in the ResNet-50 model. From the first to the last layer, the output size of a unit input cycle has a growing trend, inversely correlating with the diminishing scale of input data. Since the size of a unit output is a constant multiple of the size of the weight matrix, the proportional strategy assigns buffers to each layer in line with the growth of the weight size of the layer. The strategy generally guarantees that layers with more input cycles and less unit overhead obtain computation results mainly by reading the buffer, while layers with fewer cycles and more unit overhead obtain results mainly by regular computation. However, it ignores the fact that two layers with similar shapes may also differ significantly in the degree of input repetition, which results in the objective of reducing buffer overhead not being fully explored. Therefore, we propose a buffer allocation strategy tailored to repetitive input patterns at each layer, which will be introduced from the intra-layer part and inter-layer part.

First, we consider the scenario within a layer. Given that MVMs are serially performed in OU-level for each XBA, we map each $H_{\mathrm{OU}} \times W_M$ OU-row to an individual XBA. Thereby, the parallelism among OU-rows is maximized. Even though 20.7% layers of GoogleNet have too few columns in the weight matrix to make one OU-row completely occupy an XBA, these weights contribute only 2.4% of GoogleNet's overall weight composition. As all OU-rows perform MVMs in parallel, the efficiency of one neural layer is primarily determined by the most time-consuming OU-row. In other words, the "cask effect" influences the intra-layer buffer allocation policy. The worst performer among all OU-rows, i.e., the row that saves the least amount of computation, acts as the overall performance metric.

The computational savings for the ith OU-row with b_i buffer assigned is calculated as the sum of the top b_i terms of the frequency distribution f_i of input vectors, i.e., $si = \sum_{j=1}^{b_i} f_{i,j}$. Since the time consumption of a neural network layer is dominated by the OU-row that takes the longest time, the optimization problem for minimizing the time consumption of a layer is equivalent to maximizing the minimum saving among all OU-rows, with the limitation of buffer size. The formal problem formulation is expressed as the following Problem 1. By solving this optimization problem, we can determine the optimal buffer allocation strategy that achieves maximum computational savings for the entire neural network layer.

$$\max_{b} \min_{i} \sum f_{i,b_i} \quad \text{s.t.} \begin{cases} 1 \le i \le \lceil \frac{H_M}{H_{OU}} \rceil \\ 0 \le b_i \le 256 \\ \sum b_i \le B \end{cases} \tag{1}$$

Algorithm 1 shows the solution to the intra-layer allocation problem. The inputs include the information on each layer, which is preprocessed before the algorithm. With the OU-input streams for OU-rows, the frequency of pattern k, f_k can be counted easily. We sort patterns according to f_i in descending order so that f_i represents the most achievable saving when allocating the ith unit buffer to the layer. The profit p in Algorithm 1 stores the sorted frequencies, and maxQ represents the buffer size allocated to the layer. The algorithm follows

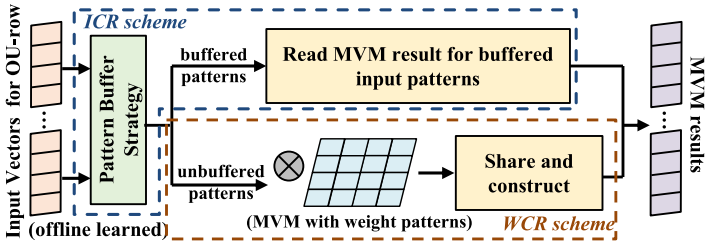


Fig. 5. The overview of Compute-Reuse Scheme for MVM.

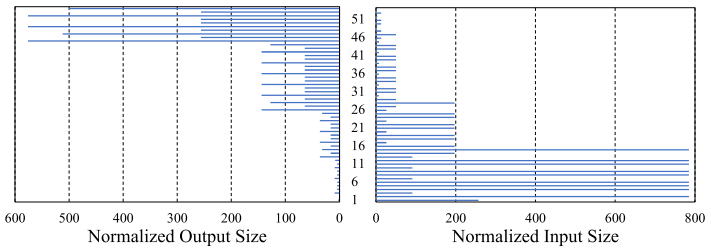


Fig. 6. The Distribution of Output and Input Data Overhead Among Neural Layers in ResNet-50.

a progressive strategy that allocates one unit buffer to the OU-row with the least saved computations each time. After each unit allocation, the marginal profit is updated for the selected OU-row.

Algorithm 1: Intra-Layer Buffer Allocation

Input : #Layers **N**, Layers **L** with: Max quantity **maxQ** and Profit **p**
Output: Layers L

```
/* Part 1:solve P1 for all layers                    */
for i = 0 to N-1 do
    Initialize L[i].p as an all-zero list
    for j = 1 to L[i].maxQ do
        Collect current profit of OU-rows to pc
        Collect marginal profit of OU-rows to pm
        r ← argmin(pc)
        L[i].p[j] ← pm[r] + L[i].p[j]
        Update marginal profit pm[r]
return L
```

Solving Problem 1 yields computational savings for one layer with given buffer sizes. We extend it to inter-layer allocation, which is transformed into a variant of the bounded knapsack problem when the total buffer capacity is considered as the knapsack size. Specifically, buffering a unit result of each layer is regarded as a commodity, where the computational savings and the required buffer size of each unit buffering serve as the profit (p) and weight (w) of the commodity, respectively. A commodity can be chosen repeatedly, and the profit is a function of the number of the commodity. Therefore, the inter-layer buffer allocation problem (Problem 2) is a variant of the bounded knapsack problem with a maximum quantity constraint and unfixed commodity values, which can be expressed as follows:

$$max \sum_j P_j[Bj] \quad \text{s.t., } B_j \geq 0 \sum_j B_j \cdot W_j \leq B \tag{2}$$

where $P_j[b]$ denotes the gain of the jth layer at a given b. B_j represents the number of unit buffers allocated to the jth layer, each capable of buffering a pattern result. W_j is the buffer space required to buffer a single pattern result for the jth layer. The sum of the buffer sizes of all layers is constrained by B. We can solve the bounded knapsack problem by using dynamic programming. Let Pmax(i, w) represent the maximal available profit with the first i commodities and w units of capacity and $P_i[k]$ be the cumulative profits of commodity i at the quantity k, and x_i be the maximal quantity of commodity i. The recurrence for searching for the maximum profit is:

$$\begin{aligned} Pmax(i,w) &= max(Pmax(i-1, w-k\cdot w_i) + P_i[k]) \\ & k \in [0, min(x_i, \lfloor \frac{w}{w_i} \rfloor)] \end{aligned} \tag{3}$$

The optimality of the strategy in Eq. (3) can be proved by induction on i. Let Opt(i,w) be the optimal profit for Problem 2 with the first i items and w units of capacity. We claim $Pmax(i,w) \leq Opt(i,w)$. **Step1.** Let i = 0. We cannot make any profit when 0 items are available. So, $Pmax(0,w) = 0 = Opt(0,w)$. **Step2.** Let $i = k > 0$ Assume that the optimality holds for $i = k > 0$, i.e., $Pmax(k,w) = Opt(k,w)$. **Step3.** Suppose $Opt(k+1,w)$ selects the quantity n' for the item k+1. The following Eqs. (4) and (5) show that $Pmax(k+1,w) = Opt(k+1,w)$, i.e., the solution generated by (3) can generate optimal profit.

$$Opt(k+1,w) = Opt(k, w - n' \cdot w_{k+1}) + P_{k+1}[n'] \tag{4}$$

$$\begin{aligned} & Pmax(k+1,w) \\ &= \max_{0 \leq n < \lfloor \frac{w}{w_{k+1}} \rfloor} Pmax(k, w - n\cdot w_{k+1}) + P_{k+1}[n] \\ &\geq Pmax(k, w - n' \cdot w_{k+1}) + P_{k+1}[n'] \\ &\geq Opt(k, w - n \cdot w_{k+1}) + P_{k+1}[n'] \\ &= Opt(k+1, w) \end{aligned} \tag{5}$$

Algorithm 2 presents the pseudo-code of the solution-searching method. The profit information of the i_{th} layer, obtained by solving Problem 1, is stored as a list member of the $L[i]$, where L is an array of the layer structure. N represents the number of layers. Since the length of an input vector is fixed to H_{OU}, the buffer space required by a pattern result exhibits a linear relationship with the width of the weight matrix (W_M) pertaining to the respective layer. Therefore, $L[i].w$ is determined by the configuration of the model. As there is no profit gain when all patterns of all OU inputs are buffered, each layer has an attribute $maxQ$ to represent the maximum number of this "commodity". We define m as a 2-D table, where $m[i,w]$ records the maximum profit obtained when the total capacity is w and only the first i commodities are available. Solving Problem 2 is equivalent to obtaining table m from bottom to top. The first row of m is initialized to all zero (Line 3), as no profit is produced in the absence of available commodities. During the iteration, for each (i,w) pair, the algorithm tries all possible quantities for the i_{th} commodity, following the strategy in (3) (Line 5–11). The global optimal profit is ultimately stored in $m[N,W]$. The top-down retrieval

starts from the tail of m. For the i_{th} commodity, once a quantity k is a feasible intermediate node to the optimal solution (Line 17), it is recorded in R (Line 18). Then, the iteration jumps to the next layer, L_{i-1}, until $R[0]$ is filled.

The solution-searching method in Algorithm 2 traverses at most $\frac{1}{2}NW(W+1)$ possible allocation strategies to obtain the maximal computation saving, where N is the number of layers, and W is the total capacity. Therefore, the time complexity of Algorithm 2 is $O(NW^2)$. Since Algorithm 1 has a time complexity of $O(NW)$, the overall time complexity for solving the whole buffer allocation problem is $O(NW^2)$. Please note that $L[i].w$ and W are normalized in order to reduce the search space. Moreover, the allocation strategies for neural layers are produced beforehand and remain static during the inference stage. In other words, the one-time learned allocation strategy can be applied to many-time inference. The time overhead of learning the allocation strategy is spread over many inference tasks.

Algorithm 2: Inter-Layer Buffer Allocation

Input : Capacity **W**, #Layers **N**, Layers **L** with: Weight **w**, Max quantity **maxQ** and Profit **p**
Output: Achievable max profit **maxP**, solution **R**

1 /* Part 2: solve P2 */
2 Init **m, R**: 2D and 1D array with size $[N, W]$ and N
3 Set all m[0,w] to 0
4 **for** $i = 1$ *to* N **do**
5 **for** $w = 1$ *to* W **do**
6 m[i,w] $\leftarrow -\infty$
7 Q $\leftarrow min(L[i-1].maxQ, \lfloor \frac{w}{L[i-1].w} \rfloor)$
8 **for** $k = 0$ *to* Q **do**
9 $q \leftarrow m[i-1, w-kL[i-1].w] + L[i-1].p[k]$
10 **if** $q > m[i,w]$ **then**
11 $m[i,w] \leftarrow q$
12 /* Part 3: get one allocation */
13 $i \leftarrow N-1; w \leftarrow W$
14 **while** $i \geq 0$ **do**
15 $Q \leftarrow min(L[i].maxQ, \lfloor \frac{w}{L[i].w} \rfloor)$
16 **for** $n = 0$ *to* Q **do**
17 **if** $m[i+1][w] - L[i].p[n] = m[i][w - n \cdot L[i].w]$ **then**
18 $R[i+1] \leftarrow n$ /* Record quantity */
19 $i \leftarrow i-1; w \leftarrow w - n \cdot L[i].w$; break
20 **return** m[N][W], R

3.3. Compute-reuse for weight patterns

ICR centers on the MVMs between OU-inputs and OU-rows, while WCR focuses on the sub-MVMs between OU-inputs and OUs. In the offline stage, solving Problem 2 determines the input patterns to be buffered for all OU-rows of all neural layers. ICR passes them to the WCR part to perform computations only once and stores the results in the pattern buffer. In the formal computation stage, the buffered input vectors are forwarded to the pattern buffer to fetch the pattern results, and the others are passed to the WCR part.

Fig. 5 shows that when WCR receives an input vector, it first computes MVMs with weight patterns only and then constructs the complete MVM results with pattern results. WCR contains Pattern matrices, Pattern Buffers, and Column Index Tables to support the process. Instead of storing the whole weight matrix for a neural layer, WCR stores a set of identical pattern matrices where all weight pattern vectors are included. Pattern Buffer stores the results of dot-products of the input vector and all weight pattern vectors. Because of the use of single-bit ReRAM cells and the OU configuration, Pattern Buffer size is linear to the $2^{H_{OU}}$ possible states. Column Index Table stores the pattern indexes of all OU-columns. An example of transforming the original MVM into WCR is shown in Fig. 7. The conventional bit-split mapping approach requires four SLC XBAs of size 4×8 to store the 4-bit

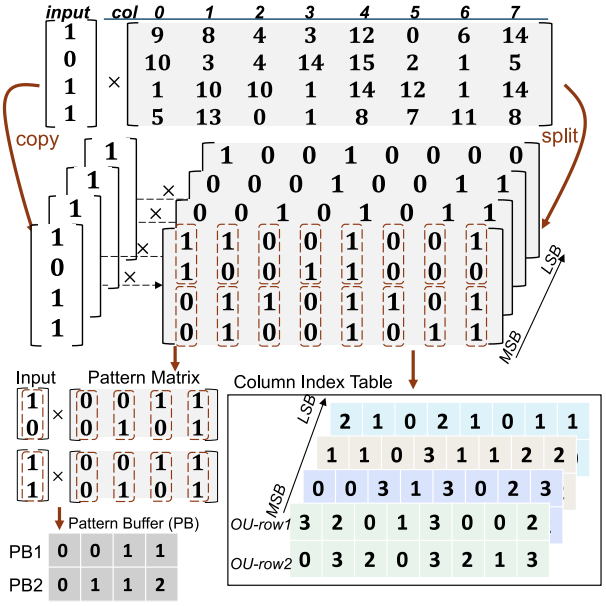


Fig. 7. Convert a regular MVM operation to a WCR process, which is supported by pattern matrices, pattern buffers, and column index tables.

4×8 weight matrix. Since an OU column of size 2 has four possible states, namely (0,0), (0,1), (1,0), and (1,1), the pattern matrix M_{patt} is of size 2×4. Given that the input vector can be split into 2 OU-inputs and the bit precision is 4, 2 Pattern Matrices, 2 Pattern Buffers, and 4 Column Index Tables are needed. Note that the two identical pattern matrices receive different input vectors at the same time and are, therefore, mapped to different XBAs to improve parallelism.

Fig. 8 illustrates the weight compute-read procedure. The pattern array stores the patt-MVM results, and the relative pattern arrays' address port sequentially receives the target index from the column index tables, which then retrieves the relevant vector–vector multiplication results and writes them to the accumulator to produce the final result. In this example, the bit-matrix M has dimensions of 4×8, the input vector V is of size 4, and OU has dimensions of 2×2. OU1 and OU2 hold all the OU-patterns, i.e., [(0,0), (0,1)], [(1,0), (1,1)]. The input vector [1,0,1,1] is split into OU-rows (1,0) and (1,1) and copied into 4-cycle input vectors, i.e., [(1,0), (1,0), (1,1), (1,1)]. The MVMs of $M_{pattern}$ with the input (1,0) and (1,1) are completed in the first and last 2 cycles, respectively. The obtained pattern-MVM results are stored in PR1 and PR2. For instance, PR1 stores the pattern results (patt0:0, patt1:0, patt2:1, patt3:1), and the index vector of the first OU-row is [3,2,0,1,3,0,0,2]. The readout vector, [1,1,0,0,1,0,0,1], is the MVM result of M[0:1,:] with input (1,0). Similarly, the MVM of M[2:3,:] with input (1,1) yields [0,2,1,0,2,1,1,2]. The accumulation vector, [1,3,1,0,3,1,1,3], represents the full MVM result of M with V.

4. CRPIM architecture and design

This section provides a comprehensive overview of the architecture designed to support the sharing process, which efficiently prunes repetitive weights and inputs.

4.1. Architecture overview

Fig. 9 illustrates the architecture of the proposed CRPIM accelerator, which consists of multiple Processing Engines (PE) and a global controller issuing control signals to all PEs. Each PE is further divided into three parts: Input Compute-Reuse (ICR), Weight Compute-Reuse (WCR), and peripherals. Within a PE, a dispatcher divides the stored

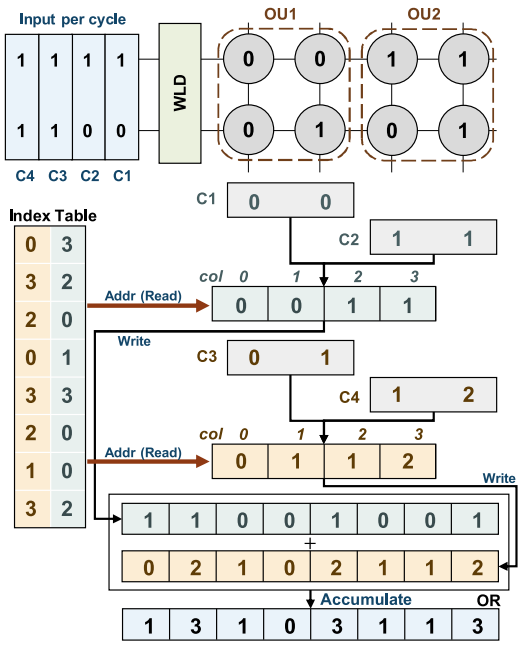


Fig. 8. Example of the process of reading the pattern results.

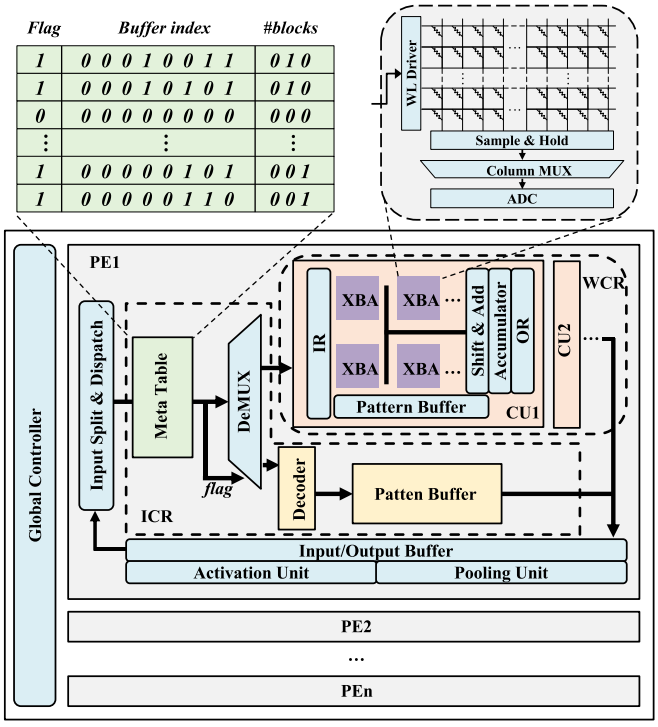


Fig. 9. CRPIM architecture. Each PE contains: Meta Table and Pattern Buffer for ICR, CUs for WCR, and peripherals.

input feature map (FM) in the input buffer and routes input vectors to the ICR's meta table. If input vectors are registered, the Matrix–Vector Multiplication (MVM) results are fetched from the pattern buffer; if not, they are derived from the WCR component. The results are further processed by pooling unit and activation unit and the final result will be stored in output buffer.

The ICR part comprises a meta table, demux, decoder and pattern buffer. The meta table records information of the $2^{H_{OU}}$ input patterns. Each entry in the meta table contains a flag bit to identify whether the corresponding computation result is buffered or not, an index to point out the starting position in pattern buffer and the number of blocks occupied by the corresponding output. The demux receives the flag and determines where to obtain the result. For those buffered, the decoder forwards the decoded starting address and the size to read to the pattern buffer. Then the read result is forwarded to output buffer. The WCR part is composed of multiple Compute Units (CUs), each of which contains input and output registers (IR and OR), shift & add (S&A), accumulators and pattern buffers. Input registers receive the assigned input vectors, a group of crossbar arrays (XBAs), The shifted and summed MVM results are accumulated by accumulators and then stored in OR.

A WCR-PE is a group of WCR-CUs, each containing an input buffer to store the assigned input vectors, n crossbar arrays to perform bit-level MVMs, and a pattern buffer to collect the pattern results. In the crossbar arrays, 1-bit weight patterns are stored repetitively so that multiple input vectors can compute their pattern results in parallel. The pattern results are then copied to the pattern registers in WCR-PEs, and the sequence of weight indexes is forwarded to the Pattern Reader (PR) to read out the buffered pattern results. Note that since a bit-level MVM is one partial result of a complete MVM operation of an M-bit matrix with an N-bit input vector, S&As and accumulators are required in WCR-PEs to construct the full result.

5. Evaluation

5.1. Experimental setup

Based on MNSIM [29], we implement a prototype of CRPIM and evaluate its performance. The latency delay of the on-chip SRAM buffer, i.e., the meta table, input register (IR), output register (OR), and pattern buffer are modeled using CACTI [30] at the 32-nm process. The hardware configuration is set referring to the parameters of ISAAC and SRE, except that ReRAM crossbars adopt Single-Level-Cell devices. The hardware configuration is set to meet the requirements of the benchmark DNN models. Table 2 summarizes the hardware configuration of the architecture. The hardware configuration is as follows: There are 512 PEs per chip, each PE has 16 CUs, and each CU groups 8 SLC crossbar arrays. The crossbar arrays in CUs are of size 128×128, where each cell stores 1 bit. MVMs proceed at the OU granularity of size (8×8). Meta tables and pattern buffers in ICR are assembled in the on-chip buffer of PE, and the pattern buffer in WCR is 8 KB for each CU.

Task and Models: The goal of CRPIM is to reduce redundant computations and decrease ReRAM device overhead in DNN tasks. We choose the Imagenet classification task as the benchmark task as Imagenet is a large-scale and diverse dataset [31], involving a number of computations and intermediate inputs. We select four representative DNN models, including Alexnet [26], GoogleNet [27], ResNet-50 [10], and VGG-16 [28], to evaluate the performance. Model weights are quantized with 8-bit using a post-training quantization algorithm [32], which quantizes pre-trained models without retraining while maintaining the inference accuracy.

Baselines and Evaluation Metrics: In this study, we compare the performance of CRPIM against two state-of-the-art weight/input pruning approaches for ReRAM-based DNN accelerators. For weight pruning, we select the pattern-based pruning technique PattPIM [21]. We choose SRE [19], and RePIM [20] as the baseline methods for joint exploitation of activation and weight pruning. We vary the dataset size in the learning stage and the buffer size in the running stage to explore the expandability of CRPIM. We consider four evaluation metrics: time speedup, crossbar pruning ratio, buffer storage, and energy consumption. To maintain consistency in the evaluation and minimize the impact of non-ideal limitations of ReRAM devices on computation accuracy, we adopt an 8×8 OU design for all the evaluated methods. We compare the results of the two buffer allocation approaches with RePIM and SRE in terms of speedup and hardware overhead, with OU sizes set to 8×8.

Table 2
Hardware Configuration.

PE configuration (1 GHz, 32 nm process, 512 PE in total)		
Component	Spec	Power
Sigmoid	Number: 2	0.52 mW
Pooling	Number: 1	0.4 mW
On-chip buffer	Size: 256 KB; banks: 8 bus width: 512 bits	197.7 mW
CU configuration (16 CUs per PE)		
Memristor Array	Number: 8; size: 128 × 128 bits-per-cell: 1 bit; OU-size: 8 × 8	2.4 mW
DAC	Number: 128 × 128; Resolution: 1 bit	7.9 mW
ADC	Number: 16; Resolution: 4 bit	11.2 mW
S+A	Number: 4	0.2 mW
S+H	8 × 128	10 uW
OR	size: 256B	0.23 mW
IR	size: 2 KB	1.24 mW
Pattern buffer	size: 8 KB	4.51 mW

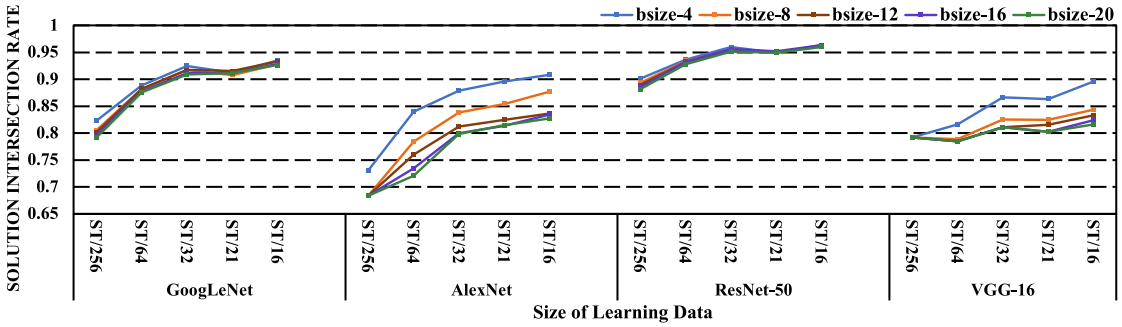


Fig. 10. Intersection ratio of solutions for different buffer sizes and learning dataset sizes.

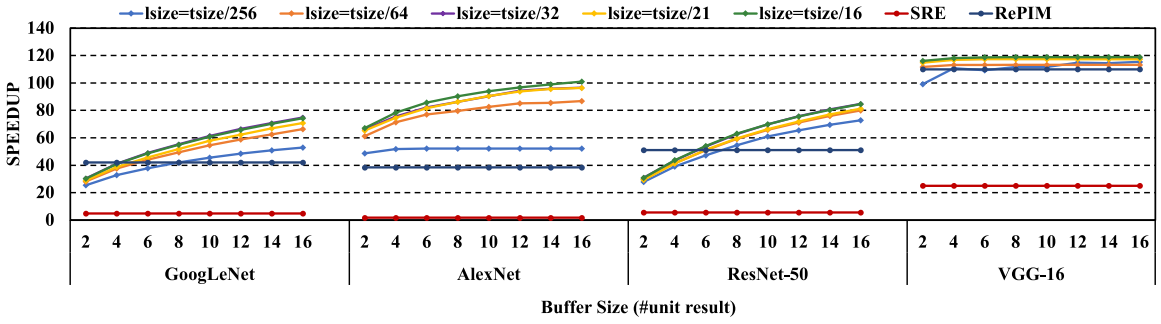


Fig. 11. Performance speedup of different approaches.

5.2. Experimental results

(1) Accuracy and Speedup: Our repetition pruning approaches buffer frequently repeated computations for fast reading and hence cause no loss of model information. The accuracy drop is consistent with the adopted post-training quantization technique [32]. It achieves around 0.5% loss of top-1 accuracy when quantizing the four models with 8 bits per weight. In the time consumption tests, we observe the performance of various models under different learning data sizes and buffer sizes. Fig. 11 presents the execution time of the proposed CRPIM and baseline methods using the same benchmark group. Note that the buffer size in Fig. 11 denotes the average number of unit pattern results that can be stored in the buffer per OU-row. It can be seen that all four models exhibit sensitivity to data size during the learning stage. As depicted in Fig. 10, a larger learning space results in more shared solutions. GoogLeNet and ResNet-50 demonstrate insensitivity to changes in buffer size (bsize), particularly when bsize exceeds 4, due to limitations in buffer hardware, such as access latency. Overall, with $lsize = \frac{tsize}{16}$, CRPIM can achieve at most 2.63× speedup compared with RePIM. The execution time reduction mainly comes from two benefits (1) Replacing massive computations of repetitive inputs with reading the pattern buffer greatly reduces the number of required computational cycles. (2) Input repetition reuse exhibits orthogonality to weight repetition reuse with respect to time-saving. ICR exclusively targets the concentration of input activations, thereby not exerting any influence on the weights. This distinctive focus makes ICR and WCR independent and complementary strategies to each other in the realm of time efficiency.

(2) ReRAM reduction: Fig. 12 presents the demand for ReRAM crossbar array from CRPIM and SRE, which is normalized to the original weights quantized to 8-bit precision. When compared to the original weights and SRE, CRPIM produces an average ReRAM compression rate of 11.9% and 37.9%, respectively. This is achieved because WCR part only performs the pattern computations in the WCR-PEs, and the size of a pattern computation does not vary with the matrix shape or input size. The difference in ReRAM cell reduction between these models falls

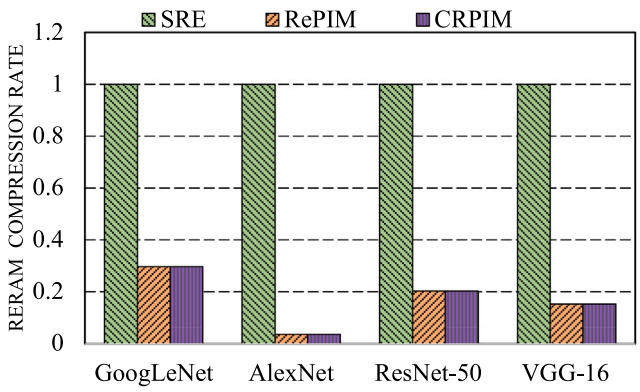


Fig. 12. ReRAM compression rate of CRPIM and SRE normalized to baseline.

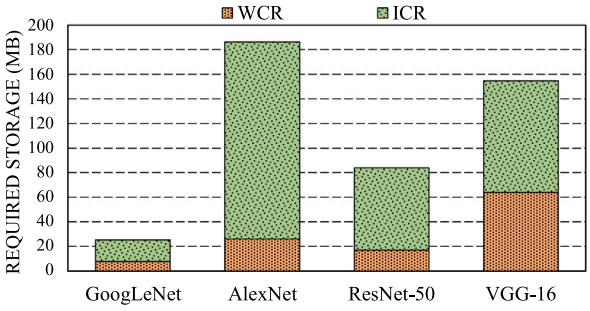


Fig. 13. Storage overhead for WCR and ICR on different models.

on the shapes of their flattened weight matrices. A model with many tall weight matrices has a high demand for pattern computation units because the computation result for a weight pattern can only be shared with the OU-columns in the same OU-rows. For example, a 64×1024 weight matrix and a 1024×64 one contain the same amount of weight values, but the former one yields a higher compression ratio on ReRAM cells, i.e., each weight pattern can be used by more weight columns.

(3) Buffer and energy overhead: The primary storage overhead of CRPIM predominantly consists of the pattern buffer and output buffer, as the size of the index table is comparatively negligible. Fig. 13 illustrates the storage overhead of the four models and their respective compositions with a fixed buffer size of 16. The space requirement of the ICR pattern buffer is contingent upon the number of pattern results to be buffered and the size of an individual result. For each pattern result, the storage is linearly proportional to the size of the corresponding weight matrix. Consequently, the ICR pattern buffer's size grows with the model's size, as the output buffer does. As the speedup of AlexNet exhibits a slower growth trend with increasing bsize, and its weight matrix occupies a substantial space, the required buffer storage is the largest among the four models. Nevertheless, as observed in Fig. 11, AlexNet is sensitive to the size of the learning data. Therefore, one can increase the lsize to achieve a trade-off with a smaller buffer size. Additionally, it is given that a buffer size exceeding 4 units of pattern result does not contribute to the performance improvement of VGG-16. In summary, scenarios with restricted buffer resources can be compensated for by increasing the learning data size during strategy formation to suspend performance degradation.

The energy consumption (Fig. 14) of our scheme outperforms the conventional computation but is behind the SRE on VGG-16. This is mainly because, though many parts of repeated computations are avoided successfully, buffer reading is still energy-consuming. However, the above overhead is deemed acceptable when contrasted with the advantages realized through speed optimization. The trade-off, therefore, skews favorably towards enhanced processing speed, as the modest increase in storage and energy demand is offset by substantial acceleration benefits.

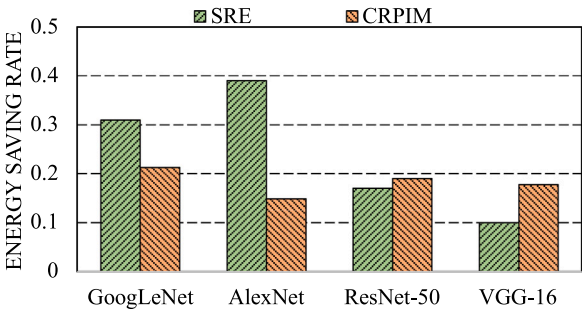


Fig. 14. Energy saving on different models.

6. Related work

This paper introduces a compute-reuse scheme for ReRAM-based DNN inference to improve the role of pattern buffers in sharing pattern results. To the best of our knowledge, this is the first work that explores the buffer allocation strategy intra- and inter-layers. In this section, we discuss recent works related to pruning computations by utilizing sparsity and repetition in inputs and weights.

6.1. Exploiting sparsity in ReRAM-based DNN

Leveraging the sparsity to prune insignificant computations and to reduce the storage overhead has been thoroughly studied in the development of ReRAM-based DNN Inference Engines [19,22,23,33–36]. For example, Yang et al. [19] designed SRE that exploits weight and input sparsity in the pruning strategy. SRE leverages row-wise weight compression combined with dynamic OU formation in order to create more all-zero OU-rows in weight matrices and OU-columns in inputs. Shin et al. [34] propose a filter reordering scheme tailored for unstructured pruning in highly sparse DNNs. The scheme optimizes the OU utilization to reduce hardware overhead and introduces a recovery scheme restoring uncompressed zero weights to balance model accuracy with resource utilization. Liu et al. [33] proposed a ReRAM-based architecture named ERA-BS, focusing on bit-level sparsity in weights and activation. Particularly, ERA-BS includes a bit-flip scheme for weights, which sacrifices little accuracy for fine-grained sparsity. PQ-PIM [36] and APQ [35] propose fusion frameworks for pruning and quantization. They automatically determine the quantization bit-width and prune the unimportant weights but influence the model performance in an acceptable range. CRPIM shares similarities with ERA-BS and SRE in recognizing and pruning invalid columns and rows in weights and inputs. However, beyond adopting sparsity pruning, CRPIM places more emphasis on utilizing repetitions for pruning rather than aiming to recognize more insignificant weights and inputs.

6.2. Exploiting repetition in ReRAM-based DNN

Accelerating DNNs by exploiting repetition has drawn wide attention [25,37–43]. Moreover, utilizing repetition to improve the performance of ReRAM-based DNN engines has also been studied [20,21,44,45]. Tsai et al. [20] proposed RePIM to boost performance by capitalizing on the repetitive nature of inputs and weights at the OU grain. RePIM introduces indexing methodologies for sharing repetitive weights and inputs, where unique patterns and sharing indexes are recorded. Chen et al. [44] introduce H-RIS aiming to achieve an optimal balance between energy consumption and accuracy for repetitive input sharing schemes. They conduct energy evaluation on buffers used in input sharing and identify an energy-efficient sharing ratio that

minimizes energy costs while ensuring computational accuracy. Shen et al. [45] propose a pruning method named PRAP-PIM, tailored for repetitive weight patterns. It introduces reusing similar weight patterns that satisfy a linear relationship instead of identical matching. H-RIS and PRAP-PIM both focus on the repetition of either weights or inputs, and both aim to achieve a higher pruning rate with minimal sacrifice to accuracy. Similar to RePIM, CRPIM encompasses pruning based on the repetition of both weights and inputs, and it does not harm model accuracy.

7. Conclusion

In this paper, we first conduct pre-experiments to show the similarity in input patterns among different datasets of the DNN computation on ReRAM-based PIM machines. Then, we propose a compute-reuse scheme called CRPIM to replace massive computations on repeated input/weight patterns with buffer reading. In CRPIM, we propose the intra-layer and inter-layer buffer allocation problem, and the latter is proved to be equivalent to a variant of the Knapsack problem. Moreover, we explore the duplicate input/weight patterns in OU-based architecture and propose an efficient compute-reuse scheme. Our evaluation shows that CRPIM achieves up to 2.63× speedups and an average of 37.9% ReRAM compression rate over the state-of-the-art practical ReRAM-based accelerator while keeping the storage and energy overhead reasonable.

CRediT authorship contribution statement

Shihao Hong: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Yeh-Ching Chung:** Supervision, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Yeh-Ching Chung reports financial support was provided by Huawei Technologies Co Ltd. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the authors used ChatGPT in order to polish the writing. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Acknowledgments

The work of this paper is supported by Huawei Technologies Co., Ltd, China under contract No. TC20220913039.

References

[1] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J.J. Yang, H. Qian, Fully hardware-implemented memristor convolutional neural network, Nature 577 (7792) (2020) 641–646.

[2] Y. Huang, L. Zheng, P. Yao, J. Zhao, X. Liao, H. Jin, J. Xue, A heterogeneous PIM hardware-software co-design for energy-efficient graph processing, in: 2020 IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE, 2020, pp. 684–695.

[3] S.A. Ghasemi, B. Jahannia, H. Farbeh, GraphA: An efficient ReRAM-based architecture to accelerate large scale graph processing, J. Syst. Archit. 133 (2022) 102755.

[4] Y. Zhong, J. Tang, X. Li, X. Liang, Z. Liu, Y. Li, Y. Xi, P. Yao, Z. Hao, B. Gao, et al., A memristor-based analogue reservoir computing system for real-time and power-efficient signal processing, Nat. Electron. 5 (10) (2022) 672–681.

[5] W. Huangfu, S. Li, X. Hu, Y. Xie, RADAR: A 3D-ReRAM based DNA alignment accelerator architecture, in: Proceedings of the 55th Annual Design Automation Conference, 2018, pp. 1–6.

[6] W. Xu, S. Gupta, N. Moshiri, T. Rosing, RAPIDx: High-performance ReRAM processing in-memory accelerator for sequence alignment, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2023).

[7] B. Hanindhito, R. Li, D. Gourounas, A. Fathi, K. Govil, D. Trenev, A. Gerstlauer, L. John, Wave-PIM: Accelerating wave simulation using processing-in-memory, in: 50th International Conference on Parallel Processing, 2021, pp. 1–11.

[8] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, Y. Xie, Prime: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory, ACM SIGARCH Comput. Archit. News 44 (3) (2016) 27–39.

[9] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J.P. Strachan, M. Hu, R.S. Williams, V. Srikumar, ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars, ACM SIGARCH Comput. Archit. News 44 (3) (2016) 14–26.

[10] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[11] Y.-W. Kang, C.-F. Wu, Y.-H. Chang, T.-W. Kuo, S.-Y. Ho, On minimizing analog variation errors to resolve the scalability issue of ReRAM-based crossbar accelerators, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (11) (2020) 3856–3867.

[12] H. Shin, M. Kang, L.-S. Kim, Fault-free: A fault-resilient deep neural network accelerator based on realistic ReRAM devices, in: 2021 58th ACM/IEEE Design Automation Conference, DAC, IEEE, 2021, pp. 1039–1044.

[13] M.V. Beigi, G. Memik, Thermal-aware optimizations of ReRAM-based neuromorphic computing systems, in: Proceedings of the 55th Annual Design Automation Conference, 2018, pp. 1–6.

[14] M.-Y. Lin, H.-Y. Cheng, W.-T. Lin, T.-H. Yang, I.-C. Tseng, C.-L. Yang, H.-W. Hu, H.-S. Chang, H.-P. Li, M.-F. Chang, DL-RSIM: A simulation framework to enable reliable ReRAM-based accelerators for deep learning, in: 2018 IEEE/ACM International Conference on Computer-Aided Design, ICCAD, ACM, 2018, pp. 1–8.

[15] Z. Zhu, H. Sun, Y. Lin, G. Dai, L. Xia, S. Han, Y. Wang, H. Yang, A configurable multi-precision CNN computing framework based on single bit RRAM, in: Proceedings of the 56th Annual Design Automation Conference 2019, 2019, pp. 1–6.

[16] S. Yang, W. Chen, X. Zhang, S. He, Y. Yin, X.-H. Sun, AUTO-PRUNE: Automated DNN pruning and mapping for ReRAM-based accelerator, in: Proceedings of the ACM International Conference on Supercomputing, 2021, pp. 304–315.

[17] H. Ji, L. Song, L. Jiang, H. Li, 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.

[18] P. Wang, Y. Ji, C. Hong, Y. Lyu, D. Wang, Y. Xie, SNrram: An efficient sparse neural network computation architecture based on resistive random-access memory, in: Proceedings of the 55th Annual Design Automation Conference, 2018, pp. 1–6.

[19] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, 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.

[20] C.-Y. Tsai, C.-F. Nien, T.-C. Yu, H.-Y. Yeh, H.-Y. Cheng, RePIM: Joint exploitation of activation and weight repetitions for in-ReRAM DNN acceleration, in: 2021 58th ACM/IEEE Design Automation Conference, DAC, IEEE, 2021, pp. 589–594.

[21] Y. Zhang, Z. Jia, H. Du, R. Xue, Z. Shen, Z. Shao, A practical highly paralleled ReRAM-based dnn accelerator by reusing weight pattern repetitions, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 41 (4) (2021) 922–935.

[22] F. Liu, W. Zhao, Z. He, Z. Wang, Y. Zhao, Y. Chen, L. Jiang, Bit-transformer: Transforming bit-level sparsity into higher preformance in ReRAM-based accelerator, in: 2021 IEEE/ACM International Conference on Computer Aided Design, ICCAD, IEEE, 2021, pp. 1–9.

[23] F. Liu, Z. Wang, Y. Chen, Z. He, T. Yang, X. Liang, L. Jiang, SoBS-X: Squeeze-out bit sparsity for ReRAM-crossbar-based neural network accelerator, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 42 (1) (2022) 204–217.

[24] C. Chu, Y. Wang, Y. Zhao, X. Ma, S. Ye, Y. Hong, X. Liang, Y. Han, L. Jiang, PIM-prune: Fine-grain DCNN pruning for crossbar-based process-in-memory architecture, in: 2020 57th ACM/IEEE Design Automation Conference, DAC, IEEE, 2020, pp. 1–6.

[25] Y.-C. Lo, R.-S. Liu, Bit-serial cache: Exploiting input bit vector repetition to accelerate bit-serial inference, in: 2023 60th ACM/IEEE Design Automation Conference, DAC, IEEE, 2023, pp. 1–6.
[26] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Commun. ACM 60 (6) (2017) 84–90.
[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.
[28] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
[29] L. Xia, B. Li, T. Tang, P. Gu, P.-Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, H. Yang, MNSIM: Simulation platform for memristor-based neuromorphic computing system, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 37 (5) (2017) 1009–1022.
[30] R. Balasubramonian, A.B. Kahng, N. Muralimanohar, A. Shafiee, V. Srinivas, CACTI 7: New tools for interconnect exploration in innovative off-chip memories, ACM Trans. Archit. Code Optim. (TACO) 14 (2) (2017) 1–25.
[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, 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.
[32] J. Zhang, Y. Zhou, R. Saab, Post-training quantization for neural networks with provable guarantees, SIAM J. Math. Data Sci. 5 (2) (2023) 373–399.
[33] F. Liu, W. Zhao, Z. Wang, Y. Chen, X. Liang, L. Jiang, Era-bs: Boosting the efficiency of ReRAM-based pim accelerator with fine-grained bit-level sparsity, IEEE Trans. Comput. (2023).
[34] H. Shin, R. Park, S.Y. Lee, Y. Park, H. Lee, J.W. Lee, Effective zero compression on ReRAM-based sparse DNN accelerators, in: Proceedings of the 59th ACM/IEEE Design Automation Conference, 2022, pp. 949–954.
[35] S. Yang, S. He, H. Duan, W. Chen, X. Zhang, T. Wu, Y. Yin, APQ: Automated DNN pruning and quantization for ReRAM-based accelerators, IEEE Trans. Parallel Distrib. Syst. (2023).
[36] Y. Zhang, X. Wang, X. Jiang, Y. Yang, Z. Shen, Z. Jia, PQ-PIM: A pruning–quantization joint optimization framework for ReRAM-based processing-in-memory DNN accelerator, J. Syst. Archit. 127 (2022) 102531.
[37] M. Riera, J.-M. Arnau, A. González, Computation reuse in DNNs by exploiting input similarity, in: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2018, pp. 57–68.
[38] E. Hanson, S. Li, H. Li, Y. Chen, Cascading structured pruning: Enabling high data reuse for sparse DNN accelerators, in: Proceedings of the 49th Annual International Symposium on Computer Architecture, 2022, pp. 522–535.
[39] M. Riera, J.M. Arnau, A. González, CREW: Computation reuse and efficient weight storage for hardware-accelerated MLPs and RNNs, J. Syst. Archit. 129 (2022) 102604.
[40] U. De Alwis, M. Alioto, Architecture for 3D convolutional neural networks based on temporal similarity removal, in: 2022 29th IEEE International Conference on Electronics, Circuits and Systems, ICECS, IEEE, 2022, pp. 1–4.
[41] V. Janfaza, K. Weston, M. Razavi, S. Mandal, F. Mahmud, A. Hilty, A. Muzahid, Mercury: Accelerating dnn training by exploiting input similarity, in: 2023 IEEE International Symposium on High-Performance Computer Architecture, HPCA, IEEE, 2023, pp. 638–650.
[42] N.M. Cicek, X. Shen, O. Ozturk, Energy efficient boosting of GEMM accelerators for DNN via reuse, ACM Trans. Des. Autom. Electron. Syst. (TODAES) 27 (5) (2022) 1–26.
[43] A. Ghanbari, M. Modarressi, Energy-efficient acceleration of convolutional neural networks using computation reuse, J. Syst. Archit. 126 (2022) 102490.
[44] Y.-C. Chen, C.-Y. Chang, A.-Y. Wu, H-RIS: Hybrid computing-in-memory architecture exploring repetitive input sharing, in: 2023 IEEE International Symposium on Circuits and Systems, ISCAS, IEEE, 2023, pp. 1–5.
[45] Z. Shen, J. Wu, X. Jiang, Y. Zhang, L. Ju, Z. Jia, PRAP-PIM: A weight pattern reusing aware pruning method for ReRAM-based PIM DNN accelerators, High-Confidence Comput. 3 (2) (2023) 100123.

Shihao Hong received his B.E. in computer science and technology from Hefei University of Technology, China, in 2016. He is currently pursuing the Ph.D. degree in Computer and Information Engineering at The Chinese University of Hong Kong, Shenzhen, China. His research interests include intelligent computing and processing-in-memory.

Yeh-Ching Chung received his Ph.D. degree in computer and information science from Syracuse University in 1992. From 2002 to 2016, he was a full professor in the Department of Computer Science at National Tsing Hua University. From 2016 to 2018, he was a deputy director of the Laboratory of Cloud Computing and Disaster Recovery Technology at the Research Institute of Tsinghua University in Shenzhen. Currently, he is a full professor in the School of Data Science at The Chinese University of Hong Kong, Shenzhen. His research interests include parallel and distributed processing, cloud computing, big data, and embedded systems.