INCA: Input-stationary Dataflow at Outside-the-box Thinking about Deep Learning Accelerators

Bokyung Kim, Shiyu Li, and Hai Li
Duke University, Durham, NC, USA
{bokyung.kim828, shiyu.li, hai.li}@duke.edu

***Abstract*—This paper first presents *an input-stationary (IS) implemented crossbar accelerator (INCA)*, supporting inference and training for deep neural networks (DNNs). Processing-in-memory (PIM) accelerators for DNNs have been actively researched, specifically, with resistive random-access memory (RRAM), due to RRAM's computing and memorizing capabilities and device merits. To the best of our knowledge, all previous PIM accelerators have saved weights into RRAMs and inputs (activations) into conventional memories—it naturally forms weight-stationary (WS) dataflow. WS has generally been considered the most optimized choice for high parallelism and data reuse. However, WS-based PIM accelerators show fundamental limitations: first, remaining high dependency on DRAM and buffers for fetching and saving inputs (activations); second, a remarkable number of extra RRAMs for transposed weights and additional computational intermediates in training; third, coarse-grained arrays demanding high-bit analog-to-digital converters (ADCs) and introducing poor utilization in depthwise and pointwise convolution; last, degraded accuracy due to its sensitivity to weights which are affected by RRAM's nonideality. On the other hand, we observe that IS dataflow, where RRAMs retain inputs (activations), can effectively address the limitations of WS, because of low dependency by only loading weights, no need for extra RRAMs, feasibility of fine-grained accelerator design, and less impact of input (activation) variance on accuracy. But IS dataflow is hardly achievable by the existing crossbar structure because it is difficult to implement kernel sliding and preserve the high parallelism. To support kernel movement, we constitute a cell structure with two-transistor-one-RRAM (2T1R). Based on the 2T1R cell, we design a novel three-dimensional (3D) architecture for high parallelism in batch training. Our experiment results prove the potential of INCA. Compared to the WS accelerator, INCA achieves up to $20.6\times$ and $260\times$ energy efficiency improvement in inference and training, respectively; $4.8\times$ (inference) and $18.6\times$ (training) speedup as well. While accuracy in WS drops to 15% in our high-noise simulation, INCA presents an even more robust result as 86% accuracy.**

I. Introduction

Deep learning has become widely popular across various applications. Advanced algorithms with endless layers, huge parameters, and complicated computations evoke the need for a new paradigm in hardware for two reasons. First, researchers have reached a consensus that it is hard to expect further development of conventional hardware to match the advancement of algorithms because conventional CMOS technology is approaching the end of Moore's Law [19]. Secondly, the fundamental structure of existing hardware is not appropriate for the learning algorithms. Whereas the hardware structure separates the computing unit and the memory, learning process requires intensively frequent movement between two units.

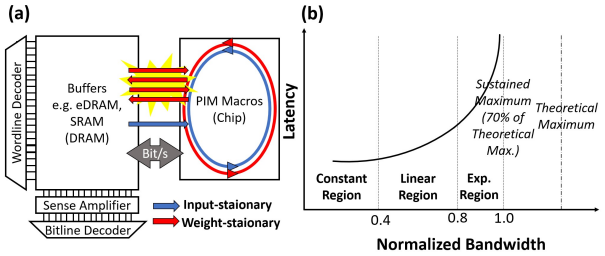

Fig. 1. (a) Interaction between conventional memories and PIM accelerators, limited by bus-width (or bandwidth). While WS needs fetching and storing data, IS requires only fetching of data. (b) Limitation in increasing bandwidth of DRAM: different latency increase rate.

The data movement between the two units creates a bottleneck phenomenon, called the von Neumann bottleneck, limiting the maximum throughput of the overall system. Currently, researchers turn to processing- or computing-in-memory (PIM, CIM) designs by performing the *in situ* computation to alleviate the high expense caused by data movement.

Specifically, PIM hardware with resistive random-access memory (RRAM, ReRAM, a.k.a. memristor) has been explored to build accelerators for deep learning models in numerous works [6], [10], [20], [42], [48]. These works exploit advantages of RRAM devices (i.e. low energy, fast switching, and small area [1], [31]) to improve throughput and energy efficiency of accelerator designs. RRAM-based PIM accelerators utilize the crossbar structure for its excellence in matrix-vector operations and high parallelism. Prior designs with the RRAM crossbar typically store weights on RRAMs while fetching input and intermediate values from external memories or buffers. Because the weight values stay in the PIM array, it naturally forms ***weight-stationary (WS)*** dataflow. The WS-based PIM removes the dependency of weight values on conventional memories; that is, DRAM and buffers only need to treat input and intermediate values. PIM hardware has mitigated the bottleneck, expediting learning process.

Deep learning accelerators based on WS PIM require loading inputs from and saving generated outputs to memories for a layer, as shown in Figure 1a. Those two movements in on-chip or off-chip memories are still the most costly in terms of latency and energy. Particularly, off-chip design accelerators cannot be the most optimized design because of a bandwidth limitation in DRAM. According to [34], [49], latency increases exponentially in the region beyond 80% of the maximum sustained bandwidth, as shown in Figure 1b. Therefore, a majority of PIM accelerators have relied on

978-1-6654-7652-2/23/$31.00 ©2023 IEEE

on-chip designs using buffers, such as eDRAM or SRAM. However, we notice that WS accelerators still experience high impacts of buffers; in other words, the reading/writing energy of buffers could dominate the overall consumption of WS. Following our observation, WS designs fundamentally demand frequent access to buffers, because generated outputs with sliding kernels should be saved in a timely manner for high performance and should be recalled in the next layer shortly afterward. The limited bus-width aggravates the efficiency in the use of buffers.

Taking a step backward from WS, we observe that ***input-stationary (IS)*** dataflow—*inputs stay in the PIM unit*—brings advantages given the inefficient use of buffers. Unlike WS, IS demands only loading weight values from buffer memories during the feedforward process. Once the corresponding weights in a layer are loaded, intermediate values (activations) are directly propagated across RRAM arrays in PIM macros (Figure 1a). Hence, IS can improve the feedforward efficiency since the required number of writes to buffers can be reduced by eliminating the redundant access. In backpropagation, weight values still need to be rewritten into buffers or even DRAM; yet writing to buffers is even more occasional than the counterpart of WS accelerators. We notice that realization of IS in PIM design is highly likely to be advantageous for the training process of DNN models.

Seeing this opportunity, in this paper, we propose an IS implemented accelerator design for inference and training, named ***INCA: Input-stationary crossbar accelerator***. To the best of our knowledge, INCA is the first-of-its-kind IS PIM architecture for DNNs. Prior PIM-based designs [9], [10], [42], [48] have been built upon the WS dataflow, especially by unrolling the weight values. The WS with unrolling approach occurs for two reasons: a seemingly impossible implementation of sliding kernel windows to the entangled 2D crossbar and the use of high parallelism of the crossbar array. In INCA, two-transistor-one-RRAM (2T1R) is used for array cells to implement sliding windows on input (activation) arrays without unrolling data. We design a novel 3D RRAM crossbar architecture, which can support high parallelism and batch training by operating multiple planes at once. Our end-to-end experiment results show that INCA achieves over one or two order(s) of magnitude better energy improvement and speedup with the ImageNet dataset, compared with a two-dimensional (2D) WS baseline architecture.

IS addresses further limitations missing or overlooked in the previous WS PIM designs. Previous WS accelerators have been optimized for only the feedforward process because of the complexity of the backward computation. As a result, few RRAM-based PIMs can support the *in situ* training process [20], [48]. But even in the papers, tremendous extra RRAMs dedicated to training parameters have been overlooked, i.e., transposed weights, errors, and gradients. Also, coarse-grained RRAM arrays in WS accelerators lead to the employment of high-precision analog-to-digital converters (ADCs) in general and the low utilization in light models. Although both points immeasurably harm hardware efficiency, previous works haven't proposed appropriate designs to resolve them simultaneously; some works could address only either one between them. In addition, *in situ* training accuracy is highly affected by RRAM nonideal properties because accuracy is more susceptible to noise in weights. On the other hand, IS uses even fewer RRAMs and its feasible overwriting scheme does not require additional RRAM arrays for training. INCA enables the natural implementation of fine-grained RRAM arrays, resolving the large ADCs and light models' utilization issues. In IS dataflow, accuracy is also less affected by RRAM nonideality by saving inputs (activations) to RRAM.

We summarize the major contributions at a glance:

- We identify the fundamental limitations of WS and analyze their impact on PIM designs—1) the remaining excessive memory access; 2) the enormous use of extra RRAMs; 3) the coarse-grained RRAM arrays causing large ADCs and poor utilization; 4) the RRAM nonideality's impact on accuracy.
- We present the first implementation of IS dataflow in PIM design. This paper explores and discusses specific advantages, challenges, and methods to implement IS dataflow for the first time. Also, we provide quantified data to support our analysis.
- We propose a new PIM architecture with horizontally-stacking technology of 3D RRAM and a 2T1R cell structure. We recognize a few challenges of IS implementation in the current hardware design. By addressing the challenges, **INCA: INput-stationary Crossbar Accelerator** was successfully implemented and demonstrated in terms of its higher efficiency than prior designs.

II. Background

A. 3D RRAM

As the design technology node scales below 10 nm, vertically integrated structure (VIS) has been considered the most feasible solution to reduce the cost per bit. A success of bit cost scalable (BiCS) architecture ignites the idea to apply the 3D stacking technology to RRAM structure [41], [54]. Specifically, two representative types have been proposed: vertically (VRRAM) and horizontally stacked RRAM (HRRAM), which are illustrated in Figure 2a and 2b, respectively. In general, pillars penetrate stacked word planes, and RRAMs are sandwiched between a shared pillar and stacked planes.

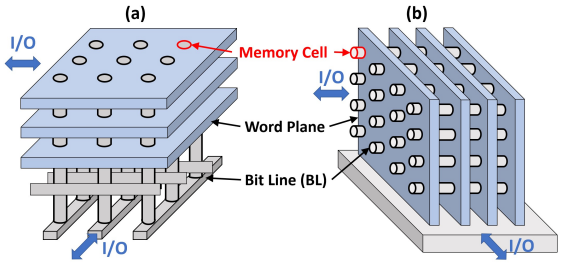

Fig. 2. Two representative 3D RRAM structures: (a) VRRAM with vertically stacked horizontal planes and (b) HRRAM with horizontally stacked vertical planes.

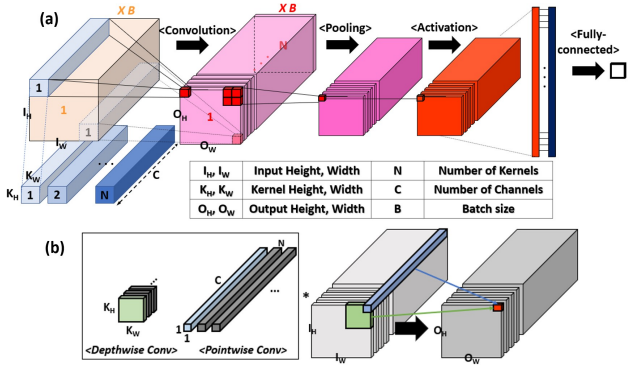

Fig. 3. (a) Feedforward process composed of convolution, pooling, activation and fully-connected layers. The inserted table summarizes parameter notation in a convolution layer. Layer's composition or size is varied according to the network architecture. (b) Depthwise and pointwise convolution.

Following the analysis in [41], VRRAM is more advantageous in 1R-based design than HRRAM to address one of the representative 1R-design issues, called sneak path current–*unintended current flows through unselected cells*. Since then, different 3D 1R VRRAMs have been fabricated for efficient in-memory logic and multiply-accumulate operation [33], [35] and analyzed [51], [68]. As such, it has been proved that 3D RRAM improves area and latency efficiency of PIM accelerators by simultaneously operating multiple word planes by one voltage supply [26], [27], [52], [66].

But the fabrication technology limits the number of vertically stacked layers in VRRAM but the plane size in HRRAM. INCA demands a design with highly stacked 3D RRAM but not a large size plane. Therefore, we chose HRRAM as a foundation of INCA architecture. It is noteworthy that our design releases the concern of sneak path current by employing transistors, which could play the role of a switch. Detailed processes of fabrication for ours or advanced 3D structures are explained in [3], [8], [12], [30], [35], [56], [62]–[64], [72]. A demonstration of 3D integration for a different 2T1R structure can be found in [50]. Specifically, our design has been developed with the encapsulation layer technique [64] and the transistor-stacking technique [45], [56]. The vertical-plane horizontal-stacking process can be built based on [8], [64].

B. DNN Training

Before an accelerator system is dedicated to a specific application, weight values should be initialized so that the values can be computed with given inputs and generate a result. The training has three phases: feedforward, backpropagation, and weight update. First, the feedforward is performed to get a classification result. Then, an error is calculated by comparing it with the ground truth, and the error is backpropagated based on the chain rule. Weights are updated based on the gradients calculated from the error. The accelerator can work its task through the testing (inference) composed of only the feedforward process.

1) Feedforward: Typical layers of neural networks are convolution, pooling, activation, and fully-connected (FC) layers, as displayed in Figure 3a. In a convolution layer, given input

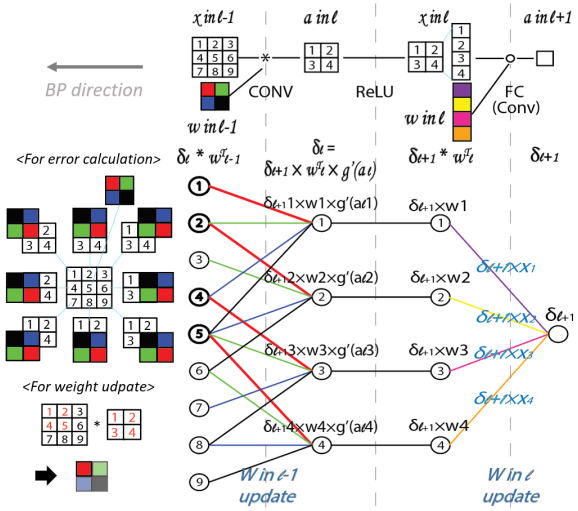

Fig. 4. Backpropagation and weight update processes. Based on the chain rule, a gradient is calculated by multiplying a coming gradient and a local gradient of the node. In the case of multiplication, operands are the local gradient of each other. Therefore, the corresponding weight (or input) value is multiplied as a local gradient of a node. Then, the final result of one error (or gradient) is computed as the sum of corresponding multiplications to the node, which means the convolution operation, as shown in the left-side subfigures.

data, x, is convolved with sliding weight filters, w. Outputs of the convolution layer, a, are calculated by

$$a_{l+1}(o_h, o_w, n) = W_l * X_l = \Sigma_{c_l=0}^{C_l-1}\Sigma_{k_w=0}^{K_w-1}\Sigma_{k_h=0}^{K_h-1} w_l(k_h, k_w, c_l, n) \cdot x_l(i_h + k_h, i_w + k_w, c_l), \quad (1)$$

where $*$ denotes the convolution operation. C_l, K_w, K_h are attributes of the kernel like the number of channels, kernel width, and kernel height, respectively. Recently, due to a huge number of kernel parameters, light models with depthwise or pointwise (a.k.a. 1×1) convolution have been proposed (Figure 3b). Depthwise convolution does not accumulate across input channels and is effective to reduce the input size, while pointwise convolution has 1×1 size of kernels to adjust the number of channels.

A maximum element or an average value under a window passes through the pooling layer. An activation function (f) is chosen from nonlinear functions such as sigmoid, rectified linear unit (ReLU), or hyperbolic tangent (tanh). In a FC layer, the dot product, symbolized as $\cdot$, is performed between unrolled input vectors and corresponding weight vectors:

$$A_{l+1} = W_l^T \cdot X_l + b_l, \quad (2)$$

where W_l is weight and b_l is bias of layer l. The feedforward process runs in a layer-by-layer manner and predicts the probability of the output classes concerning the input.

2) Backpropagation: The backpropagation proceeds to find more optimal weight values to reduce the error. Specifically, in the backward process, errors in each layer are computed from the last layer to the first layer. A loss function (J), defined by L^1, L^2, or cross-entropy loss, is used to quantitatively evaluate the last layer's error in the first place. The last layer's error, δ_L, is calculated by the gradient of the loss function. Then, based on the chain rule, errors in a layer, δ_l, are obtained by its next

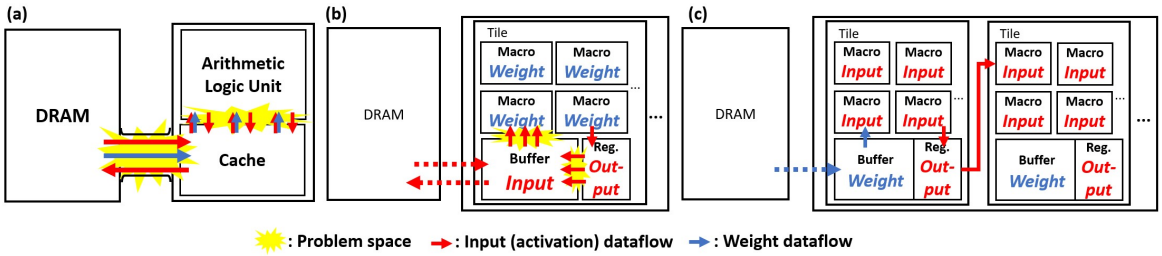

Fig. 5. Dataflow of (a) the von Neumann system, (b) weight-stationary and (c) input-stationary accelerators.

layers' errors (δ_{l+1}) and the local gradient. δ_{l+1} is convolved with the local gradient of convolution or FC layers, i.e., the transposed (rotated) weight matrix, and multiplied with the local gradient of the activation ($g'(a_l)$), as shown in Figure 4. In the max-pooling layer, the shrunken matrix dimension in the forward path is restored, and the maximum value goes to its original position while other elements are dead as 0. Eq. 3 shows the calculation of errors at a glance; with the L^2 loss function, δ_L is measured through the difference between the target (y_{target}) and predicted (y_{pred}) outputs.

$$\delta_l = \begin{cases} y_{target} - y_{pred}, & l = L \\ \delta_{l+1} * W_l^T \circ g'(a_l), & otherwise \end{cases}. \quad (3)$$

The type of pooling, activation and loss functions are varied according to the network design for high accuracy and fast convergence. For the simplicity of the explanation, we will describe INCA based on the max-pooling, ReLU activation, and L^2 loss function.

3) Weight update: Weight update is to adjust weight values for accuracy. We assume the vanilla gradient descent optimizer, which is more hardware-friendly than other optimizers. Given gradients from the next layer, *errors in layer l (δ_l)*, are convolved with inputs of the layer (x_l), *local gradients of weights*, because operands in multiplication are the local gradients for each other. In the example of Figure 4, x_l of 3×3 size are convolved with δ_l of 2×2; the first convolution is colored in red. The obtained difference is subtracted from the original weight. The step size parameter, η, is also combined into the amount of updated weights to control the learning rate. To sum up, the weight update is computed by

$$W_l^{(t+1)} = W_l^{(t)} - \eta \cdot \delta_l * X_l. \quad (4)$$

III. Dataflow

A. DNN Accelerators Trapped in WS

Explosion of DNN accelerators. As DNN algorithms have been highly developed, the limitation of the conventional hardware is looming large. The compute speed becomes much faster, while bandwidth limits the data transfer between computing and storage units. Data supply in hardware for DNNs couldn't catch up with the computing speed, therefore, lengthening the overall latency. This phenomenon has been indicated as the *von Neumann bottleneck*, as shown in Figure 5a. Consequently, a new hardware mechanism, PIM or CIM, has emerged and tremendously studied in numerous DNN accelerators: [2], [5]–[7], [9], [10], [13], [15], [17], [20], [22]–[24], [28], [36], [37], [42], [46]–[48], [55], [57], [59], [60], [67]. Due to the complexity of training, most of the prior PIM-based accelerators only focused on the inference phase. In inference, weights have static values but activation values are dynamic along the network layers. Over the past years, it has been taken for granted that weights are remaining in RRAM arrays and that inputs are moving, as shown in Figure 5b. To the best of our knowledge, all the previous PIM-based accelerator designs have adhered to the WS dataflow. However, there are limitations under the settled frame of WS as follows.

Limitation 1: Remaining Excessive Memory Access. We observed that the current WS-based PIM architecture does not fully resolve excessive memory accesses. The feedforward process still requires fetching and saving data for the current and subsequent layer, respectively. Although the amount of data in pure memory is reduced by using PIM for weights, the fundamental data movement between buffers and PIM has the same fashion as in the von Neumann architecture (Figure 5a and 5b). Figure 6 shows that DRAM and buffers dominate the energy consumption when executing commonly used networks, VGG16 and ResNet18 [18], [44].

Limitation 2: Redundant RRAMs in Training. We noticed that the WS principle causes a redundant usage of RRAMs in training from two perspectives. The first comes from the need for transposed weight matrices. Because of the different disposition of elements, previous crossbar designs should separately hold transposed weight matrices in addition to the original weight [20], [48]. The number of parameters is not negligible, when it comes to the increasing number of kernel parameters. Whereas weights of an early model (LeNet5 [32]) occupy 240KB, 553MB are necessary for VGG16 in a 32-bit system. Secondly, computed errors and gradients in the backward step should be kept along with the weight values (or in DRAM/buffers, but we assumed in RRAMs due to the cost and the design in [48]). Because the number of errors (gradients) equals that of layer's weights (inputs), the total number of errors and gradients becomes remarkable in accordance with the batch size and the model size [16].

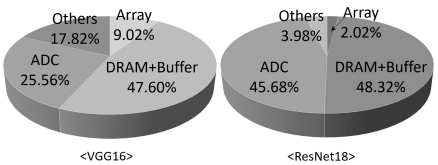

Fig. 6. Energy breakdown in execution of VGG16 and ResNet18 in WS dataflow with CIFAR10: DRAM and buffers occupy the largest portion of overall energy consumption.

TABLE I
ACCURACY DROP ACCORDING TO VARYING WEIGHT AND ACTIVATION BIT DEPTHS, COMPARED TO GPU WITH FLOATING-POINT ARITHMETIC [21].

8-bit Weight				8-bit Activation			
Act. 7	Act. 6	Act. 5	Act. 4	Wt. 7	Wt. 6	Wt. 5	Wt. 4
-0.3%	-0.4%	-1.3%	-3.5	-1.3%	-1.1%	-3.1%	-11.4%

Limitation 3: Coarse-grained Arrays. Because DNNs boast an impressive data size of weights, fine-grained computation with small arrays shall entail more peripheral circuits and post-processing. For this reason, large arrays are prevalent with a size of 128×128 or above in WS accelerators. But coarse-grained accelerators get involved in two issues: high-bit-precision ADCs and utilization. It is well-known that ADCs exponentially undermine performance and energy efficiency [67], [71]. For example, four 4-bit ADC at 2.1GHz can replace one 8-bit at 1.2GHz. On the other hand, light models, which employ depthwise and pointwise convolution, have appeared to reduce weight parameters and computational complexity. Light models in coarse-grained architectures face worse hardware efficiency because of poor utilization as well as the large ADCs. Although some works have proposed optimized designs, they are effective in only either issue and involve drawbacks. Fine-grained WS accelerators demand extra processing but do not resolve light models' utilization [67], [71]. Likewise, optimized designs for depthwise convolution are not generally applicable [70].

Limitation 4: RRAM Nonideality's Impact on Accuracy. Table I shows a more severe accuracy drop in the loss of weight information than the case of activation [21], [4]. As such, neural networks are more susceptible to weight variance. When utilizing RRAMs for storing weight values like WS accelerators, therefore, the RRAM's nonideal characteristics, such as nonlinearity and asymmetry, are highly likely to cause severe accuracy degradation. According to [66], a WS system with TaOx/HfOx RRAM shows an 11% accuracy drop in VGG8 on CIFAR10 when compared to GPU with floating-point arithmetic.

B. Input-stationary Dataflow

Breaking with the newly shaped convention in PIM designs, we first propose *input-stationary (IS)* dataflow for PIM accelerator. As shown in Figure 5c, input (activation) data is stocked in RRAM crossbars, which serve the PIM function, and weights are fetched from buffers (when exceeding buffers' capacity, DRAM as well). Following our observation, IS implementation can address the four limitations.

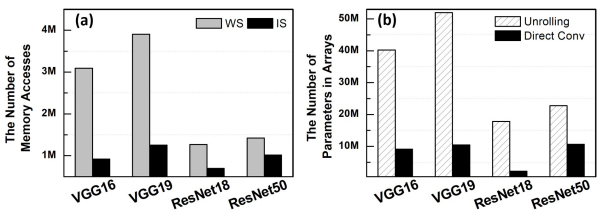

Fig. 7. (a) The number of memory accesses in WS and IS dataflow according to the network types. (b) Comparison between unrolling and direct convolution cases with IS dataflow in terms of the number of parameters in RRAM arrays.

Key Insight. The main difference of IS from WS is that generated outputs are propagated to their designated RRAM arrays, not buffers or DRAM, during feedforward. We formulate the number of memory accesses for fetching and saving to quantify the difference between WS and IS. The memory accesses are categorized into the fetching inputs/weights and the saving of outputs.

In both dataflow cases, input or weight elements in the size of one weight channel are necessary to obtain one output element. Therefore, the estimated number of memory accesses for fetching data per one output element is calculated by

$$ceil(K_H \times K_W \times C \times bit\_precision/bus\_width). \quad (5)$$

On one hand, WS repeats the fetch per an output element as many times as the number of output elements, and consequently, the number of output elements should be multiplied to eq. 5. On the other hand, IS needs the fetches as many times as the number of output channels because it reuses the obtained filter information to get a whole output channel.

In addition to fetching, WS requires more accesses to save the outputs. Following ISAAC [42], which is a representative PIM design, every output should be immediately redirected to eDRAM due to the pipelining. The access for saving data would happen as many as

$$ceil(N \times bit\_precision/bus\_width) \times O_H \times O_W. \quad (6)$$

The total number of accesses is compared in Figure 7a, based on an assumption that data has 16-bit precision and bus-width is 256-bit. It is easily noticed that WS requires from two (ResNets) to three times (VGGs) more accesses.

Next, IS accelerator design does not require a huge increase of RRAMs for training, constantly maintaining a small capacity of RRAM. IS design saves weights in the conventional memory and can load them in a different order; in other words, IS can hold the same memory space, not wasting additional memory capacity for transposed matrices. As we shall show in the following section, an IS accelerator does not need to hold activations and errors simultaneously in RRAM arrays; during the backpropagation, calculated errors can overwrite unnecessary activations because previous calculations have completed the use of the activations. Also, smaller arrays, which match well with small ADCs and light models, are feasible in IS because the number of inputs is generally less than that of weights and is similarly maintained across layers due to the increasing channels but the decreasing input size. Lastly, IS naturally mitigates the accuracy drop by storing weights on more reliable conventional memories.

Challenges. Unlike the mathematical convolution in the 2D shape, called *direct convolution*, previous PIM accelerators have chosen *the GEMM-based convolution*, which demands a process to unroll incoming data to match with the array's architectural characteristic and ensure high parallelism. In the case of IS dataflow, we observe unrolling process brings about a steep increase in the number of RRAMs because the unrolled input data have repeated elements according to the kernel sliding. Figure 7b shows a large difference between the

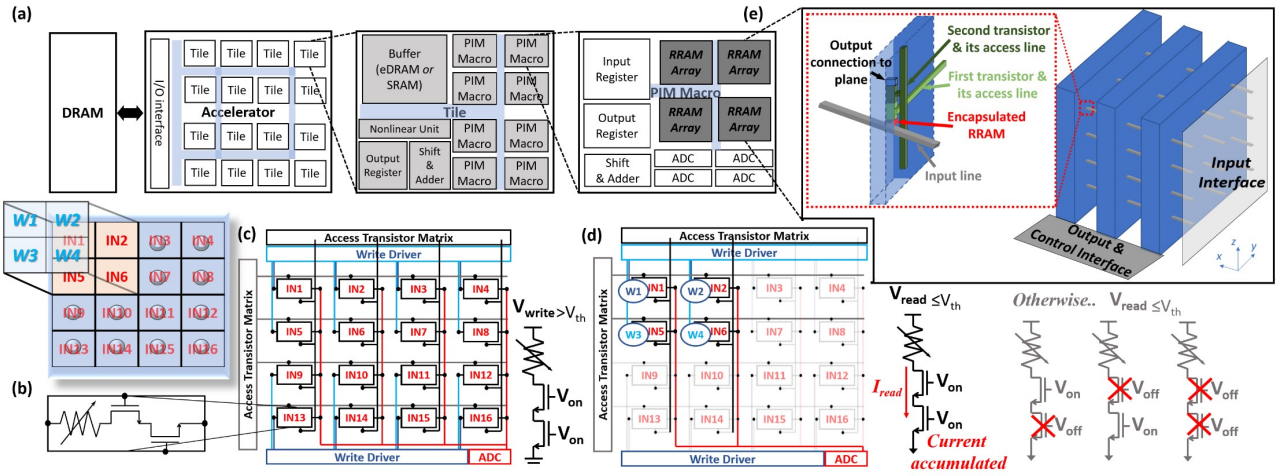

Fig. 8. (a) Overview of INCA architecture. (b) The used 2T1R cell. (c) Writing scheme to save input (activation) data. (d) Reading scheme for direct convolution: the cells under the activated 2×2 kernel window receive weight information as its shape; other cells' one or two transistors are off not to be accumulated in the convolution result. (e) The 3D array architecture, stacking vertical planes horizontally to implement multiple images in a batch, which can be built by the encapsulation layer [64] and transistor-stacking techniques [45], [56]. The vertical-plane horizontal-stacking process is based on [8], [64].

cases with and without unrolling in inputs: $4.4\times, 5.0\times, 8.0\times$, and $2.1\times$ with VGG16, VGG19, ResNet18, and ResNet50. Therefore, we decided to implement direct convolution without unrolling process. However, the current crossbar design is not appropriate for direct convolution. Kernels should be supplied across both sides of the crossbar, but the current design only provides column-wise accumulation. We designed a base cell structure by adding one more transistor to control the row-wise direction as well.

Moreover, naïve IS implementation with direct convolution couldn't achieve high parallelism, unlike WS. Specifically, WS implements weight values of all channels in crossbars after unrolling, following the order of kernel sliding. Thus, supplying one input data vector can generate an output element per channel. In contrast, IS with direct convolution holds inputs in the crossbar; therefore, only one channel's output can be obtained by giving corresponding weights. Rather than increasing and saving the same input data to calculate an output across channels, we employ 3D RRAM for high parallelism. It is worth noting that WS can hardly take advantage of 3D. Each kernel across channels requires a different input. However, 3D planes (kernels in WS) share pillars (inputs), which means receiving the same inputs regardless of planes. On the contrary, 3D RRAM can be fully employed in IS, since different images in one batch need the same weight data, which is given to the shared pillars. We implement each input feature map in one batch to each plane of 3D RRAM to process input feature maps in one batch at once, instead of channel-wise parallelism. The 3D design also enables batch training by holding all input (activation) data for the backward process.

IV. INCA: Input-stationary Crossbar Architecture

In this section, we provide details of the INCA architecture with the IS dataflow. Figure 8a presents an overview of the hierarchical architecture of INCA: tiles composed of adders, nonlinear units, buffers, and macros. Macro is the basic unit for computation, which is constituted by adders, ADCs, registers, and arrays. First, we will describe the array design and its operations at the cell and array level, as colored by dark grey in Figure 8a. The main design characteristic of INCA is that 2T1R cells for direct convolution are integrated into our novel 3D RRAM. Then, we will present mapping schemes and hardware dataflow in each training step, which corresponds to the region colored by light grey in Figure 8a.

A. 2T1R-based Array: Vertical Plane

The sneak path current is inevitable in 1R-based arrays because RRAM is like a variable resistor [41]. For this reason, 1T1R has become a standard in RRAM crossbar design to avoid the sneak path current issue by using the transistor's switch function. Unlike the general approach, here we propose a 2T1R structure to implement direct convolution in crossbar design. To the best of our knowledge, the application of the 2T1R cell to the DNN accelerator is the first of its kind for the purpose. Figure 8b shows our 2T1R cell. Two transistors are controlled in two perpendicular directions in an array.

We note two main characteristics of design at the array level as follows. **Voltages are independently supplied to all RRAM cells simultaneously** for one-time voltage supply in direct convolution. Furthermore, **all columns are tied together at the bottom of crossbar** for one-shot accumulation in a convolution operation. Figure 8c and 8d describe writing data and reading a convolution result under a window, respectively. As described in Figure 8c, the write voltage to 4×4 cells should exceed the threshold of RRAM to change the memristance. The bottom line tying all columns (i.e., all cells) is grounded so that the voltage difference applied to RRAMs is nearing the write voltage. Two transistors of 4×4 cells are on. Memristance of all cells is adjusted according to the input data voltage at the same cycle.

Figure 8d displays that a read voltage under the threshold is applied to the top electrode of RRAM, and the end of the cell is floated. The cells only under the 2×2 kernel window

receive voltages by activating the first and second vertical and horizontal lines. Other cells' one or two transistors are off not to be accumulated in the convolution result. Once one convolution is finished, by turning off the first column and on the third column, the next convolution can be computed. In other words, INCA produces results in direct convolution by keeping the original shape of inputs and weights, thanks to the two transistors. Multiplied output between the voltage and memristance can be measured as a current based on Kirchhoff's law. To sum up, input/activation data is written in arrays at the writing step first, and then a voltage set of kernel information is given to the crossbar. All written inputs and applied weights are given as their original shape. The voltages are multiplied with the adjusted memristance, and the currents from each cell are accumulated at the bottom of the crossbar at once. The convolution can be computed without any harm to the original shape of input and kernel data in one cycle. Each array plane is vertically erected in the INCA design.

B. Novel 3D RRAM Architecture

To achieve high parallelism in batch processing, we propose a new 3D RRAM design based on our 2T1R-based vertical plane. As shown in Figure 8e, our architecture design is based on HRRAM. Vertical array planes are stacked horizontally, and images in a batch are mapped to each vertical plane, respectively. Multiple pillars, which decide the array size, penetrate all the planes and receive inputs. Because the images need the same weight matrices, we can process multiply-and-accumulate (MAC) operations for all the planes at once, by giving voltages to the pillars shared by planes.

An input interface covers the front of the architecture to provide separate voltage sources to all pillars (Figure 8e). Since the input feature map size becomes smaller in deeper layers, the plane size should be optimized to be relatively smaller than the widely used size. Assuming that the crossbar size is 16×16, the number of interconnect lines for input pillars will be 256, which is feasible in the current technology. The crossbar size validity will be discussed in Section V.

RRAM is sandwiched between a pillar and the first transistor, and the transistor is also connected to the other one. 2T1R cells are embedded in a plane colored in blue in Figure 8e. The second transistor follows the first transistor, and its end is connected to the blue plane for the one-shot accumulation with other cells. Stacked planes individually interact with other planes through output and control interfaces embedded in the bottom, as described in Figure 8e. In writing, input voltages are supplied to the corresponding pillars (input lines) to input images, two transistors are turned on by controlling two perpendicular lines, and the blue planes are grounded. For the reading operation, kernel information is given to the pillars (input lines), two transistors of the cells in the window are on, and the blue planes are floated. The total currents passing through cells are accumulated in the blue plane and detected in the output interface. All the planes of one 3D independently work and are not affected by other planes.

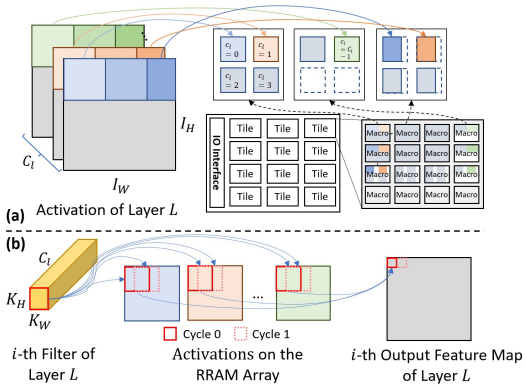

Fig. 9. (a) Activation mapping. (b) Computation process of one partition. We do not illustrate the vertical dimension of the RRAM array for readability.

C. Mapping Schemes in INCA

Intra-layer Mapping. In INCA, we spatially map input feature maps of each layer onto RRAM arrays to reduce the memory access cost. The RRAM array is the basic block for mapping in our design. Each RRAM stores one bit of input values. Thus, if we assume the input is 8-bit, we need 8 separate RRAMs for the value. We adopt the bit-serial design in INCA to reduce the overhead of peripheral circuits. The weight is fed into each array bit-by-bit, while the output is accumulated through a shift-accumulator. Since, for each array, we only need to accumulate the multiplication result of up to 9 binary values (for 3×3 kernel), a 4-bit ADC is sufficient for the conversion. For readability, we only present one of the arrays for all bits when discussing the mapping scheme. Figure 9 shows our mapping scheme. We partition the input feature maps according to the size of the array and map each partition of all channel-wise samples in the batch to each RRAM array (Figure 9a). Each plane produces one output per computation cycle, and all outputs of the 3D array are collected and written to the target array for the computation of the subsequent layers. We then assign the partitions to the RRAM array sequentially in the input channel order, i.e., the same convolution window of different input channels is assigned to one PIM macro. Thus, using the adders at each level, this mapping naturally forms an adder tree to accumulate the result from different input channels. For those positions at the edge of each partition (a.k.a. 'halo'), we calculate a partial sum within each partition and rely on the adder to gather these results. For other operations like pointwise convolution and FC layer, we fold the dimension which requires accumulation (e.g., the input channel in the pointwise convolution) to the 2D plane, slide the window with the stride that is same as the kernel size, and use different weight for each window.

Inter-layer Mapping. We align each layer to a PIM macro, i.e., start each layer from a new PIM macro. The layers are also sequentially mapped to the accelerator. Having processed by pooling and activation units, the output of each layer will be written into the location for the next layer as the activation. This mapping scheme makes it possible to overlap the activation write with the computation without the bus contention issue. The activations will remain in the array to be used in the backpropagation, until overwritten by errors.

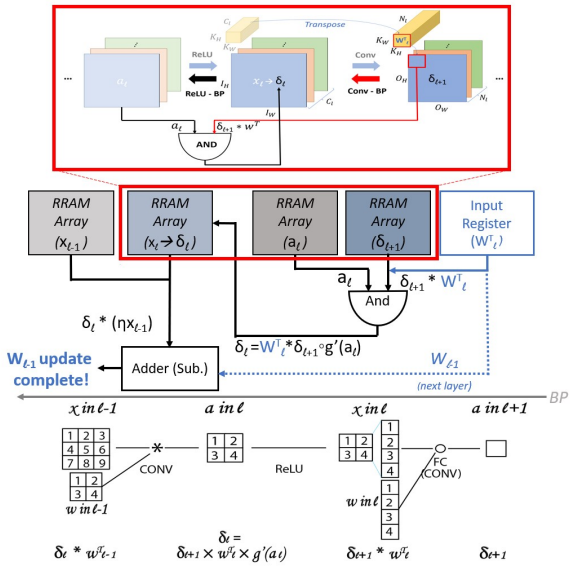

Fig. 10. Backward process according to network layers. The red box shows the mapping scheme during backpropagation in detail.

Backward. Figure 10 explains the overall backpropagation process. Error calculation and storage processes are illustrated in the red box with more details. To compute activation's errors of convolution or fully-connected layers (l), weight values of transposed matrices are fetched from memory and convolved with the previously calculated errors (δ_{l+1}). For the activation layer, AND can produce the same results as the multiplication with the gradient of ReLU. Thus, we use AND gates for the convolution between inputs of the activation layer (a_l) and the convolved values ($\delta_{l+1} * W_l^T$). The generated values are errors of the layer δ_l. INCA can reuse RRAMs, which were used for input values in l, for the calculated errors in l, since the overwritten input values will no longer be necessary for later computations. The written δ_l is also propagated in the same way. For the sake of simplicity of the figure, we omit the pooling layer; the pooling layer is processed by a lookup table (LUT) to find the original position of the maximum value, while other values' following nodes will be dead.

Finally, the weight update process for l_{l-1} is launched with δ_l–produced through the process in the red box in Figure 10. We note that l_l's weight update has already been completed with δ_{l+1}. The calculated errors are convolved with input values of the convolution layer (x_{l-1}). Weights in l_{l-1} have been loaded, while the convolution between inputs and errors. Therefore, the convolution results will be subtracted from the original weight value of l_{l-1}, not affecting the subsequent error calculations. Also, the update process is executed in parallel with the process in $l-1$, that is, $\delta_l * w_{l-1}^T$.

V. Evaluation

This section elaborates on end-to-end simulation results and analysis. Note that results of light models (MobileNetV2 and MNasNet) are separately discussed from the other networks because the two light models show a different trend due to the depthwise and pointwise convolution.

TABLE II
Architecture configuration summary with circuit simulation setup and results.

Models			
Weight & Activation Prec.	8-bit	Batch size	64
INCA Accelerator			
Subarray Size	16×16	# of Stacked Layers	64
Macro Size	8	Tile Size	12
Cell Prec.	1-bit	ADC Prec.	4-bit
Subarrays Per ADC	16	Technology	22nm
Buffer Size	64KB	Buffer Bitwidth	256-bit
DRAM	8GB HBM2		
Baseline Accelerator			
Subarray Size	128×128	Macro size	8
Tile Size	12	Cell Prec.	1-bit
ADC Prec.	8-bit	Technology	22nm
Buffer Size	64KB	Buffer Bitwidth	256-bit
DRAM	8GB HBM2		
Circuit			
On Resistance	240KΩ	Off Resistance	24MΩ
Technology	65nm	Scale factor	0.34
Read Voltage	0.5V	Write Voltage	1.1V
Read Pulse Width	10ns	Write Pulse Width	50ns
Off-cell Power	10.42nW	On-cell Power	1.03uW
INCA Cell Width	600nm	INCA Cell Length	700nm
Baseline Cell Width	540nm	Baseline Cell Length	485nm
GPU - Titan RTX			
Memory Bandwidth	672GB/s	Graphic Memory	24GB GDDR6
CUDA Cores	4608	Power Consumption	280W
Area	$754mm^2$	Peak Performance	16.3TFLOPs

A. Experiment Setup

To evaluate the performance of INCA, we selected six representative CNN models on ImageNet [11], including VGG16, VGG19 [44], ResNet18, ResNet50 [18], MobileNetV2 [40], and MNasNet [53]. These models span different computation regimes and contain a rich set of layer variations. We compared INCA to a baseline 2D WS architecture and a GPU. For the baseline, we referred to a representative 2D accelerator design, ISAAC [42]. However, ISAAC only can support the feedforward phase for inference. Thus, we reflected the PipeLayer [48] in the baseline design for training.

For the evaluation, a customized simulator was implemented based on our circuit simulation results and NeuroSim+ [38], [39]. NeuroSim+ is an end-to-end simulation framework and has been used for the performance and energy evaluation of accelerators [14], [29], [57], [61]. Even though its recent version supports inference and training for 2D, we customized the simulator in our way because the framework has different dataflow and provides only inference in case of 3D. Specifically, we simulated our 2T1R circuit and obtained its layout in a vertical plane by Cadence Virtuoso with the TSMC 65nm technology library. Then the circuit simulation results were scaled down according to the rules of scaling to match the technology node selected in the accelerator simulation. We then integrated the key circuit-level parameters of the 2T1R cell into the simulator, replacing the original cell data. We also modified the dataflow and mapping in NeuroSim+ to reflect INCA and baseline designs. We adopted 32pJ per 8-bit as an approximated estimation of the HBM2 access cost, as provided in the framework. Table II summarizes the key parameters for INCA, the baseline, and the used GPU. For the sake of fair comparison, the simulation of INCA and the baseline employed the same peripheral components of NeuroSim+.

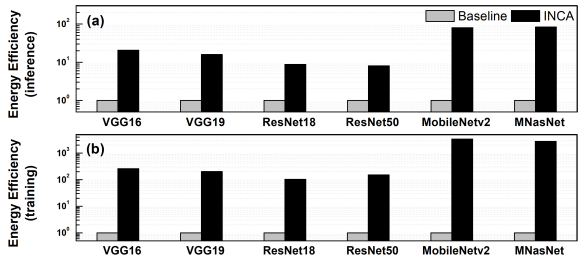

Fig. 11. Energy efficiency comparison of INCA to the baseline architecture according to different network models: (a) inference and (b) training.

TABLE III
THE ESTIMATED NUMBER OF ACCESSES TO BUFFERS IN BASELINE AND INCA DURING INFERENCE.

# of accesses	Baseline	INCA
Eqs.	Eq. (5) $\times O_H \times O_W$ + Eq. (6)	Eq. (5) $\times N$
VGG16	1,544,496	460,000
VGG19	1,952,176	625,888
ResNet18	632,880	349,024
ResNet50	711,022	508,950
MobileNetV2	258,024	66,832
MNasNet	244,656	92,333

B. Results and Analysis

1) Energy: Figure 11 exhibits comparison results of energy efficiency between the baseline and INCA during inference and training. During inference, INCA shows $20.6\times$, $15.9\times$, $8.7\times$, and $8.0\times$ improvement in VGG16, VGG19, ResNet18, and ResNet50 than the results of the baseline. In training, the batch parallelism of INCA boosts energy efficiency. INCA presents the energy efficiency for two orders of magnitude of improvement over the baseline: $260\times$, $202\times$, $103\times$, and $152\times$ when executing each of the four models.

As summarized in Table III, we estimated the number of accesses to buffers according to the different networks during inference. We only take into consideration of the pipelined design in ISAAC because a non-pipelined design needs an even higher capacity of buffers and longer latency. Therefore, access to load and save is necessary at each convolution to execute the proposed pipelining smoothly. The number of accesses is calculated based on the equations in Table III. As discussed in Section III, the estimation implies that INCA would reduce the number of accesses to buffers and that VGGs would experience higher improvement than ResNets, which are well-matched with the results in Figure 11. Note that the training process may double the accesses in INCA to fetch transposed weight matrices. However, most networks still take advantage of the IS dataflow during training as well.

We then plotted the layerwise energy of the baseline and INCA in Figure 12 with VGG16, in linear and log scales for a thorough analysis. In Figure 12a, the baseline features a tremendous gap between some early layers and the others in terms of energy consumption. The gap comes from the fact that the early layers carry out most of the convolutions in WS and inputs (activations) are loaded and saved during the remarkable convolution operations. On the contrary, INCA consumes a constant level of energy due to the similar size and reuse of kernels across layers. Figure 12b in log scale reveals that the trend in layerwise energy consumption also shows a good match with the estimation in Table III. Although

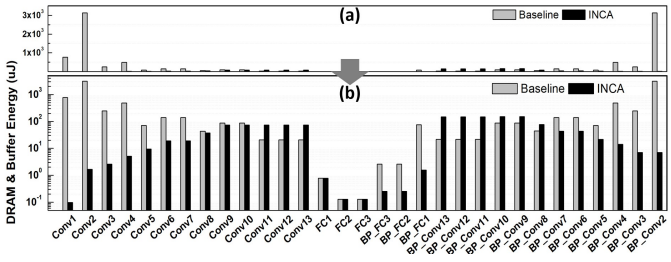

Fig. 12. Layerwise energy consumption by DRAM and buffer energy in the WS baseline and INCA when executing VGG16 with ImageNet, plotted in (a) linear and (b) log scales.

the output size keeps shrinking as approaching the last layer, the increasing number of channels and kernels (C and N) continuously produces frequent access to buffers in WS. In contrast, INCA only needs kernel information, thereby not affected by the input size. The fetched weights are reused for different sliding windows; thus, the feedforward shows a trend of increasing energy consumption under the increase of C and N. Accordingly, INCA consumes more energy than the baseline in a few later layers but it negligibly impacts on the overall consumption due to the small number of computations in those layers. In training, as aforementioned with Table III, while the baseline uses similar energy as during inference, the number of accesses of INCA becomes twice than during inference due to transposed matrices.

Moreover, INCA is effective in diminishing the energy consumption by ADCs, thanks to the elimination of high-bit-precision ADCs. While 128×128 baseline arrays demand high-bit-precision ADCs to support coarse-grained computations, small ADCs can cover INCA due to small arrays. As mentioned early, one 8-bit ADCs consumes energy as much as four 4-bit ADCs, not two [67]. Therefore, ADCs of INCA spend $5\times$ less energy in total than ADCs of the baseline, as depicted in Figure 13a. Lastly, we also plotted a pie chart of the overall energy breakdown to compare with the WS case of Figure 6. The smaller DRAM and buffer segment in Figure 13b indicates reduced access to buffers of INCA. All the given evidence supports the improved energy efficiency of INCA.

2) Latency: Figure 14 describes the latency comparison results between the baseline and INCA. INCA shortens the inference and training time, compared to the baseline. But it is noteworthy that the magnitude of improvement is less than that of energy efficiency because of the pipelined execution of the baseline. During inference, INCA attains $4.6\times$ and $3.7\times$ speedup in VGG16 and VGG19; $1.9\times$ and $4.8\times$ speedup in ResNet18 and ResNet50. The training process of INCA presents a higher speedup than the inference result: $18.6\times$,

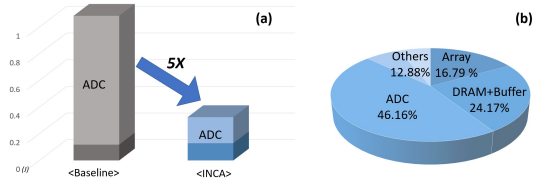

Fig. 13. (a) Comparison of energy consumption reduction by ADCs between baseline and INCA with VGG16. (b) Energy breakdown for an apple-to-apple comparison with WS-based accelerator in Figure 6.

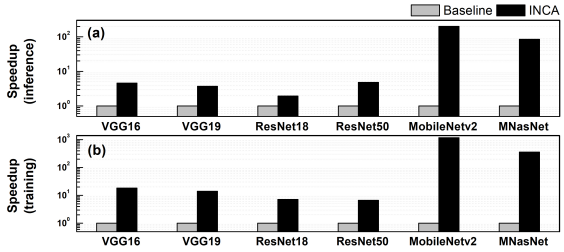

Fig. 14. Speedup comparison of INCA to the baseline architecture according to different network models: (a) inference and (b) training.

14.2×, 7.2×, and 6.8 in VGG16, VGG19, ResNet18, and ResNet50, respectively. The batch parallelism in INCA largely contributes to the improvement in training. While the WS baseline needs repeated operations for each image in the same batch, INCA can compute all images in parallel.

Generally, writing to RRAM necessitates a longer latency than reading. Here we clarify the speedup of INCA despite its repeated RRAM writing operations. In INCA, a pipeline-fashioned execution partly hides the RRAM write latency by the read latency. To be specific, while a convolution result is written to its corresponding RRAM cell, the next convolution is launched to read. Yet the write latency still increases the overall time for one convolution since writing spends about 2× longer than reading in INCA. The read latency in the baseline, however, is about 2× than the write latency of INCA as a result of the large size arrays and the high-bit-precision ADCs. Therefore, INCA could achieve a latency-saving effect.

3) Comparison with GPU: We also compare INCA with GPU to confirm the effectiveness of INCA over a non-RRAM-based accelerator. Figure 15a shows normalized energy efficiency, while Figure 15b displays normalized throughput per area improvement for an iso-area comparison. The comparison results present that INCA outperforms GPU in both and is particularly conducive to energy saving across network models and to throughput in light models.

4) Light models: INCA accomplishes outstanding hardware efficiency, especially in MobileNetV2 and MNasNet with one or two order(s) of magnitude higher energy improvement/speedup than the results of VGGs and ResNets. Inference of MobileNetV2 and MNasNet results in 80× and 83× better energy efficiency in INCA than in the baseline (Figure 11a); 201× and 85× speedup as well (Figure 14b). Training brings about even phenomenal improvement: 3873× and 2790× in energy (Figure 11b) and 1187× and 363× speedup (Figure 14b) in MobileNetV2 and MNasNet, respectively.

The efficiency result from the light models can be explained by utilization. We illustrated Figure 16 to compare utilization

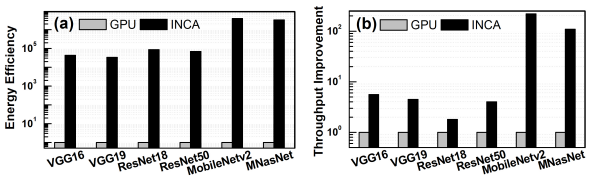

Fig. 15. (a) Energy efficiency comparison and (b) an iso-area throughput comparison between INCA and GPU in training, according to different network models.

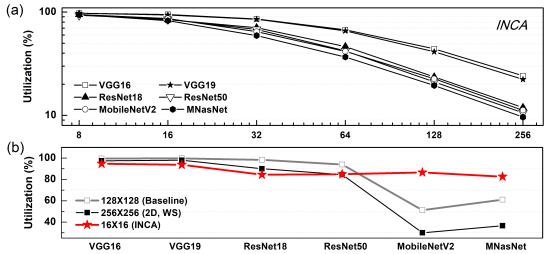

Fig. 16. (a) Dropped utilization of INCA as the array size is increasing. (b) Utilization comparison between INCA and the baseline in different network models; while ours maintain the utilization constant, the WS accelerator undergoes a drastic drop in the light models.

TABLE IV
REQUIRED MEMORY FOOTPRINT (RRAM ARRAYS AND BUFFERS) OF THE BASELINE AND INCA FOR BOTH INFERENCE AND TRAINING.

Memory footprint [MB]	Baseline		INCA	
	RRAM	Buffers	RRAM	Buffers
VGG16	272.57	8.69	8.69	131.94
VGG19	283.94	9.94	9.94	137.00
ResNet18	24.36	2.08	2.08	11.14
ResNet50	58.79	10.15	10.15	24.32
MobileNetv2	13.05	6.45	6.45	3.31
MNasNet	13.57	5.29	5.29	4.14

of convolution layers in IS and WS. It is confirmed that 16 × 16 is the optimized size for INCA because it achieves competitive utilization (Figure 16a). Also, INCA maintains utilization across different networks because the usage of RRAMs is not affected by kernel variance. However, the WS baseline utilization dramatically reclines in MobileNetV2 and MNasNet, unlike the utilization in VGGs and ResNets. The two light networks include depthwise and pointwise convolution, whose accumulation scope is much smaller than regular convolution due to smaller kernels. For instance, 3×3 kernels in depthwise convolution only use nine of 128 cells in a column since each output pixel only needs the accumulation of nine products between inputs and weights. The low utilization terribly aggravates WS hardware efficiency. In training, the batch parallelism effect in INCA causes the dramatic efficiency gap between the baseline and INCA.

5) Memory footprint: In Table IV, we estimate the requirement of memory footprint to support both inference and training phases. As indicated in Limitation 2 of the WS dataflow, WS accelerators should separately save additional parameters for training into RRAMs, along with original weights. Transposed weights and errors occupy the same capacity as original weights and inputs (activations) in WS. Gradients in WS are obtained by AND and used for error calculation in an instant. Consequently, the minimum footprint of RRAM arrays is the sum of the number of inputs (activations) and the doubled amount of weights, while buffer size is as much as the capacity of inputs (activations). On the other hand, INCA only needs to save weights in buffers because it can fetch transposed weights from the original weight buffer according to the different element dispositions. In case of RRAMs, all the inputs (activations) are necessary for the backward process to update the errors. Therefore, the number of activation values and the bit precision determine the minimum capacity of buffers. It is worth noting that INCA does not demand

TABLE V
AREA BREAKDOWN OF BASELINE AND INCA.

	Baseline		INCA	
	Spec	**Area [mm^2]**	**Spec**	**Area [mm^2]**
Buffer	64KB 168	13.944	64KB 168	13.944
Array	128×128 168, 12, 8	7.927	$16 \times 16 \times 64$ 168, 12, 8	0.793
ADC	8-bit 168, 12, 8	30.298	4-bit 168, 12, 8	4.5864
DAC	1-bit 168, 12, 8×128	0.343	1-bit 168, 12, 8×256	0.686
Post-processing	ReLU Max-pooling 168	3.656	ReLU Max-pooling 168	3.656
Others	-	27.920	-	24.249
Total	**84.088**		**47.914**	

additional RRAMs except for inputs (activations) because INCA aggressively recycles the activation cells already used for the error calculation.

6) Area: INCA is a new 3D architecture proposed in this paper, therefore, we defined the INCA area by its projected 2D area. As illustrated in Figure 8e, the lengths on the x-axis (along output and control interface) and the y-axis (input interface) determine width (W) and length (L) of INCA. W of one plane is estimated as a doubled thickness of a transistor to ensure sufficient space to other planes [69]. We obtained L and the baseline cell area by our circuit layout of the 2T1R and 1T1R cells, respectively. Note that INCA vertically stacks 16 cells on one cell and that its high stackability offsets the sacrificed area by 2T1R—even brings area advantage, thanks to the 3D high stackability with a minimum impact on area [41], [58]. Hence, 16 cells of INCA occupy only $0.048\mu m^2$, while the baseline one-cell area is $0.030\mu m^2$ (after scaling). The summary in Table V verifies area-saving effect of INCA. For a fair iso-capacity comparison, we built INCA to possess the same capacity of RRAM as the baseline. The number of RRAMs in one 3D architecture ($16 \times 16 \times 64$) equals that of one crossbar in the baseline (128×128). Therefore, we organized the same number of crossbars and macros in a tile as the baseline design. While one crossbar of the baseline needs $491.52\mu m^2$, one 3D architecture of INCA demands $49.152\mu m^2$. Buffer and ADC/DAC were estimated based on [42], [67] and the others were measured by NeuroSim+. Due to the advantage of 3D design and small ADCs, INCA could effectively save area.

7) Accuracy: We also examine the impact of the RRAM's nonideality on accuracy. We chose the zero-centered normal distribution to model the noise caused by nonideal properties like variation, nonlinearity, and asymmetry, following [65]. The noise strength (σ) was adjusted from 0.5% to 5%, since the range between 2% and 5% is practically adopted for modeling. We used a pretrained ResNet18 model of the torchvision library and trained the network for ten epochs in every case. The noise was directly added to activations or weights during the training process. Table VI summarizes evaluation results. While accuracy dramatically declines in the case of noisy weight, the result of noisy activation shows a meager drop in accuracy. Therefore, the result proves that RRAM's nonideal properties more negatively affect the accuracy of the WS accelerator than the INCA accuracy.

TABLE VI
TRAINING ACCURACY IN ACCORDANCE WITH DIFFERENT NOISE STRENGTH (σ) APPLIED TO WEIGHTS AND ACTIVATIONS.

σ	**0.005**	**0.01**	**0.02**	**0.03**	**0.05**
Wt.	82.13	77.03	58.36	48.57	15.17
Act.	89.21	89.02	88.50	87.54	85.59

VI. FUTURE WORK FOR ENDURANCE

RRAM devices were found in relatively recent years under the name of memristor. Due to device immaturity, RRAM-based hardware has struggled with unstable reliability and robustness. INCA is also unable to avoid the endurance issue of RRAMs like other trainable accelerators. The previously proposed accelerators have assumed that an accelerator can hold all required weight parameters for networks. But accelerators need to rewrite RRAMs frequently as large networks have begun to be implemented in accelerators [16]. RRAM-based digital PIMs [20] have also evoked an extreme concern regarding endurance. However, researchers in the field are continuously addressing reliability, and fortunately, fast development of device technologies promises robust RRAM-based hardware [25], [43]. Meanwhile, IS dataflow is widely applicable to PIM designs beyond RRAM, therefore, we leave IS implementation into other designs as our future work to exploit more stable properties of other hardware candidates.

VII. CONCLUSION

This paper proposes an input-stationary PIM accelerator for the first time, named *INCA: input-stationary (IS) implemented crossbar accelerator*, supporting inference and training. We observed and identified limitations of previous PIM designs from the weight-stationary (WS) dataflow: 1) the remaining excessive memory access; 2) the enormous use of extra RRAMs; 3) the need for large ADCs and the poor array utilization in light models; 4) the RRAM nonideality's impact on accuracy. IS dataflow can naturally resolve the limitations; however, the current crossbar design impedes the IS implementation, presenting two main challenges: kernel sliding and parallelism. For kernel sliding, this work implements 2T1R cells with direct convolution. We then attain high parallelism and batch training implementation by a novel 3D RRAM architecture to take the structural advantage of 3D. Our experiment with ImageNet demonstrates that INCA shows higher hardware efficiency over one or two order(s) of magnitude, or even more in light models. The reduced memory accesses, low-bit ADCs, and sufficient utilization contribute to the improvement results of INCA. Furthermore, INCA is expected to release the redundant usage of RRAMs and generate more reliable *in situ* classification results. Across various hardware designs and applications, IS dataflow could be a compelling candidate for PIM accelerator designs.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation (NSF) under grant 2023752 and 2233808. The opinions and/or findings contained in this paper are those of the authors and should not be interpreted as representing the official views of the NSF.

REFERENCES

[1] G. C. Adam, B. Chrakrabarti, H. Nili, B. Hoskins, M. A. Lastras-Montaño, A. Madhavan, M. Payvand, A. Ghofrani, K.-T. Cheng, L. Theogarajan *et al.*, "3d reram arrays and crossbars: Fabrication, characterization and applications," in *IEEE 17th International Conference on Nanotechnology (IEEE-NANO)*, 2017, pp. 844–849.

[2] O. M. Awad, M. Mahmoud, I. Edo, A. H. Zadeh, C. Bannon, A. Jayarajan, G. Pekhimenko, and A. Moshovos, "Fpraker: A processing element for accelerating neural network training," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 857–869.

[3] Y. Bai, H. Wu, K. Wang, R. Wu, L. Song, T. Li, J. Wang, Z. Yu, and H. Qian, "Stacked 3d rram array with graphene/cnt as edge electrodes," *Scientific reports*, vol. 5, no. 1, pp. 1–9, 2015.

[4] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[5] F. Chen and H. Li, "Emat: an efficient multi-task architecture for transfer learning using reram," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–6.

[6] F. Chen, L. Song, and Y. Chen, "Regan: A pipelined reram-based accelerator for generative adversarial networks," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 178–183.

[7] F. Chen, L. Song, H. H. Li, and Y. Chen, "Zara: A novel zero-free dataflow accelerator for generative adversarial networks in 3d reram," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[8] H.-Y. Chen, S. Brivio, C.-C. Chang, J. Frascaroli, T.-H. Hou, B. Hudec, M. Liu, H. Lv, G. Molas, J. Sohn *et al.*, "Resistive random access memory (rram) technology: From material, device, selector, 3d integration to bottom-up fabrication," *Journal of Electroceramics*, vol. 39, no. 1, pp. 21–38, 2017.

[9] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH computer architecture news*, vol. 44, no. 3, pp. 367–379, 2016.

[10] 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.

[11] 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*, 2009, pp. 248–255.

[12] P. Dimitrakis, *Charge-trapping non-volatile memories*. Springer, 2015.

[13] M. Drumond, L. Coulon, A. Pourhabibi, A. C. Yüzügüler, B. Falsafi, and M. Jaggi, "Equinox: Training (for free) on a custom inference accelerator," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 421–433.

[14] Z. Fan, Z. Li, B. Li, Y. Chen, and H. Li, "Red: A reram-based deconvolution accelerator," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1763–1768.

[15] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 751–764.

[16] S. Hall, R. Schreiber, and S. Lie, "Training giant neural networks using weight streaming on cerebras wafer-scale systems," Cerebras Systems, Inc, 1237 E ARQUES AVE, SUNNYVALE, CA 94085 USA, Tech. Rep., 2021. [Online]. Available: https://www.cerebras.net/blog/scaling-up-and-out-training-massive-models-on-cerebras-systems-using-weight-streaming/

[17] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[19] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[20] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 802–815.

[21] 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.

[22] 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.

[23] L. Jiang, M. Kim, W. Wen, and D. Wang, "Xnor-pop: A processing-in-memory architecture for binary convolutional neural networks in wide-io2 drams," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[24] H. Kal, S. Lee, G. Ko, and W. W. Ro, "Space: locality-aware processing in heterogeneous memory for personalized recommendations," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 679–691.

[25] T. Kempen, R. Waser, and V. Rana, "50x endurance improvement in taox rram by extrinsic doping," in *2021 IEEE International Memory Workshop (IMW)*. IEEE, 2021, pp. 1–4.

[26] B. Kim, E. Hanson, and H. Li, "An efficient 3d reram convolution processor design for binarized weight networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 5, pp. 1600–1604, 2021.

[27] B. Kim and H. Li, "Leveraging 3d vertical rram to developing neuromorphic architecture for pattern classification," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2020, pp. 258–263.

[28] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 380–392, 2016.

[29] H. Kim, J. Sim, Y. Choi, and L.-S. Kim, "Nand-net: Minimizing computational complexity of in-memory processing for binary neural networks," in *2019 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2019, pp. 661–673.

[30] C. Kügeler, M. Meier, R. Rosezin, S. Gilles, and R. Waser, "High density 3d memory architecture based on the resistive switching effect," *Solid-state electronics*, vol. 53, no. 12, pp. 1287–1292, 2009.

[31] D. Kuzum, S. Yu, and H. P. Wong, "Synaptic electronics: materials, devices and applications," *Nanotechnology*, vol. 24, no. 38, p. 382001, 2013.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[33] H. Li, K.-S. Li, C.-H. Lin, J.-L. Hsu, W.-C. Chiu, M.-C. Chen, T.-T. Wu, J. Sohn, S. B. Eryilmaz, J.-M. Shieh *et al.*, "Four-layer 3d vertical rram integrated with finfet as a versatile computing unit for brain-inspired cognitive information processing," in *2016 IEEE Symposium on VLSI Technology*. IEEE, 2016, pp. 1–2.

[34] S. Li, D. Reddy, and B. Jacob, "A performance & power comparison of modern high-speed dram architectures," in *Proceedings of the International Symposium on Memory Systems*, 2018, pp. 341–353.

[35] P. Lin, C. Li, Z. Wang, Y. Li, H. Jiang, W. Song, M. Rao, Y. Zhuo, N. K. Upadhyay, M. Barnell *et al.*, "Three-dimensional memristor circuits as complex neural networks," *Nature Electronics*, vol. 3, no. 4, pp. 225–232, 2020.

[36] M. Mahmoud, I. Edo, A. H. Zadeh, O. M. Awad, G. Pekhimenko, J. Albericio, and A. Moshovos, "Tensordash: Exploiting sparsity to accelerate deep neural network training," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 781–795.

[37] H. Mao, M. Song, T. Li, Y. Dai, and J. Shu, "Lergan: A zero-free, low data movement and pim-based gan architecture," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 669–681.

[38] 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, vol. 40, no. 11, pp. 2306–2319, 2020.

[39] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, "Dnn+ neurosim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *2019 IEEE international electron devices meeting (IEDM)*. IEEE, 2019, pp. 32–5.

[40] 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.

[41] J. Y. Seok, S. J. Song, J. H. Yoon, K. J. Yoon, T. H. Park, D. E. Kwon, H. Lim, G. H. Kim, D. S. Jeong, and C. S. Hwang, "A review of three-dimensional resistive switching cross-bar array memories from the integration and materials property points of view," *Advanced Functional Materials*, vol. 24, no. 34, pp. 5316–5339, 2014.

[42] 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.

[43] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, G. De Micheli, and R. Drechsler, "Endurance management for resistive logic-in-memory computing architectures," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. Ieee, 2017, pp. 1092–1097.

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

[45] M. Sivan, Y. Li, H. Veluri, Y. Zhao, B. Tang, X. Wang, E. Zamburg, J. F. Leong, J. X. Niu, U. Chand *et al.*, "All wse2 1t1r resistive ram cell for future monolithic 3d embedded memory integration," *Nature communications*, vol. 10, no. 1, pp. 1–12, 2019.

[46] L. Song, F. Chen, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Accpar: Tensor partitioning for heterogeneous deep learning accelerators," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 342–355.

[47] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Hypar: Towards hybrid parallelism for deep learning accelerator array," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 56–68.

[48] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2017, pp. 541–552.

[49] S. Srinivasan, "Prefetching vs the memory system: Optimizations for multi-core server platforms," Ph.D. dissertation, 2007.

[50] C. Su, M. Huang, K. Lee, V. Hu, Y. Huang, B. Zheng, C. Yao, N. Lin, K. Kao, T. Hong *et al.*, "3d integration of vertical-stacking of mos 2 and si cmos featuring embedded 2t1r configuration demonstrated on full wafers," in *2020 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2020, pp. 12–2.

[51] P. Sun, N. Lu, L. Li, Y. Li, H. Wang, H. Lv, Q. Liu, S. Long, S. Liu, and M. Liu, "Thermal crosstalk in 3-dimensional rram crossbar array," *Scientific reports*, vol. 5, no. 1, pp. 1–9, 2015.

[52] W. Sun, S. Choi, B. Kim, and J. Park, "Three-dimensional (3d) vertical resistive random-access memory (vrram) synapses for neural network systems," *Materials*, vol. 12, no. 20, p. 3451, 2019.

[53] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[54] H. Tanaka, M. Kido, K. Yahashi, M. Oomura, R. Katsumata, M. Kito, Y. Fukuzumi, M. Sato, Y. Nagata, Y. Matsuoka *et al.*, "Bit cost scalable technology with punch and plug process for ultra high density flash memory," in *2007 IEEE Symposium on VLSI Technology*. IEEE, 2007, pp. 14–15.

[55] S. Venkataramani, V. Srinivasan, W. Wang, S. Sen, J. Zhang, A. Agrawal, M. Kar, S. Jain, A. Mannari, H. Tran *et al.*, "Rapid: Ai accelerator for ultra-low precision training and inference," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 153–166.

[56] C.-H. Wang, C. McClellan, Y. Shi, X. Zheng, V. Chen, M. Lanza, E. Pop, and H.-S. P. Wong, "3d monolithic stacked 1t1r cells using monolayer mos 2 fet and hbn rram fabricated at low (150° c) temperature," in *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2018, pp. 22–5.

[57] 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 *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[58] B. Yan, B. Li, X. Qiao, C.-X. Xue, M.-F. Chang, Y. Chen, and H. Li, "Resistive memory-based in-memory computing: From device and large-scale integration system perspectives," *Advanced Intelligent Systems*, vol. 1, no. 7, p. 1900068, 2019.

[59] D. Yang, A. Ghasemazar, X. Ren, M. Golub, G. Lemieux, and M. Lis, "Procrustes: a dataflow and accelerator for sparse deep neural network training," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 711–724.

[60] 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.

[61] 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.

[62] D. Yoo, Y. Song, J. Jang, W.-T. Hwang, S.-H. Jung, S. Hong, J.-K. Lee, and T. Lee, "Vertically stacked microscale organic nonvolatile memory devices toward three-dimensional high integration," *Organic Electronics*, vol. 21, pp. 198–202, 2015.

[63] M. Yu, Y. Cai, Z. Wang, Y. Fang, Y. Liu, Z. Yu, Y. Pan, Z. Zhang, J. Tan, X. Yang *et al.*, "Novel vertical 3d structure of taox-based rram with self-localized switching region by sidewall electrode oxidation," *Scientific reports*, vol. 6, no. 1, pp. 1–10, 2016.

[64] M. Yu, Y. Fang, Z. Wang, G. Chen, Y. Pan, X. Yang, M. Yin, Y. Yang, M. Li, Y. Cai *et al.*, "Encapsulation layer design and scalability in encapsulated vertical 3d rram," *Nanotechnology*, vol. 27, no. 20, p. 205202, 2016.

[65] S. Yu, "Neuro-inspired computing with emerging nonvolatile memorys," *Proceedings of the IEEE*, vol. 106, no. 2, pp. 260–285, 2018.

[66] S. Yu, W. Shim, X. Peng, and Y. Luo, "Rram for compute-in-memory: From inference to training," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2021.

[67] 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)*. IEEE, 2021, pp. 265–278.

[68] L. Zhang, S. Cosemans, D. J. Wouters, B. Govoreanu, G. Groeseneken, and M. Jurczak, "Analysis of vertical cross-point resistive memory (vrram) for 3d rram design," in *2013 5th IEEE International Memory Workshop*. IEEE, 2013, pp. 155–158.

[69] Y. Zhang, H. Li, H. Wang, H. Xie, R. Liu, S.-L. Zhang, and Z.-J. Qiu, "Thickness considerations of two-dimensional layered semiconductors for transistor applications," *Scientific reports*, vol. 6, no. 1, pp. 1–7, 2016.

[70] Q. Zheng, X. Li, Z. Wang, G. Sun, Y. Cai, R. Huang, Y. Chen, and H. Li, "Mobilattice: A depth-wise dcnn accelerator with hybrid digital/analog nonvolatile processing-in-memory block," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[71] Z. Zhu, J. Lin, M. Cheng, L. Xia, H. Sun, X. Chen, Y. Wang, and H. Yang, "Mixed size crossbar based rram cnn accelerator with overlapped mapping method," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[72] , "A simulation study of self-organization in neural network by spike-timing dependent plasticity," Ph.D. dissertation, , 2015.