CIMAT: A Compute-In-Memory Architecture for On-chip Training Based on Transpose SRAM Arrays

Hongwu Jiang, Xiaochen Peng, Shanshi Huang, and Shimeng Yu, *Senior Member, IEEE*

Abstract—Rapid development in deep neural networks (DNNs) is enabling many intelligent applications. However, on-chip training of DNNs is challenging due to the extensive computation and memory bandwidth requirements. To solve the bottleneck of the memory wall problem, compute-in-memory (CIM) approach exploits the analog computation along the bit line of the memory array thus significantly speeds up the vector-matrix multiplications. So far, most of the CIM-based architectures target at implementing inference engine for offline training only. In this article, we propose CIMAT, a CIM Architecture for Training. At the bitcell level, we design two versions of 7T and 8T transpose SRAM to implement bi-directional vector-to-matrix multiplication that is needed for feedforward (FF) and backprogpagation (BP). Moreover, we design the periphery circuitry, mapping strategy and the data flow for the BP process and weight update to support the on-chip training based on CIM. To further improve training performance, we explore the pipeline optimization of proposed architecture. We utilize the mature and advanced CMOS technology at 7 nm to design the CIMAT architecture with 7T/8T transpose SRAM array that supports bi-directional parallel read. We explore the 8-bit training performance of ImageNet on ResNet-18, showing that 7T-based design can achieve 3.38× higher energy efficiency (∼6.02 TOPS/W), 4.34× frame rate (∼4,020 fps) and only 50 percent chip size compared to the baseline architecture with conventional 6T SRAM array that supports row-by-row read only. The even better performance is obtained with 8T-based architecture, which can reach ∼10.79 TOPS/W and ∼48,335 fps with 74-percent chip area compared to the baseline.

Index Terms—SRAM, deep neural network, compute-in-memory, on-chip training

1 Introduction

Recently, DNNs have achieved remarkable improvement for a wide range of intelligence applications, from image classification to speech recognition an autonomous vehicle. The main elements of DNNs are convolutional layers and fully connected layers. To achieve incremental accuracy improvement, state-of-the-art DNNs tend to increase the depth and size of the neural network aggressively, which requires large amount of computational resources and memory storage for high-precision multiply-and-accumulate (MAC) operation. For example, Resnet-50 [1] can achieve ∼70 percent accuracy with 25.6M parameters. Although graphic processing units (GPUs) are the most popular hardware for DNN training at the cloud, there have been many efforts from academia and industry on the design of application specific integrated circuit (ASIC) accelerators for inference or even training on-chip. However, the memory wall problem remains in the conventional CMOS ASIC accelerators such as TPU [2] where the parameters (i.e., weights and activations) are stored in global buffer and the computation is still performed at the digital MAC arrays. Despite parallel computation with optimized data is realized across multiple processing elements (PE), the weights and intermediate data still require inefficient on-chip or off-chip memory access. This drawback is exacerbated for DNN training due to frequent back and forth data movement. To alleviate the memory access bottleneck, compute-in-memory (CIM) is a promising solution for DNN hardware acceleration. Weight movements can be eliminated by in-memory computing. CIM could also improve the parallelism within the memory array by activating multiple rows and use the analog readout to conduct multiplication and current summation. However, most of the proposed CIM architectures so far [3], [4], [5] could support the DNN inference only. Some efforts are made to accelerate training in CIM [6], [7], [8] but based on resistive random access memory (RRAM), however, the relatively large write latency/energy, and asymmetry and nonlinearity in the conductance tuning of RRAM prevent it from ideal candidates for extensive weight updates [9]. In addition, for backpropagation (BP) process in the DNN training, CIM array needs to perform convolution computation with transposed weight matrices. If the regular memory array with row-wise input and column-wise output is used, the parallel analog read-out could not be achieved when processing the transposed weight matrices. Instead, the sequential column by column read-out is required during BP, which significantly decreases the throughput and energy-efficiency. On the other hand, network compression is also a promising approach to reduce energy and area cost of the

• *The authors are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332. E-mail: {hjiang318, xpeng76, shuang406}@gatech.edu, shimeng.yu@ece.gatech.edu.*

Manuscript received 14 Oct. 2019; revised 25 Jan. 2020; accepted 1 Mar. 2020. Date of publication 13 Mar. 2020; date of current version 9 June 2020. (Corresponding author: Shimeng Yu.)
Recommended for acceptance by Xuehai Qian and Yanzhi Wang.
Digital Object Identifier no. 10.1109/TC.2020.2980533

storage. There have been many efforts in reducing the precision of parameters even to 1-bit during inference (e.g., BNN [10] and XNOR-net [11]). Due to the incremental accumulation in stochastic gradient descent (SGD) optimization, the precision demand and computational complexity for training is much higher than inference. Recently, discrete training techniques (e.g., DoReFa-Net [12] and WAGE [13]) are proposed to process both training and inference with low-bit-width parameters, which could be in favor of on-chip training compared to full precision in floating range.

In this paper, we propose a transpose SRAM-based *CIM Architecture for multi-bit precision DNN Training*, namely CIMAT, with two different bit-cell design, and explore the corresponding weight mapping strategies, data flow and pipeline design. SRAM is a mature CMOS technology and recent silicon prototype chips have demonstrated the efficacy of SRAM based CIM for inference only [14], [15], [16]. Therefore, the next step is to explore the architectural design for SRAM based CIM for training. Compared to other CIM architectures, we make the following key contributions:

1. *Novel hardware to support DNN training.* To support on-chip training, we present 7T and 8T transpose SRAM bit-cell design, which could perform bi-direction read access. In addition, we propose a near-memory hardware solution to perform weight update process.
2. *Transpose crossbar structure which can implement both feed-forward and backpropagation computation.* With the novel bit-cell design, we explored the mapping strategy and data flow of transpose crossbar structure to perform feed-forward and backpropagation on the same memory array.
3. *CIM solution for weight gradient calculation.* A CIM approach for matrix-to-matrix gradient calculation is proposed to improve the performance of CIM on-chip training.
4. *Pipeline optimization for CIM training with 7T and 8T SRAM design.* We propose a pipeline design to speed up forward and error calculation process with 7T SRAM design, respectively. To further improve energy efficiency and processing speed, we optimize the hardware with 8T SRAM design to pipeline the entire training process.

The rest of paper is organized as follows: Section 2 introduces the basics of the DNN training and CIM approach. Section 3 shows two novel transpose SRAM cell design and peripheral circuity to support the entire DNN training process. Section 4 presents the proposed CIM architecture, mapping strategies and data flow for all four steps of DNN training, namely feed-forward/inference, error calculation, weight gradient calculation and weight update. Section 5 shows pipeline design for 7T and 8T memory cell design, respectively. Especially, we optimize architecture for 8T design to accelerate pipeline, which obtains significant improvement on energy efficiency and throughput. Section 6 presents specifications of our proposed architecture to implement the ResNet-18 training. The chip-level evaluation is performed in the NeuroSim simulator [17] on 7 nm technology node. We also compare the performance of energy efficiency, frame rate and chip area between the proposed 7T/8T transpose SRAM based CIM array and a baseline design with conventional 6T SRAM array. Section 5 concludes the paper.

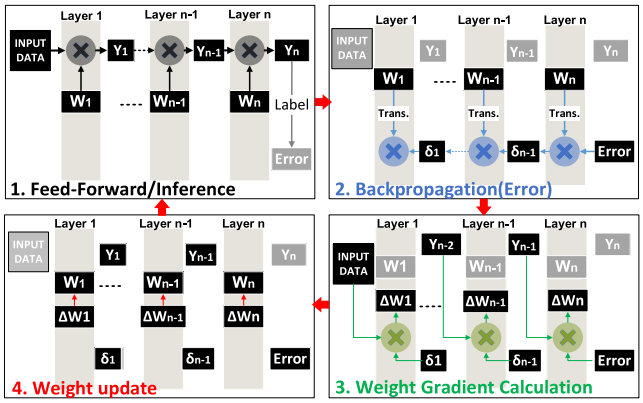


Fig. 1. Basic diagram of DNN training process.

This work is an extension of our prior conference paper [18]. The new materials added include 1) a more advanced 8T transpose design to perform forward and backward training pass simultaneously; 2) the pipeline optimization of CIMAT is also explored deeply to improve training performance.

2 Background

2.1 DNN Training Basics

Convolutional neural network (CNN) is one of the most popular DNN models. The training process of CNN could be divided into four steps, as shown in Fig. 1, namely, 1) feed-forward (FF), 2) BP for error calculation, 3) weight gradient (ΔW_n) calculation and 4) weight update. These four steps run in a loop to obtain a well-trained model through iterations.

In the FF process, it takes input data and calculates the error between the predicted output and the label (ground truth). The intermediate activations in each layer need to be stored into buffer to for later usage in step 3). This is different from the inference engine design where the intermediate activations could be discarded. For a given layer n, the FF operation is shown in Eq. (1),

$$Y_n = f(W_n Y_{n-1} + b_n), \tag{1}$$

where f is the neuron activation function such as ReLU, W_n is the weight of the current layer, Y_n is the output of the current layer, Y_{n-1} denotes the output from the previous layer, which acts as the activation to the current layer and b_n represents the bias.

During the BP process, the main goal is to calculate the gradient on the weights of each layer. A method based on stochastic gradient descent (SGD) is used to calculate gradient layer by layer from the back to the front. For a given layer n, with the chain rule, the error $\partial L/\partial Y_n$ is calculated by the convolution between the error and the "transpose" weight matrix W_{n+1}^T from the next layer, as shown in Eq. (2). Buffer is also needed for storing the error output in BP.

Then the weight gradient ΔW_n of the current layer (layer n) is obtained by another convolution between the error $\partial L/\partial Y_n$ and the activation Y_{n-1} that is obtained in the FF process, as shown in Eq. (3). Finally, the weights of the

current layer are updated by ΔW_n modulated by the learning rate (LR), as shown in Eq. (4).

$$\frac{\partial L}{\partial Y_n} = \frac{\partial L}{\partial Y_{n+1}} \cdot \frac{\partial Y_{n+1}}{\partial Y_n} = \frac{\partial L}{\partial Y_{n+1}} \cdot W_{n+1}^T \tag{2}$$

$$\Delta W_n = \frac{\partial L}{\partial W_n} = \frac{\partial L}{\partial Y_n} \cdot \frac{\partial Y_n}{\partial W_n} = \frac{\partial L}{\partial Y_n} \cdot Y_{n-1} \tag{3}$$

$$W_n^t = W_n^{t-1} - LR \times \frac{\partial L}{\partial W_n}. \tag{4}$$

As Eq. (2) suggested, the critical challenge to implement on-chip training is that the training hardware needs to support the read-out of the transposed weight matrix. Another issue is how to deal with massive intermediate data, which are generated during the entire training process.

2.2 Compute-in-Memory Basics

CIM, or in-memory computing, is an attractive solution for the extensive MAC operations in both DNN inference and training as it combines memory access and computation. In general, CIM architecture performs mixed-signal computation, i.e., analog current summation along the column/or the row, then the analog to digital conversion (ADC) at the edge of the array. Due to the increased parallelism and reduced data movement, CIM is expected to significantly improve the throughput and energy efficiency. As a trade-off, the limited precision of ADCs and their variations lead to approximate computation results in CIM, which generally result in a slight degradation of the inference accuracy [19].

SRAM has been considered as a mature candidate for CIM. The general approach is to modify the SRAM bit-cell and periphery to enable the parallel access. For example, the design in [3] expanded 6T cells into 8T to support bitwise XNOR. The VMM is done in a parallel fashion where the input vectors activate multiple rows and the dot-product is obtained as column voltage or current. Sense amplifier (SA) is also replaced by ADC to produce quantized output. Multi-bit inference [20] is possible with SRAM based CIM architectures.

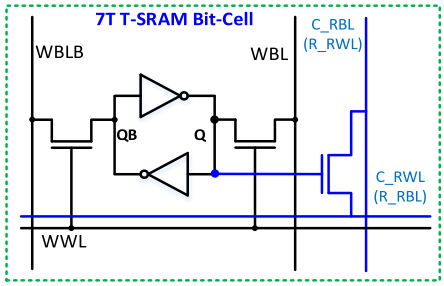
If using the FF only CIM design, when it processes the transpose weight matrix in BP, the input vector is applied to the column, and it is only possible to read-out the weights in a particular row and perform the summation in digital adders along that row. CIM could not be realized directly. Therefore, it is imperative to design a new CIM architecture to support both FF and BP calculation.

3 Hardware Support

3.1 Transpose Bit-Cell Design

3.1.1 7T Transpose SRAM Design

7T transpose SRAM is used as the bit-cell as shown in Fig. 2a. The feasibility of this 7T transpose SRAM has validated in silicon chips as in [21]. Such 7T design only has a very small cell area overhead, while providing bi-directional read access and read-disturb-free access. The regular 6T cell is used for the data storage and row-by-row write (controlled by WWL), while the innovation is on the additional transistor (in blue color) for bi-directional read access. As shown in Fig. 2b, this 7T transpose SRAM design has two read modes to support forward and backward process,



(a)

Mode	Column Read Wordline(C_RWL)/ Row Read Bitline(R_RBL)	Column Read Bitline(C_RBL)/ Row Read Wordline(R_RWL)
Forward	Activation input	Readout by Column
Backward	Readout by Row	Error input

(b)

Fig. 2. (a) 7T transpose SRAM bit cell; (b) Operation modes of 7T SRAM bit cell.

respectively. In forward mode, C_RWL is enabled as neuron input and C_RBL is used as bit line for partial sum read-out. For backward mode, these two lines exchange their roles: C_RWL acts as R_RBL, and C_RBL acts as R_RWL. R_RWL is enabled as neuron input and R_RBL is used as bit line for partial sum read-out. Both column and row paths have separate sets of WL writers and ADCs. The analog value of current along C_RBL/R_RBL represents the MAC results and this partial sum is digitized and quantized by the ADCs.

3.1.2 8T Transpose SRAM Design

Despite 7T transpose SRAM design could perform bi-direction read access to support both FF and BP calculation, the weight can be only stored and readout through Q point, which means FF and BP cannot perform simultaneously in same cell. In batch mode, 7T SRAM-based CIM could only realize pipiline in FF and BP seperately. To further improve processing speed and energy efficiency, we propose 8T T-SRAM bit-cell structure as shown in Fig. 3a. There is an additional PMOS transitor compared with 7T design. The gate of PMOS transistor is connected to QB to support read access from two sides of bit-cell. As shown in Fig. 3b, this 8T SRAM design also have two modes to support forward and backward process. In forward process, additional NMOS transistor in blue color is activated. C_RWL acts as activation input and C_RBL is used as bitline to collect analog current as partial sum readout. In backward mode, additional PMOS transistor in red color is activated. For BP calculation, error is fed into cell through R_RWL as input and R_RBL is used as bitline for partial sum readout. Since forward and backward mode have seperate wordline, bitline and storing point for read access, bi-direction read can be performed simultaneously. Thus, with 8T design, pipeline can be implemented between FF and error calculation to speedup training process further.

The 6-transistor part of 7T/8T design has the same structure as conventional 6T SRAM design which can be used for data storage and normal read/write is controlled by WWL as shown in Figs. 2 and 3. The compact foundry design rule

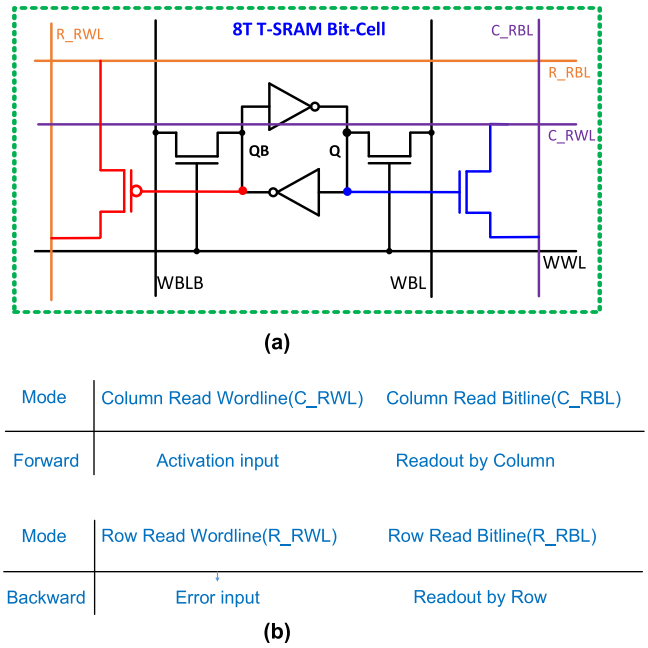


Fig. 3. (a) 8T transpose SRAM bit cell; (b) Operation mode of 8T SRAM bit cell.

could be applied there without any modification. The additional transistors implement bi-directional read access for in-memory computation. At this stage, these additional transistors could be added using logic design rule if foundry has not provided the optimized design rule.

3.2 Periphery Circuit Design for Training

As shown in Fig. 4, the transpose 7T/8T SRAM array has the typical periphery circuits for memory array, including word line (WL) writers for both column and row access, WL decoders for weight write and pre-charge circuit. In addition,

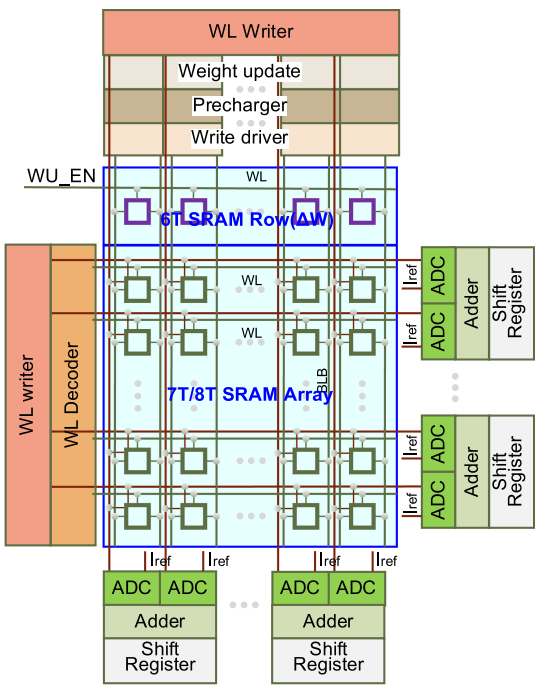


Fig. 4. Block diagram of transpose SRAM sub-array and periphery circuity.

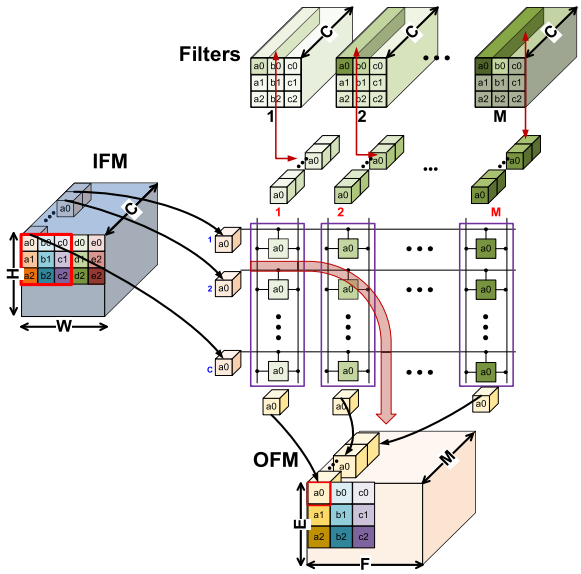


Fig. 5. Data flow of FF process showing one PE that implements a 0 element in the 3 × 3 filter only. (To implement all the elements in the filter, 9 PEs are needed.)

flash-ADCs (i.e., multilevel sense amplifiers with different references) are employed to quantize the partial sum. Shift and adder accumulates digitalized partial sum from least significant bit (LSB) input cycle to most significant bit (MSB) input cycle to support multi-bit input activation. In order to support bi-directional access, two groups of periphery circuits are needed. Besides the transpose SRAM bit-cell, there is also a row of 6T SRAM connected to the same BL/BLB, which is used for weight update inspired by [21].

4 Proposed Architecture

CIMAT, proposed for *C*ompute-*I*n-*M*emory *A*rchitecture for *T*raining, not only could perform the DNN inference but also could implement the BP calculation and weight update. Section 4.1 shows the mapping strategy of FF process. Section 4.2 presents the dataflow and approach to implement error calculation with the same memory array of FF process. The CIM solution of weight gradient calculation are proposed in Section 4.3. Finally, Section 4.4 introduces how to perform weight update with designed periphery circuit.

4.1 Feed-Forward Process

The FF process of the CIM array is shown in Fig. 5. Here M is the number of filters/output feature map (OFM) channels, C is number of input feature map (IFM)/filter channels, H/W is the IFM plane height/width, and E/F is the OFM plane height/width. The filters are unrolled to a weight matrix that is stored in the memory array. The IFM vector is applied to the row in parallel, and the OFM vector is obtained from the column in parallel. In general, there are two types of weight mapping schemes for CIM convolution. One scheme is to unroll all the elements in a filter (e.g., a 3 × 3 filter with 128 channels) to one long vector and put them into the same column (e.g., with a length of 1,152) in one PE [23]. The other scheme is to put each element in a filter to the same column in a subarray and use different PEs for different locations of the elements (e.g., 9 subarrays for 3 × 3 filter and each PE has

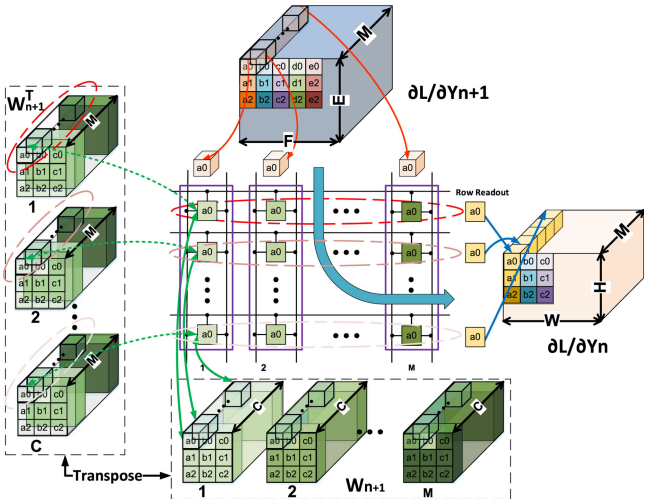


Fig. 6. Weight mapping for BP process (Eq. (2)) in the PE that implements a0 element in the 3 × 3 filter.

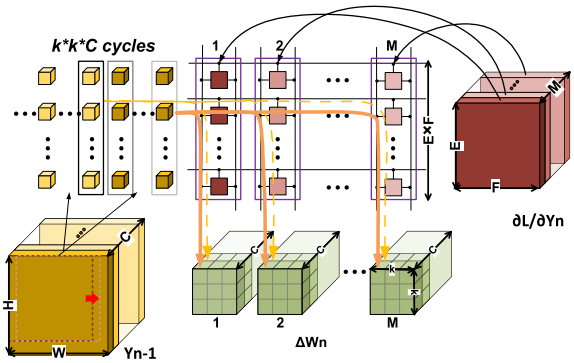


Fig. 7. Weight mapping of CIM approach for weight gradient calculation (Eq. (3)), error is stored in the 6T CIM SRAM array, while the activation Y_{n-1} is loaded as input from cycle to cycle. k = 3 for 3 × 3 filter.

a column length of 128), then sum up the partial sums from different PEs through an adder tree [24]. Although both schemes can be used for the transpose CIM architecture, the first one will make the forward and backward operation asymmetric because the column output in FF is directly the entire partial sum, while the row output in BP is just part of the partial sum. To keep the balance between FF and BP, we choose to use the second mapping scheme (as shown in Fig. 5) in our transpose CIM architecture. First, weights are pre-written into 7T/8T SRAM cells by write-wordlines (WWLs) in regular read mode. Then, for in-memory MAC operation, activations are first fed into SRAM cells through read-wordlines (RWLs). One-bit multiplication can be implemented by NAND gate. Thus, in computing mode, the read-out current from read-bitline (RBL) represents product of multiplication. The summed current along one column/row, namely partial sum, represents final value of MAC operation. It is called 'partial sum' since this value is only the summed dot-product from one kernel of the filter. To get the final output feature map, partial sums from different kernels need to be summed again. For the fully connected layers, they are treated as a special case of convolution layers with 1 × 1 filters, thus the same mapping scheme could be applied.

4.2 Backpropagation Process

Fig. 6 shows the details of error calculation in Eq. (2) for the first channel group (a0 element) of the filters. The backward pass for a convolution operation is also a convolution (but with spatially-flipped filters). For the FF, the product of input sliding window with the same filter across all the channels is summed up to generate one output, which means all dot products in same column of the PE is summed up. However, for the BP, the product of input sliding window and the same channel across different filters is summed up, which means all dot-products in same row need to be summed up. Essentially, we will process the transposed version of weight matrix in the BP. As shown in Fig. 6, W_{n+1} is the weights in FF while W_{n+1}^T is the transposed weights for BP, and they are mapped to the same memory array. In BP, the input vector (i.e., the error in the next layer $\partial L/\partial Y_{n+1}$) is applied to the column in parallel, and the output vector (i.e., the error in the current layer $\partial L/\partial Y_n$) is obtained from the row in parallel. With such transpose architecture, FF and error calculation can be performed within the same PE. No additional memory access is needed in error calculation, which means an improvement in throughput and energy efficiency.

4.3 Weight Gradient Calculation

For the calculation of weight gradient matrix ΔW_n in Eq. (3), we propose using CIM approach with additional 6T non-transpose CIM SRAM arrays to perform the outer dot product multiplication between error matrix $\partial L/\partial Y_n$ and related input activation matrix Y_{n-1}. The mapping method for ΔW_n calculation is shown in Fig. 7. $\partial L/\partial Y_n$ that is calculated in the previous step by Eq. (2) is written in 6T CIM SRAM array as weight first and then Y_{n-1} is loaded (from off-chip DRAM to on-chip buffer) as activation. Each plain of $\partial L/\partial Y_n$ is stretched into one long column which length equals E×F, the number of $\partial L/\partial Y_n$ channels is M, which means there are M columns in total. Thus, $\partial L/\partial Y_n$ can be mapped to a large weight matrix, whose height and width equal to E×F and M. Sliding windows on each plain of the input, Y_{n-1}, are also unrolled to a group of columns. The activation columns, whose length also equal to the height of weight matrix, are fed into the array cycle by cycle, thereby performing bitwise multiplication and accumulation. Partial sums from same column with multiple cycles generate all the weight gradients of one filter while partial sums from different columns form the entire gradient matrix ΔW_n. In the normal batch training mode, ΔW_n of each image is sent to off-chip DRAM for storing, and at the end of each batch, weight gradients are loaded back and accumulated on-chip. The averaged ΔW_n is used as input of weight update that is performed in periphery circuit of the array, as describe in the next sub-section.

4.4 Weight Update

Fig. 8 shows the structure of weigh update module. The 6T SRAM row is disabled during the FF and BP. When the accumulated weight gradients are ready after one batch, they are fed into shift register to realize multiplication with learning rate by shift (Eq. (4)) in a read-modify-write scheme that is done row-by-row. First, one row of the ΔW_n

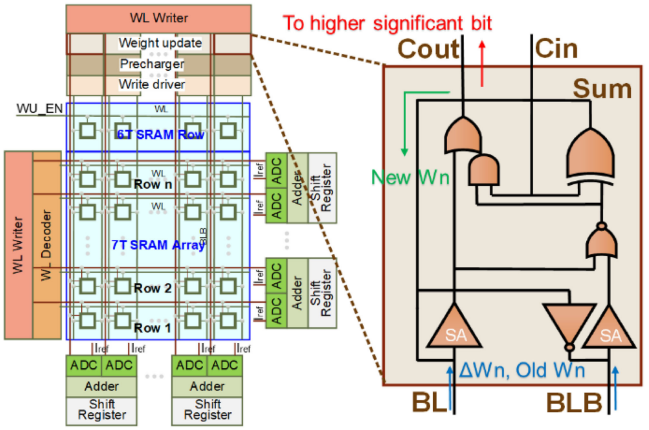


Fig. 8. The structure of weight update modules performing read-modify-write.

matrix is written into the 6T row of the sub-array. Then, WLs of 6T row and the to-be-updated weight row are activated simultaneously to send stored information to the weight update module. The weight update module has an adder to calculate the sum of ΔW_n and W_n using in-memory computing technique. This is to obtain the new W_n, which is then written back to the weight row and C_{out} of the adder is forwarded to the higher significant bit, which provides C_{in} for higher significant bit update. Since this process is repeated row by row in one PE, we could treat the update of different significant bits in a pipeline to speed up. The entire process of weight update is completed when all the multi-bit weights, from LSB to MSB, are updated.

5 Pipeline Design

5.1 7T SRAM Bit-Cell-Based Pipeline Design

The pipeline dataflow for FF and BP with 7T SRAM bit-cell design is shown in Fig. 9a. As an example of implementing ResNet-18, the forward pass and BP pass of error calculation are realized using 7-stage pipeline design. The latency of the 1st to 5th layers to process an entire image is almost the same while the total latency of 6th to 17th convolution layers is only half of the previous layers. Therefore, the 1st to 5th layers are treated as pipeline stage 1 to stage 5 respectively. To implement pipeline for all layers and match the

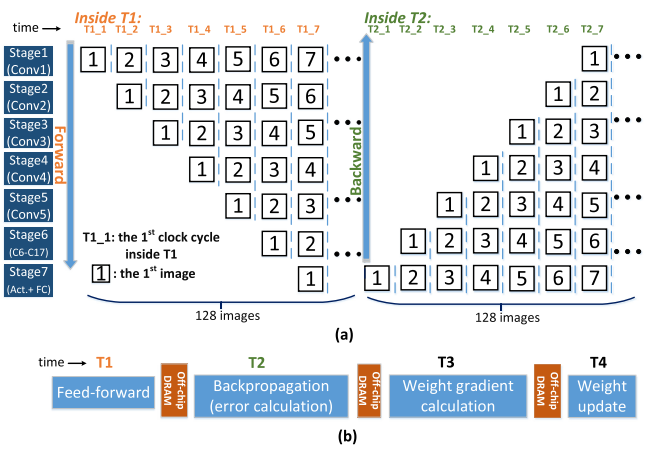


Fig. 9. (a) Intra-pipeline design inside FF and BP. (b) Training process in timeline of 7T-based architecture.

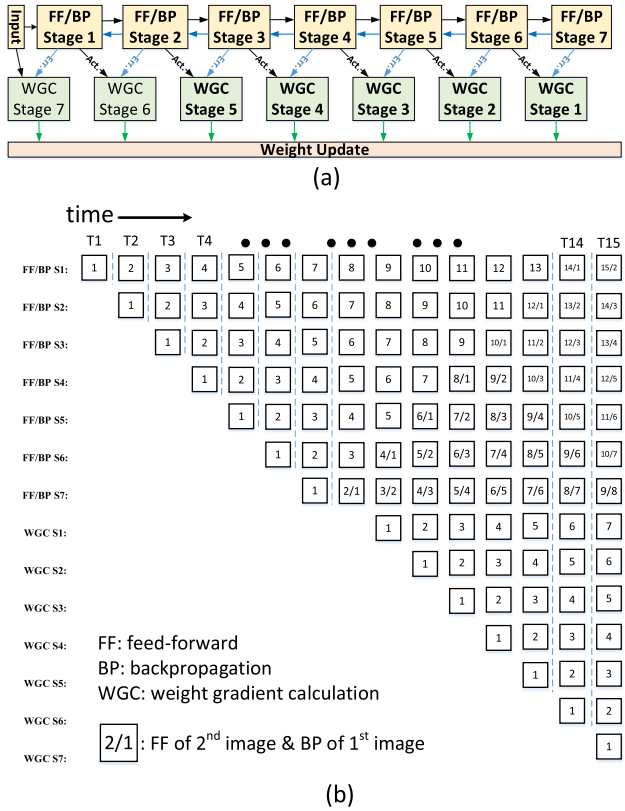


Fig. 10. (a) Pipeline structure of 8T SRAM-based CIM training. (b) The state of each stage as a function of time. 15 super clock cycles are illustrated.

latency of each stage, we group 6th to 17th layers as one stage (stage 6). Fully-connected layer and other activation function circuit form stage 7.

Fig. 9b shows the entire training process of 7T design in timeline. First, for FF process, one batch of images is fed into training system stage by stage in forward direction. After finishing FF process of one batch (T1), the error caulcation starts to operate stage by stage in backward direction (T2). The generated intermediate data in FF and BP process needs to be saved off-chip for gradient calculation since we will obtain hundred groups of activations and errors in the batch mode, which are too large to be saved on-chip. Typically, 128 images form one batch. For gradient calculation process, after the errors are obtained for one batch, they are used togehther with the activations to calculate the gradients (to be applied to the weights) image by image (T3), which means we will get 128 groups of weight gradients after 128 runs. The gradient calculation is performed after the batch FF and BP process. Finally, the 128 groups of weight gradients are averaged across the batch and the weights are updated in one step using this averaged weight gradients (T4).

5.2 8T SRAM Bit-Cell-Based Pipeline Design

As described in Section 3.1.2, 8T SRAM bit-cell can perform bi-directional read simultaneously, which means 8T based subarray is able to support bi-directional vector-to-matrix MAC calculation synchronously. Thus, we further optimize the pipeline design of 8T SRAM as shown in Fig. 10a. The stage configuration of FF and BP process is the same as 7T pipeline design. However, instead of waiting for the complete

of FF process, error calculation process of 8T design could form pipelines together with FF process to increase throughput significantly. In addition, as long as activation and error of the 1st image are ready, the weight gradient calculation (WGC) could start to work. The process of gradient calculation with CIM approach in Section 4.3 can be accelerated by duplicating CIM arrays. If the latency of WGC stage is approximately equals to the latency of FF/BP stage, gradient calculation could also work in a pipeline fashion together with FF and BP process avoiding off-chip ΔW_n movement. The state of each stage as a function of time is shown in Fig. 10b. For example, at the 14th cycle, FF/BP stage 7 is performing FF calculation of image 8 and error calculation of image 7 simutaneously. Meanwhile, WGC stage 1 is able to calculate weight gradient for image 6 because necessary activations and error of image 6 have been obtained since the 12th cycle and the 13th cycle, respectively.

Compared to intra-pipeline of 7T based architecutre, 8T design could also implement inter-pipeline between different training processes, which aggressively improves the throughput of training. Besides, the optimized pipeline is also beneficial to energy saving due to the reduced off-chip memory access and standby leakage of SRAM. The overhead of 8T based training architecture is on-chip buffer has to be large enough to run the pipeline.

6 Evaluation Results

We evaluate our CIMAT architecture design by implementing the ResNet-18 model for on-chip training on ImageNet. We first describe the experiments setup, and then present the simulation results of chip-level performance using modified NeuroSim [25]. NeuroSim is a circuit macro model that supports flexible CIM array design options with different device technologies (from SRAM to emerging nonvolatile memories) with various peripheral circuitry modules [15], which has been validated with SPICE simulations and actual device data.

6.1 Experiments Setup

Recent progresses in algorithms have proved that DNN training with 8 bits is sufficient to maintain the accuracy of for large-scale data set [11], [26], thus we use 8-bit weight and 8-bit activations in both CIMAT and baseline settings. To enable the bi-directional access, weight bits with different significance are stored on different tiles (i.e, 8 tiles) with shift-add to combine them together after obtaining their partial sums. The multi-bit activations are sent to the weight arrays from LSB to MSB using eight cycles and the output are accumulated with shift-add as well.

The batch size for training is 128, which means 128 images go through the FF/BP operations and the averaged weight gradients after 128 runs are used to update after the entire batch. The typical weight matrix size of one PE in ResNet-18 could be several hundreds up to thousands, but we use matrix partition technique and limit the subarray size to 128 × 128 considering the practical maximum SRAM array size when accessed in parallel [3]. Hence, 16 subarrays are formed into one PE and 9 PEs (corresponding to 3 × 3 filters) are formed into one tile, which means any layer in ResNet-18 can fit into 8 tiles (for 8-bit weight precision).

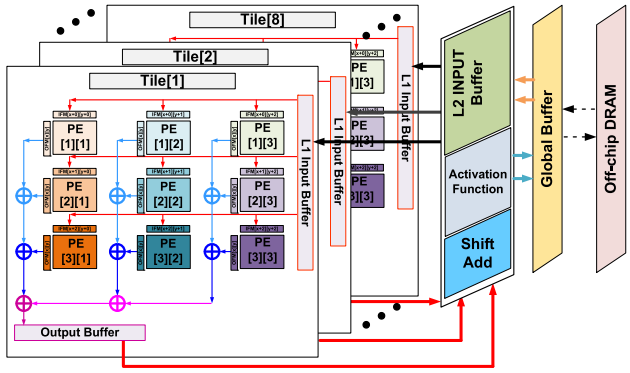


Fig. 11. Top-level CIMAT architecture for one convolution layer.

To simplify periphery circuit of transposable subarray, weight bits with different significance are stored on different tiles (i.e., 8 tiles) with shift-add to combine them together after obtaining their partial sums. Fig. 11 shows the top-level CIMAT architecture for one convolution layer, which contains 8 tiles, shift-add units, activation function units, L2 buffer and the global buffer. A tile contains multiple PEs, adder trees and L1 buffer. In both FF and BP, the multi-bit inputs are sent to the weight arrays from LSB to MSB using eight cycles and the output are accumulated with shift-add calculation. Then the adder trees accumulate results for all the subarrays in one PE for one element of the filter. Accumulated results from PEs in the same tile accumulate again through adder trees outside PE to finish bitwise MAC for the entire filter, but only for single significance of the weight. Outside tiles, shift-add will be performed again for eight tiles to obtain the eventual 8-bit output of OFM.

For the baseline, we use the regular 6T SRAM arrays to store weights and do near-memory computation (i.e., row-by-row read-out with digital adders at the edge of the array to accumulate the partial sum). For the BP, since non-transpose SRAM is used, each time when we read out a row, we obtain all the elements for one transpose filter. Then a group of 8-bit multipliers is used to get the products of the filter elements and the inputs. These products are accumulated by adders to get the final sum of error in Eq. (2). For weight gradient calculation in Eq. (3), the FF activations of each layer are first fetched from off-chip DRAM and then multipliers and adder trees are used to realize bitwise matrix-to-matrix multiplication and accumulation of the activation and error. All the weight gradients will be stored off-chip for weight update. Finally, ΔW_n for batch inputs are accumulated by adders and new weights are calculated by digital circuits and then written back to 6T SRAM arrays. We propose this near-memory computation solution as baseline to verify the advantages of CIM approach.

6.2 Circuit-Level Parameter

We model the CIMAT design according to state-of-the-art 7 nm high performance (HP) CMOS library and build the CIMAT architecture with a modified NeuroSim framework by adding the considerations of the on-chip buffer and off-chip DRAM access. Table 1 shows circuit-level parameters based on 7T SRAM design, including the hardware configuration, precision, area and energy for key circuit modules. The energy is given in energy per operation (or bit), for

TABLE 1
7T CIMAT Parameters

Block	Spec.		Area(um^2)	Energy
Technode: 7nm HP		***Input Precision: 8-bit***		
Subarray level (7T)				
7T SRAM Array	Size	128*128	143.41	0.22pJ/op
	Precision	1-bit		
ADC (MLSA)	Number	32	278.95	2.25pJ/op
	Precision	4-bit		
Shift-Add &Register	Number	32	174.34	8.35pJ/op
	Precision	11-bit		
Weight Update	Number	128	52.88	5.76pJ/op
Drivers, MUX and others			200.62	14.95pJ/op
Subarray Total			797.33	25.75pJ/op
PE level				
Subarray	Number	4*4	1.27E04	418.44pJ/op
Adder Tree	Number	128	2,865.37	6.51pJ/op
	Precision	13-bit		
L1 Buffer	Size	128*128	2,066.30	0.01pJ/bit
Output Buffer	Size	32*54	216.30	0.003pJ/bit
Tile level(357 tiles on chip)			***ResNet-18***	
PE	Number	3*3	1.62E05	3,795pJ/op
Adder Tree	Number	128	25,634	29.26pJ/bit
	Precision	17-bit		
L2 Buffer	Size	256*512	16,435	0.01pJ/bit
Output Buffer	Size	32*68	284.09	0.003pJ/bit
Global Buffer(7T)	Size	8MB	8.41E06	0.05pJ/bit

TABLE 2
8T CIMAT Parameters

Block	Spec.		Area(um^2)	Energy
Technode: 7nm HP		***Input Precision: 8-bit***		
Subarray level (8T)				
8T SRAM Array	Size	128*128	163.90	0.45pJ/op
	Precision	1-bit		
ADC (MLSA)	Number	32	278.95	2.25pJ/op
	Precision	4-bit		
Shift-Add &Register	Number	32	174.34	8.35pJ/op
	Precision	11-bit		
Weight Update	Number	128	52.88	5.76pJ/op
Drivers, MUX and others			200.62	14.95pJ/op
Subarray Total			817.82	25.98pJ/op
PE level				
Subarray	Number	4*4	1.30E04	422.12pJ/op
Adder Tree	Number	128	2,865.37	6.51pJ/op
	Precision	13-bit		
L1 Buffer	Size	256*128	4132.60	0.01pJ/bit
Output Buffer	Size	64*54	432.60	0.003pJ/bit
Tile level(357 tiles on chip)			***ResNet-18***	
PE	Number	3*3	1.68E05	3,828pJ/op
Adder Tree	Number	128	25,634	29.26pJ/bit
	Precision	17-bit		
L2 Buffer	Size	512*512	32,870	0.01pJ/bit
Output Buffer	Size	64*68	568.18	0.003pJ/bit
Global Buffer(8T)	Size	20MB	2.1E07	0.05pJ/bit

example, sub-array total energy is 25.75 pJ/op, which means the total energy for one single sub-array to do one vector-matrix multiplication, and L1 buffer energy is 0.006pJ/bit, which means total averaged energy to write one-bit data. The estimated energy cost of off-chip DRAM access is 4.2pJ/bit from prior work [27] assuming 3D high-bandwidth memory is used. As described in section 6.1, the multi-bit activations are sent to the weight arrays from LSB to MSB using eight cycles and the outputs are accumulated with shift-add. Calculated MAC value of LSB after ADC is 4-bit and first stored in register. Then, MAC value of MSB will be shifted and added to stored value in the register. The adder is designed to be 11-bit, which is enough to keep carry-bit information for each shift-add operation of 8 bit.

Intermediate data during training process is massive and hard to store completely in on-chip SRAM buffer. For 7T SRAM cell based CIMAT design, the global buffer size is 8 MB. To store massive generated intermediate data, in FF, activation outputs of each layer Y_n will be sent to off-chip DRAM for reuse in weight gradient calculation. The calculated errors of each layer for batch input also need to be stored off-chip for gradient calculation. After the weight gradient calculation for each image, ΔW_n will be stored off-chip DRAM for weight update. Without including energy cost of off-chip data transfer, energy efficiency could reach $\sim$20 TOPS/W. Considering DRAM access, energy efficiency decreases to $\sim$6 TOPS/W as shown in Table 3. To further improve energy efficiency and throughput, we proposed 8T SRAM cell based CIMAT design to implement inter-/intra-pipeline. The circuit-level paramters of 8T transpose SRAM based architecture are shown in Table 2. The 8T memory array is a little larger than 7T due to area overhead of additional transistors. For 8T based architecture, since FF calculation and error calculation can perform simultaneously, the L1, L2 and output buffer in tile is increased to support cocurrent bi-directional computation. Moreover, global buffer size also needs to be enlarged to support inter-pipeline among FF, error calculation and gradient calculation. As shown in table 2, global buffer of 8T design is 20 MB compared to 8 MB of 7T design. Such large buffer size is feasible as TPU uses 24 MB on-chip buffer [2]. Other hardware of 8T based chip remains similar to 7T based chip. Totally, we need 357 tiles to store all 8-bit weights from ResNet-18 networks. Despite 8T-based pipeline could reduce DRAM access during feedforward and error calculation, DRAM access during weight gradient calculation still cannot be eliminated. This observation can be proved by our evaluation as shown in Table 3. Compared to $\sim$6 TOPS/W of 7T design, 8T could achieve $\sim$10 TOPS/W. However, total energy efficiency is still limited by massive off-chip memory access while 55 TOPS/W can be reached without considering DRAM access.

6.3 Benchmark Evaluation

We compare the proposed CIMAT architectures with a baseline design with regular 6T SRAM array with row-by-row read. As described in 6.1, the baseline architecture is a

TABLE 3
Benchmark Results

Batch input(128 images)			ImageNet
Architecture	**TOPs/W**	**FPS**	**Area(mm^2)**
7T SRAM			
CIMAT_FF	28.11	50,585	67.83
CIMAT_BP	17.23	4,376	5.41
CIMAT_weight update	95.36	2.11E06	negligible
CIMAT Subtotal without DRAM Access	19.84	4,020	81.80
CIMAT_Total (7T)	**6.02**	**4,020**	**81.80**
8T SRAM			
CIMAT_FF	55.53	49,469	78.23
CIMAT_BP			43.28
CIMAT_weight update	95.36	2.11E06	negligible
CIMAT Subtotal without DRAM Access	55.37	48,335	121.51
CIMAT_Total (8T)	**10.79**	**48,335**	**121.51**
Baseline			
Baseline_FF	1.98	10,430	64.15
Baseline_BP	2.38	1,018	96.54
Baseline_weight update	197.25	9.60E07	1.79
Baseline Subtotal without DRAM Access	2.23	927	164.02
Baseline_Total	**1.78**	**927**	**164.02**

TOPS/W means total operation number per second per watt. FPS is the frame rate per second.

near-memory computation solution that places digital computation units at the edge of the memory while SRAM array only serves as weight storage unit. Table 3 shows the comparison of performance between 7T/8T SRAM based architecture and the baseline. The batch size for training is fixed to 128 ImageNet images per batch. The energy efficiency in terms of TOPS/W, the frame rate in terms of FPS and the chip area are evaluated for FF (Eq. (1)), BP (Eq. (2), (3)) and weight update (Eq. (4)), respectively. Here, BP performance includes both error calculation and gradient calculation.

According to Table 3, energy efficiency of CIMAT for both FF and BP process is much improved over that of baseline, which verifies our hypothesis that CIM architecture can minimize memory access for convolution computation. Besides, with the transpose SRAM design, area for BP process is much smaller than baseline due to shared CIM arrays and elimination of additional digital circuit for error calculation. For BP process, hardware of gradient calculation of 8T CIMAT has 8x area size of 7T design, which accelerates gradient calculation by duplicating CIM arrays to implement inter-pipeline in training. As shown in Table 3, compared to near-memory computation baseline, 7T SRAM based CIMAT training architecture can achieve overall ~4.34× speed up and ~3.38× improvement in energy efficiency with ~0.5× chip area while 8T based CIMAT can further increase energy efficiency and throughput by ~6.06× and ~52.14×, respectively with 0.74× chip area.

Figure 12 compares the performance and energy efficiency of CIMAT with the GPU-based implementation, proposed near-memory computation baseline and Neural Cache [28] which is a state-of-the-art hardware accelerating DNN inference using SRAM-based in-memory computing architecture. For GPU platform, the experiments are performed using Pytorch running on NVidia Titan RTX. We use NVidia-SMI tool for GPU power measurement. For Neural Cache, we used 28 TOPS and 52.92W average power

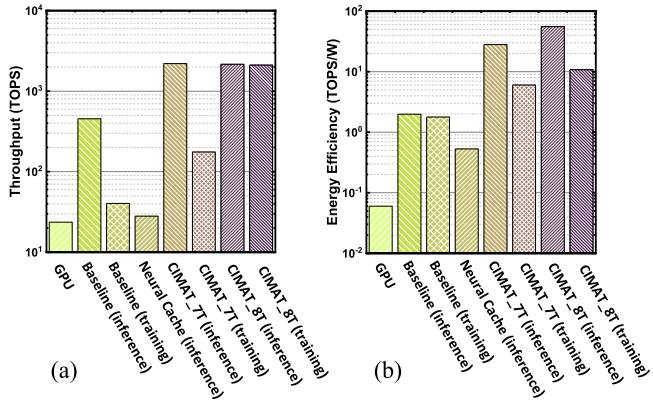


Fig. 12. Performance benchmark: (a) Throughput comparison. (b) Energy efficiency comparison.

at 22nm technology as reported in the reference paper [28]. Our evaluation shows that CIMAT 7T design can achieve on average 7.4× speedup and 100× energy efficiency in training as compared to GPU-based approach. Compared to Neural Cache, CIMAT provides 78.7× speedup and 53x higher energy efficiency in feed-forward and could support training instead of inference only. Figure 12 also compares CIMAT training performance over inference with 7T and 8T structure. Our evaluation shows that training could only achieve 0.25× energy efficiency over inference due to massive off-chip intermediate data access during training. For throughput, 8T design could provides almost same speed for both inference and training while the training speed of 7T design is limited to less than 1/10 of inference. The higher throughput of 8T design comes from the inter-pipeline between training processes as shown in Section 5.2.

The TPU v3 (for training) energy efficiency is estimated to be approximately ~0.45TOPS/W [29]. Compared to other custom designed architecture for training, for example, the TIME [7] obtains ~5.72 TOPS/W by reducing tunning cost of RRAM with look up table policy. The energy efficiency of Pipelayer [8] is only ~0.14 TOPS/W since all intermediate data is written to RRAM arrays. In this work, we employed the fast-write SRAM technology, 7T design achieves 6.02 TOPS/W and 8T design achieves 10.79 TOPS/W, showing the benefits of our proposed CIMAT architecture.

7 Related Work

Since training of DNNs that involves BP and weight update is more complicated, most prior designs only support inference. A limited number of works present techniques to implement DNN training. We now discuss these works. Song *et al.* [8] presented a RRAM-based design named PipeLayer. They separate the RRAM-based memory into two types: morphable subarrays and memory array. The results of FF process are stored in memory array for future backward computation and morphable array is used as both compute unit and storage. Their design exploits both intra- and inter-layer parallelism to implement pipelined training. Compared to GPU, their design could achieve improvement in speed and energy efficiency. Imani *et al.* [30] proposed FloatPIM, a CIM-based DNN training architecture that exploits analog properties of the memory without converting

data into the analog domain. FloatPIM could support both floating-point and fixed-point precision. To overcome the internal data movement issue, the author design a switch to enable in-parallel data transfer between the neighboring blocks. The evaluation result shows that FloatPIM can achieve on average 6.3× and 21.6× higher speedup and energy efficiency as compared to PipeLayer. Fujiki *et al*. [31] proposed Duality Cache, developing a single instruction multiple thread (SIMT) architecture by enabling in-situ floating point and transcendental functions to improve data-parallel acceleration. Their design could improve performance by 3.6x and 4.0x for GPU benchmarks and OpenACC benchmarks respectively.

8 Conclusion

We propose a SRAM-based Compute-In-Memory Training Architecture, namely CIMAT, which can maximize hardware reuse with the transpose array based on novel 7T/8T bit-cell design. A new CIM solution for error calculation is proposed with low hardware overhead. We focus on ResNet-18 impelementation in this work, but our proposed methodologies could be applied to implement other DNN models.

The experiment results show that, 7T SRAM based CIMAT can achieve 3.38× higher energy efficiency (∼6.02 TOPS/W), 4.34× frame rate (∼4,020 fps) and only 50 percent chip size (∼81.80 mm^2) compared to the baseline architecture with conventional 6T SRAM array that supports row-by-row read only. With more advanced 8T bit cell and optimized pipeline design, 8T SRAM based CIMAT can further achieves more energy saving (∼10.79 TOPS/W) and aggressively more than 10× throughput (48,335 fps) with tolerable area overhead (121.51 mm^2) compared to 7T CIMAT. Our results reveal that CIM is a promising solution to implement on-chip DNN training, which can reduce off-chip talk significantly. Limited on-chip buffer will be a constraint for pipeline implementation of deeper neural networks. Possible replacement of SRAM buffer with larger but slower RRAM buffer is worthy of future exploration. Another limitation for CIM approach is that extreme large DNN model is hard to be fully stored on-chip with today's silicon technology. On one side, CIM approach is more attractive to edge device due to much improved energy efficiency. From the algorithm's perspective, there are many efforts on developing small networks for edge AI application. Besides, advanced algorithmic methods like transfer learning also help reduce the load for training. From the hardware perspective, 5nm process [32] (512 Mb = 64 MB) SRAM cache is available from the foundry recently, which could increase capacity of CIM solution for larger networks in the near future considering the possible scaling to 3nm node.

Acknowledgments

This work was supported in part by Samsung GRO program.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[2] N.P. Jouppi *et al*., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2017, pp. 1–12.
[3] L. Rui *et al*., "Parallelizing SRAM arrays with customized bit-cell for binary neural networks," in *Proc. IEEE/ACM Design Autom. Conf.*, 2018, Art. no. 21.
[4] P. Chi *et al*., "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2016, pp. 27–39.
[5] A. Shafiee *et al*., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2016, pp. 14–26.
[6] B. Li, L. Song, F. Chen, X. Qian, Y. Chen, and H. H. Li, "ReRAM-based accelerator for deep learning," in *Proc. ACM/IEEE Des. Autom. Test Europe Conf.*, 2018, pp. 815–820.
[7] M. Chen *et al*., "Time: A training-in-memory architecture for memristor-based deep neural networks," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2017, Art. no. 26.
[8] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 541–552.
[9] X. Sun and S. Yu, "Impact of non-ideal characteristics of resistive synaptic devices on implementing convolutional neural networks," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 3, pp. 570–579, Sep. 2019.
[10] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.
[11] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.
[12] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016. *arXiv:1606.06160*.
[13] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," in *Proc. Int. Conf. Learn. Representations*, 2018.
[14] Z. Jiang, S. Yin, M. Seok, and J.S. Seo "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," in *Proc. IEEE Symp. VLSI Circuits*, 2018, pp. 173–174.
[15] W. S. Khwa *et al*., "A 65nm 4Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processors," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2018, pp. 496–498.
[16] J. Zhang, Z. Wang, and N. Verma, "A machine-learning classifier implemented in a standard 6T SRAM array," in *Proc. IEEE Symp. VLSI Circuits*, 2016, pp. 1–2.
[17] P. Y. Chen, X. Peng, and S. Yu, "NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 12, pp. 3067–3080, Dec. 2018.
[18] H. Jiang, X. Peng, S. Huang and S. Yu, "CIMAT: A transpose SRAM-based compute-in-memory architecture for deep neural network on-chip training," in *Proc. ACM Int. Symp. Memory Syst.*, 2019, pp. 490–496.
[19] X. Sun, S. Yin, X. Peng, R. Liu, J. S. Seo, and S. Yu, "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks," in *Proc. ACM/IEEE Des. Autom. Test Europe Conf.*, 2018, pp. 1423–1428.
[20] X. Si *et al*., "A twin-8T SRAM computation-in-memory macro for multiple-bit CNN-based machine learning," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2019, pp. 396–398.
[21] K. Bong, S. Choi, C. Kim, S. Kang, Y. Kim, and H. J. Yoo, "A 0.62 mW ultra-low-power convolutional-neural-network face-recognition processor and a CIS integrated with always-on haar-like face detector," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2017, pp. 248–249.
[22] J. Wang *et al*., "A compute SRAM with bit-serial integer/floating-point operations for programmable in-memory vector acceleration," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2019, pp. 224–226.
[23] T. Gokmen and Y. Vlasov, "Training deep convolutional neural networks with resistive cross-point devices," *Front. Neurosci.*, vol. 11, 2017, Art. no. 538.
[24] X. Peng, R. Liu, and S. Yu, "Optimizing weight mapping and data flow for convolutional neural networks on RRAM based processing-in-memory architecture," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2019, pp. 1–5.
[25] [Online]. Available: https://github.com/neurosim/DNN_NeuroSim_V1.0
[26] R. Banner *et al*., "Scalable methods for 8-bit training of neural networks," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 5151–5159.

[27] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and efficient neural network acceleration with 3D memory," in *Proc. ACM Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2017, pp. 751–764.
[28] C. Eckert *et al.*, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2018, pp. 383–396.
[29] P. Teich. [Online]. Available: https://www.nextplatform.com/2018/05/10/tearing-apart-googles-tpu-3-0-ai-coprocessor/
[30] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM:: In-memory acceleration of deep neural network training with high precision," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2019, pp. 802–815.
[31] D. Fujiki, S. Mahlke, and R. Das, "Duality cache for data parallel acceleration," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2019, pp. 397–410.
[32] G. Yeap *et al.*, "5nm CMOS production technology platform featuring full-fledged EUV, and high mobility channel FinFETs with densest 0.021um^2 SRAM cells for mobile SoC and high performance computing applications," *IEEE Int. Electron Devices Meeting*, 2019, pp. 36.7.1–36.7.4.

Hongwu Jiang received the BS degree from the Dalian University of technology, in 2012, and the MS degree in electrical engineering from Arizona State University, in 2014, he is currently working towards the PhD degree in electrical and computer engineering at the Georgia Institute of Technology in Atlanta, Georgia. His research interests include SRAM-/eNVM- based hardware architecture and accelerator design of deep learning.

Xiaochen Peng received the BS degree in automatic system from the Hefei University of Technology, in 2014, and the MS degree in electrical engineering from Arizona State University, in 2016, she is currently working toward the PhD degree in electrical and computer engineering at the Georgia Institute of Technology in Atlanta, Georgia. Her research interests include development of device-to-system benchmarking framework for machine learning accelerators, and design of emerging-device-based hardware implementation for neural networks.

Shanshi Huang received the BS degree in communication engineering from the Beijing Institute of Technology, in 2012, and the MS degree in electrical engineering from Arizona State University, in 2014, she is currently working toward the PhD degree in electrical and computer engineering at the Georgia Institute of Technology in Atlanta, Georgia. Her current research interests include deep learning algorithm & hardware co-design and deep learning security.

Shimeng Yu (Senior Member, IEEE) received the BS degree in microelectronics from Peking University, Beijing, China, in 2009, and the MS and PhD degrees in electrical engineering from Stanford University, Stanford, California, in 2011 and in 2013, respectively. He is currently an associate professor of electrical and computer engineering at the Georgia Institute of Technology in Atlanta, Georgia. From 2013 to 2018, he was an assistant professor of electrical and computer engineering at Arizona State University, Tempe, Arizona. His research interests are nanoelectronic devices and circuits for energy-efficient computing systems. His expertise is on the emerging non-volatile memories (e.g., RRAM, ferroelectrics) for different applications, such as machine/deep learning accelerator, neuromorphic computing, monolithic 3D integration, and hardware security, etc. He was a recipient of the NSF Faculty Early CAREER Award, in 2016, the IEEE Electron Devices Society (EDS) Early Career Award, in 2017, the ACM Special Interests Group on Design Automation (SIGDA) Outstanding New Faculty Award, in 2018, the Semiconductor Research Corporation (SRC) Young Faculty Award, etc.