A Two-way SRAM Array based Accelerator for Deep Neural Network On-chip Training

Hongwu Jiang[1], Shanshi Huang[1], Xiaochen Peng[1], Jian-Wei Su[2], Yen-Chi Chou[2], Wei-Hsing Huang[2], Ta-Wei Liu[2], Ruhui Liu[2], Meng-Fan Chang[2], and Shimeng Yu[1*]
[1]Georgia Institute of Technology, Atlanta, GA, USA [*]E-mail: shimeng.yu@ece.gatech.edu
[2]National Tsing Hua University, Hsinchu, Taiwan

ABSTRACT

On-chip training of large-scale deep neural networks (DNNs) is challenging due to computational complexity and resource limitation. Compute-in-memory (CIM) architecture exploits the analog computation inside the memory array to speed up the vector-matrix multiplication (VMM) and alleviate the memory bottleneck. However, existing CIM prototype chips, in particular, SRAM-based accelerators target at implementing low-precision inference engine only. In this work, we propose a two-way SRAM array design that could perform bi-directional in-memory VMM with minimum hardware overhead. A novel solution of signed number multiplication is also proposed to handle the negative input in backpropagation. We taped-out and validated proposed two-way SRAM array design in TSMC 28nm process. Based on the silicon measurement data on CIM macro, we explore the hardware performance for the entire architecture for DNN on-chip training. The experimental data shows that proposed accelerator can achieve energy efficiency of ~3.2 TOPS/W, >1000 FPS and >300 FPS for ResNet and DenseNet training on ImageNet, respectively.

1 INTRODUCTION

In the recent years, deep neural networks (DNNs) become deeper and more complex, number of operations and parameters also increase significantly. Consequently, data movements between computing and storage units become the bottleneck of conventional von Neumann architectures, known as memory wall. Many application specific integrated circuit (ASIC) accelerators are proposed to overcome the memory bottleneck. However, the memory wall problem still remains in most of the accelerators such as Eyeriss [1] and TPU [2] where the parameters are stored in embedded memory for data reuse and the computation is still performed at the digital multiply-and-cumulate (MAC) units. Although optimized dataflow and paradigm could speed up computation across multiple processing elements (PEs), the weights and intermediate data still require inefficient on-chip or off-chip memory access in a row-by-row fashion.

To overcome this problem, network compression and compute-in-memory (CIM) are two promising solutions to reduce high computational cost of DNN. There have been many efforts in reducing the network size during inference (e.g. BNN [3] and XNOR-Net [4]). On the other hand, CIM is an efficient paradigm to address the memory wall problem [5]. Convolution operation essentially contains VMM, which occupies most of the computations in DNNs. The crossbar CIM structure could implement VMM operations by activating multiple rows and summing current along columns in analog domain.

The common application scenario of DNN is to perform training in cloud and implement inference on edge device. However, several limitations exist for this scenario. First, on-device inference of DNN models, which are pre-trained with global dataset, may lead to degraded performance due to local environment. Second, if new scenes are captured in the field, the pre-trained model could not respond to the new scenes. Therefore, an adaptive and continuous learning mode is preferred for the edge device. One solution is to send the newly captured data back to the cloud for the model updates. However, this may burn additional power on data communications (e.g. over wireless network) and increase the burden of the battery lifetime for edge device. Besides, privacy and security also raise concerns to share the personal data to the cloud for retraining purpose. Therefore, exploiting on-device learning (or on-chip training) has become desired and essential.

However, several challenges exist in designing a practical accelerator that supports both training and inference. First, most of the current low-precision techniques could maintain high accuracy only for moderate-sized network, which is not scalable to more advanced network like ResNet [6] and DenseNet [7]. Second, most of the CIM training solution still based on architecture-level design without chip-level validation. The feasibility of CIM array for training needs to be proven in silicon first. Third, dataflow for in-memory training is largely unexplored as most prior work focused on in-memory inference. One critical problem is how to implement signed number multiplication during backpropagation as negative inputs are involved. This issue deserves deeply explored since CIM performs mixed-signal computation instead of pure digital. Furthermore, the impact of analog-to-digital converter (ADC) resolution for CIM is rarely studied on training performance.

In this paper, we propose a SRAM-based CIM accelerator for DNN training, and evaluate the hardware performance based on several large DNN models with experimental data. SRAM is mature in manufacturing and faster/less energy in write operations

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

(than emerging non-volatile memories), which benefit the write-intensive on-chip training. Compared to prior works, we make the following contributions.

- We design a two-way SRAM array to support both row-wise and column-wise computation. With bi-directional access, forward and backward propagation calculation can be implemented in the same array, improving the area efficiency over the prior design [8] that uses two copies of arrays.
- During DNN training, both errors and weights can be negative value in backpropagation. We propose an efficient solution to implement in-memory signed multiplication, which perform unsigned VMM in memory first and then multiple rounds of shift-add or invert operations in the periphery.
- We build our accelerator and optimize the workflow for DNN in-memory training, which can perform feed-forward calculation, error calculation and gradient calculation within the same proposed SRAM array design. The impact of ADC quantization loss is also explored for DNN training.
- The proposed two-way SRAM array macro is fabricated and validated in TSMC 28nm process. According to experimental results, our accelerator could achieve ~3.2 TOPS/W on ImageNet dataset training.

2 PRELIMINARY

2.1 Low-precision DNN training

Fig. 1 shows the generic diagram of training process in hardware, which could be divided into four steps, namely, 1) feed-forward (FF), 2) error calculation (EC), 3) gradient (ΔW) calculation (GC) and 4) weight update (WU). Theoretically, the above four steps run in a loop through batch based training, in which steps 1) to 3) occur every input cycle within the batch but step 4) occurs only once at end of the batch. Weight matrix (WM) is utilized in FF process while transpose weight matrix (TWM) is employed in EC process.

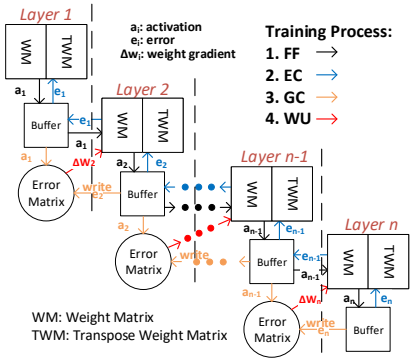


Figure 1: Generic diagram of DNN training.

Processors such as GPUs and TPUs that are widely used in servers for DNN online training, support 64-bit floating-point or 16-bit fixed-point operations. However, for edge device, area overhead and energy cost will be significant if such high precision is used. Thus, many efforts are made to reduce computational cost by parameter quantization. Several methods have been proposed to achieve low bit precision training. BNN [3] and XNOR-Net [4] proved the successful of training with 1-bit weight in FF but all other parameters are still floating-point. DoReFa-Net [9] further used low gradient precision to eliminate the high precision copy needed for GC, showing that 1b-weight, 2b-activation and 6b-gradient is sufficient for training. Despite DoReFa-Net reported low precision gradient, its errors are still floating-point. WAGE [10] further introduced quantization in EC to realize low-precision training with 2b-weight, 8b-activation, 8b-gradient and 8b-error. While these methods work pretty well on small/moderate sized network for MNIST and CIFAR-10 datasets, they all show noticeable accuracy drop for large networks on ImageNet dataset. Most of the results are reported so far are for the simple AlexNet and VGG-16, which are large in terms of parameters but relatively low in accuracy. In [11], authors show that 16b-gradient could achieve little accuracy loss on ResNet-18&34. Recently, a newer version of WAGEBU [12] also claims that high precision gradient is needed for scalability. Based on these observations, we choose to use 8b-weight, 8b-activation, 8b-error and 16b-gradient as a safe choice for the low precision scheme for our on-chip training accelerator design.

2.2 SRAM-based CIM

In principle, CIM could be implemented by different device technologies. Emerging non-volatile memories (eNVMs) such as phase change memory (PCM) [13] and resistive random-access memory (RRAM) [14] provide promising solutions due to small cell size. However, grand challenges of eNVMs including asymmetric conductance tuning, low endurance and large write energy/latency limit the on-chip training accuracy [15]. 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 [16] 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 partial sum output. The feasibility of SRAM based CIM is proven by several fabricated prototype chips [17-20] for small to moderate size networks. However, most of these chips is at macro-level for a single array, targeting at inference only, without architectural optimizations of data quantization, pipeline and weight mapping strategy for training. Previous CIM training architectures [8, 21], store WM and TWM with two separate memory arrays to implement FF and EC process, respectively. CIM approach for gradient calculation is also shown in prior work with additional CIM arrays [22]. In our design, we will reuse SRAM arrays for all three processes, namely, FF, EC and GC with no additional hardware cost. The details are described in the following section.

3 CIM ACCELERATOR DESIGN

3.1 Design of Two-way SRAM Array

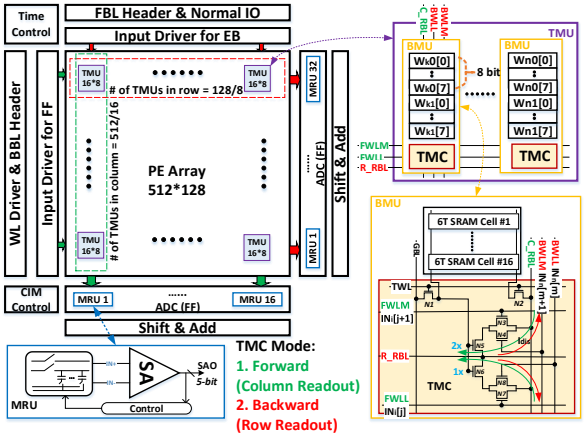
Fig. 2 (a) presents the circuit diagram of two-way CIM SRAM array. One single array comprises 16×32 transpose-multiply-units (TMUs), 16 multibit-readout-units (MRUs) for FF process, 32 MRUs for error calculation, input drivers for FF and EC, and other peripheral circuits. Each TMU has 8 bitwise-multiply-units (BMUs) across 8 columns, in which two 8-bit weights is stored in the 16 SRAM cells on the same column. Each BMU includes 1 transpose-multiply-cell (TMC) and 16 regular 6T SRAM cells.

Each TMC consists of 10 transistors including 2 pass-gate transistors (N1, N2) and 2 multiply branches (N3-N5 and N6-N8). In standby mode, all N1/N2 are on and the global BLs charge local BLs/BLBs to a precharge voltage. As shown in Fig. 2 (a), TMC has two read modes to support bi-directional bitwise multiplication, namely forward mode and backward mode. In forward mode, row-read-bitlines (R_RBLs) are at 0V. In each input cycle, a 2-bit input is applied to the gates of N3 (FWLM) and N7 (FWLL), corresponding to $IN_i[j+1]$ and $IN_i[j]$, respectively. When a wordline (WL) is activated, the weight stored in the selected 6T SRAM is read out, multiplying 2-bit input through 2 multiply branches at local BL. N3-N5 pair and N6-N8 pair corresponds to more-significant-bit (MSB) multiplication and less-significant-bit (LSB) multiplication of input, respectively. The transistor width of N3-N5 is designed to be twice of N6-N8, which enables the N3-N5 pair to produce discharge current twice of the N6-N8 branch. When the two currents of both N3-N5 and N6-N7 pairs are summed up, the voltage swing (ΔV) contributed by the TMC cell to column-read-bitline (C_RBL) is proportional to the result of multiplying 2-bit input and 1-bit weight. In backward mode, different significant inputs, $IN_n[m+1]$ and $IN_n[m]$, are fetched into BWLM and BWLL and C_RBL is set at 0V. The N4-N5 pair in TMC performs the bitwise multiplication of $IN_n[m+1]$ × W, while the N6-N8 pair performs $IN_n[m]$ × W. Then the two discharge currents from N4-N5 and N6-N8 branches are combined, the voltage swing contributed by TWC cell to R_RBL is proportional to the multiplication result of column input and stored weight. All the TMUs in the same column share the same C_RBL while all the TMUs in the same row share the same R_RBL.

Fig. 2 (b) lists the truth table with possible combination patterns of 2-bit input×1-bit weight and the corresponding multiplication results. The difference of ΔV value on R_RBL/C_RBL reflects the product of 2-bit input and binary weight effectively. For FF, forward mode is activated: 16 selected vertical TMCs from same column are grouped and all the discharge currents from same group is summed as a MAC operation. The generated analog voltage V_{C_RBL}, representing partial sum of 2-bit input multiplication with 1-bit weight, is quantized to digital output by MRU. Final sum of multibit input multiplication with multibit weight can be obtained by shift-adding the partial sum of different significant bits of weight and input successively. For EC, 16 selected horizontal TMCs from same row are activated in backward mode. Each R_RBL accumulates products from TMUs in the same row and generate analog sum voltage V_{R_RBL}, which is then read out by MRU. The MRU basic structure is built with low-offset sense amplifier (SA), control block and capacitor sharing unit as shown in Fig. 2 (a). MRUs repeats SA operation with 5 reference voltages through 5 cycles to generate a 5-bit digitalized output, effectively serving as 5-bit ADC.

3.2 Signed Number Multiplication

For the mathematical calculation in hardware, data is usually represented by 2's complement code for digital calculation. In CIM, dot-products are summed up in analog current (which is always positive from power supply to ground). Thus the multiplication is true only when input and weight are both positive. Signed number multiplication may not be a concern for inference engine design because the FF usually only involves positive input data (as ReLU is often used for activations). However, for typical training process, the errors in the backpropagation could be either positive or negative. The problem of 2's complement multiplication is that, when a positive value multiplied by a negative value, both the multiplicand and the multiplier have to be sign extended. As shown in Fig. 3 (a), sign extension is essential to obtain correct result of the signed number multiplication to calculate $0.25 \times -0.75 = -0.1875$ in binary fashion. For convolution, sign extension of input could be realized by more input cycles. However, sign extension of weights means extra arrays. For example, previous CIM training architecture [21] stores negative and positive weights separately on different arrays, which thus doubles the memory size.



	Forward/Backward	Binary value							
Input	FWLM/BWLM	1	1	0	0	1	1	1	1
	FWLL/BWLL	1	0	1	0	1	0	1	0
Weight	LBL	1				0			
Product (In * W)	R_RBL/C_RBL	Sum value							
		$3I_{dis}$	$2I_{dis}$	I_{dis}	0	0	0	0	0
		$-3\Delta V$	$-2\Delta V$	$-\Delta V$	0	0	0	0	0

(b)

Figure 2: (a) Overall structure of two-way SRAM array design. (b) Truth table and operation mode for TMC.

Instead, we propose using a single memory array to perform the unsigned VMM and then utilize periphery to shift-add or invert the partial sums according to the scale, as shown in Fig. 3 (b). This is in fact a method to convert a 2's complement value back to decimal. Though the 2's complement value has sign information in itself, in the weighted sum operation we neglect the sign information at first. For the same example above, 001×101 will generate two 1's (in the red circle): The first 1 has a scale 0.25×0.25, thus we need to shift it by 4 times to obtain 0.0001; the second 1 has a scale -0.25, thus we need to shift it by twice and then perform 2's complement conversion (invert and add 1) to obtain 1.1100. Finally we add these two values together, $0.0001 + 1.1100 = 1.1101$, which is the correct 2's complement code for -0.1875.

In our proposed SRAM array, we treat the binary input/weight sequence as the unsigned weighted sum in the memory array so that

there is no need to worry about sign information since they are all positive. Then we multiply back the scale and sign information of the partial sums to the 2's complement representation. Fig. 4 shows our design: we use different rows in same TMU to store different

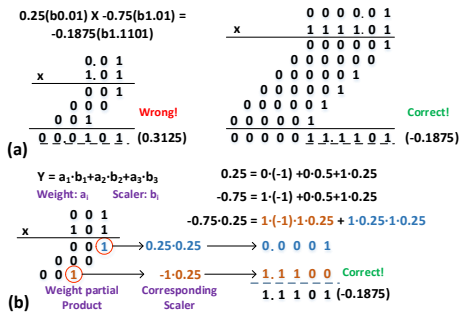


Figure 3: (a) Normal signed extension method. (b) Proposed new multiplication method.

significance of the multi-bit weight, e.g. W[0], W[1], and W[2] shown in Fig.4, and we use multiple cycles to represent the multi-bit input data. For same significant bit of weights in the same column, the MRU outputs of each cycles are multiplied by the input scales and summed up by shift-add. For example, if input precision is 3-bit, Cycle 1's output will be shifted, and Cycle 2's output will be shifted and added with Cycle 1's , and then Cycle 3's output will be inverted and added one first (to represent the negative sign) and then added with the first two cycles' partial sum. For different significant bit of weight, partial sums are further weighted and summed by the weight scales according to the significance. For example, W[0]'s partial sum will be shifted, and W[1]'s partial sum will be shifted and added with W[0]'s, and then W[2]'s partial sum will be inverted and added one first (to represent the negative sign) and then added with the other two bits' partial sums. The result will be the final sum. To generalize, the range of weights and inputs is between [-1,1], and the quantization scales will always be -2^0, 2^{-1} to $2^{-(n-1)}$. Therefore, we can use shift-adder to operate the scaling-back except for the highest bit. For the highest bit corresponding to -2^0(-1 sign), we directly invert all the bits and add 1 to obtain the 2's complement value.

3.3 Overall Architecture

The overall architecture is shown in Fig. 5. From bottom-level to top-level, the accelerator is hierarchically composed of subarrays, tiles, digital blocks and on-chip buffers. One tile consists of 32 proposed subarrays and digital blocks include accumulators and adder trees. The weight capacity of each tile is 256kB. The training dataflow is also shown in Fig. 5. According to proposed mapping strategy [23], for the FF, the products of input sliding window with the same filter across all the channels are summed up to generate one output, which means all the dot products in same column of the memory array are summed up. This pattern can be implemented by most conventional CIM architectures. However, for the BP, the products of input sliding window and the same channel across different filters are summed up, which means all the dot-products in same row need to be summed up, which is not supported by current CIM architectures. A naive solution is to store transpose weight matrix with another copy of memory arrays, which certainly increase area overhead significantly. Our bi-directional SRAM array design, which is able to perform perpendicular MAC operations, could eliminate the additional hardware for error calculation.

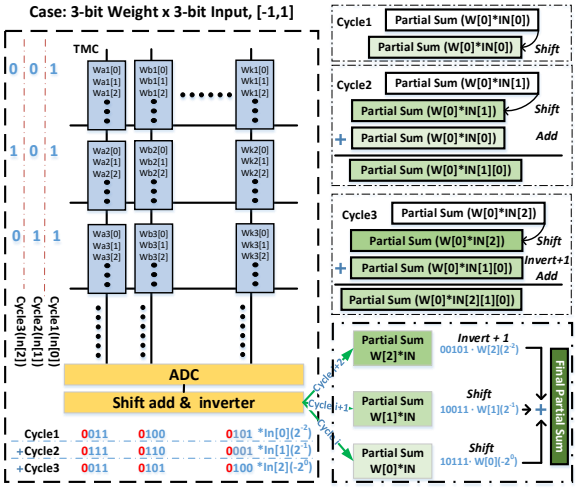


Figure 4: In-memory dataflow of signed VMM.

First, the green line and arrow indicate that activations are fed into two-way SRAM arrays to perform forward MAC and then final activation output of each layer is obtained by accumulating partial sums, which will be sent to off-chip buffer for further reuse. Then, after FF, subarrays performs backward MAC to implement error backpropagation as the red lines show. Inspired by [22], we also implement gradient computation inside memory. Unlike prior work, which requires additional SRAM arrays to perform in-memory computing of gradient, we use the same hardware for FF and EC process to calculate gradient, which can reduce both chip area and total SRAM leakage. As shown in blue line, during GC, calculated errors will be first mapped into SRAM array and activations of each layer is fed into memory array to perform VMM calculation to obtain weight gradient. Then the new weights are calculated inside the digital blocks. Finally, as shown in purple line, new weights will be sent back to each subarrays to update new weights by regular SRAM's write operation.

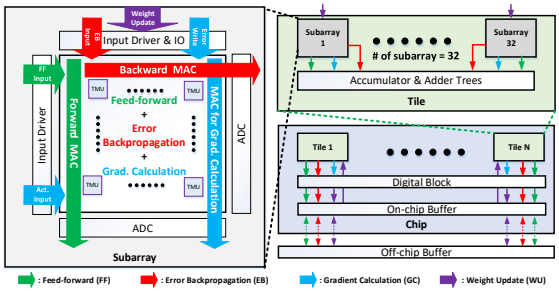


Figure 5: The overall architecture and DNN training dataflow on proposed design.

4 HARDWARE CONFIGURATION

In the proposed accelerator design, different configurations of the SRAM array will directly affect the accuracy performance and the hardware cost. The main design parameters of CIM subarrays include choices of the array size and ADC resolution. To determine array size, firstly we need to consider the weight precision. As we

discussed in Section 2.2, 8-bit weight & error precision is safe to maintain high accuracy for large dataset. To support different applications, we assign 16 SRAM cells to share one TMC, which means we can support 1-bit to 16-bit weight precision theoretically. The array size influences the ideal precision of partial sum and hardware performance. Theoretically, larger array size leads to higher throughput while ADC offset becomes worse compared to smaller array size. Besides, according to different filter size from various networks, the most efficient size is also different. Combining all the above considerations, we choose 512×128 as our SRAM array size, which can store 64×128 8-bit weights. This array size is appropriate for various moderate network from VGG-8 to large networks such as ResNet-18/34 and DenseNet-121/169 with 7 ~ 22 millions of parameters.

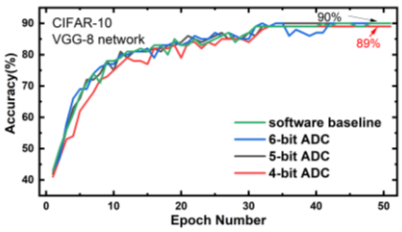


Figure 6: The impact of ADC resolution on training performance.

To determine ADC resolution, we need demonstrate the impact of ADC quantization loss on accuracy performance but prior work [16] only analyzed ADC impact on inference. Ideally, high resolution ADC can be applied to avoid any quantization loss. However, the area and power of ADC will increase dramatically as ADC precision increases. Our two-way SRAM design fetches 2-bit input and sums 16 row simultaneously, which means ideal full precision of each MAC operation is 6-bit. Thus, we use PyTorch platform to explore the impact of ADC resolution from 4-bit to 6-bit on training performance of VGG-8 network on CIFAR-10 dataset. As shown in Fig. 6, 5-bit ADC design could maintain the same accuracy as 6-bit full precision, which means 1-bit quantization loss is tolerable while 4-bit resolution degrades the accuracy by 1%. Depending on above observation, we choose 5-bit as our ADC resolution, which could obtain same accuracy but with smaller area cost compared with full precision design.

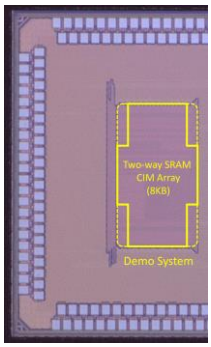
5 EVALUATION

In this section, the fabricated two-way SRAM CIM macro is presented. The proposed architectural methods in this work are also evaluated. Finally, the benchmark for various DNN models and comparison are discussed. We use four DNN models in our benchmarks for ImageNet: ResNet-18. ResNet-34, DenseNet-121 and DenseNet-169. SRAM CIM macro parameters are obtained from silicon measurements and the peripheral digital blocks in the tile level are evaluated with NeuroSim framework [24].

5.2 Prototype Implementation

We taped-out and validated the proposed two-way SRAM array in TSMC 28nm process. Fig. 7 (a) shows the fabricated die photo and chip measurement results are shown in Fig. 7 (b). The SRAM array size is 64kb (512 row × 128 column). Each BMU is only 1.68x of the area of 16 compact-rule 6T SRAM cells. Subarray partial sum precision is 20-bit for 8-bit input and weight. The energy cost of EC is about twice of FF process since forward MAC is processed in 16 columns simultaneously while backward MAC is processed in 32 rows in parallel. Compared to conventional SRAM array for cache with same array size, the area overhead of proposed two-way SRAM array is 2.98× due to transpose-multiplication-cells and ADCs.

(a)



(b)

Technology	TSMC 28nm			
Supply voltage(V)	0.85 – 1			
SRAM array size	8KB (512 row × 128 column)			
Mode	Forward		Backward	
Input precision (bit)	8		8	
Weight precision (bit)	4	8	4	8
Output precision (bit)	16	20	16	20
Process time (ns)	15.2-17.2	30.4-34.4	18.5-21.0	37.0-42.0
Energy (pJ)	67.48-73.10	67.51-73.20	136.02-142.02	136.25-142.23
memory area (no periphery)	1.68×			
Array area (total)	1.49×	2.98×	1.49×	2.98×

Figure 7: (a) Die photo of Two-way SRAM macro. (b) Summary table of silicon measurement data.

5.3 Benchmark Results and Discussion

Fig. 8 (a) shows the architecture-level performance comparison between conventional CIM SRAM design and two-way SRAM array design. The proposed two-way SRAM design could reduce chip area by ~25% with no throughput or energy efficiency loss, which is observable across different networks. Fig. 8 (b) shows the evaluation of proposed signed number multiplication compared to conventional solution, which directly performs sign extension multiplication. With the novel multiplication method, hardware performance including area, throughput, energy efficiency are all improved significantly. For instance, for ResNet-18, our proposed design could achieve 2× throughput and 1.6× energy efficiency with only 0.6× area cost. ResNet shows larger throughput than DenseNet because ResNet needs less computations with less layers. Further speed-up is possible by weight duplication for slow layers in a pipeline design. Taking the example of DenseNet-121, the trade-off between weight duplication and performance is presented in Fig. 8 (c). We can see that the throughput is proportional to the hardware area cost.

Fig. 9 (a) shows the energy breakdown among training steps. Gradient calculation takes more energy than FF and EC due to less reuse of error matrix and frequent write to reload error matrix. Fig. 9 (b) shows energy breakdown of CIM computing, SRAM leakage, and off-chip DRAM access. SRAM leakage is almost eliminated since we keep reusing SRAM array during the entire training process to shorten standby time. DRAM access is still a significant part of the total energy since GC and WU process needs to access DRAM frequently due to limited on-chip buffer. Table 1 compares this work to state-of-the-art accelerator designs from digital ASIC to CIM approaches. It is seen that our proposed design achieves highest energy efficiency and throughput, which could support multi-bit training on large dataset while most prior works only support low-bit inference on small to moderate dataset. Our SRAM CIM accelerator also holds advantages (~3.2 TOPS/W) compared to GPU (~0.1 TOPS/W) or TPU (~0.45 TOPS/W for the training version).

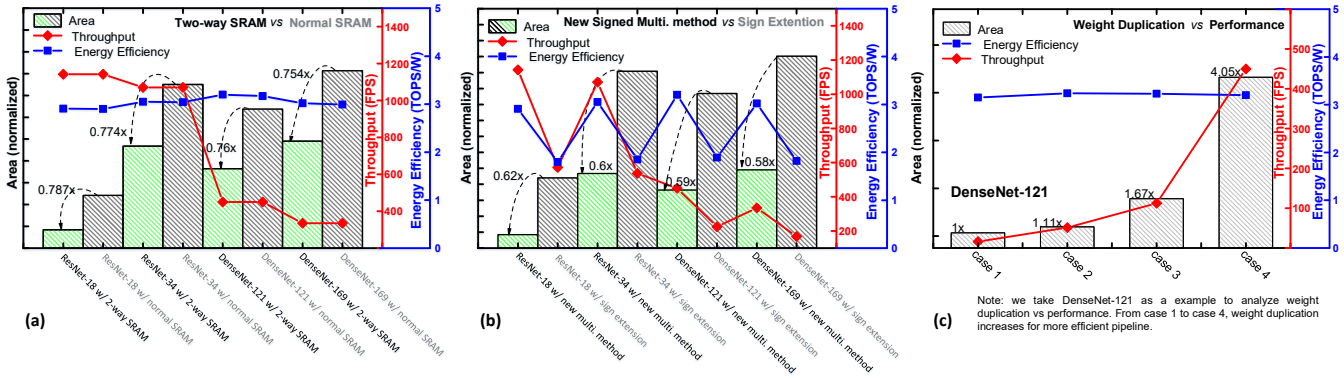


Figure 8: (a) Performance improvement with two-way SRAM design. (b) Performance improvement with proposed signed number multiplication. (c) Performance comparison when increasing the weight duplication on DenseNet-121.

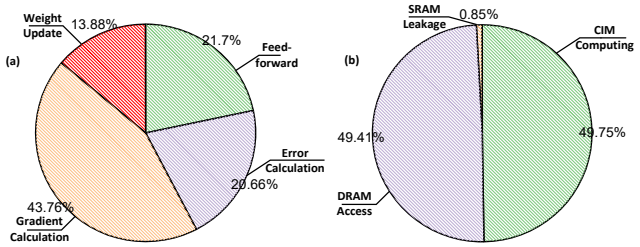


Figure 9: Two-way SRAM architecture-level benchmark result: (a) Energy breakdown of training steps (b) Energy breakdown of operations.

Table 1: Comparison with recent ASIC and CIM hardware

	JSSCC'17[1]	ISSCC'18[18]	DAC'18[16]	JSSCC'19[20]	ISSCC'19[19]	This work
Technology	65nm	65nm	65nm	65nm	55nm	28nm
Algorithm	AlexNet	SVM	BNN, XNOR-Net	CNN	CNN	VGG,ResNet DenseNet
Dataset	ImageNet	MIT-CBCL	CIFAR10	MNIST, CIFAR10	MNIST, CIFAR10	CIFAR10, ImageNet
Training support	No.	Yes	No.	No.	No.	Yes
Architecture	Digital	SRAM-based CIM	SRAM-based CIM	SRAM-based CIM	SRAM-based CIM	SRAM-based CIM
Precision	16-bit	8-bit	1-bit	1-bit	4-bit	8-bit
MAC TOPS/W	-	3.125	50	-	18.37	17.2
Total TOPs/W	0.023	-	-	1.25	-	3.15-3.2
TOPS	0.12	0.004	-	0.0432	0.09	6.1-23.2

6 CONCLUSIONS

In this paper, we propose an efficient SRAM based CIM accelerator aimed at DNN training. First, a two-way SRAM array for CIM training is presented to significantly decrease energy cost and area overhead. Meanwhile, by applying a new signed number multiplication approach for in-memory computing, unnecessary input cycles, memory arrays and energy cost for sign extension are eliminated. Moreover, we evaluate ADC quantization impact on training performance. Finally, we validated the proposed two-way SRAM array design with a tape-out in TSMC 28 nm process. The architecture-level performance for DNN on-chip training is estimated from the measured silicon data of CIM macro, which shows about 3.2 TOPS/W energy efficiency with over 1000 FPS on ResNet-18/34 and over 300 FPS on DenseNet-121/169.

ACKNOWLEDGEMENT

This work is in part supported by Samsung GRO program.

REFERENCES

[1] Y.H. Chen et al., "Eyeriss: An energy- efficient reconfigurable accelerator for deep convolutional neural networks," in *IEEE JSSCC,* 2017.
[2] N.P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *IEEE ISCA,* 2017
[3] I. Hubara et al., "Binarized neural networks," in *NIPS*, 2016.
[4] M. Rastegari et al, "XNOR-net: Imagenet classification using binary convolutional neural networks," in *ECCV,* 2016.
[5] S. Yu., "Neuro-inspired computing with emerging nonvolatile memorys," in *Proc. IEEE*, 2018.
[6] K. He et al., "Deep residual learning for image recognition," in *IEEE CVPR,* 2016.
[7] H. Gao et al., "Condensenet: An efficient densenet using learned group convolutions," in *IEEE CVPR,* 2017.
[8] L. Song et al., "Pipelayer: A pipelined ReRAM-based accelerator for deep learning," in *IEEE HPCA*, 2017.
[9] S. Zhou et al., "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160,* 2016.
[10] S. Wu et al., "Training and inference with integers in deep neural networks," in *ICLR,* 2018.
[11] R. Banner et al., "Scalable methods for 8-bit training of neural networks," *Advances in Neural Information Processing Systems*, 2018.
[12] Y. Yang et al., "Training high-performance and large-scale deep neural networks with full 8-bit integers," *arXiv preprint arXiv:1909.02384*, 2019.
[13] I. Giannopoulos et al., "8-bit precision in-memory multiplication with projected phase-change memory," in *IEEE IEDM,* 2018.
[14] C.X. Xue et al., "A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors," in *IEEE ISSCC,* 2019.
[15] G. W. Burr, et al., "Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power)," in *IEEE IEDM,* 2015.
[16] R. Liu et al., "Parallelizing SRAM arrays with customized bit-cell for binary neural networks," in *IEEE DAC,* 2018.
[17] Z. Jiang et al., "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," in *IEEE Symp. VLSI Technology*, 2018.
[18] S.K. Gonugondla et al., "A 42pJ/decision 3.12 TOPS/W robust in-memory machine learning classifier with on-chip training," in *IEEE ISSCC*, 2018.
[19] X. Si, et al., "A twin-8T SRAM computation-in-memory macro for multiple-bit CNN-based machine learning," in *IEEE ISSCC*, 2019.
[20] H. Valavi et al., "A 64-tile 2.4-Mb in-memory-computing CNN accelerator employing charge-domain compute," in *IEEE JSSC,* 2019.
[21] M. Chen et al., "TIME: A training-in-memory architecture for memristor-based deep neural networks," in *IEEE DAC*, 2017.
[22] H. Jiang et al., "CIMAT: A transpose SRAM-based compute-in-memory architecture for deep neural network on-chip training," in *ACM MEMSYS,* 2019.
[23] X. Peng et al., "Optimizing weight mapping and data flow for convolutional neural networks on RRAM based processing-in-memory architecture," in *IEEE ISCAS,* 2019.
[24] P.Y. Chen et al., "NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," in *IEEE TCAD,* 2018.