RRAM Based Buffer Design for Energy Efficient CNN Accelerator

Kaiyuan Guo, Jincheng Yu, Xuefei Ning, Yiming Hu, Yu Wang, Huazhong Yang
Department of Electronic Engineering, Tsinghua University
{gky15@mails.tsinghua.edu.cn, yu-wang@tsinghua.edu.cn}

***Abstract*—Convolutional Neural Network (CNN) has become the state-of-the-art algorithm for many computer vision tasks. But its high computation complexity and high memory complexity makes it hard to be deployed on traditional platforms like CPUs. Memory energy can take up a large part of the system energy, which limits the energy efficiency of CNN processing. The emerging metal-oxide resistive switching random-access memory (RRAM) has been widely studied because of its good properties like high storage density and the compatibility with CMOS. In this paper, a system level energy analysis of using RRAM as on-chip weight buffer is carried out for a typical CNN accelerator. Hardware and scheduling optimizations are proposed to fully utilize the large RAM and avoid high read/write energy overhead. Experimental results show that RRAM based designs save 12-18% system energy with 15-75% smaller on-chip RAM area compared with SRAM designs.**

***Index Terms*—RRAM, Convolutional Neural Network, Hardware Accelerator**

I. Introduction

Convolutional Neural Network (CNN) has become a state-of-the-art algorithm for a wide range of applications like image classification [1], object detection [2] and other image-based tasks. Compared with traditional hand-crafted-feature-based methods, CNN introduces a uniform model for different tasks and adjusts the model based on different training data set. Thus CNN can be adopted in various tasks and keeps high model accuracy.

But CNN is still not widely applied in real applications because of its high computation and memory complexity. A typical network like AlexNet [3] consists of more than 240MB parameters and 1.4G FLOPs (floating-point operations) for the inference of a single 224×224 image. More advanced networks [1], [4] require much more computation and memory than AlexNet. The energy cost for CNN computation is thus high, especially on traditional platforms like CPU.

Various works explore energy efficient hardware designs for CNN accelerators. Many designs focus on CMOS based circuit design targeting at efficient data path and memory system [5], [6]. In these designs, the size of the on-chip SRAM is quite limited. Thus external memory like DRAM is always needed in real applications. The high energy cost brought by off-chip data transfer dramatically limits the system energy efficiency.

One solution to reduce memory access is to perform in-memory computing. RRAM cross bar based design is one of the popular research topics. Chi et al. propose the PRIME [7] architecture which implements the matrix-vector multiplication directly with the RRAM crossbar. Work by Cheng et al. [8] implements the RRAM crossbar to support both inference and training of neural networks. In memory computation with RRAM shows attractive energy efficiency. But the application range of the design is limited by the scalability and computation accuracy of the RRAM crossbar.

Another way to reduce off-chip data transfer is to implement large on-chip memory, where RRAM is a good candidate. We compare RRAM with SRAM and DRAM in Table I according to the performance reported in [9], [10] and simulation result by NVSim [11]. Compared with SRAM, the storage density of RRAM is similar to DRAM, which is about $20\times$ higher. Compared with DRAM, RRAM can be integrated on-chip while the former one can not.

TABLE I
Comparison between SRAM, DRAM, and RRAM

	SRAM	DRAM	RRAM
Cell Size	$140F^2$	$6 \sim 8F^2$	$6F^2$
Frequency	>1GHz	100-400MHz	<100MHz
Integrated on Chip	yes	no	yes
I/O energy / Byte	<1pJ	∼10pJ	∼40pJ

But RRAM is also limited in some aspects. First, the I/O bandwidth of RRAM is smaller than SRAM. For applications like CPU cache, where latency is critical to performance, RRAM may not be a good choice. For CNN, the data access pattern is static. This means we can design data storage to achieve sequential access at run-time. So we can use more banks to compensate for the limited bandwidth.

Second, the I/O dynamic energy cost of RRAM is high In this paper, we show that a simple implementation with RRAM increases the total system energy cost. But there is a chance that we utilize the property of CNN computation to reduce on-chip memory access to overcome the energy overhead of using RRAM.

In this paper, we introduce RRAM as the weight memory for a typical CNN accelerator design and optimize the design in hardware and scheduling level to overcome the RRAM energy overhead. The contributions of this paper are as follows:

- Hardware optimization is proposed to reduce the RRAM dynamic energy overhead.
- Dedicated scheduling strategy optimization is proposed to fully utilize the large RRAM buffer to reduce off-chip data transfer.
- Design space exploration is done with state-of-the-art networks to show the effect of using RRAM as the on-chip buffer.

Experimental results on state-of-the-art CNN models show that RRAM based design saves $12-18\%$ system energy with a $15-75\%$ smaller on-chip RAM area compared with SRAM design. The proposed hardware and scheduling optimization reduce up to 96% on-chip RAM access energy and 98% off-chip data transfer.

The rest of this paper is organized as follows: Section II introduces the related work for CNN accelerator and RRAM research. Section III introduces the hardware design. The scheduling strategies are introduced in section IV. The experimental results are shown in section V. Section VI concludes this paper.

II. Related Work

A. Convolutional Neural Network

A CNN mainly consists of two types of layers, convolution (CNV) layers and fully connected (FC) layers, which contribute to most of the computation and parameters in a CNN. The computation for a CNV layer is shown as the pseudo code below. F_{in} and F_{out} are the input and output feature maps. K and b denote the convolution kernels and bias. The fully connected layer can be expressed as matrix-vector multiplication and can be treated as a special case of CNV layers where all the feature maps are 1×1, and convolution kernels are 1×1.

CnvLayer(F_{in}, F_{out}, K, b)

```
1  for m = 1 → M // output channel loop
2      for n = 1 → N // Input channel loop
3          // feature map pixel loop
4          for each F_out[m][x][y] ∈ F_out[m]
5              F_out[m][x][y] = 0
6              // convolution kernel loop
7              for each K_mn[xx][yy]
8                  F_out[m][x][y]+ = K_mn[xx][yy]*
                       F_in[n][x + xx][y + yy]
9              F_out[m][x][y]+ = b_m
```

Table II shows the number of parameters and the computation complexity of some state-of-the-art network models. Usually a CNN consists of 10-100 million parameters and requires from 1-50G operations for the inference of each input. Compared with that, general purpose platforms like CPU only offers 1-100GOP/s computation capacity and consumes 1-100W power. This hardly meets the real-time requirement in video processing or the power limit of mobile platforms. There is urgent need of fast and energy efficient computation platform for processing CNN.

B. CNN Accelerator

The design of a CNN Accelerator involves two aspects: computation units and scheduling strategy. For computation units, various techniques have been studied to reduce the bit-width used for the data expression in CNN to reduce both the memory requirement and energy cost for computation and data transfer. Experimental results from [12] show that 8-bit linear data quantization introduces negligible accuracy loss for common CNN models compared with the original 32-bit floating-point model.

TABLE II
Computation complexity and parameter size of state-of-the-art networks.

	Computation (GOP)			Parameter (M)		
	CNV	FC	Total	CNV	FC	Total
AlexNet [3]	1.33	0.12	1.45	2.33	58.62	60.95
VGG-11 [1]	14.97	0.25	15.22	9.22	123.63	132.85
VGG-16 [1]	30.69	0.25	30.94	14.71	123.63	138.34
ResNet-34 [4]	7.28	0.001	7.281	21.1	0.51	21.61

On scheduling aspect, most of the CNN accelerators [5], [6] uses single layer scheduling strategies, where all the computation units work for the same layer at any time. Different layers are executed one by one. Single-layer design requires that the result of a layer to be written back to external memory when it is larger than the on-chip buffer. Alwani et al. [13] suggest that adjacent layers can be fused together as a scheduling unit to reduce the data transfer with the external memory.

As suggested by [14], typical energy for a 32bit DRAM access is $100\times$ more than that of a 32-bit SRAM access, $200\times$ more than a 32bit multiplication and $6400\times$ more than a 32-bit addition. This indicates that the memory access energy contributes a major part of the total system cost.

C. RRAM

RRAM is one of the promising candidate for large on-chip memory. The cell size of SRAM is typically 10-20x larger than that of RRAM with the same technology. Some memory designs [15], [16] try to store multiple bits within a cell to further increase storage density. Despite the high storage density, the limited bandwidth of RRAM is always a consideration when using as on-chip memory. But recent work [10] shows that even high-density array of 32Gb achieves 1GB/s read bandwidth, and 200MB/s write bandwidth. In this work, we adopt [11] as a tool to generate RRAM modules of different sizes in our experiments.

Using RRAM crossbar for both computation and storage in NN accelerators is another research topic [7] [8] [17] [18]. In these designs, extremely high energy efficieny is achieved because there is no energy needed to read network parameters from memory. But these designs are limited by the storage and computation accuracy of RRAM devices and the analog-digital converter overhead. In this paper, we only focus on using RRAM as on-chip memory, not as computation units.

III. Hardware Design

This section introduces the CNN accelerator we adopt in this work. We first introduce the architecture and energy model, and then the optimization to reduce the RRAM buffer dynamic energy cost.

A. Architecture

The overall architecture is shown in Figure 1(a). The system adopts DDR as external memory to support enough memory space. The on-chip memory consists of two buffers:

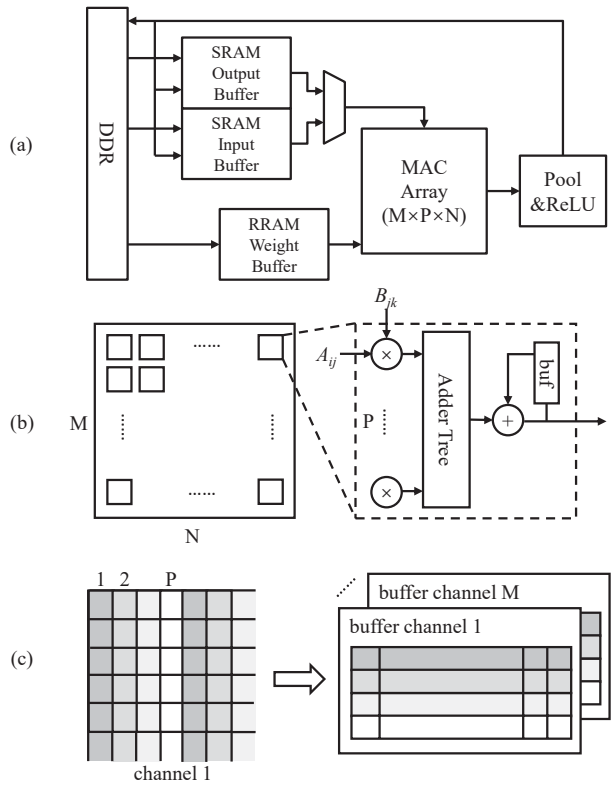

Fig. 1. The CNN accelerator architecture. (a) The architecture mainly consists of a MAC array for computation and two levels of memory with on-chip buffers and external DDR. (b) MAC array structure for an $A_{p\times m} \times B_{m\times n}$ matrix-matrix multiplication. (c) Feature map buffer organization for sliding window data selection.

input/output buffer and weight buffer. Input and output buffer stores feature maps during the process of a CNN. As the output of one layer serves as the input of the next, the input data of MAC array can be chosen from both of the buffers by a MUX unit. All the buffers work in a double-buffer way to cover the off-chip data transfer time with calculation time.

We first show how the process of CNN is parallelized with this design. As introduced in Section II A, a CNN layer consists of 4 nested loops. In this case, we unroll the feature map pixel loop, input channel loop and output channel loop as a $A_{m\times p} \times B_{p\times n}$ matrix-matrix multiplication. Each row of matrix A denotes the p pixels read from a feature map and each B_{ij} denotes a pixel in the convolution kernel corresponds to input channel i and output j. Thus the convolution of p pixels with $K\times K$ convolution kernels will be done with $\lceil M/m \rceil * K^2$ steps. The structure of the computation core is shown in Figure 1(b). The array consists of $p \times n$ processing elements (PEs). Each of the PE implements a size-m vector inner product in a pipelined manner. Each PE has an accumulator for its result.

To support the above parallelization, a window selection function is needed for each channel stored in input/output buffer. In this design, we adopt the data mapping format as shown in Figure 1(c). The pixels of each channel are stored in p independent RAMs so that a size-p window at any position can be selected from the buffer within a single cycle. Weight buffer can be implemented with a normal simple dual port RAM.

Consider the data access pattern, implement input/output buffers with RRAM is not practical. First, they require high write bandwidth to receive computation results from fast on-chip logic. Second, they require a high random access bandwidth, not sequential access bandwidth. Compared with input/output buffer, weight buffer only requires a high sequential access bandwidth. So in this work, we only consider implementing weight buffer with RRAM.

B. Choosing the Loop Order

With the parallelization strategy decided as introduced in section III-A, the data access behavior is affected by the order of the loops in the whole process. If we do not consider data transfer with DDR, the on-chip buffer energy cost comes from three aspects: input/output buffer read, weight buffer read, and input/output buffer write. To minimize this energy cost, we consider two loop execution order: kernel first and pixel first.

Kernel first order computes the $K \times K$ convolution on the p pixels with K^2 cycles first, then move on to the next p pixels. With this order, during the K^2 cycles, as the feature map window slides, the spatial locality of 2-d convolution is explored. Each feature map pixel is accessed only once. Pixel first order reuses the same weight in a $K \times K$ kernel and computes multiple p-pixel groups, then move on to the next weight. With this order, more feature map access is needed, and intermediate result for each pixel group should be stored. This is not commonly adopted because intermediate result requires more bits for fixed-point data process. Suppose c pixel groups are processed together, we compare the data access in Table III.

TABLE III
DATA ACCESS COMPARISON BETWEEN KERNEL FIRST ORDER AND PIXEL FIRST ORDER TO CALCULATE c GROUPS OF p PIXELS WITH $K \times K$ CONVOLUTION.

	Kernel First	Pixel First
Input Feature Map	$cK(p+K-1)$	cpK^2
Weight	cK^2	K^2
Output Feature Map Rd	0	$cp(K^2-1)$
Output Feature Map Wr	cp	$cp(K^2-1)$

Directly using input/output buffer to store the intermediate result is not feasible because the intermediate result for fixed-point multiplication requires more bits. As RRAM has high read/write dynamic, we should try to reuse the weight to reduce access to weight buffer. In this design, we use a local accumulation buffer of size $2n$ as shown in Figure 1(b) to enable n pixel groups processed together before written back to input/output buffer. This introduces extra on-chip memory cost. We examine the area and energy overhead in our experiment.

IV. SCHEDULING STRATEGY

In the previous section, we choose the loop execution order to reduce data access. In this section, we analyze schedule

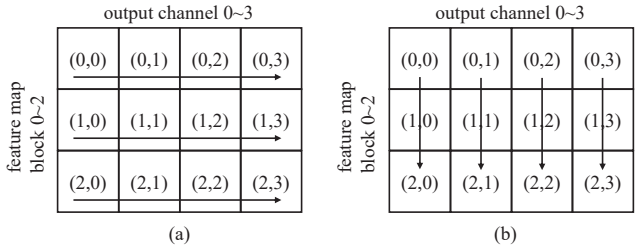

Fig. 2. An example of how loop order affects data transfer behavior. Block (i, j) denotes the computation for the i^{th} feature map block on the j^{th} channel. (a) Reuse feature map first and load each weight 3 times. (b) Reuse weight first and load each feature map 4 times.

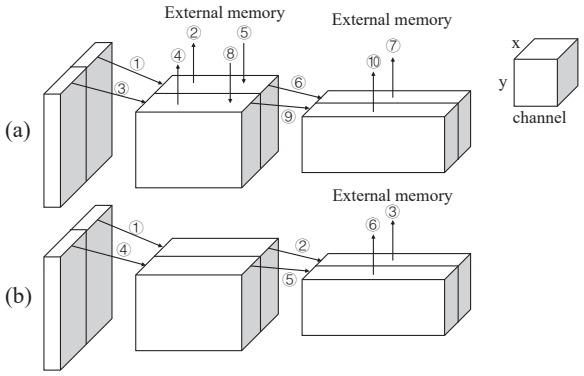

Fig. 3. An example of a cross-layer schedule over 3 layers. (a) single-layer schedule. (b) cross-layer schedule.

strategy, which decides the off-chip data transfer behavior during the process of a network. Three kinds of scheduling are considered: single-layer, cross-layer and fixing weight on-chip.

Single-layer Schedule Strategy. In the case where on-chip buffer size is smaller than the weight size and feature map size of a single-layer, data needs to be loaded multiple times. An example is shown in Figure 2 where the weight buffer can hold only $1/4$ of the weight and input buffer can hold only $1/3$ of the feature maps. Choosing reuse weights or reuse feature map will cause a difference in data transfer behavior and further affects the data transfer energy and data transfer time.

Cross-layer Schedule Strategy. As suggested by [13], when the network is processed layer by layer, the result of one layer needs to be written back to external memory it is larger than the output buffer. But if the weight buffer is large enough to contain more than one layers' weight, these layers can be scheduled together to avoid writing the intermediate layers result to external memory. An example is shown in Figure 3, where the cross-layer schedule avoids the second layer's feature map to be transferred to and from external memory. As we will implement weight buffer with RRAM, the size of weight buffer can be much larger and thus benefits from this strategy.

Fixing Weight On-chip. An observation is that if the weight buffer is large enough to hold all the layer's weight on-chip, no weight will need to be read from external memory. With a slightly smaller buffer, we can still follow this idea by fix some of the layer's weights in the on-chip buffer to reduce external memory access. So we need to find which layers' weight is to be fixed in the buffer as to minimize the total energy cost.

The solution space of this problem is binary tree of depth $(n+1)$. n denotes the number of CNV layers in the network. To avoid searching the whole $O(2^n)$ solution space, we prune the binary tree with the buffer size limitation. If the the weights on a path already exceeds the weight buffer size, we ignore the search of its subtree. The pseudo code of the optimization process is as follows:

SEARCHFIXEDWEIGHT($layers$)
1 Let $is\_fixed[1..layers.size]$ be a boolean vector
2 IterSearchFixedWeight($layers, 1, is\_fixed$)
3 **return** is_fixed

ITERSEARCHFIXEDWEIGHT($layers, n, f$)
1 **if** $l > layers.size$
2 **return** OptEnergy($layers, f$)
3 **if** available buffer size $< layers[n].weight\_size$
4 $f[n] = false$
5 **return** IterSearchFixedWeight($layers, n+1, f$)
6 $f1 = f;\ f1[n] = false$
7 $f2 = f;\ f2[n] = true$
8 $e1$ = IterSearchFixedWeight($layers, n+1, f1$)
9 $e2$ = IterSearchFixedWeight($layers, n+1, f2$)
10 $f = (e1 < e2)\ ?\ f1 : f2$
11 **return** Min(e1, e2)

V. EXPERIMENTS

Experiments are carried out on the architecture introduced in Section III, with the scheduling strategy in section IV applied. We use a behavior level simulator to analyze the memory access and computation energy for running a certain network. Memory access dynamic energy cost, memory static energy cost and computation energy cost are considered in our model.

A. Experiment setup

The MAC array in the architecture is configured as an $8 \times 8 \times 8$ array running at 1GHz. This offers a peak performance of 1TOP/s. 8-bit multiplication and 32-bit accumulation are adopted in this model. Multiplication and addition energy is scaled down from the data in [14] to 22nm technology.

The above configuration requires the read bandwidth of input/output buffer and weight buffer to be at least 64GB/s. We implement each buffer with 8 banks and each of them should offer 8GB/s read bandwidth. On-chip memory parameters are generated from NVSim [11] with different memory size configurations. RRAM buffer bit width is configured as 256bit to achieve enough bandwidth.

The external memory parameter is generated from MICRON DDR4 power calculator [19]. The generated dynamic I/O power is further converted to energy per read or write byte. In our experiment, we use 2 DDR chips as external memory because the bandwidth can support the configured MAC array with proposed schedule strategy and the size is enough. To reduce background power overhead, we use the least number

TABLE IV
DEVICE PARAMETERS USED FOR I/O BUFFER, WEIGHT BUFFER AND DDR.

	bitwidth (B)	size (B)	Rd BW (GB/s)	Wr BW (GB/s)	Rd Ene (pJ)	Wr Ene(pJ)	Leakage (mW)	Area (um^2)
22nm LSTP RRAM	32	128K	15.169	1.556	67.690	195.286	0.04000	21224
	32	256K	11.693	1.534	75.071	217.468	0.04104	26707
	32	512K	8.005	1.490	88.483	260.073	0.04314	37308
	32	1M	11.056	1.534	133.189	268.319	0.05282	61090
	32	2M	10.306	1.534	231.750	357.190	0.07806	107007
22nm LSTP SRAM	8	16K	9.145	5.147	3.057	0.556	0.00134	10031
	8	32K	8.609	4.973	6.432	1.315	0.00270	20048
	8	64K	16.831	11.763	6.780	3.777	0.00600	42803
	8	128K	10.977	10.451	7.931	2.792	0.01153	82032
	8	256K	10.977	10.451	11.562	6.424	0.02306	164065
DDR4	4	128M	3.2		80.300	82.719	52.80000	N/A

of chips. All the detailed data and configuration is shown in Table IV.

The buffer in the accumulator is also considered in our experiments. 4 types of buffers are chosen for design space exploration. Corresponding parameters are generated by NVSim and are shown in Table V.

TABLE V
32-BIT ACCUMULATION BUFFER PARAMETERS.

depth	16	32	64	128
energy per read (pJ)	0.045	0.056	0.107	0.12
energy per write (pJ)	0.022	0.031	0.083	0.094
area (um^2)	57.673	103.422	188.714	354.138
Leakage (nW)	6.162	11.133	22.385	44.823

B. Energy Cost Analysis

We do experiments on all the combinations of the RAM configurations in Table IV and V, which means $5 \times 5 \times 5 = 125$ choices for an SRAM or RRAM based accelerator. The three schedule strategies in section IV is applied to each choice. Figure 4 shows the experimental result with the convolution layers of VGG-11 network for 1 input. The energy cost of computation and DDR leakage is marked and is the same for all the designs because the network is fixed and all the designs are computation bounded. So the processing time is a constant to all the designs. In general, the RRAM based design consumes less energy compared with an SRAM design of the same area. The minimal energy design for SRAM and RRAM costs 3086uJ and 2532uJ respectively, meaning that using RRAM can save 18% system energy. The RRAM design is also 15% smaller than the SRAM design. We also do experiments with the convolution layers of VGG-16 and AlexNet, where RRAM design saves 18% and 12% energy with 15% and 75% less on-chip RAM area compared with SRAM design.

We examine the effect of the proposed hardware optimization and scheduling strategy. First, the loop order is the key effect to the RRAM based design, as shown in Figure 4(b). Note that some of the design points with the kernel first loop order are out of the figure because the energy cost is larger than 8000uJ. The large read energy of RRAM dominates the system energy as the capacity of RRAM increases. The reduction in DDR access cannot compensate for this overhead. Even on the smallest design, changing the loop order to pixel first will save at least 1/3 of the system energy cost.

A breakdown of the on-chip buffer energy cost is shown in Figure 5(a). The largest designs, in our experiment the designs with 4MB SRAM input/output buffer and 2MB SRAM (or 16MB RRAM) weight buffer are used. Here we only consider the read energy of input/output buffer and weight buffer and the dynamic energy of accumulation buffer. The write energy to input/output buffer and leakage energy are not affected by the accumulation buffer and thus not considered. Up to 96% energy is saved for the RRAM based designs consider the overhead brought by the accumulation buffer. SRAM design also benefits from the change in loop order but not as effective as to the RRAM designs.

Second, fixing the weight on-chip helps the hardware fully utilize the on-chip RAM to reduce off-chip data transfer. As can be seen from Figure 4(b), using the pixel first loop order, the accelerator cannot benefit from a larger on-chip buffer with single-layer scheduling and cross-layer scheduling. By fixing some of the network weights on-chip, the system energy cost is gradually reduced as the on-chip RAM size increases.

A breakdown of the DDR transfer cost is shown in Figure 5(b). All the designs implement no accumulation buffer and 1MB SRAM as input/output buffer. The single layer scheduling and cross-layer scheduling can only achieve 6.4% and 13.5% energy saving respectively when the weight buffer increases from 1MB to 16MB. With the fix weight strategy, the energy saving is 98.5%.

VI. CONCLUSION

In this paper, we introduce RRAM into CNN accelerator as on-chip weight buffer. To address the high read/write dynamic energy of RRAM, we change the loop order and implement the accumulation buffer to reduce RRAM access. Up to 96% energy is saved with this optimization. To fully utilize the large capacity of RRAM buffer, we explore the possibility to keep weights in the on-chip buffer to further reduce off-chip memory access based on existing scheduling strategies. Experimental results show that this strategy reduces 85% more DDR read as the weight buffer size increases when compared with existing scheduling strategy. With the hardware and scheduling optimization, RRAM based design saves 12-18% system energy with a 15-75% smaller on-chip RAM area compared with SRAM design. Future work should focus on the rest part of the energy cost like the leakage energy of DDR and the computation energy.

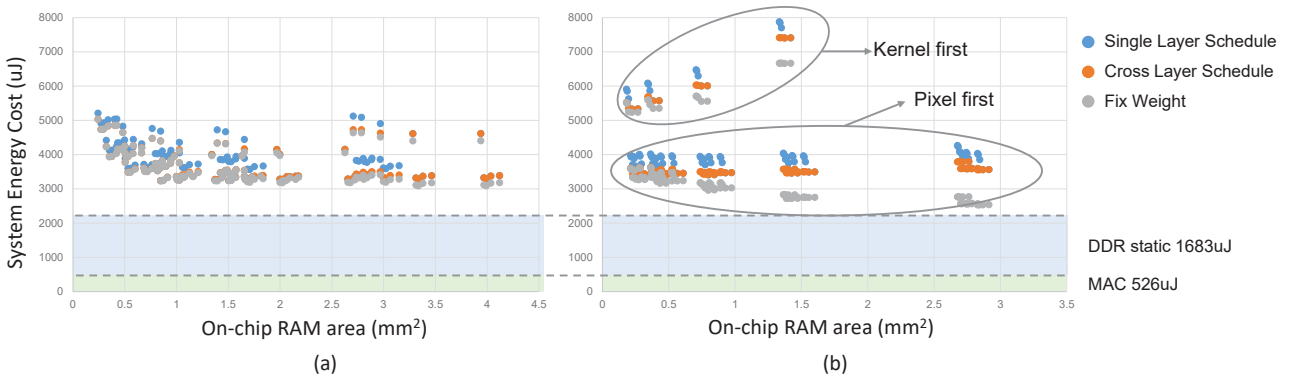

Fig. 4. Design space exploration on different hardware choices and schedule strategies on the convolution layers of VGG-11 model. (a) SRAM weight buffer design. (b)RRAM weight buffer design.

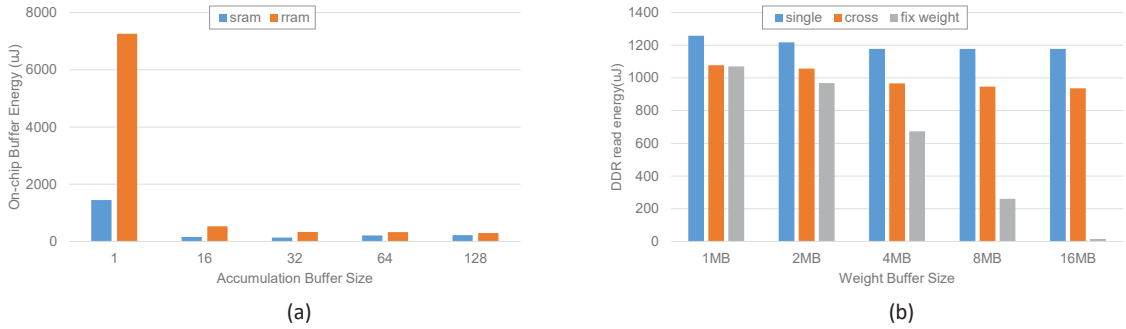

Fig. 5. (a) On-chip buffer energy cost for a series of designs only differs in accumulation buffer size. (b) DDR access energy cost for a series of designs only differs in weight buffer size.

ACKNOWLEDGEMENT

This work was supported by National Key R&D Program of China 2017YFA0207600, National Natural Science Foundation of China (No. 61622403, 61621091),Joint fund of Equipment pre-Research and Ministry of Education (No. 6141A02022608), Beijing National Research Center for Information Science and Technology, and DeePhi Technology.

REFERENCES

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

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *arXiv preprint arXiv:1506.02640*, 2015.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[5] C. Zhang, P. Li, G. Sun *et al.*, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *FPGA*. ACM, 2015, pp. 161–170.

[6] J. Qiu, J. Wang, S. Yao *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA*. ACM, 2016, pp. 26–35.

[7] P. Chi, S. Li, C. Xu *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*. IEEE Press, 2016, pp. 27–39.

[8] M. Cheng, L. Xia, Z. Zhu *et al.*, "Time: A training-in-memory architecture for memristor-based deep neural networks," in *DAC*. ACM, 2017, p. 26.

[9] S. Yu. Overview of memory technology. [Online]. Available: https://www.coursehero.com/file/15370931/EEE-598-Section-1-Overview-of-Memory-Technology/

[10] R. Fackenthal, M. Kitagawa, W. Otsuka *et al.*, "19.7 a 16gb reram with 200mb/s write and 1gb/s read in 27nm technology," in *ISSCC*. IEEE, 2014, pp. 338–339.

[11] X. Dong, C. Xu, N. Jouppi, and Y. Xie, "Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory," in *Emerging Memory Technologies*. Springer, 2014, pp. 15–50.

[12] K. Guo, S. Han, S. Yao *et al.*, "Software-hardware codesign for efficient neural network acceleration," *IEEE Micro*, vol. 37, no. 2, pp. 18–25, 2017.

[13] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–12.

[14] M. Horowitz, "Energy table for 45nm process, stanford vlsi wiki.[online]." https://sites.google.com/site/seecproject.

[15] W. Chien, Y. Chen, K. Chang, E. Lai, Y. Yao, P. Lin, J. Gong, S. Tsai, S. Hsieh, C. Chen *et al.*, "Multi-level operation of fully cmos compatible wox resistive random access memory (rram)," in *Memory Workshop, 2009. IMW'09. IEEE International*. IEEE, 2009, pp. 1–2.

[16] W.-C. Chien, M.-H. Lee, F.-M. Lee *et al.*, "Multi-level 40nm wo x resistive memory with excellent reliability," in *Electron Devices Meeting (IEDM), 2011 IEEE International*. IEEE, 2011, pp. 31–5.

[17] L. Xia, T. Tang, W. Huangfu, M. Cheng, X. Yin, B. Li, Y. Wang, and H. Yang, "Switched by input: Power efficient structure for rram-based convolutional neural network," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[18] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on rram," in *Design Automation Conference*, 2017, pp. 782–787.

[19] Power calc. [Online]. Available: https://www.micron.com/~/media/documents/products/power-calculator/ddr4_power_calc.xlsm?la=en