RRAM Based Buffer Design for Energy Efficient CNN Accelerator

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 applied with traditional platforms like CPUs. Memory energy can take up more than 50% 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.

KEYWORDS

RRAM, Convolutional Neural Network, Hardware Accelerator

ACM Reference Format:
. 2017. RRAM Based Buffer Design for Energy Efficient CNN Accelerator. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Convolutional Neural Network (CNN) has become a state-of-the-art algorithm for a wide range of applications like image classification [19][13] , object detection [18] 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 different tasks and keeps high classification accuracy or detection accuracy.

But CNN is still not widely applied in real applications because of its high computation and memory complexity. A typical network like AlexNet [15] consists of more than 240MB parameters and 1.4GFLOPs for the inference of a single 224×224 image. More advanced networks [19][13] requires much more computation and memory than AlexNet. The energy cost for CNN computation is thus high, especially on traditional platforms like CPU.

Conference'17, July 2017, Washington, DC, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Various works explore energy efficient hardware designs for CNN accelerators. One kind of researches base on CMOS technology and focus on efficient data path and memory system designs, for example the data tiling strategy in [21] and the convolution kernel in [17][8]. In these designs, the on-chip memory is implemented with SRAM, which means the size is quite limited. Thus external memory like DRAM is always needed in real applications, which means a large energy cost on the off-chip data transfer. This energy cost greatly limits the energy efficiency of this kind of design.

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. proposes the PRIME [4] architecture which implements the matrix vector multiplication directly with the RRAM cross bar. Work by Cheng et al. [3] implements the RRAM cross bar to support not only the inference but also the training phase of neural networks. In memory computation with RRAM shows great energy efficiency. But the application range of the design is limited by the scalability and computation accuracy of the RRAM cross bar.

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 1 according to the performance reported in [9, 20] and simulation result by NVSim [7]. Compared with SRAM, the storage density of RRAM is similar to DRAM, which is about 20× higher. Compared with DRAM, RRAM can be integrated on-chip while the former one can not.

Table 1: 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. The first is the small bandwidth. 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.

The second is the high I/O dynamic energy cost. In this paper, we show that a simple implementation with RRAM increase the total system energy cost. But there is 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 is 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 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 reduces up to 96% on-chip RAM access energy and 98% off-chip data transfer.

The rest of this paper is organized as follows: Section 2 introduces the related work for CNN accelerator and RRAM research. Section 3 introduces the hardware design. The scheduling strategies are introduced in section 4. The experimental results are shown in section 5. Section 6 concludes this paper.

2 RELATED WORK

2.1 Convolutional Neural Network

Convolutional neural networks(CNNs) mainly consist 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 denotes 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($F_{in}, F_{out}, K, b$)
// output channel loop
for $m = 1 \to M$
    // Input channel loop
    for $n = 1 \to N$
        // feature map pixel loop
        for each $F_{out}[m][x][y] \in F_{out}[m]$
            $F_{out}[m][x][y] = 0$
            // convolution kernel loop
            for each $K_{mn}[xx][yy]$
                $F_{out}[m][x][y] += K_{mn}[xx][yy] * F_{in}[n][x+xx][y+yy]$
            $F_{out}[m][x][y] += b_m$
```

CNNs are both computation and memory intensive. Table 2 shows the number of parameter and the computation complexity of some state-of-the-art network models. Usually a CNN consists of 10 ∼ 100 million parameters with most of them in the fully connected layers. The computation complexity varies from 1 ∼ 50G operations for each inference with most of them in the convolution layers. This means the convolution layers are computation limited while the fully connected layers are bandwidth limited. The large size of parameter and high computation complexity also brings challenge for energy efficient CNN processing.

Table 2: Computation complexity and parameter size of state-of-the-art networks.

	Computation (GOP)			Parameter (M)		
	CNV	FC	Total	CNV	FC	Total
AlexNet [15]	1.33	0.12	1.45	2.33	58.62	60.95
VGG-11 [19]	14.97	0.25	15.22	9.22	123.63	132.85
VGG-16 [19]	30.69	0.25	30.94	14.71	123.63	138.34
ResNet-34 [13]	7.28	0.001	7.281	21.1	0.51	21.61

2.2 CNN Accelerator

The design of a CNN Accelerator involves two aspects: computation units and scheduling strategy. For computation units, various researches have been carried out to reduce the bitwidth used for the data expression in CNN as this reduce both the memory requirement and energy cost for computation and data transfer. Experimental results from [10] shows that 8-bit linear data quantization introduces negligible accuracy loss for common CNN models compared with the original 32-bit floating-point model. Nonlinear quantization can further reduce the bitwidth [11] but needs decoding logic and computation units with higher accuracy [12];

On scheduling aspect, most of the CNN accelerators [21][17][16] implement single layer designs, where all the computation units work for the same layer at any time. Different layers are executed one by one. As concluded in [16], one this kind of design should focus on three aspects: loop tiling, loop unrolling and loop interchange. 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. This is unnecessary because the result of one layer is the input of the next layer. Alwani, et al. [2] suggests that adjacent layers can be fused together as a schedule unit to reduce this overhead.

As suggested by [14], typical energy for a 32bit DRAM access is 100× more than that of a 32bit SRAM access, 200× more than a 32bit multiplication and 6400× more than a 32bit addition. This indicates that the memory access energy contributes a major part of the total system cost. Reducing data bitwidth will further increase the memory access cost as the energy for multiplication We also show this conclusion with our experiments later in this paper.

2.3 RRAM

Metal-oxide resistive random access memory (RRAM) is one of the promising solution for integrating large memory on-chip. The cell size of RRAM is typically $4 \sim 6F^2$ while that for SRAM is more than $100F^2$. Some works [5][6] even try to store multiple bits within a cell. Despite the high storage density, the limited bandwidth is always a consideration for its usage as on-chip memory. But recent work [9] shows that even high density array of 32Gb achieves 1GB/s read bandwidth and 200MB/s write bandwidth. In this work, we adopt [7] as a tool to generate RRAM modules of different sizes in our experiments.

3 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 release the RRAM buffer dynamic energy cost.

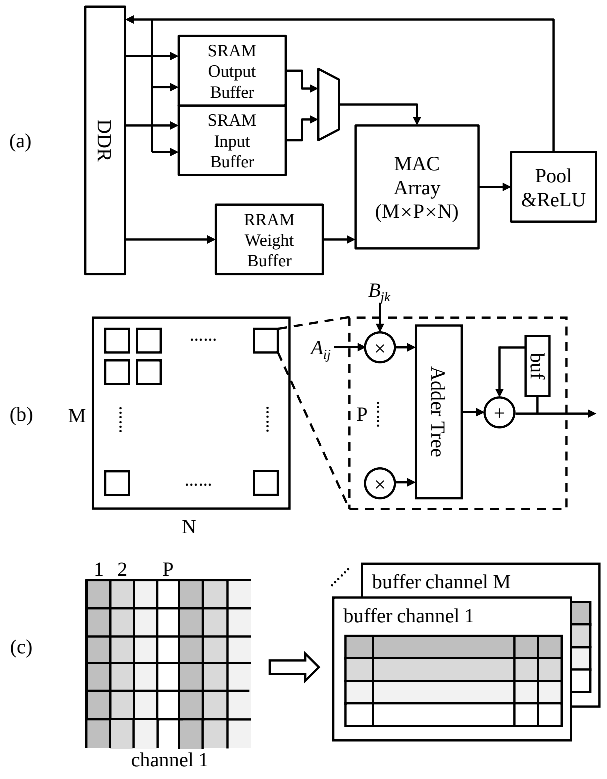

Figure 1: The CNN accelerator architecture. (a) The architecture mainly consists of a MAC array for computation and two levels of memory with on-chip buffers and external DDR. (b) MAC array structure for an $A_{p\times m} \times B_{m\times n}$ matrix-matrix multiplication. (c) Feature map buffer organization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

3.1 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: 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 works 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 $A_{p\times m} \times B_{m\times n}$ 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 $p \times n$ 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 the case where weight buffer is implemented with RRAM.

3.2 Choosing the Loop Order

With the parallelization strategy decided as introduced in section 3.1, the behavior of data access 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: kerel 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 convoution is explored. Each feature map pixel is accessed only once. Pixel first order reuse 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 3.

Table 3: 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 show in Figure 1(b) to enable n pixel groups processed together before written back to input/output buffer. This introduce extra on-chip memory cost. We examine the area and energy overhead in our experiment.

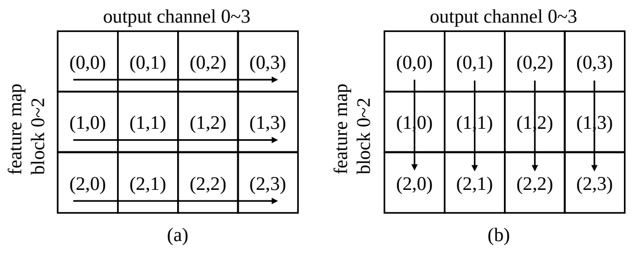

Figure 2: An example of how loop order affects data transfer behavior. Block (i, j) denotes the computation for the i^{th} feature map block on the j^{th} channel. (a) Reuse feature map first and load each weight 3 times. (b) Reuse weight first and load each feature map 4 times.

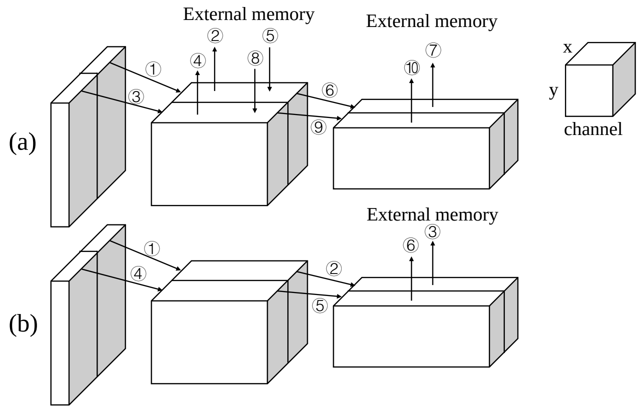

Figure 3: An example of a cross layer schedule over 3 layers. (a) Single layer schedule. (b) Cross layer schedule.

4 SCHEDULING STRATEGY

In the previous section, we choose the loop execution order to reduce data access. In this section, we analyze schedule strategy, which decides the off-chip data transfer behavior during the process of a network. Three kinds of schedule is 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 difference on data transfer behavior and further affects the data transfer energy and data transfer time.

Cross Layer Schedule Strategy. As suggested by [2], 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 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 a depth-$(n + 1)$ binary tree. 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)
```

5 EXPERIMENTS

Experiments are carried out on the architecture introduced in Section 3, with the scheduling strategy in section 4 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.

5.1 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. 8bit multiplication and 32bit accumulation is 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 [7] with different memory size configuration. To achieve enough bandwidth, RRAM buffer bit width is configured as 256bit.

Table 4: 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

The external memory parameter is generated from MICRON DDR4 power calculator [1]. 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. In order to reduce background power overhead, we use the least number of chips. All the detailed data and configuration is shown in Table 4.

The buffer in the accumulator is also considered in our experiments. 4 types of buffer is chosen for design space exploration. Corresponding parameters are generated by NVSim and are shown in Table 5.

Table 5: 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

5.2 Energy Cost Analysis

We do experiments on all the combinations of the RAM configurations in Table 4 and 5, which means $5 \times 5 \times 5 = 125$ choices for a SRAM or RRAM based accelerator. The three schedule strategies in section 4 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 for computation and DDR leakage is marked. These energy cost 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 is able to 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 on 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 brough 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 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 on 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 respecitvely when the weight buffer increases from 1MB to 16MB. With the fix weight strategy, the energy saving is 98.5%.

6 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 on-chip buffer to further reduce off-chip memory access based on existing scheduling strategies. Experimental results show that this strategy reduces 85% more

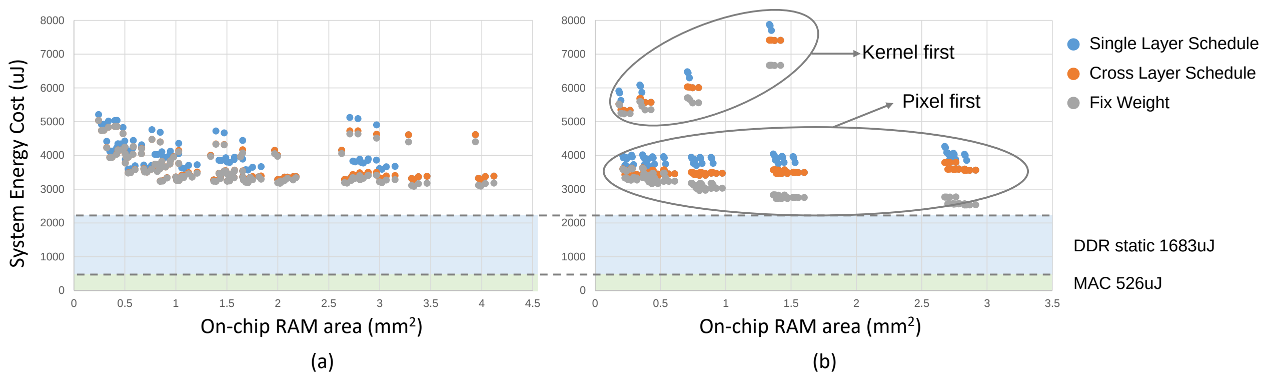

Figure 4: Design space exploration on different hardware choices and schedule strategies on the convolution layers of VGG-11 model. (a) SRAM weight buffer design. (b)RRAM weight buffer design.

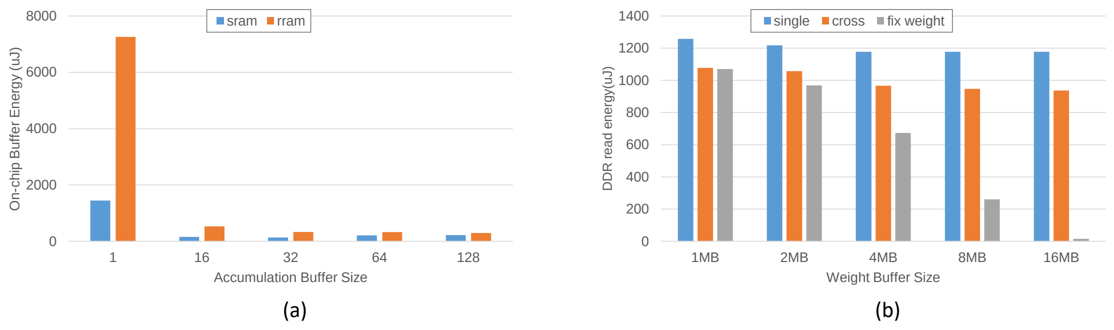

Figure 5: (a) On-chip buffer energy cost for a series of designs only differs in accumulation buffer size. (b) DDR access energy cost for a series of designs only differs in weight buffer size.

DDR read as the weight buffer size increases when compared with existing scheduling strategy. With the hardware and scheduling optimiation, 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.

REFERENCES

[1] [n. d.]. Power Calc. ([n. d.]). Retrieved December 21, 2017 from https://www.micron.com/~/media/documents/products/power-calculator/ddr4_power_calc.xlsm?la=en
[2] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–12.
[3] Ming Cheng, Lixue Xia, Zhenhua Zhu, et al. 2017. TIME: A Training-in-memory Architecture for Memristor-based Deep Neural Networks. In *DAC*. ACM, 26.
[4] Ping Chi, Shuangchen Li, Cong Xu, et al. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ISCA*. IEEE Press, 27–39.
[5] WC Chien, YC Chen, KP Chang, EK Lai, YD Yao, P Lin, J Gong, SC Tsai, SH Hsieh, CF Chen, et al. 2009. Multi-level operation of fully CMOS compatible WOx resistive random access memory (RRAM). In *Memory Workshop, 2009. IMW'09. IEEE International*. IEEE, 1–2.
[6] Wei-Chih Chien, Ming-Hsiu Lee, Feng-Ming Lee, et al. 2011. Multi-level 40nm WO X resistive memory with excellent reliability. In *Electron Devices Meeting (IEDM), 2011 IEEE International*. IEEE, 31–5.
[7] Xiangyu Dong, Cong Xu, Norm Jouppi, and Yuan Xie. 2014. NVSim: A circuit-level performance, energy, and area model for emerging non-volatile memory. In *Emerging Memory Technologies*. Springer, 15–50.
[8] Zidong Du, Robert Fasthuber, Tianshi Chen, et al. 2015. ShiDianNao: shifting vision processing closer to the sensor. In *ISCA*. ACM, 92–104.
[9] Richard Fackenthal, Makoto Kitagawa, Wataru Otsuka, et al. 2014. 19.7 A 16Gb ReRAM with 200MB/s write and 1GB/s read in 27nm technology. In *ISSCC*. IEEE, 338–339.
[10] Kaiyuan Guo, Song Han, Song Yao, et al. 2017. Software-Hardware Codesign for Efficient Neural Network Acceleration. *IEEE Micro* 37, 2 (2017), 18–25.
[11] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
[12] Han, Song and Liu, Xingyu and Mao, Huizi and others. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *ISCA*. 243–254.
[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385* (2015).
[14] M. Horowitz. [n. d.]. Energy table for 45nm process, Stanford VLSI wiki.[Online]. https://sites.google.com/site/seecproject. ([n. d.]).
[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.
[16] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2017. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In *FPGA*. ACM, 45–54.
[17] Jiantao Qiu, Jie Wang, Song Yao, et al. 2016. Going deeper with embedded fpga platform for convolutional neural network. In *FPGA*. ACM, 26–35.
[18] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2015. You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640* (2015).
[19] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[20] Shimeng Yu. [n. d.]. Overview of Memory Technology. ([n. d.]). Retrieved June 15, 2017 from https://www.coursehero.com/file/15370931/EEE-598-Section-1-Overview-of-Memory-Technology/
[21] Chen Zhang, Peng Li, Guangyu Sun, et al. 2015. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *FPGA*. ACM, 161–170.