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 on the effect of using RRAM as on-chip buffer is carried out for a typical CNN accelerator. With the proposed hardware optimization and schedule strategy optimization, DDR access energy is reduced by $25\% \sim 98\%$ and on-chip buffer dynamic energy is reduced by 86% on a state-of-the-art CNN model.**

I. Introduction

Convolutional Neural Network (CNN) has become a state-of-the-art algorithm for a wide range of applications like image classification [1] [2] , object detection [3] 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 [4] consists of more than 240MB parameters and 1.4GFLOPs for the inference of a single 224×224 image. More advanced networks [1] [2] requires 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. 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 [5] and the convolution kernel in [6] [7]. 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 [8] architecture which implements the matrix vector multiplication directly with the RRAM cross bar. Work by Cheng et al. [9] 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 I according to the performance reported in [10]–[12]. 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	low	high	high

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 runtime. 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.

- The data access pattern is static.
- The weight sharing idea of CNN means we need much less memory access than computation.
- The high storage capacity of RRAM helps reduce memory refresh.

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 RRAM buffer.
- Design space exploration is done with state-of-the-art networks to show the effect of using RRAM as on-chip buffer.

Experimental results show that DDR access energy is reduced by $25\% \sim 98\%$ and on-chip buffer dynamic energy is

reduced by 86% on a state-of-the-art CNN model with the proposed methods. Introducing RRAM as on-chip buffer, the total energy cost of the system is reduced by 18% compared with the optimal SRAM design.

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 schedule methods 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

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)

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

CNNs are both computation and memory intensive. Table II shows the number of parameter and the computation complexity of some state-of-the-art network models. Usually a CNN consists of $10 \sim 100$ million parameters with most of them in the fully connected layers. The computation complexity varies from $1 \sim 50$G 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.

B. 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 [13] 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 [14] but needs decoding logic and computation units with higher accuracy [15];

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

	Computation (GOP)			Parameter (M)		
	CNV	FC	Total	CNV	FC	Total
AlexNet [4]	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 [2]	7.28	0.001	7.281	21.1	0.51	21.61

On scheduling aspect, most of the CNN accelerators [5] [6] [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. [17] suggests that adjacent layers can be fused together as a schedule unit to reduce this overhead.

As suggested by [18], typical energy for a 32bit DRAM access is $100\times$ more than that of a 32bit SRAM access, $200\times$ more than a 32bit multiplication and $6400\times$ 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.

C. 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 [19] [20] 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 [11] shows that even high density array of 32Gb achieves 1GB/s read bandwidth and 200MB/s write bandwidth. In this work, we adopt [12] as a tool to generate RRAM modules of different sizes in our experiments.

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 release 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

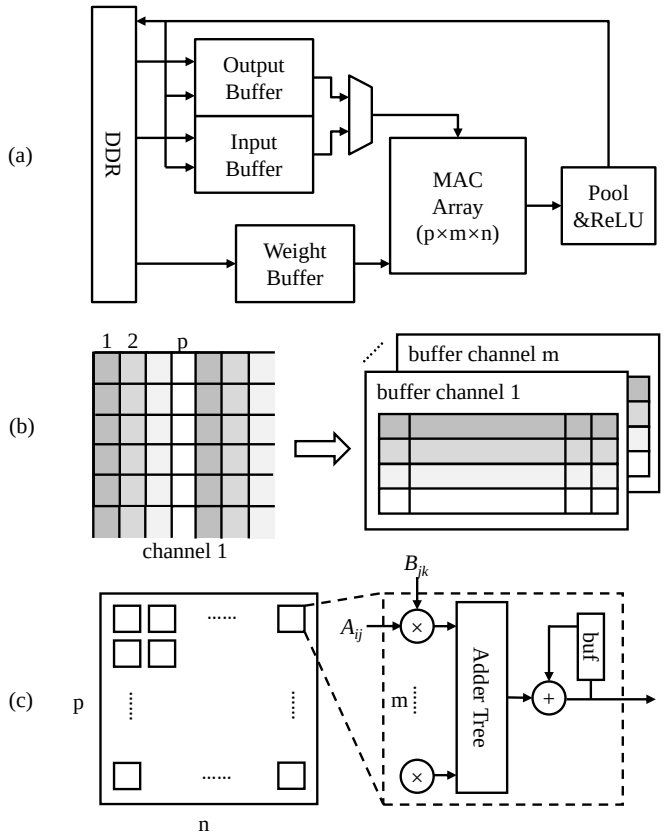

Fig. 1: The CNN accelerator architecture. (a) The architecture mainly consists of a MAC array for computation and two levels of memory with on-chip buffers and external DDR. (b) Feature map buffer organization for sliding window data selection. (c) MAC array structure for an $A_{p\times m} \times B_{m\times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

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.

To support the above function, 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(b). 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.

The structure of the computation core is shown in Figure 1(c). 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.

B. Energy Model

We only focuses on memory access energy, computation energy and background energy in this work.

For the memory access energy, we count the number of data read from or write to each buffer and multiplies them with a statistical energy cost generated from simulation tools. This highly relates to the schedule strategy because it decides how data is reused during the process of a CNN.

For the computation energy, we calculate it by multiply the average energy cost of one operation, i.e. addition and multiplication, with the total number of operations in a network. So this part is fixed regardless of the schedule strategy or hardware design.

For the background energy, we sum all the background power and multiplies it with the total processing time. The processing time of a network is given layer by layer, as the longer one between off-chip data transfer time and computation time.

With the hardware design and the target network given, the computation energy cannot be optimized. Neither the background energy can be optimized if the system is not bounded by bandwidth. So we mainly focus on off-chip data transfer energy, which is decided by schedule strategy and the on-chip buffer energy for computation.

C. RRAM Buffer and Hardware Optimization

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.

As RRAM has high read/write dynamic, we should try to reuse the weight to reduce access to weight buffer. Here we use a small buffer rather than a register for the accumulation in PE as shown in Figure 1(c). Each time a weight is read from weight buffer, we can utilize the buffer to store intermediate result for different output pixels before moving on to the next weight. With a buffer of size d, the time a weight will be read from weight buffer is expressed as equation 1

$$N_w = \lceil \lceil \frac{F_{out}.x * F_{out}.y}{p} \rceil / d \rceil \tag{1}$$

Though this method reduce weight access to approximately $1/d$, more buffer energy is needed as d increases. We search for the optimal point in our experiment.

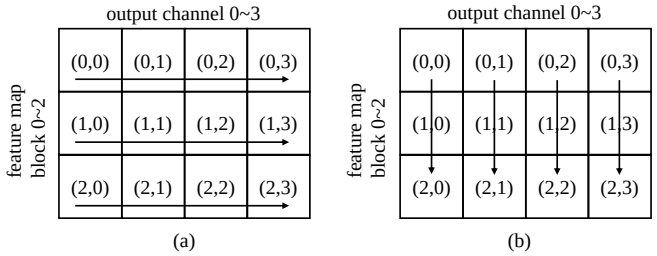

Fig. 2: An example of how loop order affects data transfer behavior. Block (i, j) denotes the computation for the i^{th} feature map block on the j^{th} channel. (a) Reuse feature map first and load each weight 3 times. (b) Reuse weight first and load each feature map 4 times.

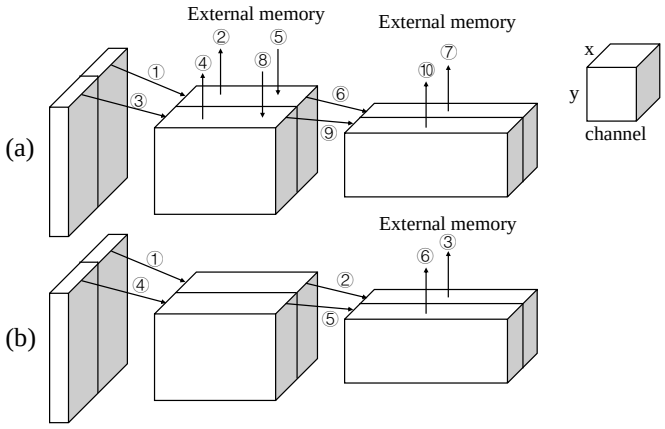

Fig. 3: An example of a cross layer schedule over 3 layers. (a) Single layer schedule. (b) Cross layer schedule.

IV. Scheduling Strategy

In this section, we analyze schedule strategy, which decides the off-chip data transfer behavior during the process of a network. Three levels of schedule is considered: single layer, cross layer and network weight arrangement.

A. 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.

B. Cross Layer Schedule Strategy

As suggested by [17], 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.

C. Network Weight Arrangement

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 **if** $e1 < e2$
11 $f = f1$
12 **return** $e1$
13 **else**
14 $f = f2$
15 **return** $e2$

V. Experiments

Experiments are carried out on the architecture introduced in Section III, with the scheduling strategy in section IV applied. Design space exploration is also done to optimize the energy cost of a state-of-the-art network.

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. 8bit multiplication and 32bit accumulation is adopted in this model. Multiplication and addition energy is scaled down from the data in [18] to 22nm technology.

The above configuration requires the read bandwidth of input/output buffer and weight buffer to be at least 64GB/s. We

TABLE III: 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$\hat{2}$)
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

implement each buffer with 8 banks and each of them should offer 8GB/s read bandwidth. On-chip memory parameters are generated from NVSim [12] with different memory size configuration. To achieve enough bandwidth, RRAM buffer bit width is configured as 256bit.

The external memory parameter is generated from MICRON DDR4 power calculator [21]. 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 III.

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 IV

TABLE IV: 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

B. DDR Access energy optimization

DDR access behavior is totally decided by schedule strategy. Here we use the convolution layers of VGG-11 to test the effectiveness of the schedule strategy optimization. Figure 4 shows the experimental results. Input/output buffer is 128KB SRAM and weight buffer varies from 1MB to 16MB RRAM in this experiment. Three levels of optimization is compared: only single layer schedule, cross layer schedule, cross layer schedule and consider fixing weight in buffer. When the weight buffer is small, the main energy saving comes from the cross layer schedule. When the buffer gets larger, fixing weight becomes more important. Up to 20% energy saving can be achieved by the our schedule strategy optimization. From Figure 4(b), we see that strategy optimization reduces the DDR access energy by $25\% \sim 98\%$ under different memory configurations.

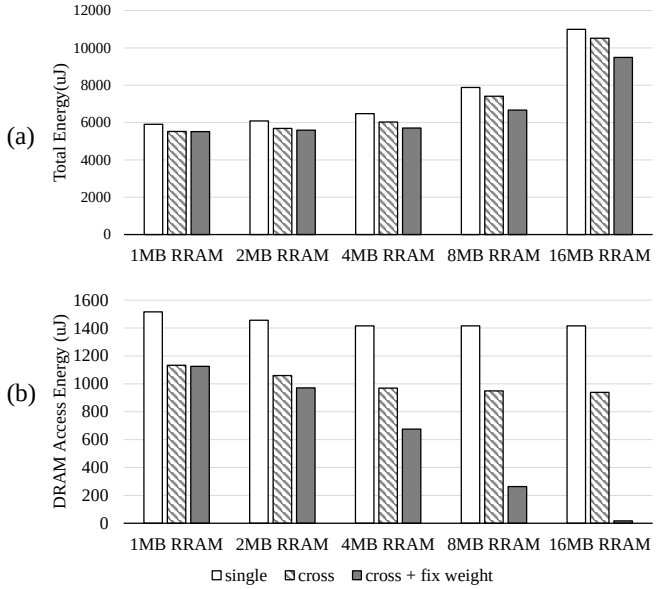

Fig. 4: System energy with different level of schedule optimization. (a) Total system energy. (b)DRAM access energy.

C. On-chip buffer energy optimization

From Figure 4(a), we see that using a larger buffer is not better. This is caused by the increasing buffer I/O energy and leakage power. An example of energy breakdown is shown in Table V. Both of the design adopts 1MB SRAM I/O buffer. The SRAM weight buffer is 128KB and the RRAM weight buffer is 1MB. Though the RRAM design has less DDR access energy because of a larger weight buffer, the weight buffer read energy is much higher. This shows the necessity of using buffer, rather than a single register, for accumulation.

We still use the VGG-11 convolution layers to search for an optimal accumulator buffer depth. For each depth choice, we search the optimal buffer size configuration with all the schedule strategy optimization applied. The results are shown in Figure 5. As can be seen from the figure, both methods benefits from the accumulation buffer, but the RRAM design benefits more. With the accumulation buffer, the total on-chip buffer energy cost can be reduced by 86%. Combining schedule optimization and accumulation buffer design, we can finally achieve 18% energy reduction by introducing RRAM

TABLE V: Example energy breakdown for a pair of SRAM and RRAM design

	RRAM weight buffer		SRAM weight buffer	
read iobuf	0.29	0.01%	0.29	0.01%
read weight	2119	40.50%	383	10.85%
read ddr	841	16.10%	929	26.34%
write iobuf	1.04	0.02%	1.04	0.03%
write weight	50.4	0.96%	0.64	0.02%
write ddr	2.08	0.04%	2.08	0.06%
standby	1691	32.33%	1686	47.79%
calculate	526	10.05%	526	14.90%
Total	5230		3527	

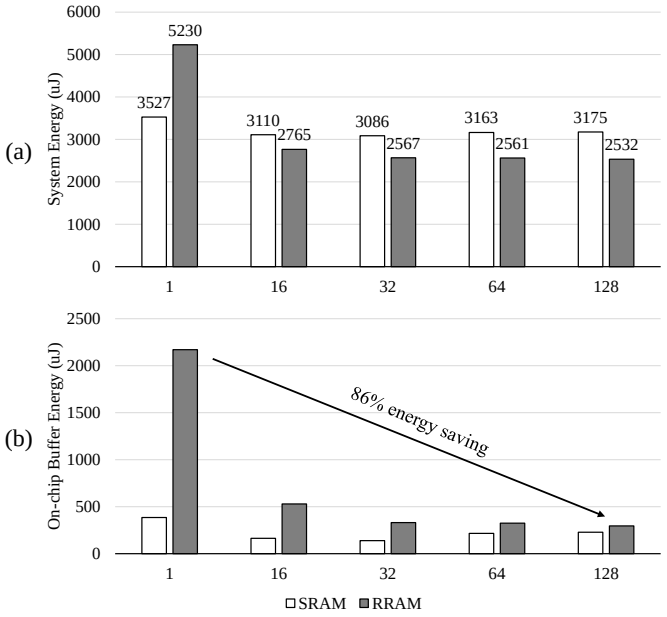

Fig. 5: System energy with different accumulation buffer size. (a) System energy cost. (b) On-chip buffer energy cost.

into on-chip buffer design.

VI. Conclusion

In this paper, we try to introduce RRAM into CNN accelerator on-chip buffer design. The accumulation buffer is introduced to relief the I/O energy overhead brought by RRAM. A set of schedule strategy optimizations is proposed to fully utilize the high storage density of RRAM. Experimental results show that by combining the hardware and software optimization and do design space exploration, $25\% \sim 98\%$ DRAM access energy and 86% on-chip buffer energy can be saved. RRAM based design achieves 18% less system energy compared with an SRAM based design. Future work should focus on the rest part of the energy cost like the standby energy and the computation energy.

References

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

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

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

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

[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] Z. Du, R. Fasthuber, T. Chen *et al.*, "Shidiannao: shifting vision processing closer to the sensor," in *ISCA*. ACM, 2015, pp. 92–104.

[8] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 2016, pp. 27–39.

[9] M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, and H. Yang, "Time: A training-in-memory architecture for memristor-based deep neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 26.

[10] "Overview of memory technology," https://www.coursehero.com/file/15370931/EEE-598-Section-1-Overview-of-Memory-Technology/.

[11] R. Fackenthal, M. Kitagawa, W. Otsuka, K. Prall, D. Mills, K. Tsutsui, J. Javanifard, K. Tedrow, T. Tsushima, Y. Shibahara *et al.*, "19.7 a 16gb reram with 200mb/s write and 1gb/s read in 27nm technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 2014, pp. 338–339.

[12] 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.

[13] K. Guo, S. Han, S. Yao, Y. Wang, Y. Xie, and H. Yang, "Software-hardware codesign for efficient neural network acceleration," *IEEE Micro*, vol. 37, no. 2, pp. 18–25, 2017.

[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[15] Han, Song and Liu, Xingyu and Mao, Huizi and Pu, Jing and Pedram, Ardavan and Horowitz, Mark A. and Dally, William J., "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, 2016, pp. 243–254.

[16] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 45–54.

[17] 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.

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

[19] 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.

[20] W.-C. Chien, M.-H. Lee, F.-M. Lee, Y.-Y. Lin, H.-L. Lung, K.-Y.

Hsieh, and C.-Y. Lu, "Multi-level 40nm wo x resistive memory with excellent reliability," in *Electron Devices Meeting (IEDM), 2011 IEEE International*. IEEE, 2011, pp. 31–5.

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