2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture

RANA: Towards Efficient Neural Acceleration with Refresh-Optimized Embedded DRAM

Fengbin Tu, Weiwei Wu, Shouyi Yin*, Leibo Liu, Shaojun Wei
Beijing National Research Center for Information Science and Technology (BNRist)
Institute of Microelectronics, Tsinghua University, Beijing, China
Email: yinsy@tsinghua.edu.cn

***Abstract*—The growing size of convolutional neural networks (CNNs) requires large amounts of on-chip storage. In many CNN accelerators, their limited on-chip memory capacity causes massive off-chip memory access and leads to very high system energy consumption. Embedded DRAM (eDRAM), with higher density than SRAM, can be used to improve on-chip buffer capacity and reduce off-chip access. However, eDRAM requires periodic refresh to maintain data retention, which costs much energy consumption.**

Refresh is unnecessary if the data's lifetime in eDRAM is shorter than the eDRAM's retention time. Based on this principle, we propose a Retention-Aware Neural Acceleration (RANA) framework for CNN accelerators to save total system energy consumption with refresh-optimized eDRAM. The RANA framework includes three levels of techniques: a retention-aware training method, a hybrid computation pattern and a refresh-optimized eDRAM controller. At the training level, CNN's error resilience is exploited in training to improve eDRAM's tolerable retention time. At the scheduling level, RANA assigns each CNN layer with a computation pattern that consumes the lowest energy. At the architecture level, a refresh-optimized eDRAM controller is proposed to alleviate unnecessary refresh operations. We implement an evaluation platform to verify RANA. Owing to the RANA framework, 99.7% eDRAM refresh operations can be removed with negligible performance and accuracy loss. Compared with the conventional SRAM-based CNN accelerator, an eDRAM-based CNN accelerator strengthened by RANA can save 41.7% off-chip memory access and 66.2% system energy consumption, with the same area cost.

***Keywords*-Neural Network; Embedded DRAM (eDRAM); Refresh Optimization; Retention Time**

I. Introduction

Convolutional neural networks (CNNs) are widely used in modern artificial intelligence applications, like image classification, object detection and video surveillance, with unprecedented accuracy. However, to achieve higher accuracy, CNN's size keeps growing and produces large amounts of intermediate data storage. Table I lists the maximum layer storage requirements in four typical CNN models [1–4]. Their layer's input/output/weight storage requirement reaches 0.3∼6.27MB in 16-bit precision, for the standard ImageNet [5] input image size (224×224×3). The numbers will greatly increase when the networks process higher resolution images.

Table I
Data Storage Requirements of CNNs (16-bit)

CNN Models Input (224×224×3)	Max. Layer Inputs	Max. Layer Outputs	Max. Layer Weights
AlexNet [1]	0.30MB	0.57MB	1.73MB
VGG [2]	6.27MB	6.27MB	4.61MB
GoogLeNet [3]	0.39MB	1.57MB	1.30MB
ResNet [4]	1.57MB	1.57MB	4.61MB

Table II
Characteristics Comparison of SRAM and eDRAM (32KB, in the 65nm Technology Node)

	SRAM	eDRAM
Data Storage	Latch	Capacitor
Area	$0.181mm^2$	$0.047mm^2$
Access Latency	1.730ns	1.541ns
Access Energy	1.139pJ/bit	0.662pJ/bit
Refresh Energy	-	0.788μJ/bank
Retention Time	-	$< 100\mu s$ ($45\mu s$ in [6])
Features	(+) Fast (-) Large area	(+) Small area (-) Refresh

Many CNN accelerators [7–11] have been designed to enable highly energy-efficient CNN processing, but the small footprint limits the on-chip memory capacity. Their on-chip memory size is usually no more than 500KB with area cost of $3 \sim 16mm^2$ (normalized to 65nm). Extra off-chip memory access is required when they compute the CNN models in Table I.

Embedded DRAM (eDRAM), known for its high density, is adopted into modern CNN accelerators to provide large on-chip memory capacity [12–15]. As shown in Table II, eDRAM's area cost is 26.0% of SRAM in the 65nm technology node (simulated with Destiny [16]). However, as an eDRAM cell stores data as charge on a capacitor, the charge will leak over time and cause retention failures. Thus, eDRAM cells need periodic refresh to maintain data retention. The refresh interval usually equals to the weakest cell's retention time ($45\mu s$ in [6]). We simulate ResNet

*Corresponding author: Shouyi Yin (yinsy@tsinghua.edu.cn).

2575-713X/18/$31.00 ©2018 IEEE
DOI 10.1109/ISCA.2018.00037

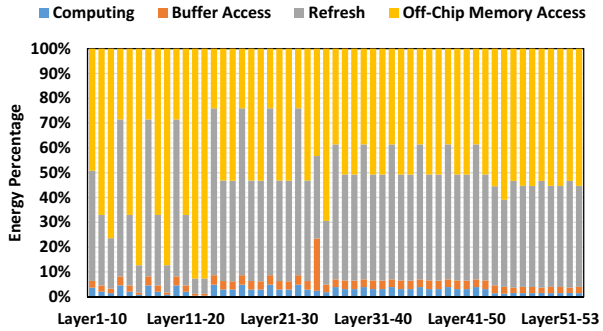


Figure 1. Energy consumption breakdown of ResNet on the evaluation platform.

[4] on our evaluation platform with eDRAM buffers, and find out that eDRAM's refresh energy takes up a quite large part in the total system energy, as presented in Figure 1. Although large eDRAM buffers alleviate many off-chip memory access, the refresh energy consumption becomes a new problem that we can't neglect.

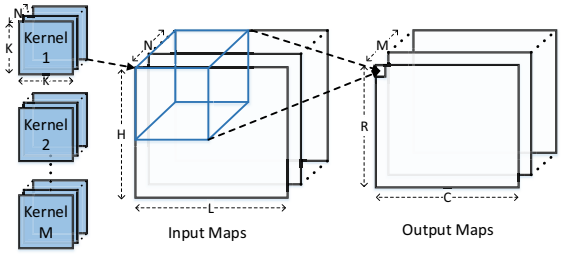
We find out that, refresh is no longer required if the data's lifetime in eDRAM buffers is shorter than the eDRAM cell's retention time. In this paper, we propose a Retention-Aware Neural Acceleration (RANA) framework that strengthens CNN accelerators with refresh-optimized eDRAM, to save total system energy consumption. The framework solves two problems - the buffer storage problem caused limited on-chip memory capacity, and the data retention problem produced by eDRAM refresh. RANA includes three levels of optimization techniques:

(1) **Training Level**: A retention-aware training method is proposed to improve eDRAM's tolerable retention time with no accuracy loss. Bit-level retention errors are injected during training, so the network's tolerance to retention failures is improved. A higher tolerable failure rate leads to longer tolerable retention time, so more refresh can be removed.

(2) **Scheduling Level**: A system energy consumption model is built in consideration of computing energy, on-chip buffer access energy, refresh energy and off-chip memory access energy. RANA schedules networks in a hybrid computation pattern based on this model. Each layer is assigned with the computation pattern that costs the lowest energy.

(3) **Architecture Level**: RANA independently disables refresh to eDRAM banks based on their storing data's lifetime, saving more refresh energy. A programmable eDRAM controller is proposed to enable the above fine-grained refresh controls.

The rest of this paper is organized as follows: Section II provides a preliminary on CNN, CNN accelerators and eDRAM. In Section III, we build an evaluation platform for energy analysis, and point out the opportunities to optimize



(a) CONV layer in CNNs.

```
for(r = 0; r < R; r + +)                    // Loop R.
  for(c = 0; c < C; c + +)                  // Loop C.
    for(m = 0; m < M; m + +)                // Loop M.
      for(n = 0; n < N; n + +)              // Loop N.
        O[m][r][c] +=                       // Convolution.
          Σ_{i=0}^{K-1} Σ_{j=0}^{K-1} W[m][n][i][j] * I[n][r * S + i][c * S + j];
```

(b) Pseudo code of a CONV layer.

Figure 2. Illustration for a convolutional (CONV) layer.

overall system energy consumption. We then propose the RANA framework in Section IV, including three levels of optimization techniques. Section V presents the experimental results of RANA on the evaluation platform. Section VI concludes this paper.

II. Preliminary

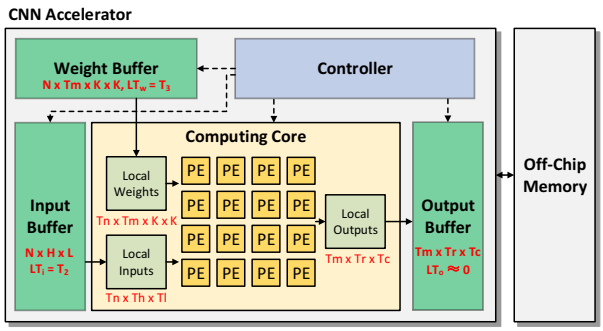
A. Convolutional Neural Network (CNN)

Most CNNs are mainly composed of three types of layers: convolutional (CONV) layers, pooling layers and full connection layers. CONV layers extract different levels of features from the input image. Pooling layers follow CONV layers and perform sub-sampling onto the feature maps. Full connection layers are usually used in the last few layers and act as a classifier. In this paper, our discussion is focused on acceleration for CONV layers, for two reasons: 1) CONV layers account for the majority of CNN's runtime [17, 18]; 2) Other layers can be transformed to execute in a similar way with the CONV layer acceleration [11, 19–21].

Figure 2(a) illustrates a CONV layer in CNN. It takes $N \times H \times L$ feature maps as the inputs, and has M 3D CONV kernels ($N \times K \times K$). Each kernel performs a 3D convolution on the input maps with a sliding stride of S, which generates a $R \times C$ output map. Therefore, the output map number equals to the kernel number M. The computation of a CONV layer can be expressed as the multi-level loop in Figure 2(b), where Matrice I, O and W stand for the input maps, output maps and kernel weights, respectively. The basic operation of a CONV layer is multiply-accumulation, *i.e.* the inner-most loop in Figure 2(b).

B. CNN Accelerator

Figure 3(a) shows a typical CNN accelerator architecture used in many hardware CNN implementations like



(a) Typical CNN accelerator architecture.

```
for(m = 0; m < M; m += Tm)                          // Loop M.        ┐
  for(r = 0; r < R; r += Tr)                        // Loop R.        │ Memory Control
    for(c = 0; c < C; c += Tc)                      // Loop C.        │ Part
      for(n = 0; n < N; n += Tn)                    // Loop N.        ┘
        for(tm = m; tm < min(m + Tm, M); tm + +)    // Loop Tm.       ┐
          for(tn = n; tn < min(n + Tn, N); tn + +)  // Loop Tn.       │
            for(tr = r; tr < min(r + Tr, R); tr + +)// Loop Tr.       │ Core Computing
              for(tc = c; tc < min(c + Tc, C); tc + +)// Loop Tc.     │ Part
                O[tm][tr][tc] +=                    // Convolution.   │
                ∑_{i=0}^{K-1} ∑_{j=0}^{K-1} W[tm][tn][i][j] * I[tn][tr * S + i][tc * S + j];  ┘
```

(b) Pseudo code of a CNN accelerator [10].

Figure 3. Illustration for a CNN accelerator.

[7–11]. In the CNN accelerator, there are a controller, weight/input/output buffers and a computing core. Previous works usually use a layer-by-layer accelerating style to compute CNNs. That is, the layers in a network are sequentially mapped onto the accelerator. During runtime, the controller schedules the other components based different layers' configurations. The input buffer and weight buffer store the input maps and kernel weights of the current layer. Due to the limited local storage in the computing core, only part of the inputs and weights are loaded each time, and computed by the processing elements (PEs). PEs perform convolutions, activate and pooling functions to generate part of output feature maps. The partial results are cached in the output buffer. After many iterations, the final outputs are sent to the off-chip memory, and will be loaded again for the successive layer.

Figure 3(b) illustrates a typical computation pattern [10], which indicates how a CONV layer executes in a CNN accelerator. Compared with the original pseudo code in Figure 2(a), Loop R, C, M and N are tiled by Tr, Tc, Tm and Tn, due to the computing core's limited local storage. The inner-most four loops (Loop $Tm/Tn/Tr/Tc$) represent the core computing part: $N \times H \times L$ input feature maps are tiled by $Th \times Th \times Tl$, and then convolved by Tm 3D kernels ($Tn \times K \times K$) to generate $Tm \times Tr \times Tc$ output feature maps. The loops out of the gray box (Loop $M/R/C/N$) are scheduled by the controller, which works as follows:

(1) Loop N: All the **N** input channels are loaded into the core to compute the $Tm \times Tr \times Tc$ output feature maps.
(2) Loop RC: The whole $N \times H \times L$ input maps are loaded into the core to compute all the $Tm \times \mathbf{R} \times \mathbf{C}$ output map by repeating Loop N.
(3) Loop M: All the $\mathbf{M} \times R \times C$ output maps are computed by repeating Loop N and Loop RC.

The above process is the memory control part of the computation pattern. It only determines the memory access order and doesn't influence the core's computing.

Table III
ENERGY COST IN THE 65NM TECHNOLOGY NODE

Operation	Energy	Relative Cost
16-bit Fixed-Point MAC[a]	1.3pJ	1.0x
16-bit 32KB SRAM Access[b]	18.2pJ	14.3x
16-bit 32KB eDRAM Access[b]	10.6pJ	8.3x
16-bit 32KB eDRAM Refresh[b]	48.1pJ	37.7x
16-bit 1GB DDR3 Access[c]	2112.9pJ	1653.7x

[a] Extracted from the TSMC 65nm GP technology.
[b] Simulated with Destiny [16].
[c] Simulated with CACTI [26].

C. Memory Challenge for CNN Acceleration

Many previous CNN accelerators focus on optimizing the core computing part to achieve high performance in computing CONV layer [7, 9, 10, 22–25]. However, to achieve state-of-the-art accuracy, CNN's size grows significantly, leading to massive memory access in CNN acceleration. Table III shows the energy costs of basic arithmetic and memory operations, extracted through simulation in the 65nm technology node. Off-chip memory access (DDR3) costs orders of higher energy consumption than on-chip memory access and the MAC operation. Notice that we use DDR3 as the off-chip memory in this paper. Large CNN models may not fit in on-chip buffer capacity and hence require costly off-chip memory access. Therefore, off-chip memory access energy usually takes up the most in total system energy consumption, as pointed out in [7, 11, 19]. **This is the buffer storage problem, known as a big challenge for CNN acceleration.**

D. Embedded DRAM (eDRAM)

Embedded DRAM (eDRAM), with higher density than SRAM, is adopted into modern CNN accelerators to provide large on-chip capacity [12–15]. Figure 4 shows the typical structure of eDRAM. Each eDRAM cell is composed of an access transistor and a storage capacitor. The logic state is stored as electrical charge in the capacitor. The capacitor loses charge over time through the current I_{off} in the access transistor. Therefore, an eDRAM cell requires periodic refresh to maintain the correct logic state. Current eDRAM designs pessimistically assume the worst-case retention time, and end up refreshing all the eDRAM cells in a module at the same, conservatively-high rate. Previous works [27, 28] indicate that eDRAM refresh can be a bottleneck in total energy consumption. As shown in Figure 1, although large eDRAM buffers can help alleviate many off-chip memory

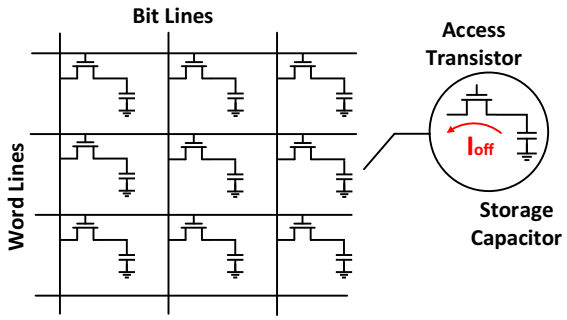


Figure 4. Embedded DRAM (eDRAM) Structure.

access, the refresh operations lead to a big cost of energy. **This is the data retention problem, caused by introducing eDRAM into CNN acceleration.**

III. Motivation

In this section, we build an evaluation platform to study the buffer storage and data retention problems in eDRAM-based CNN accelerators. Through the analysis, we point out three opportunities to optimize overall system energy consumption, leading to the RANA framework proposed in Section IV.

A. Evaluation Platform

The evaluation platform is built as described in Figure 5. We first implement a SRAM-based test CNN accelerator with totally 256 PEs working at 200MHz, where each PE has one multiply-accumulator (MAC) inside. With 384KB SRAM buffer capacity, the accelerator's area is 5.682mm^2 in the TSMC 65nm GP technology. The resource and area costs are both close to many moderate-sized CNN accelerators [7–11]. Our design is synthesized with Synopsys Design Compiler, whose layout is presented on the bottom of Figure 5. The core's computing logic is similar to a state-of-the-art CNN accelerator Envision [10]: The 256 PEs are organized in a 16x16 PE array, where the 16 rows of PEs share the same inputs to compute 16 output channels in parallel. The core's local storage reaches 36KB in total, so more data can be reused in the core to save buffer access [19]. The controller is designed to enable flexible reconfiguration of the memory control part, so we can study different computation patterns on the same evaluation platform. The input/output/weight buffers are organized as a unified buffer system to support flexible data mapping for different patterns. We run RTL-level cycle-accurate simulation on the SRAM-based test CNN accelerator, for performance estimation and memory access tracing. Thus, we can analyze different computation pattern's real performance, data lifetime and memory access behaviors.

According to Table II, 1.454MB eDRAM can be obtained with the same area cost of 384KB SRAM. Thus in our evaluation platform, we simulate eDRAM-based designs with 1.454MB eDRAM buffers, while other hardware parameters stay the same as the SRAM-based design. The eDRAM's retention time distribution is in accordance with that in [6], with typical retention time of 45μs. One layer in ResNet [4] (Layer-A: "res4a_branch1") and one layer in VGG [2] (Layer-B: "vgg_conv9") are selected for the running cases in our discussions.

Evaluation Platform Configurations	
CNN Accelerator	256 MACs, 384KB SRAM, 200MHz, 5.682mm², 65nm
eDRAM	1.454MB, retention time = 45μs, 65nm
Layer-A (res4a_branch1)	M = 1024, N = 256, R = C = 14, K = 1, S = 2
Layer-B (vgg_conv9)	M = 512, N = 256, R = C = 28, K = 3, S = 1

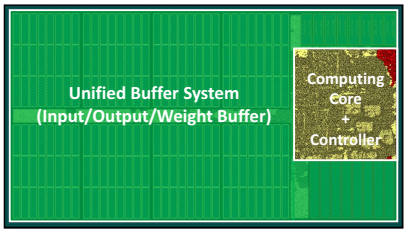


Figure 5. Evaluation platform configurations and the accelerator's layout.

B. Problem Analysis

1) Buffer Storage Problem: We begin with an in-depth study on the typical computation pattern described in Figure 3(b). Since Loop M is the outer-most loop, all the inputs should be stored on chip to be reused by all the M output feature maps, to avoid extra off-chip memory access. The outputs are computed out in a tile size of $Tm \times Tr \times Tc$, and only Tm kernels are needed for convolution. Therefore, the buffer storage requirements for inputs/outputs/weights (BS_i, BS_o, BS_w) can be calculated by:

$$BS_i = N \cdot H \cdot L \tag{1}$$

$$BS_o = Tm \cdot Tr \cdot Tc \tag{2}$$

$$BS_w = N \cdot Tm \cdot K^2 \tag{3}$$

Take Layer-A as an example, the minimum total buffer storage requirement BS equals to $BS_i+BS_o+BS_w$=785KB in 16bit-precision, which exceeds the SRAM-based accelerator's buffer capacity (384KB, $Tm, Tn, Tr, Tc = 1$). Within the same area, eDRAM-based buffers of 1.454MB can be placed in the design, so Layer-A's intermediate data can be cached on chip to avoid extra off-chip memory access. However, it's still possible that buffer capacity of 1.454MB cannot meet the storage requirement of too large layers according to Table I, producing extra off-chip memory access energy.

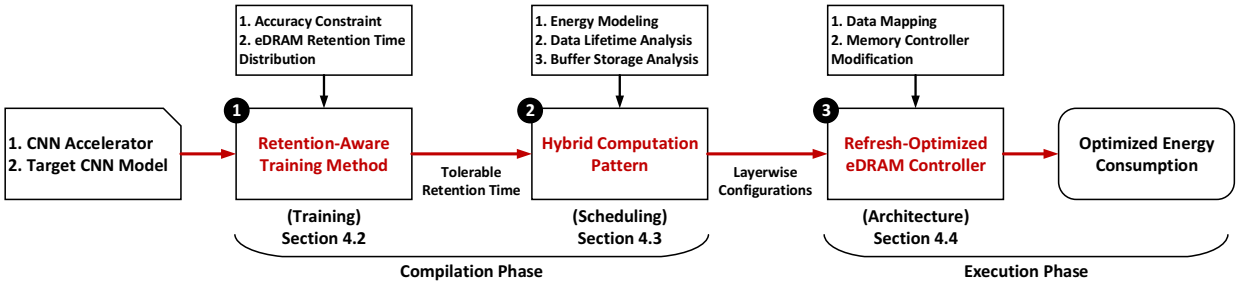


Figure 6. RANA Framework: Optimizations are introduced to the training, scheduling and architecture levels.

2) Data Retention Problem: Moreover, eDRAM-based buffers need periodic refresh to maintain data retention, causing extra energy consumption. As mentioned before, refresh can be avoid if the storing data's lifetime is shorter than the retention time. Thus, we take a further look at the computation pattern's input/output/weight lifetime. Since all the inputs should be stored on chip to for reuse, the inputs' lifetime in buffers equals to the total execution time of a layer:

$$LT_i = \frac{M \cdot N \cdot R \cdot C \cdot K^2}{MAC \cdot Frequency \cdot \eta} \tag{4}$$

, where MAC, $Frequency$ and η are the number of hardware MAC units (=256), working frequency (=200MHz) and PE utilization. As for weights, they are fully used during Loop RC, so their lifetime is

$$LT_w = \frac{Tm \cdot N \cdot R \cdot C \cdot K^2}{MAC \cdot Frequency \cdot \eta} \tag{5}$$

It's notable the outputs are kept accumulating in the PEs without communicating with the output buffer. They are only stored in the output buffer at the end of Loop M, when the Tm outputs complete all the accumulations. The outputs are quickly sent to the off-chip memory, so we regard their lifetime in the output buffer (LT_o) as 0. The accumulation of outputs is a good property that can be utilized to reduce lifetime. We will make a further discussion in Section IV-C1.

We run Layer-A on the test accelerator and measure the lifetime of inputs/outputs/weights: $LT_o < LT_w < LT_i = 2294\mu s$. Since the input lifetime exceeds eDRAM's retention time ($45\mu s$), refresh cannot be avoided, leading to the data retention problem.

C. Optimization Opportunities

In Section III-B, we have analyzed how the buffer storage and data retention problems come about in eDRAM-based CNN accelerators. The produced off-chip memory access energy and eDRAM energy are the two main sources of total system energy consumption, as shown in Figure 1. To solve the problems, the following two conditions should be met in CNN acceleration:

(1) **Buffer Storage Requirement** $<$ **Buffer Capacity**: If the buffer storage requirement is smaller than the accelerator's buffer capacity, no extra off-chip memory access is needed, reducing much off-chip energy consumption.
(2) **Data Lifetime** $<$ **Retention Time**: If the data lifetime is shorter than eDRAM's retention time, no refresh is required, which saves lots of on-chip energy consumption.

The buffer capacity can't be changed once the accelerator is designed, but the other three factors still have optimization opportunities. In Section IV, we will propose techniques to increase "retention time", reduce data lifetime and shrink buffer storage requirement.

IV. RANA Framework

A. Overview

Although eDRAM's high density increases on-chip buffer capacity, the buffer storage problem might still exist if the network is too large. Meanwhile, the data retention problem caused by eDRAM will produce considerable refresh energy consumption. In this section, we propose a Retention-Aware Neural Acceleration (RANA) framework that strengthens CNN accelerators with refresh-optimized eDRAM, to save total system energy consumption. Figure 6 is an overview of the whole framework. RANA takes the CNN accelerator and target CNN model's parameters as the inputs, and then produces a 3-stage workflow. The first two stages lie in the compilation phase, to generate configurations for the accelerator. The last stage is in the execution phase, where the accelerator runs based on the configurations, with optimized energy consumption.

Stage ❶ (Training): A retention-aware training method is used to train the target CNN model to improve its tolerance to retention failures. Retention failures are injected by adding bit-level error masking to each layer of the model. Under the given accuracy constraint, the training method obtains the highest tolerable retention failure rate. According the eDRAM's retention time distribution, the tolerable retention time is obtained according to the failure rate, which is usually longer than the eDRAM's typical retention time.

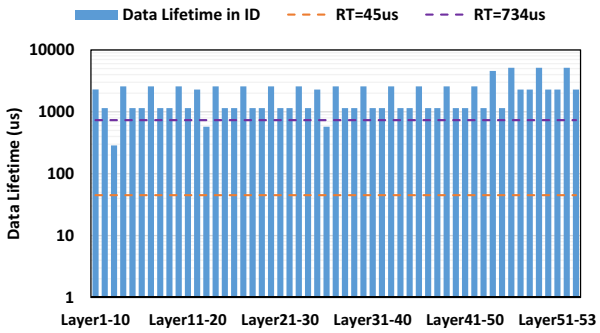


Figure 7. ResNet's data lifetime before optimized.

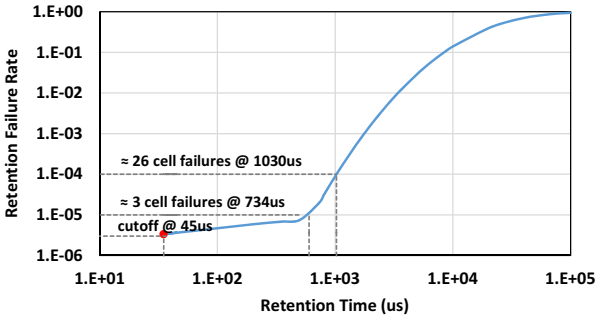


Figure 8. Typical eDRAM retention time distribution [6].

Stage ❷ (Scheduling): A system energy consumption model is built in consideration of computing energy, on-chip buffer access energy, refresh energy and off-chip memory access energy. For each CNN layer, RANA explores different computation patterns, and estimates their energy consumption based on data lifetime and buffer storage analysis. RANA assigns the pattern with the lowest energy to each layer separately, and results in a hybrid computation pattern for the whole network. RANA generates layerwise configurations in this stage, which contains the tolerable retention time, each layer's computation pattern and eDRAM refresh flags to be used in the next stage.

Stage ❸ (Architecture): A refresh-optimized eDRAM controller is designed to enable independent refresh controls on each buffer bank, to further optimize refresh energy. During the execution phase, the refresh interval is initialized as the tolerable retention time. The accelerator loads the configurations layer by layer. The control logic and buffer data mapping are adjusted for the current layer's computation pattern. The eDRAM controller only issues refresh to the bank whose refresh flag is valid, thus saving many unnecessary refresh operations.

The proposed RANA framework can be applied to current CNN hardware architectures, with only small modifications on their memory control logic. The performance loss is negligible, because RANA doesn't change their core computing part and the eDRAM refresh overhead is minimized in this framework.

B. Retention-Aware Training Method

We run ResNet on the evaluation platform and find out that, not just Layer-A, all the layers' data lifetime is longer than eDRAM's typical retention time ($RT = 45\mu s$), as presented in Figure 7. Thus, refresh operations cannot be avoided under such short retention time.

It would be quite meaningful if longer retention time can be obtained, for two reasons: 1) The condition "**Data Lifetime** $<$ **Retention Time**" can be reached to avoid refresh; 2) The refresh interval gets longer even if refresh is still needed. Although eDRAM's retention time is fixed

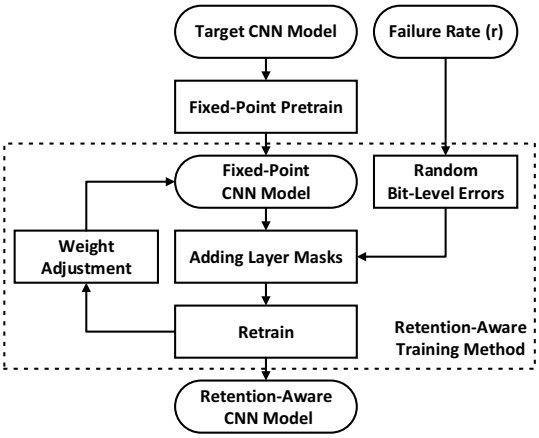
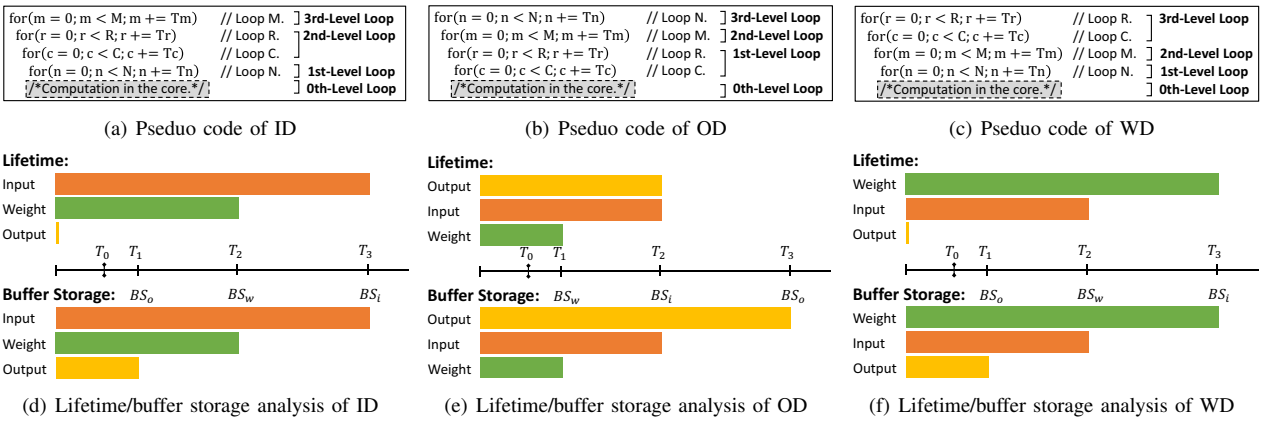


Figure 9. Retention-aware training method.

once fabricated, we can still improve the "retention time" by exploiting CNN's error resilience. This "retention time", *i.e.* tolerable retention time, refers to the retention time that can be tolerated with no accuracy loss.

Figure 8 shows a typical eDRAM retention time distribution [6], which is used in our evaluation platform. The X axis is the retention time. The Y axis is the retention failure rate, which equals to the fraction of the cells under the given retention time. In conventional eDRAM design, the refresh interval is conservatively decided by the cell with the shortest retention time. For a 32KB-eDRAM buffer, the weakest cell typically appears at the 45μs point [6] with an failure rate of 3×10^{-6} (see Figure 8). However, the majority of cells have much longer retention time than 45μs. As shown in Figure 8, only ≈ 3 cells have retention time of no more than 734μs, which means we can use a 16x refresh interval with a cell failure rate of only 10^{-5}. Thus, the tolerable retention time can be relaxed to orders longer, which greatly improves the benefits of our method.

We propose a retention-aware training method to improve CNN's tolerance to eDRAM cell failures, and thereby to increase the tolerable retention time. As shown in Figure 9, we introduce retention failures in the forward propagation



(a) Pseduo code of ID (b) Pseduo code of OD (c) Pseduo code of WD

(d) Lifetime/buffer storage analysis of ID (e) Lifetime/buffer storage analysis of OD (f) Lifetime/buffer storage analysis of WD

Figure 10. Pseudo code and lifetime/buffer storage analysis of Input Dominant (ID), Output Dominant and Weight Dominant (WD) computation patterns.

during the model's training phase, by adding a mask to each layer's inputs and weights. The network has been previously trained in fixed-point precision (typically 16-bit or 8-bit) for hardware execution. The mask models retention failures by injecting errors to each bit at a failure rate of r. An error means the bit has a random value of 0 or 1 with equal probability. The fixed-point model is retrained to maintain its accuracy. During each iteration in the training, bit-level errors are randomly injected, so the weights would be adjusted to the errors in the forward propagation. If the final accuracy loss is acceptable, it means the model can tolerate the retention failure rate. Therefore, cell failures under the corresponding retention time cause little accuracy loss, leading to a longer tolerable retention time to be used in our method.

The proposed retention-aware training method is tested on four CNN models, AlexNet [1], VGG [2], GoogLeNet [3] and ResNet [4], with retention failure rates of 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}. Figure 11 presents the relative top-1 accuracy under different retention failure rates. The baseline accuracy is trained with no retention failure, in 16-bit fixed-point precision. The results prove CNN's error resilience to retention failures. All the four benchmarks show no accuracy loss at the failure rate of 10^{-5}. From the failure rate of 10^{-4}, the accuracy gradually decreases. According to Figure 8, the failure rate of 10^{-5} is corresponding to tolerable retention time of 734μs. As shown in Figure 7, although only three layers' data lifetime is shorter than 734μs, many layers are quite close to the purple line. If we can reduce the data lifetime, more layers' data lifetime will be shorter than 734μs and refresh operations can be removed from those layers.

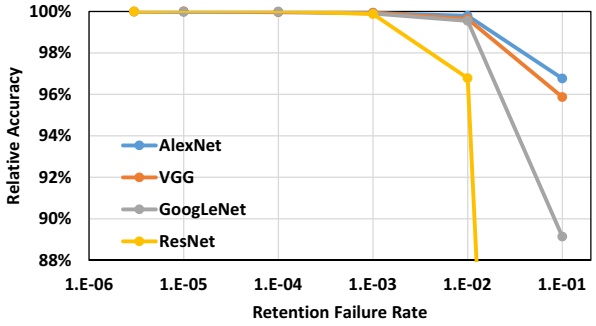


Figure 11. Relative accuracy under different retention failure rates.

C. Hybrid Computation Pattern

1) OD: Reducing Data Lifetime: We will discuss how we can optimize the computation pattern to optimize data lifetime. Let's first go back to the previously discussed typical computation pattern. As shown in Figure 10(a), we divide the whole pattern into 4 levels of loops: the core computing part as the 0th-level loop, and the memory control part (Loop N, RC, M) as the 1st/2nd/3rd-level loops respectively. We find out that **data lifetime and buffer storage are strongly related to the loop ordering in the memory control part**, as illustrated in Figure 10(d): Loop M for inputs, Loop RC for weights, and Loop N for outputs. For example, as Loop M is the outer-most loop, the input lifetime equals to the 3rd-level loop's execution time (T_3), and all the inputs require storing on chip ($BS_i = N \cdot H \cdot L$). Since both buffer storage and lifetime are dominated by inputs, we name this typical pattern as the input dominant (ID) computation pattern.

Through the previous analysis on ID, we find out that the output lifetime is quite different from the other two data types' lifetime. Unlike inputs/weights that are statically stored in buffers for access, outputs are dynamically updated by accumulations. For each update, the outputs are rewritten to the buffer, which recharges the eDRAM cells and recovers data retention just like eDRAM's periodic refresh operations.

We exploit the output's self-refresh property in accumulation, and propose an output dominant (OD) computation pattern. As shown in Figure 10(b), we change the memory

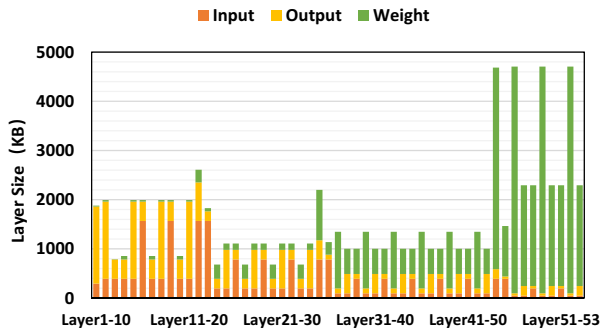


Figure 12. Layer size analysis of ResNet (16-bit precision, input image size = 224×224×3).

control part: Loop N is switched to the outer-most loop, while the other loops keep the same ordering as in ID. In this way, the computation pattern is reorganized: The 2nd-level loop generates all the output points with partial accumulations and then store them in the output buffers. The outputs are updated in the 3rd-level loop, which automatically refresh the values stored in memory. The buffer storage requirements and lifetime of inputs/outputs/weights are:

$$BS_i = Tn \cdot H \cdot L \tag{6}$$

$$BS_o = M \cdot R \cdot C \tag{7}$$

$$BS_w = Tn \cdot Tm \cdot K^2 \tag{8}$$

$$LT_i = LT_o = T_2 = \frac{M \cdot Tn \cdot R \cdot C \cdot K^2}{MAC \cdot Frequency \cdot \eta} \tag{9}$$

$$LT_w = T_1 = \frac{Tm \cdot Tn \cdot R \cdot C \cdot K^2}{MAC \cdot Frequency \cdot \eta} \tag{10}$$

As illustrated in Figure 10(e), both buffer storage and lifetime are dominated by outputs in the OD pattern. Although all the outputs should be stored on chip, the 3rd-level loop refreshes their values and lifetime. Thus, the output lifetime (LT_o) is reduced to the 2nd-level loop's execution time (T_2). We run Layer-A in the OD pattern with tiling parameters of $Tm, Tn, Tc = 16, Tr = 1$. The layer's data lifetime equals to $LT_o = 72\mu s$, which is shorter than the eDRAM's tolerable retention time ($734\mu s$), so no eDRAM refresh is needed during Layer-A's computation.

According to Equation (9), OD's data lifetime can be changed by adjusting Tn. For some large layers like Layer-B in Figure 5, reducing Tn from 16 to 8 can alleviate all the refresh, because the lifetime declines from $1290\mu s$ to $645\mu s$. However, a smaller Tn means fewer data are stored in the core, which will result in more buffer access energy. Therefore, we should explore the tiling parameters to find out the configuration with the lowest total system energy. We will discuss it in Section IV-C3.

2) WD: Shrinking Buffer Storage Requirement: Unfortunately, OD might still suffer from the buffer storage problem, as all the outputs should be stored on chip. Figure 12 shows ResNet's layer sizes in 16-bit precision. Some layers' output size even exceeds the 1.454MB buffer capacity. The situation will become worse if higher resolution images are processed. We find out that the inputs and outputs dominate the size of shallow layers. As the layer goes deeper, the input/output size gets smaller while the weight size grows significantly. The reason is that when the layer gets deeper, its input/output feature map size ($H \times L$, $R \times C$) shrinks as well as the total input/output size, while the channel count increases (from 3 to 512, 1024, or even 2048), leading to larger weight size. As a result, inputs and outputs usually have similar counts, while weights and outputs are quite complementary in size.

We propose a weight dominant (WD) computation pattern as a supplement to OD in shallow layers, to shrink buffer storage requirement when OD exceeds buffer capacity. As illustrated in Figure 10(c), in WD, Loop RC is switched to the 3rd-level loop, while the other loops keep the same ordering as in ID. Similar to ID and OD, WD's buffer storage is dominated by the 3rd-level loop. All the weights should be stored on chip, while only part of the input and output feature maps are buffered. The buffer storage requirements can be expressed as:

$$BS_i = N \cdot Th \cdot Tl \tag{11}$$

$$BS_o = Tm \cdot Tr \cdot Tc \tag{12}$$

$$BS_w = N \cdot M \cdot K^2 \tag{13}$$

3) Scheduling Scheme: Through the previous analysis, we find out that OD and WD shows complementary properties in buffer storage for different layers in a network. Meanwhile, the tiling parameters (Tm, Tn, Tr, Tc) also influence total system energy. Usually, increasing tiling parameters will reduce memory access times, but raise data lifetime and buffer storage. Therefore, RANA explores different computation patterns with different tiling parameters, and assigns the pattern with the lowest energy to each layer separately. ID is not included in the exploration space, for two reasons: 1) ID's lifetime is always longer than OD; 2) ID's buffer storage is usually similar to OD's.

A system energy model is built for energy estimation:

$$Energy = \alpha \cdot E_{mac} + \beta_b \cdot E_{buffer} + \gamma \cdot E_{refresh} + \beta_d \cdot E_{ddr} \tag{14}$$

System energy is the sum of computing energy, buffer access energy, refresh energy and off-chip memory access energy. E_{mac}, E_{buffer}, $E_{refresh}$ and E_{ddr} are the energy consumption for each MAC operation, buffer access, refresh operation and off-chip memory access, as listed in Table III. The coefficients refer to the total times of the above operations. α equals to the layer's total MAC operation count. The memory access times β_b, β_d can be accurately

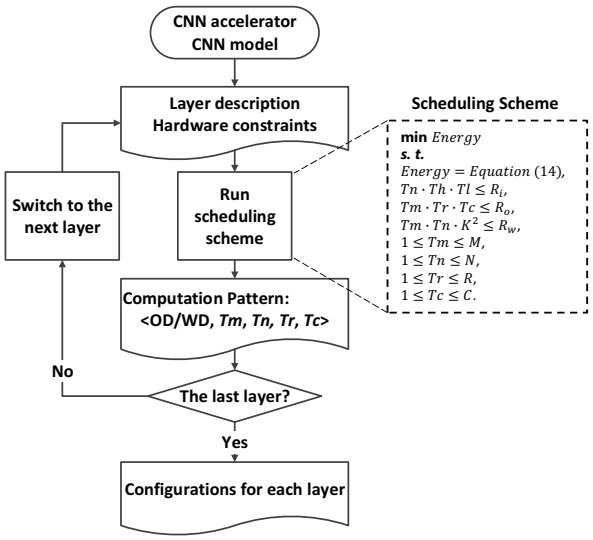


Figure 13. RANA's scheduling scheme for the hybrid computation pattern.

modeled as done in [11]. The refresh operation count γ is obtained through simulation on the evaluation platform, with data lifetime analysis.

Figure 13 presents RANA's layer-based scheduling scheme. The scheme takes the CNN accelerator and target CNN model's parameters as the inputs, and schedules each layer's computation pattern one by one. For each layer, the exploration for patterns is formulated as an optimization problem that minimizes total system energy consumption. The tiling parameters are constrained by the local storage in the core: $Tn{\cdot}Th{\cdot}Tl \leq R_i, Tm{\cdot}Tr{\cdot}Tc \leq R_o, Tm{\cdot}Tn{\cdot}K^2 \leq R_w$, where R_i, R_o, R_w refer to core's local input, output and weight storage, measured in the data count. After the exploration, we can obtain the best computation pattern for each layer with the optimal energy consumption, under the accelerator's hardware constraints. Each layer's computation pattern is described as $\langle$OD/WD$, Tm, Tn, Tr, Tc\rangle$, leading to a hybrid computation pattern for the whole network. In the end, the scheduling result is complied into layerwise configurations, which contains the tolerable retention time, each layer's computation pattern and eDRAM refresh flags to be used in the next stage.

D. Refresh-Optimized eDRAM Controller

1) Unified Buffer System: As discussed in Section IV-C, the hybrid computation pattern shows different storage requirements for different data types: OD's buffer storage is dominated by outputs ($BS_o = M \cdot R \cdot C$), while WD's buffer storage is dominated by weights ($BS_w = M \cdot N \cdot K^2$). Thus in WD, the weight buffer storage requirement (BS_w) might exceeds the weight buffer capacity (BS_w), but there's still much space in the output buffer, leading to unnecessary off-chip memory access. A unified buffer system for all the data types, as in [8, 10], is required to better utilize the on-chip buffer capacity. Thus, data mapping in the buffers can be easily adjusted to storage changes between layers: More eDRAM banks are allocated to outputs in OD, while in WD more eDRAM banks are used to store weights.

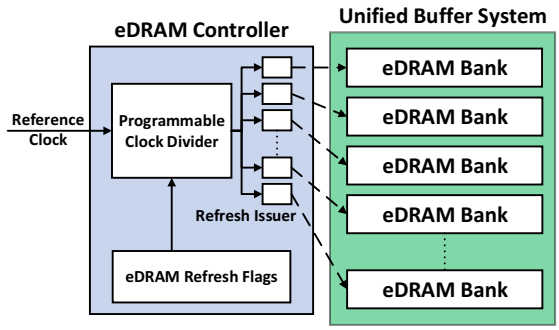


Figure 14. Refresh-optimized eDRAM Controller.

2) Memory Controller Modification: In conventional eDRAM-based designs, all the eDRAM banks are conservatively refreshed at the same rate. Through the analysis in Section IV-C, we observe that different data types have different lifetime. For example in Layer-B (see Figure 5), the input, output and weight lifetime on the test accelerator are $LT_i = LT_o = 1290\mu s, LT_w = 40\mu s$ when $Tn = 16$, so the eDRAM cells storing weights don't need to refresh under the tolerable retention time of $734\mu s$. Moreover, for small networks like AlexNet, some eDRAM banks are not used but still periodically refreshed.

Refresh energy can be further optimized by making more fine-grained refresh controls on the eDRAM. Figure 14 shows the eDRAM controller structure that is modified to support retention-aware refresh optimizations. The controller has a programmable clock divider, independent refresh issuers and eDRAM refresh flags for each eDRAM bank. The refresh flags indicate whether the data stored in one eDRAM bank needs refresh. They are contained in the configurations generated by **Stage ❷** of RANA, as different layers have different refresh needs for eDRAM banks. For a bank, if no data is stored in it, or its storing data's lifetime is shorter than the tolerable retention time, it will be set with a flag of "disable".

The programmable clock divider takes the accelerator's reference clock as the input and generates a refresh pulse. The pulse period equals to the eDRAM's tolerable retention time obtained from the proposed training method. At each refresh pulse, the eDRAM controller issues refreshes to the eDRAM banks as indicated by the refresh flags. When the current layer is completed, the next layer's refresh flags will be loaded into the controller. With the above modifications to the eDRAM buffer controller, RANA independently enables or disables refresh for each eDRAM bank, so unnecessary refresh operations are largely alleviated and refresh energy can be further reduced.

Table IV
DESIGN CONFIGURATIONS FOR RANA EVALUATION

Design Name	Buffer Type	Buffer Capacity	Computation Pattern	Failure Rate	Refresh Interval	Memory Controller
S+ID	SRAM	384KB	ID	-	-	-
eD+ID	eDRAM	1.454MB	ID	0 (3×10^{-6})	45μs	Normal
eD+OD	eDRAM	1.454MB	OD	0 (3×10^{-6})	45μs	Normal
RANA (0)	eDRAM	1.454MB	Hybrid (OD+WD)	0 (3×10^{-6})	45μs	Normal
RANA (E-5)	eDRAM	1.454MB	Hybrid (OD+WD)	10^{-5}	734μs	Normal
RANA*(E-5)	eDRAM	1.454MB	Hybrid (OD+WD)	10^{-5}	734μs	Optimized

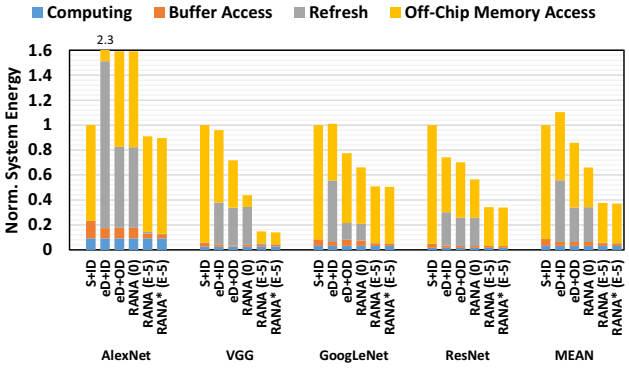


Figure 15. Total system energy comparison.

V. EXPERIMENTAL RESULT

A. Experiment Setup

We evaluate the RANA framework on the platform with a test CNN accelerator implemented in the TSMC 65nm GP technology, as described in Section III-A. We select four famous CNN models as the benchmarks: AlexNet [1], VGG [2], GoogLeNet [3] and ResNet [4]. The retention-aware training framework is implemented with modifications to the Caffe framework [29].

We will study the energy consumption of six designs with the configurations in Table IV, under the same area ($5.682mm^2$), working frequency (200MHz) and MAC count (256) constraints. For briefness, S+ID, eD+ID and eD+OD are used to represent the three baseline designs SRAM+ID, eDRAM+ID and eDRAM+OD, respectively. RANA (0) strengthens eD+OD with a hybrid computation pattern. RANA (E-5) runs the benchmarks trained with a failure rate of 10^{-5} (no accuracy loss), so its eDRAM refresh interval equals to 734μs. RANA*(E-5) further optimizes RANA (E-5) with a refresh-optimized eDRAM controller. As for energy measurement, we simulate the designs on the evaluation platform to obtain the operation counts $\alpha, \beta_b, \beta_d, \gamma$. The basic energy E_{mac}, E_{buffer}, $E_{refresh}$ and E_{ddr} are in accordance with the numbers in Table III. Total system energy consumption is estimated based on Equation (14).

B. RANA Evaluation

1) General Comparison: Figure 15 presents a general comparison of the six designs on total system energy consumption. All the numbers are normalized to S+ID's system energy for better comparison. Compared with the S+ID, eD+ID has more on-chip buffer capacity (1.454MB eDRAM vs. 384KB SRAM), thus saving off-chip memory access by 40.3% on average. However, the total energy is raised by 13.3% because of the extra eDRAM refresh energy. An energy increase of 2.3x is observed on AlexNet, because it's a relatively small network with no extra off-chip memory access, and the refresh takes up a large part of the total energy. In comparison with eD+ID, eD+OD removes more refresh operations due to its shorter lifetime and saves refresh energy by 43.7%.

The rest designs are three versions of RANA strengthened accelerators. RANA (0)'s total energy is 19.4% lower than eD+OD, because the hybrid computation pattern saves more off-chip memory access than OD. Through the retention-aware training method, tolerable retention time of 734μs is obtained at the failure rate of 10^{-5}. Thus in RANA (E-5), 98.5% refresh operations are removed from RANA (0) and the total energy is further reduced by 45.4%. With the refreshed-optimized eDRAM controller, RANA*(E-5) becomes almost refresh-free. Refresh accounts for only 0.4% of the total energy consumption. No significant energy reduction over RANA (E-5) is observed because RANA (E-5) has already greatly optimized refresh energy. Moreover, for accelerators with large on-chip memory, RANA*(E-5) shows more advantages in saving unnecessary refresh operations, which will be further discussed in Section V-B4.

Combing all the proposed techniques, RANA*(E-5) saves 41.7% off-chip memory access and 66.2% system energy consumption, with the same area cost of the SRAM-based baseline (S+ID). In comparison the eDRAM-based baseline (eD+ID), it reduces 99.7% eDRAM refresh operations with no accuracy loss.

2) Benefits on the Data Retention Problem: In Figure 15, a sharp energy drop is observed from RANA (0) to RANA (E-5), because the increased tolerable retention time and refresh interval remove 98.5% refresh operations. We will go further to study RANA's benefits on the data retention problem. As shown in Figure 16, the accelerator energy (excluding off-chip memory access energy) of eD+ID, eD+OD and RANA(0) on ResNet is estimated under different retention time (from 45μs to 1440μs). Refresh

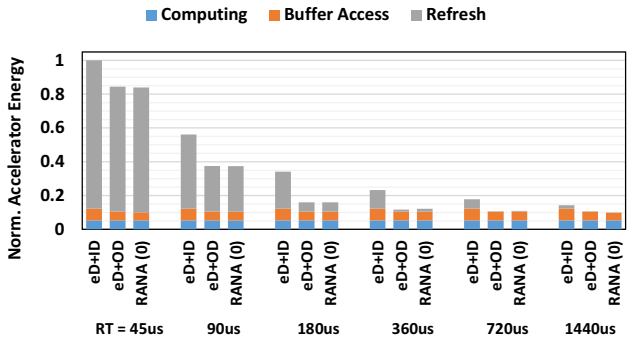


Figure 16. Accelerator energy comparison on ResNet: eD+ID vs. eD+OD vs. RANA (0), with retention time (RT) increasing from 45μs to 1440μs.

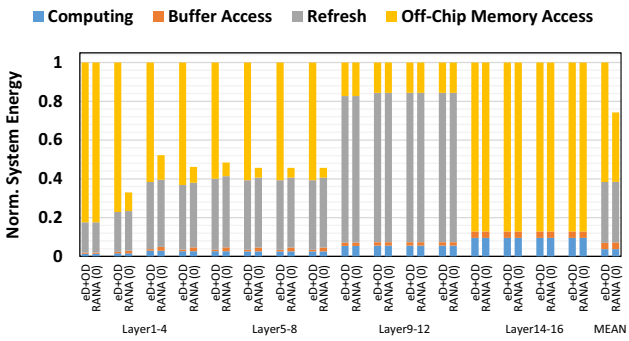
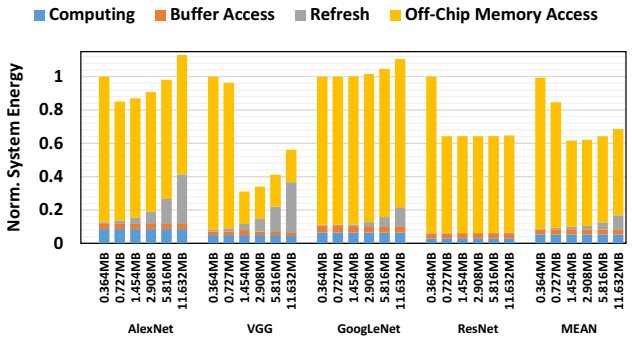
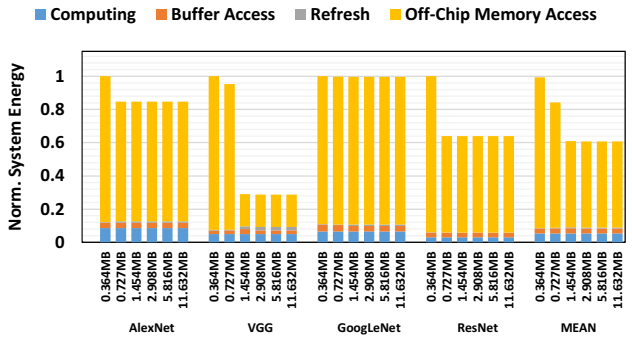


Figure 17. Layerwise system energy comparison on VGG: eD+OD vs. RANA (0).



(a) System energy of RANA (E-5).



(b) System energy of RANA*(E-5).

Figure 18. Total system energy comparison: RANA (E-5) vs. RANA*(E-5), with buffer capacity increasing from 0.364MB to 11.632MB (0.25x∼8x of 1.454MB).

interval also gradually increases with the retention time. We exclude off-chip memory access in Figure 16 because it's not significantly affected by retention time. eD+OD always has lower refresh energy than eD+ID, and shows more advantages when retention time is relatively long. For instance, when retention time grows from 90μs to 180μs, eD+ID's refresh energy declines by 50.0% as refresh interval doubles with the retention time. In comparison, eD+OD's refresh energy is reduced by 80.1%, because OD's short data lifetime makes more layers to meet the condition "**Data Lifetime** $<$ **Retention Time**" to avoid refresh. Notice that refresh still accounts for a large part in the accelerator energy even if retention time grows to 720μs in eD+ID, but eD+OD is almost refresh-free in this case. RANA (0) has similar refresh energy with eD+OD, although some of the layers are assigned with the WD pattern by RANA. But the hybrid pattern helps RANA (0) achieve lower system energy with the optimized off-chip memory access, as presented in Figure 15.

As illustrated in Figure 16, refresh energy can be greatly reduced by increasing retention time. However, retention time is usually not too long (45μs [6] in this paper, typically $< 100\mu s$) and it's fixed once eDRAM is fabricated. Owing to the retention-aware training method, we achieve longer tolerable retention time by exploiting CNN's error resilience. In RANA (E-5), tolerable retention time of 734μs is obtained at the failure rate of 10^{-5} with no accuracy loss. It removes 98.5% refresh operations from RANA (0), whose retention time is only 45μs.

3) Benefits on the Buffer Storage Problem: To evaluate RANA's benefits on the buffer storage problem, we conduct a layerwise system energy comparison between eD+OD and RANA (0) on VGG, as shown in Figure 17. For each layer, RANA (0) is normalized to eD+OD's energy for better comparison. On Layer2∼8, RANA (0) achieves significant energy reduction, because WD, instead of OD, is selected as the computation pattern. The seven layers' buffer storage requirements all exceed the eDRAM buffer capacity (1.454MB). With the help of WD, the buffer storage requirements are lowered down, saving 79.5∼91.6% off-chip memory access. Although the refresh energy slightly rises due to WD, the total energy still declines by 47.8∼67.0%. As for the other layers, RANA (0) selects OD according to the system energy model, so the energy stays the same as eD+OD. On the whole VGG, thanks to the hybrid computation pattern, RANA (0) saves 19.4% system energy in total.

4) Sensitivity to Buffer Capacity: We study our method's sensitivity to buffer capacity, by comparing RANA (E-5) and RANA*(E-5)'s system energy consumption, with buffer capacity sweeping from 0.364MB to 11.632MB (0.25x∼8x of

1.454MB). Figure 18(a) presents RANA (E-5)'s normalized system energy with different buffer capacity, measured on the four benchmarks. The off-chip memory access decreases with the growing of buffer capacity, since more data can be cached on chip. However, the total energy consumption gradually rises when buffer capacity increases beyond 1.454MB. The reason is that, in conventional eDRAM design, the memory controller conservatively refresh all cells whether they store data or not. Thus, increasing buffer capacity directly produces more refresh operations. Once the buffer capacity is larger than the required buffer storage, the refresh for the unused eDRAM cells leads to a big waste of energy. On AlexNet, eDRAM refresh energy takes up 26.3% of the system energy when the buffer capacity reaches 11.36MB, making its total energy even higher than that obtained with 363.5KB buffer capacity.

The problem is well solved in RANA*(E-5) by removing the unnecessary refresh operations, as shown in Figure 18(b). In RANA*(E-5)'s refresh-optimized eDRAM controller, refresh is only issued to the eDRAM bank that stores data with lifetime longer than the tolerable retention time, based on the refresh flags contained in the configurations. As a result, refresh energy no longer increases once the buffer capacity meets the intermediate data storage requirement. Compared with RANA (E-5), 65.5∼92.3% refresh energy is reduced in RANA*(E-5).

It's notable that, with 1.454MB buffer capacity, no extra off-chip memory access exist for all the four benchmarks, in both RANA (E-5) and RANA*(E-5). According to Table I, the maximum layer size reaches 6.27MB for inputs/outputs, and 4.61MB for weights. The hybrid computation pattern of RANA, the buffer storage requirements are significantly reduced, saving both refresh energy and chip area.

C. Scalability Analysis on DaDianNao

We apply the RANA framework to the state-of-the-art eDRAM-based CNN accelerator DaDianNao [12], to study RANA's scalability to other architectures, as demonstrated in Figure 19. Since we don't have DaDianNao's design files, we build a cycle-accurate simulator to model its data lifetime, with the following design parameters extracted from one node of DaDianNao: 4096 PEs, organized in a tree-like structure with tiling parameters of $Tm = Tn = 64, Tr = Tc = 1$, 36MB on-chip eDRAM capacity, working at 606MHz. For fair comparison, the eDRAM's retention time distribution is in accordance with that in Figure 8 [6]. DaDianNao is strengthened with three versions of RANA configurations, *i.e.* RANA (0), RANA (E-5), RANA*(E-5), with the same hardware parameters as DaDianNao's. Their optimization techniques related to RANA are configured as in Table IV.

In DaDianNao, buffer access energy takes up 23.5% of the total system energy consumption, because it only uses the WD computation pattern and produces frequent

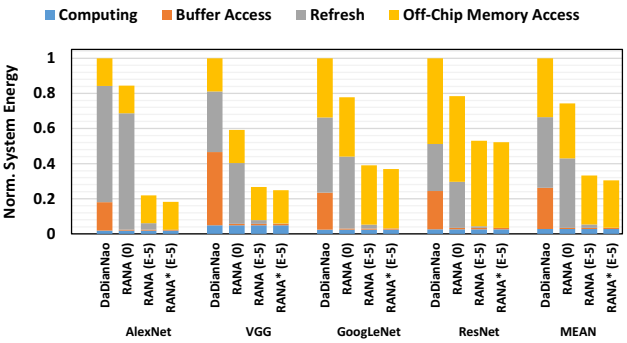


Figure 19. Scalability analysis on DaDianNao [12].

access to its weight buffer. Owing to the hybrid computation pattern, RANA (0) tends to use OD in most layers, saving 97.2% buffer access energy from the original DaDianNao. With longer tolerable retention time (734μs), RANA (E-5) reduces 94.9% refresh energy compared with RANA (0). However, refresh energy still accounts for 36.9% of the accelerator energy, because large parts of the 36MB eDRAM are not used and still require periodic refresh. RANA*(E-5) removes the unnecessary refresh operations with the optimized eDRAM controller.

In comparison with the original DaDianNao, RANA*(E-5) reduces 99.9% eDRAM refresh operations and saves 69.4% system energy consumption. No reduction in off-chip memory access energy is observed because the 36MB eDRAM stores all the intermediate data and alleviates all the extra off-chip memory access. In such a high performance accelerator with large on-chip memory like DaDianNao, the RANA framework greatly optimizes both accelerator and system energy, showing very good scalability.

VI. Conclusions

In this paper, we propose RANA, a Retention-Aware Neural Acceleration framework, for CNN accelerators to save system energy consumption with refresh-optimized eDRAM. The RANA framework solves the buffer storage and data retention problem, with three levels of techniques: a retention-aware training method, a hybrid computation pattern and a refresh-optimized eDRAM controller. RANA can be applied to current CNN hardware architectures, with only small modifications for the memory control logic. The performance is maintained because RANA doesn't change their core computing part and the eDRAM refresh overhead is minimized. Experimental results show that, an eDRAM-based CNN accelerator becomes almost refresh-free after strengthened by RANA. Meanwhile, RANA saves 41.7% off-chip memory access and 66.2% system energy consumption, compared with the conventional SRAM-based CNN accelerator in the same area.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[2] A. Z. Karen Simonyan, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[6] W. Kong, P. C. Parries, G. Wang, and S. S. Iyer, "Analysis of retention time distribution of embedded dram-a new method to characterize across-chip threshold voltage variation," in *Test Conference, 2008. ITC 2008. IEEE International*. IEEE, 2008, pp. 1–7.

[7] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine learning," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[8] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2016, pp. 262–263.

[9] L. Cavigelli and L. Benini, "Origami: A 803-gop/s/w convolutional network accelerator," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 11, pp. 2461–2475, Nov 2017.

[10] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*. IEEE, 2017, pp. 246–247.

[11] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2220–2233, Aug 2017.

[12] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47. Washington, DC, USA: IEEE Computer Society, 2014, pp. 609–622. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2014.58

[13] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 1–13. [Online]. Available: https://doi.org/10.1109/ISCA.2016.11

[14] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[15] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017, pp. 382–394. [Online]. Available: http://doi.acm.org/10.1145/3123939.3123982

[16] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "Destiny: A tool for modeling emerging 3d nvm and edram caches," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 1543–1546.

[17] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *International Conference on Artificial Neural Networks (ICANN)*. Springer, 2014, pp. 281–290.

[18] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 27–40.

[19] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016, pp. 367–379.

[20] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks," in *International Conference on Computer-Aided Design (ICCAD)*. ACM, 2016, p. 12.

[21] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2016, pp. 26–35.

[22] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in *CVPR 2011 WORKSHOPS*, June 2011, pp. 109–116.

[23] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015, pp. 92–104. [Online]. Available: http://doi.acm.org/10.1145/2749469.2750389

[24] A. Rahman, J. Lee, and K. Choi, "Efficient fpga acceleration of convolutional neural networks using logical-3d compute array," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 1393–1398.

[25] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li, "C-brain: A deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, pp. 1–6.

[26] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP Laboratories*, pp. 22–31, 2009.

[27] M. T. Chang, P. Rosenfeld, S. L. Lu, and B. Jacob, "Technology comparison for large last-level caches (l3cs): Low-leakage sram, low write-energy stt-ram, and refresh-optimized edram," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2013, pp. 143–154.

[28] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-l. Lu, "Reducing cache power with low-cost, multi-bit error-correcting codes," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 83–93.

[29] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM International Conference on Multimedia (MM)*. ACM, 2014, pp. 675–678.