공학박사 학위논문

In-DRAM Neural Network Accelerator Architecture for Binary Neural Network

이진 뉴럴 네트워를 위한 DRAM 기반의 뉴럴 네트워크 가속기 구조

2021 년 2 월

서울대학교 대학원

컴퓨터공학부

최 해 랑

In-DRAM Neural Network Accelerator Architecture for Binary Neural Network

이진 뉴럴 네트워를 위한 DRAM 기반의 뉴럴 네트워크 가속기 구조

지도교수 유 승 주

이 논문을 공학박사 학위논문으로 제출함

2020 년 11 월

서울대학교 대학원

컴퓨터공학부

최 해 랑

최해랑의 공학박사 학위논문을 인준함

2020 년 12 월

위 원 장	윤 성 로	(인) [illegible]
부위원장	유 승 주	(인) [illegible]
위 원	이 재 욱	(인) [illegible]
위 원	김 재 준	(인) [illegible]
위 원	권 대 한	(인) [illegible]

Abstract

In-DRAM Neural Network Accelerator Architecture for Binary Neural Network

Haerang Choi

Department of Computer Science and Engineering

The Graduate School

Seoul National University

In the convolutional neural network applications, most computations occurred by the multiplication and accumulation of the convolution and fully-connected layers. From the hardware perspective (i.e., in the gate-level circuits), these operations are performed by many dot-products between the feature map and kernel vectors. Since the feature map and kernel have the matrix form, the vector converted from 3D, or 4D matrices is reused many times for the matrix multiplications. As the throughput of the DNN increases, the power consumption and performance bottleneck due to the data movement become a more critical issue. More importantly, power consumption due to off-chip memory accesses dominates total power since off-chip memory access consumes several hundred times greater power than the computation. The accelerators' throughput is about several hundred GOPS$\sim$several TOPS, but Memory bandwidth is less than 25.6 or 34GB/s (with DDR4 or LPDDR4).

By reducing the network size and/or data movement size, both data movement power and performance bottleneck problems are improved. Among the algorithms, *Quantization* is widely used. Binary Neural Networks (BNNs) dramatically reduce precision down to 1 bit. The accuracy is much lower than that of the FP16, but the accuracy is continuously improving through various studies. With the data flow control, there is a method of reducing redundant data movement by increasing data reuse. The above two methods are widely applied in accelerators because they do not need additional computations in the inference computation.

In this dissertation, I present 1) a DRAM-based accelerator architecture and 2) a DRAM refresh method to improve performance reduction due to DRAM refresh. Both methods are orthogonal, so can be integrated into the DRAM chip and operate independently.

First, we proposed a DRAM-based accelerator architecture capable of massive and large vector dot product operation. In the field of CNN accelerators to which BNN can be applied, a computing-in-memory (CIM) structure that utilizes a cell-array structure of Memory for vector dot product operation is being actively studied. Since DRAM stores all the neural network data, it is advantageous to reduce the amount of data transfer. The proposed architecture operates by utilizing the basic operation of the DRAM.

The second method is to reduce the performance degradation and power consumption caused by DRAM refresh. Since the DRAM cannot read and write data while performing a periodic refresh, system performance decreases. The proposed refresh method tests the refresh characteristics inside the DRAM chip during self-refresh and increases the refresh cycle according to the characteristics. Since it operates independently inside DRAM, it can be applied to all systems using DRAM and is the same for deep neural network accelerators.

We surveyed system integration with a software stack to use the in-DRAM accelerator in the DL framework. As a result, it is expected to control in-DRAM accelerators with the memory controller implementation method verified in the previous experiment. Also, we have added the performance simulation function of in-DRAM accelerator to PyTorch. When running a neural network in PyTorch, it reports the computation latency and data movement latency occurring in the layer running in the in-DRAM accelerator. It is a significant advantage to predict the performance when running in hardware while co-designing the network.

Keywords: Neural network, Memory bottleneck, convolutional neural network, binary neural network, Computing-in-Memory, DRAM-based accelerator, DRAM refresh

Student Number: 2016-34620

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the convolutional neural network applications [?], most computations occurred by the multiplication and accumulation (MAC) operation of the convolution and fully-connected layers. In AlexNet, the computation of the convolution and fully-connect layer accounts for 89% of the total [5]. Since the feature map and kernel have the matrix form, the vector which converted from 3D or 4D matrices reused many times for the matrix multiplications. At the hardware perspective (i.e., in the gate-level circuits), these operations performed by many dot-products between the feature map and kernel vectors [6]. These operations are computation-intensive and suitable to apply parallel processing because it is a repetition of simple MAC operations. Therefore, the neural network accelerator increases the throughput of the network by integrating small and simple computation units in large quantities [7–10].

As the throughput of the deep neural network operations increases, the performance bottleneck and power consumption due to the data movement between Memory to Process unit become more critical issues. The data movement size for neural network increases with the network throughput. For example, the VGG-16 network's kernel size is about 264MB in 16b Floating-point precision (FP16), and the data movement latency is about 11.2ms (with 25.6GB/s Memory bandwidth). When the network throughput increases twice, the kernel data movement also increases. Since the network size is larger than the cache size, the kernel is newly loaded from Memory for every layer. Throughput of the accelerators are about several hundred Giga-operations per second (GOPS)$\sim$several Tera-operations per second (TOPS), but Memory bandwidth is less than 25.6 Giga-byte per second (GB/s) with DDR4 [4] (or 34 GB/s with LPDDR4 [11]). Computation latency will be shorter than the data movement latency. The memory-bound performance bottleneck can be predicted as the ratio of the memory bandwidth and the throughput [12].

More importantly, power consumption due to off-chip memory accesses dominates total power since off-chip memory access consumes several hundred times greater power than the computation. For instance, at the 65nm process, the DRAM access power is two hundred times larger than the computation power [9]. DDR4 DRAM consumed 22.6mJ energy to read the VGG-16 network kernel once [13]. The DRAM read power overhead of about 2.1W is significantly large compared to the neural network accelerator [13, 14]. Neural network accelerators use a local buffer (SRAM) to reduce DRAM access power, but the data movement still has a large proportion of power consumption. (i.e., 54.4 % [7], 57.6 % [8], 45.0 % [9], 70.0 % [10])

By reducing the network size and/or data movement size, both the performance bottleneck and data movement power are improved. Among the algorithms, *Quantization* is widely used. When 4b integer precision (INT4) is applied to FP16 precision, the data size is reduced by 75 %. In addition, it can be applied to various applications, and additional computation is not required during the inference. In terms of data flow, there is a method of reducing redundant data movement by increasing data reuse. The above two methods are widely applied in accelerators.

Binary neural network (BNN) dramatically reduce the precision down to 1 bit [15–17]. The accuracy is much lower than that of the FP16, but the accuracy is continuously improving through various studies. In recent years, with the increasing accuracy of BNN, BNN have been applied to various tasks such as image classification, object detection, and segmentation in computer vision [18].

In the field of CNN accelerators to which BNN can be applied, a computing-in-memory (CIM) architecture that utilizes a cell-array structure of Memory for vector dot product operation is being actively studied [6, 19–25]. CIM reduces data movement and overheads by performing operations inside the memory where data are stored. Since all the data needed to run the neural network are loaded into the main memory, DRAM-based CIM can minimize the data movement than other memory types. The DRAM-based CIM can be classified into two types, charge sharing and gate logic methods, according to how the logic operation is implemented [20, 21]. Among DRAM-based accelerators, the method of implementing accelerators in the peripheral area [26, 27] is classified as Logic-in-memory [28]. Area overhead is larger than DRAM-based CIM, and the design target is a stand-alone accelerator. In this dissertation, we focused on DRAM-based CIM.

The memory wall does not occur only as a data movement. Refresh hearts the power efficiency and system performance. Refresh increases the system background power because it continues to run while power is supplied to the DRAM. The DRAM periodically performs refresh, data cannot be read or written during the refresh. Refresh interval (tREFI of 1x mode) is 7.8 µs and refresh execution time is 350 ns and 550 ns for 8Gb DDR4 and 16Gb DDR4 respectively. It reduces the NN accelerator performance of data-intensive applications by about 4.5 % and 7.1 %. When refresh executes with 4x mode, the performance drop is 8.1 % and 13.3 %. The DRAM-based CIM operating inside the DRAM also cannot operate during refresh. There are various methods of reducing DRAM refresh overhead have been proposed [2, 29–38]. Also, for the accelerator using eDRAM, a refresh scheme to reduce overhead due to refresh was proposed [3].

In this dissertation, we proposed 1) a DRAM-based accelerator architecture and 2) a DRAM refresh method to improve performance reduction due to DRAM refresh. Both methods are orthogonal, so can be integrated into the DRAM chip and operate independently.

We proposed a DRAM-based accelerator architecture capable of massive and large vector dot product operation. Since DRAM stores all the data of the neural network, it is advantageous to reduce the amount of data transfer. The proposed architecture operates by utilizing the basic operation of the DRAM.

We proposed a refresh method to improve power consumption and performance degradation caused by DRAM refresh. Whenever the DRAM performs a refresh, the system performance decreases because it cannot read or write data. The proposed refresh method tests the refresh characteristics inside the DRAM chip during self-refresh, and increases the refresh cycle according to the characteristics. Since it operates independently inside DRAM, it can be applied to all systems using DRAM, and is the same for deep neural network accelerators.

This dissertation is organized as follows. Chapter 2 introduces background for the neural network. Chapter 3 explains the proposed in-DRAM neural network accelerator. Chapter 4 explains the proposed the refresh method. Chapter 5 explains the system integration of the proposed methods. Chapter 6 concludes the dissertation.

Chapter 2

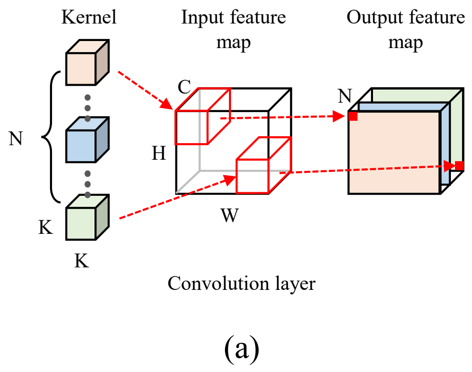
Background

2.1 Neural Network Operation

In the convolutional neural network applications, most computations occurred in the convolution and fully-connected layers.For example, in AlexNet, the convolution and fully-connect layer take up 89 % of the total computation [5].

As shown in Figure 2.1 (a), the convolution layer performs convolution between the input feature map (or activation matrix) and N number of the kernels (weight matrix). Since the kernel is smaller than the input feature map, the kernel is used repeatedly to calculate the entire input feature map. Convolution is an element-wise multiplication between matrices and then accumulates all values. When it is executing at the hardware level, all matrices converted to vectors and computed as a dot product.

Fully connected layer performs a weighted sum between input vector (or converted vector from matrix form) and kernel vector. As shown in Figure 2.1 (b), it performs a weighted sum across all activation of input



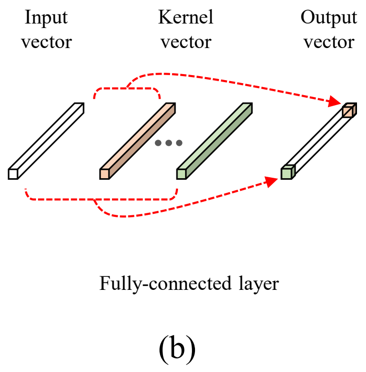


Figure 2.1 Typical DRAM Memory Structure.

vector to create one output. Input vector and kernel vector have a same dimension. At the hardware level, it is calculated by the dot product of the two vector, and the input vector is repeatedly used.

2.2 Data Movement Overhead

Since the neural network computation is data-intensive, performance drop due to data movement latency occurs [12]. In CPU and GPU, memory access is hidden by scheduling, and data movement overhead is reduced by optimizing data reuse patterns. However, in a customized accelerator, instead of hiding memory access, it moves data to a local buffer and then operates in a way that increases data reuse [9]. Also, even in accelerators that can hide memory access, memory access power is not hidden.

Talbe 2.1 shows the data size, data movement latency, and MAC op-

Table 2.1 Data size, data movement latency and required MAC operations (example of batch size 1)

		Data size (MB)		Memory access latency		
		FP16	w/ im2col	FP16	w/ im2col	MAC#
CIFAR-10	VGG-9	28.2	32.5	1.2 ms	1.3 ms	0.62 G
	ResNet-14	10.6	20.3	0.4 ms	0.8 ms	0.65 G
	ResNet-18	28.4	63.8	1.2ms	2.6 ms	1.79 G
	MobileNetV2	9.6	21.9	0.4 ms	0.9 ms	0.08 G
ImageNet	VGG-16	309.0	455.1	12.7 ms	18.6 ms	16.04 G
	ResNet-34	58.5	117.1	2.4 ms	4.8 ms	4.0 G
	MobileNetV2	33.8	93.2	1.4 ms	3.8 ms	0.31 G

erations required when the neural network processes CIFAR-10 [39] and ImageNet [40]. CIFAR-10 is a 32x32 small image dataset, and ImageNet is a 224x224 large image dataset. The table summarizes the required values when processing one image on VGG, ResNet, and MobileNetV2 [41–43] with a data set (bath size 1). Data size is the sum of the input feature map, kernel, and output feature map. Assume FP16 precision. im2col is an input reshape method required for the GEMM method that accelerates the convolution operation into matrix-matrix multiplication. Since the customized accelerator does not apply im2col in many cases, the matrix form data is compared by default. Memory access latency is the data movement latency when reading and writing data to DRAM. Since data reuse methods vary depending on the accelerator structure, it is assumed that only one read or write to Memory is performed.

For ease of understanding, we assume an accelerator that performs MAC operations with 1 TOPS. This accelerator takes 0.62 ms to compute one CIFAR-10 image in VGG-9, and it takes 1.2 ms to read the input and kernel to compute it and save the output. Memory access latency is 19 times longer than computation latency. ResNet-14 is a smaller version of ResNet-18 in order to make the amount of computation similar to VGG-9. Memory access latency of ResNet-14 and MobileNetV2 is smaller than VGG-9. It means that the data movement overhead of these networks is smaller than that of VGG. In terms of computation resource and Memory bottleneck, it is advantageous to use ResNet and MobileNetV2 for large images. In VGG and ResNet with ImageNet, Memory access latency is less than computation latency, but both values are large. MobilenetV2's Memory access latency is the smallest at 1.4ms, but its computational latency is 0.31 seconds, so its performance is limited by Memory access.

Therefore, by increasing the batch size, the kernel is reused to reduce memory access. Also, accelerators use local buffers (SRAM) to reduce DRAM access. If the kernel is still being stored in the local buffer, there is no need to read kernel data from DRAM. However, since the kernel size of the small network VGG-9 with CIFAR-10 is 26.7MB, most accelerators load the kernel every time the layer is executed.

Also, the read power of 3200 DDR4 is 2.2W, which is larger than most customized accelerators. Moreover, considering the ratio of memory access latency and computation latency, it is easy to predict the power consumption problem in memory access.

2.3 Binary Neural Networks

Quantization. Quantization is a method of converting data of high precision into integer data of low precision bits. As data precision bits decrease, data movement size, and computation resource requirements decrease. Moreover, since quantized weights and activations are calculated as integers, the overhead of implementing arithmetic logic is also reduced. Research on quantization has been actively conducted, and high accuracy has been obtained with INT8 in image classification [**?**, 44], and most customized accelerators are designed based on INT8 operation [44–46].

Binary neural network. BNN is an extreme quantization case that reduces the precision of weight and activation to a binary bit. The advantage of BNN is that there is a simple way to implement the ALU with bit multiplication (XNOR operation) and pop counting [15, 17]. Also, binary quantization can be implemented simply by using the sign bit of data as binary data. Despite the loss of accuracy in BNN, to utilize the advantage of small computing resources, a training method to increase the accuracy of BNN and a structure of a BNN accelerator are actively studied [18]. In particular, focusing on bit-wise operation, memory-based accelerators that utilize a memory array for binary neural network operation have been proposed. Memory-based accelerator is explained in the next section.

2.4 Computing-in-Memory

Computing-in-memory refers to an accelerator structure that performs bit-wise operations directly on a memory array where raw bit vectors are stored, without moving data from memory to the arithmetic core of the

CPU or GPU. It supports dot product operation and logic operation between two vectors. Because it operates on a column or row unit of a memory array, it is advantageous to calculate many bits at once. It can operate in parallel on physically separate memory arrays. A latency required for fetching and moving data is reduced because it operates directly in memory. Since there is no repetitive weight and activation matrix loading, the power consumed by memory I/O will be greatly reduced. Memory I/O power consumption is tens of times that of simple ALU operations [47].

Recently, NN accelerator based on ReRAM has been proposed the most [24,48–56]. It is easy to implement multi-bit multiplication and accumulation using voltage and current because the data is determined by the current flowing by applying the voltage to the resistive memory cell. The memory structure is the same as DRAM, and it can operate several K bit-wise operation at a time. Because different sub-arrays or banks can operate in parallel, it is easy to increase throughput. Also, since it is not a commodity product yet, it is possible to propose various structures. However, since there is a reliability problem such as limited write endurance, resistance drift over time, susceptibility to process variation (PV), etc., using it in an actual system is the current challenge [57,58].

SRAM based NN accelerators generally target the implementation in the last level cache [19]. Compared with other memories, it is advantageous that the latency is very short. However, since the data width accessed at once is 64b, the data access method of accelerator mode must be implemented separately from memory mode [6, 19]. Since the memory size is as small as several MB, it is difficult to map a large network to the SRAM.

DRAM based NN accelerator targets the implementation of an accelerator in main memory [20, 21]. Unlike other memories, main memory does not need to load weight and activation data because the data needed to run the program is already stored. DRAM chip density is as large as several Giga-bit and can be operated more than 8 K bit-wise operation at a time, so it is more advantageous than other memory when mapping large networks. It can operate in parallel on many sub-arrays and banks to increase throughput. In addition to running in parallel in memory, performance can also be scaled up by increasing the number of DRAM. However, there are problems such as limitation of cell array structure change, single bit operation only, and data destructive access method [20–22].

2.5 Memory Bottleneck due to Refresh

In the computing system, the memory bottleneck caused by DRAM refresh is gradually becoming a critical issue [2, 59, 60]. As DRAM chips' density increases, the number of cells that the DRAM needs to refresh increases [2]. Also, as the technology shrinks, the DRAM cell capacitor's size decreases, so we need to refresh more often to maintain cell data [61].

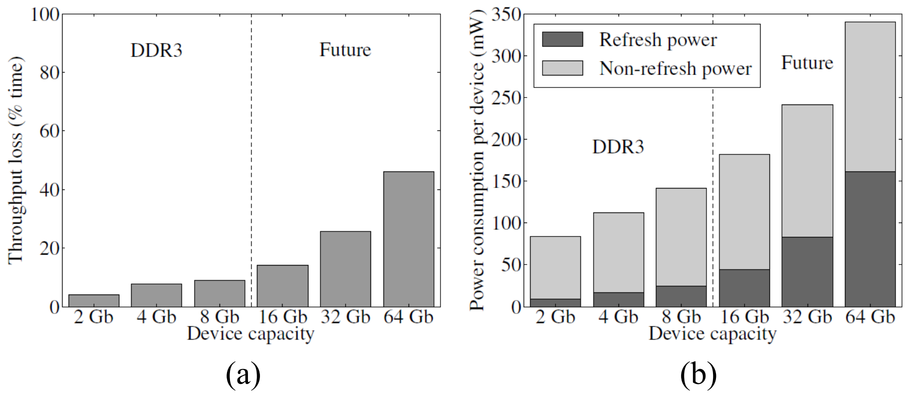


Figure 2.2 Refresh overhead, (a) throughput loss over DRAM density and (b) refresh power overhead [2].

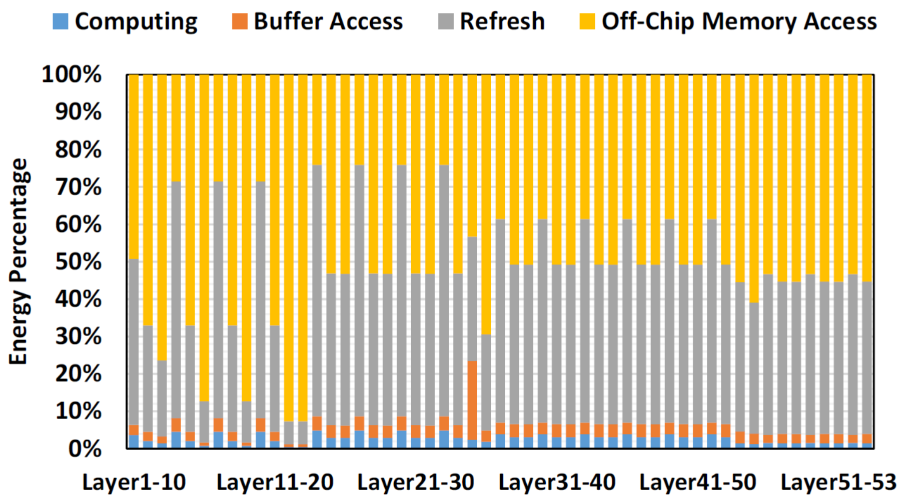


Figure 2.3 Energy consumption breakdown of ResNet on the evaluation platform [3].

As shown in the Figure 2.2, J. Liu analyzed that the throughput loss due to refresh will occur about 50% in 64Gb DRAM chip [2]. Throughput loss is a value obtained by dividing the refresh period (tREFI) by the refresh time tRFC. Since DRAM cannot perform read/write access during tRFC, system throughput loss occurs. In the current DDR4, the throughput loss is smaller than the Figure. In normal refresh mode (AREF x1), throughput loss of 8Gb and 16Gb occurs 4.5 % and 7.1 %, respectively. Moreover, it increases up to 8.1 % and 13.3 % in AREF x4 mode. Also, the refresh overhead is still increasing due to the tech scaling and density increasing.

Refresh overhead is a problem that needs to be solved in the neural network accelerator. In accelerators that use eDRAM to reduce DRAM access overhead, the refresh overhead can be more significant. Since the cell retention time of eDRAM is very short (45 µs), the refresh should be performed about 1400 times more often than DRAM [62]. Figure 2.3 shows the breakdown of the energy consumption of the accelerator when running ResNet [3]. Although power consumption is different for each layer, refresh power accounts for 30 70 % of the layer's power.

Chapter 3

In-DRAM Neural Network Accelerator

Computation in memory (CIM) is a structure that reduces data movement by performing operations inside the memory where data are stored. CIM executes a large amount of bit-wise operation by utilizing a memory array structure. Hence, simple and massively parallel operations are implemented in memory while the other operations are performed in the CPU. Although it can be implemented in various memory types, the main memory (i.e., DRAM) based structure [20–22] can minimize the data movement. This is because all the data needed to run the neural network are loaded into the main memory. Cache memory (SRAM) based structure [6, 19] cannot eliminate data movement between cache and main memory. ReRAM based structure [24,48–56] is capable of various structures, and has advantages in the possibility of multi-bit operation and low power consumption. However, since there is a reliability problem such as resistance drift over time, using it in an actual system is the current challenge [57, 58]..The storage memory (NAND) based structures [25] use an asynchronous memory access, which can yield longer latency than DRAM.

The DRAM based CIM structure can be classified into two types, charge sharing [20] and gate logic methods [21, 22], according to how the logic operation is implemented. The gate logic method adds logic circuits to all sense-amp array (S/A), so the area overhead is significant (e.g., 24 % for DRISA 1T1C-nor) [21]. In addition, the performance of the added circuit is limited by the low leakage transistor in the cell array. The charge sharing method does not change the S/A array, and the area overhead is very small (e.g., 1 % of DRAM chip area) [20]. However, in the existing charge sharing method, the implementation of NOT operation requires a new structure of DRAM cell [20] or a gate logic (incurring area overhead).

In this work, we focus on the charge sharing method on the existing DRAM which exhibits better area efficiency than the gate logic method.

- We proposed a novel CIM architecture on DRAM which judiciously exploits existing DRAM operations and charge sharing for both NOT operation and accumulation, which enables us to significantly reduce off-chip accesses and area overhead.

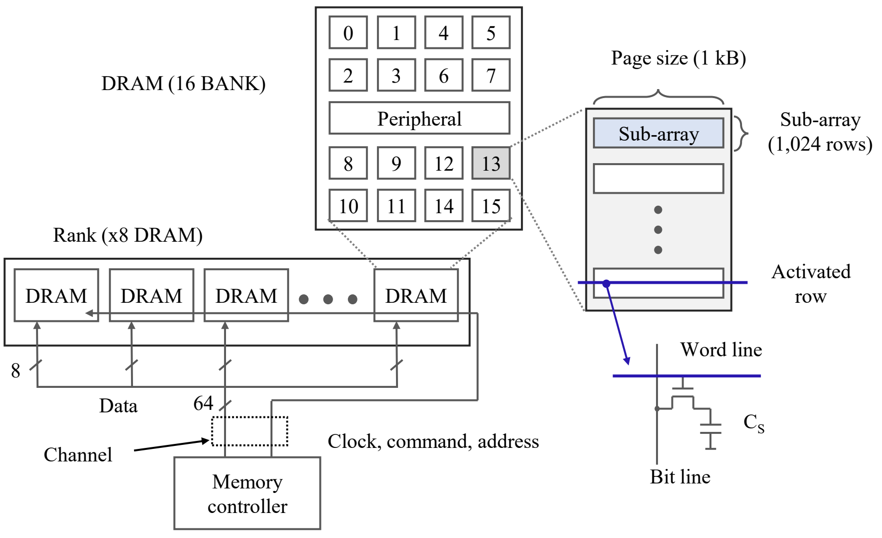
- We propose a novel charge sharing method of realizing NOT operation and accumulation on DRAM utilizing the existing commands of activation and pre-charge.

- We also present a new circuit for charge sharing-based operation which makes best use of the existing structure of sense amplifier and bit lines to obtain the inverted values and partial sum results.
- Our experiments report that the proposed method significantly (2.6 times in latency per image) outperforms the existing methods without compromising the quality of neural networks.

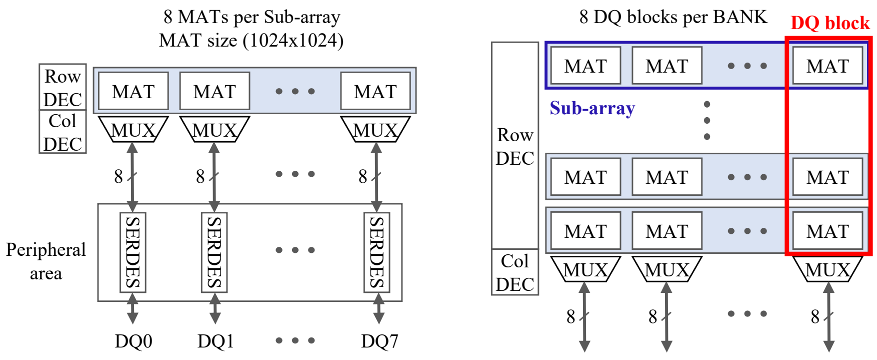
3.1 Backgrounds

3.1.1 DRAM hierarchy

Figure 3.1 (a) shows a typical DRAM memory structure. The DIMM (dual in-line memory module) is a PCB module where DRAM chips are mounted. The rank is a unit to access DRAM and is a sub-unit of DIMM. All DRAM chips of a given rank are controlled by the same command and address signals. A DRAM chip consists of multiple banks, where each bank comprises sub-arrays. For example, a bank of 4Gb DDR4 DRAM chip consists of 64 sub-arrays each of which has 1,024 rows. As shown in the figure, a DRAM cell consists of a capacitor to store 1-bit data and a switching transistor for access control (1T-1C structure). One-bit data (0 or 1 represented by the amount of charge) is stored in the capacitor. A DRAM chip needs an activation command to select a row and load all the data of the selected row to the sense amplifier (S/A)



(a)



(b)

Figure 3.1 Typical DRAM Memory Structure.

array. Then, a read or write command with a column address is issued to access the desired data in the S/A array. After accessing the data, the S/A is turned off, and the bit-lines are pre-charged by a pre-charge command. To retain the data, DRAM periodically refreshes the cell before it is discharged below a critical level [63]. Refresh is similar to performing pre-charge after activation without the read or write operation.

Figure 3.1 (b) shows the structure in which data is mapped to a memory array inside the DRAM. In Figure 3.1 (a), the DRAM chip has 8 Data I/O pins (DQ), and the sub-array consists of the same number of MAT blocks as the DQ. MAT is the physical unit of the memory cell array that composes the BANK. As shown on the left of Figure 3.1 (b), data input from the outside is converted into low-speed parallel data through the SERDES block in the peripheral area. The MAT of the sub-array stores the data input to each DQ. In DRAM design, as shown in the figure on the right, BANK is divided into DQ block units, which are units mapped to DQ. In other words, the column of the BANK is divided into DQ block, and the row is divided into sub-array.

MAT and DQ blocks are units used only inside DRAM. Since it is not a unit that can be distinguished by the address, most memory architecture levels do not use these units. However, when the in-DRAM accelerator is running, it is necessary to identify where the data of each DQ is stored in memory. Therefore, in this paper, DQ block and MAT are used together with the sub-array.

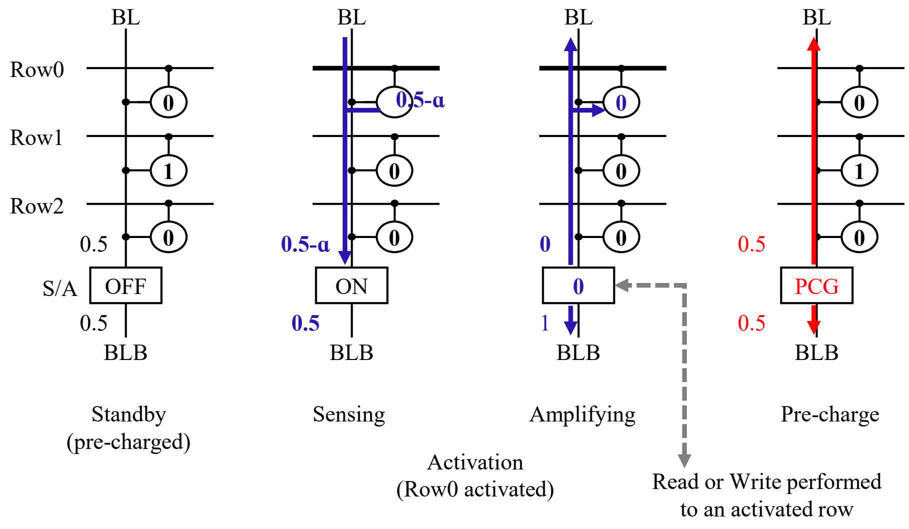


Figure 3.2 DRAM basic operation example.

3.1.2 DRAM Basic Operation

The figure 3.2 shows a simplified DRAM structure has 3 rows and a single bit line. The voltage level is represented as ‘1’ instead of the actual voltage VCORE. Sub-array is a unit of DRAM cell array. In general, the sub-array has 1024 rows, 8192 bit-lines and sense amps. Sense-amp is a circuit that evaluates the data from the charge stored in the cell. Since two sub-arrays share the sense-amps, one is called bit-line, and the other is called bit-line-bar. Activation is the operation that converts the charge stored in the cell to logical data. When we are activating the row 0, charge sharing occurs between the cell in row 0 and the bit-line. Since the data of the cell is ’0’ and the bit-line voltage is half, the charge sharing result is half - alpha. And the sense amp amplifies the voltage difference between bit-line and bit-line-bar. Simply, when the charge sharing results is smaller than half, the data evaluated to ‘0’. After activation, the data in the sense amps can be accessed to read or write. And, before activates other rows, bit-line, bit-line-bar, and sense amp must be initialized. This is called pre-charge, and the voltage is made to half by the pre-charge circuit inside a sense-amp. We exploit these activation and pre-charge.

3.1.3 DRAM Commands with Timing Parameters

There are four basic operations in DRAM. These are Activation (ACT), Read/Write (RD/WR), pre-charge (PRE), and Refresh (AREF) commands. In DRAM prior to DDR DRAM, it was divided into row address strobe (RAS), column address strobe (CAS), pre-charge (PRE), and refresh (REF). As described above, after the row is activated and data is loaded into the S/A array, read and write are possible. Before activating another row in the same sub-array, a pre-charge operation to reset the S/A array is required. In other words, when reading data from DRAM, a command sequence of ACT-RD-PRE is required. The figure shows an example of the command sequence for the basic operation of DRAM and the related timing parameters.

Figure 3.3 (a) shows the interval of ’ACT to PRE’ and ’ACT to ACT’ command. In the figure, tRAS is the latency from the row activation until the S/A fully charges/discharges the cell and bit-line. tRP is the latency

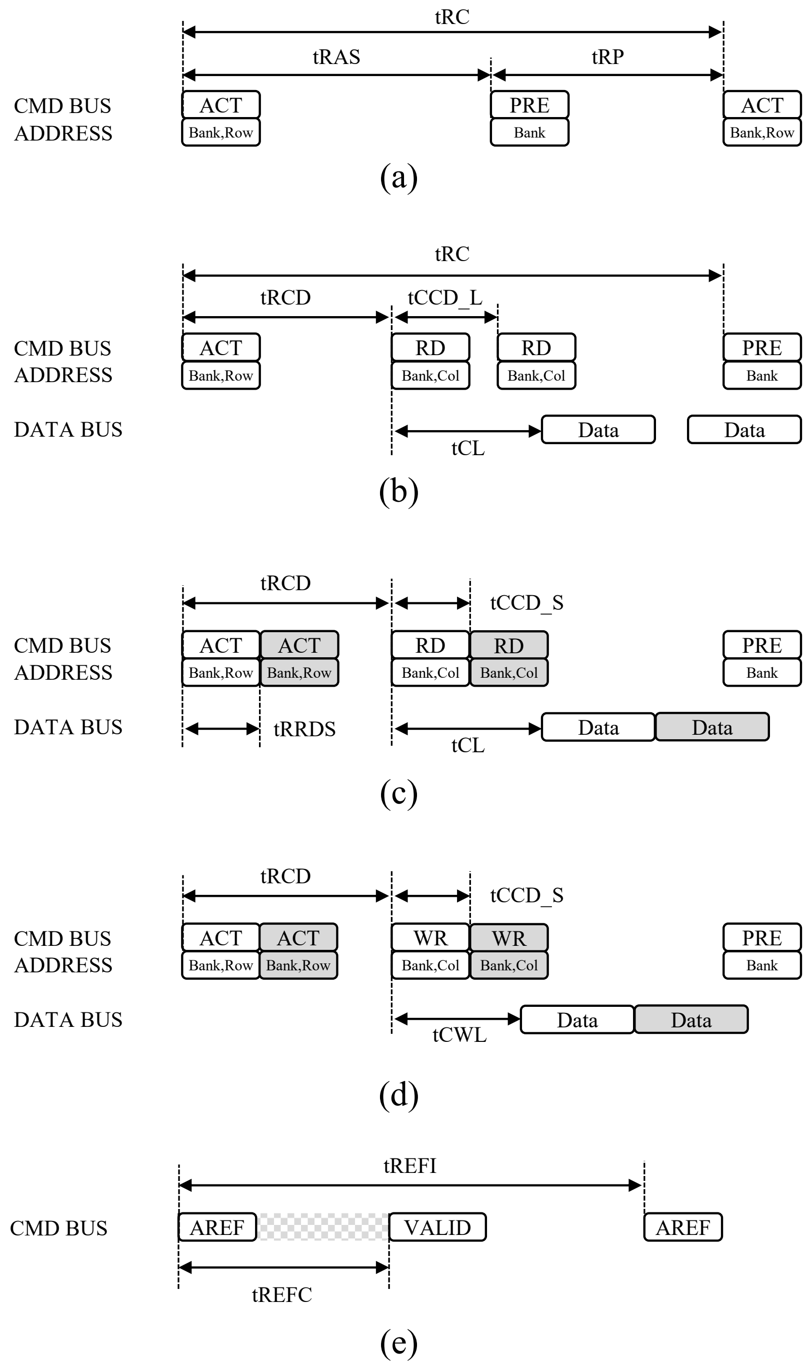


Figure 3.3 DRAM basic commands and command intervals.

to reset the fully charged bit line and S/A. When the row is activated, S/A does not directly sense the cell capacitor's voltage. It senses the charge-shared voltage between the cell capacitor and the bit-line connecting the cell and S/A. Since the several cells connected to one S/A along the bit-line. The number of cells connected through the bit-line is the same as the row number of the sub-array. Therefore, when the row is activated, the cell capacitor's voltage and the bit-line become a 'voltage' smaller than that of VCORE, and S/A senses this 'voltage' and fully charges/discharges to VCORE and VSS. In the field of DRAM design, this 'voltage' is called delV. In this paper, we assume that C_BL is 20 times C_S, and delV is VCORE*C_S/(C_S+C_BL). For simplicity, we use 1 and 0 as VCORE (high potential voltage) and VSS (low potential voltage), interchangeably.

Figure 3.3 (b) shows an example of executing read after activating a row. tRCD is the interval at which the read command can be input after ACT, and tCL is the latency until data is output to the DRAM DQ after the read command. The read command is input with the column address, and the DRAM reads the data located at the column address in the column MUX connected to the output of the S/A array. Note: tRCD is a smaller value than tRAS. That is because even before the S/A output is fully charged, if the value is greater or less than the column MUX circuit's logic threshold voltage, the S/A output is recognized as logical '1' and '0'.

Figure 3.3 (c) is different from Figure 3.3 (b) in RD to RD interval. In the same bank group (BG) that shares GIO, the following read data can be transferred through the GIO after the GIO data is completely transferred to the read FIFO. The interval at which read commands can be executed continuously in the same BG is tCCD_L. As shown in Figure 3.3(c), the serial read commands between the other BG can be executed with tCCD_S. As shown in Figure 3.3 (d), the same interval is applied to the write command. tCWL is the latency that data is input to the DRAM DQ pin after the write command. During tCWL, DRAM decodes the write command and turns on the DQ I/O circuit to receive data.

Figure 3.3 (e) shows an example of the refresh command. DRAM receives AREF at regular intervals, and this interval is tREFI. When the AREF is received, the DRAM refreshes several rows inside. tREFC is the time the AREF command is performed. Since other commands cannot be performed during tREFC, tREFC/tREFI means performance degradation by refresh.

3.1.4 Bit-wise Operation in DRAM

Depending on how logical operations are implemented, DRAM based accelerators are divided into charge sharing and gate logic methods.

Figure 3.4 (a) shows an example of AND and OR implementation of the charge sharing method [20]. We call a cell array connected to a sense-amp (S/A) array in DRAM a sub-array. Typically, a sub-array has

1k rows and 8k columns. For simplicity, the figure shows an example with three rows and one column. In the figure, since the cells (circles) are connected to the same bit line (BL0), S/A can read the cell of the selected row among them. This operation is called activation. If some of the rows, e.g., the above three rows, are selected at the same time, charge sharing occurs between cells, which yields, as the result, the majority of 0 and 1 values in the selected cells. This charge sharing can be used to implement logic operations.

In Figure 3.4 (a), Row(W), Row(A), and Row(S) have cells with stored weight, input activation, and select data, respectively. We call the activation of the neural network as the input or output data to distinguish it from DRAM row activation. Select data is a value for selecting an AND operation or an OR operation. Figure 3.4 (a) shows an example of performing an AND operation. In this case, since the select data is '0', the charge sharing result of the three cells, i.e., the select data '0', the weight '1' and the input '0', becomes 0.33. For simplicity, we use 1 and 0 as VCORE (high potential voltage) and VSS (low potential voltage), interchangeably. S/A evaluates the charge sharing result as '1' if it is higher than 0.5 and '0' when it is lower than 0.5. Thus, the result of the AND operation is '0'. In other words, if the select data is '0', S/A evaluates a charge sharing result as '1' only when both the weight and input are '1'. Figure 3.4 (a) also shows that the OR operation is enabled by the select data of '1'. In such a case, the S/A yields the output of '1'.

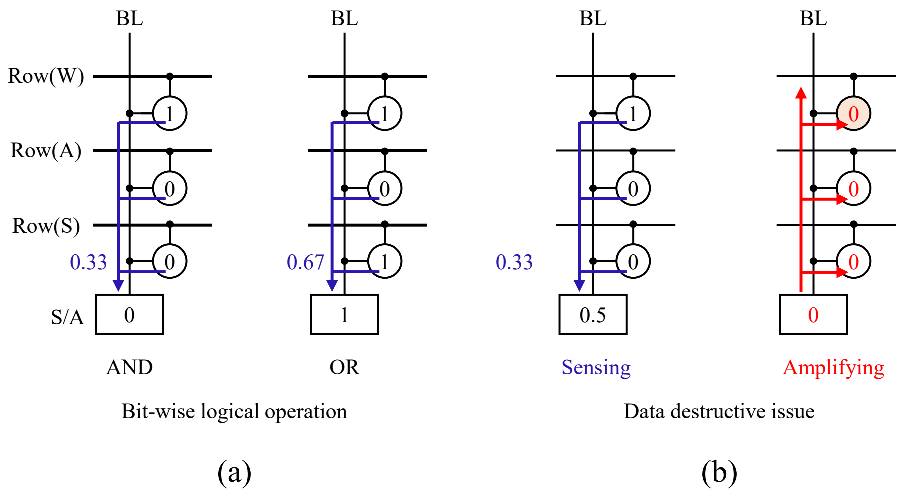


Figure 3.4 (a) Logic operation examples (AND and OR), and (b) data destructive access issue that the result is overwritten to the data cells.

There are two key issues to be considered in the charge sharing method, process variation and data destructive access. The effect of process variation on the computational results is small at 0.3 % at a process variation of 10 % [20]. Data destructive access issue means that the logic operation result is overwritten in the DRAM cell where the data is stored, as shown in Figure 3.4 (b) thereby losing the original data (of Row(W) in the figure). In order to address this issue, the original data are preserved by copying them using the row clone technique [64]. Since such a data backup is required for each operation, the latency of the charge sharing method is 100ns$\sim$1µs longer than the row cycle time, tRC, typically 50ns [20].

The gate logic method is a structure in which gate logic and latch are

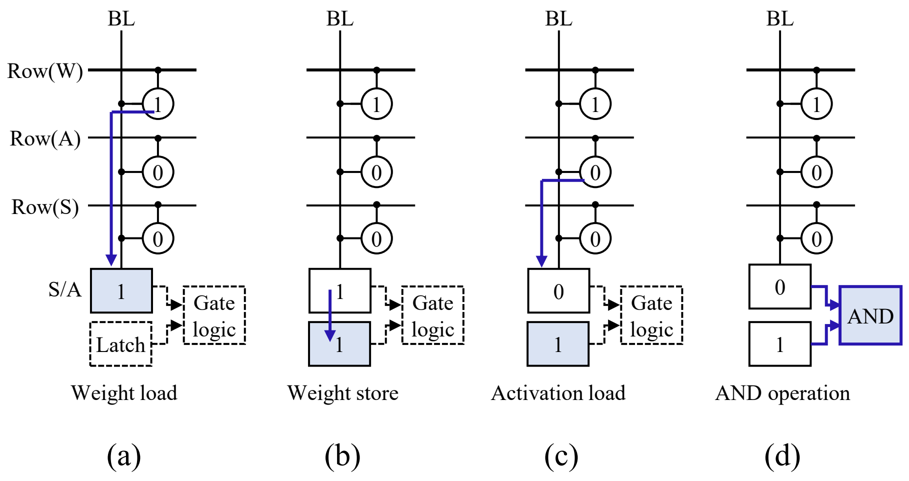


Figure 3.5 AND operation examples of gate logic method, (a) weight load, (b) weight store to latch, (c) activation load, (d) AND operation.

added to the output nodes of all S/A for bit-wise operation. Figure 3.5 is a simplified example of AND operation of the gate logic method. Activate Row(W) to load data '1' of Row(W) into S/A (see Figure 3.5 (a)), and store this value in a latch connected in parallel to the S/A output node (see Figure 3.5 (b))). Activate Row(A) again and load the data of Row(A) to S/A (see Figure 3.5 (c)). Figure 3.5 (d) shows that the data of Row(A) loaded in S/A and data of Row(W) stored in the latch are operated by AND gate logic connected to the S/A output terminal. However, the gate logic methods add circuits (latch and required gate blocks) to every sense amp, so the area overhead is very large (e.g., 24 % for DRISA 1T1C-nor) [21]. Since the accelerator is integrated to the memory, DRAM design issues should be considered. In the case of DRAM, manufacturing cost is the most important, and a method with large area overhead is unacceptable for DRAM.

3.2 Motivations

The previous works of the charge sharing method have two critical limitations: lack of NOT operation and high accumulation cost. First, the previous study did not implement a NOT operation using the charge sharing method, but instead added a new structure of DRAM cell [20]. Second, in the previous works, the large amount of intermediate results has to be transferred from DRAM to CPU for the accumulation. The size of the intermediate results is much larger than the sum of input, kernel, and output, e.g., 73.1MB vs. 2.6MB in a binary VGG-9 model with CIFAR-10 [39, 41]. Handling the intermediate results on the CPU incurs significant overhead in power consumption (and area). According to our experiments, especially, data movement latency is significant in both cases because the time to read the intermediate results from the DRAM cells is dominant. This problem occurs in DRAM-based accelerators and in other memory types that use large cell-arrays such as ReRAM. There is a need for a method to reduce latency and power to compute many intermediate results generated after executing a large number of bit-wise operations at once.

Moreover, a method of reducing input data movement is needed.

DRAM-based accelerator and ReRAM-based accelerator increase throughput by operating multiple sub-arrays (or banks) at once. For that, the operation must be executed after input data is written to all sub-arrays. The input data movement overhead that occurs, in this case, has not yet been addressed.

3.3 Proposed architecture

Figure 3.6 shows our proposed in-DRAM accelerator architecture based on 8Gb DDR4 DRAM. The basic structure of DRAM is maintained for operation in memory mode, and the circuits added for neural network (NN) mode are indicated in green. M2V (Matrix2Vector) converts the input matrix (or vector) into a vector, especially for im2col, which matches the DRAM structure and stores it in the DRAM row. Row operator performs the bit-wise logic operation and generates the partial sum by charge sharing in the DRAM sub-array. SiD (Sum-in-DRAM) accumulates partial sums (which obtained by the charge sharing method using Row operator) and generates the output of binary multiplication. The proposed structure accelerates the computation of the convolution and fully-connected layer. We assumed that the host processor executes output reshape (col2im), batch normalization (BN), activation, and pooling.

In our in-DRAM accelerator, for logic operations, we run one sub-array per bank and, thus, a total of 128 sub-arrays per rank in parallel

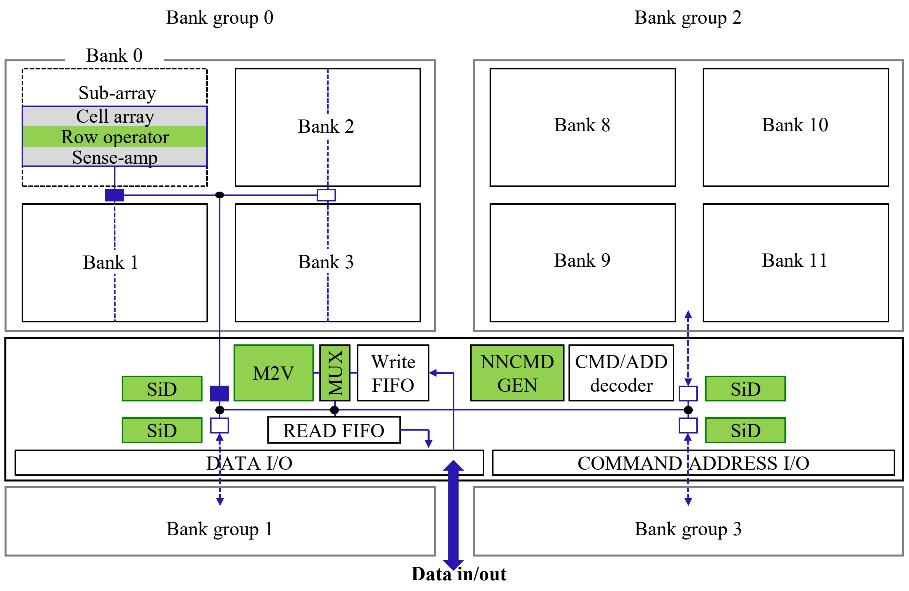


Figure 3.6 Proposed architecture (based on DDR4 architecture).

(=one sub-array per bank * 16 banks per chip * 8 DRAM chips per rank). Each bank stores different weight kernels and takes the same input. The same input data are broadcast to all banks. In Figure 3.6, the blue line represents the data bus, and the blue box the data repeater. The input is broadcast to all the banks by using all repeaters (blue boxes) of the write data path. Note that data movement latency can be hidden by operation latency via sub-array interleaving.

After running logic operations on bit lines in parallel, we obtain the final summation of them in two steps. First, we perform charge sharing-based accumulation on each sub-array. Second, we run the counter in SiD (sum-in-DRAM) module to count the partial sum results and obtain the final accumulation result.

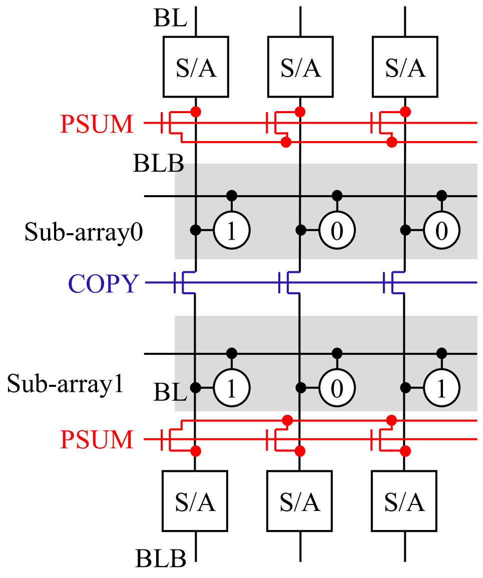


Figure 3.7 Proposed row operator (operand row is in sub-array).

3.3.1 Operation Examples of Row Operator

Figure 3.7 shows our proposed row operator structure, showing the newly proposed part in color in the existing structure of Figure 3.4. In terms of circuit, we add two types of transistors, i.e., switches, COPY (for NOT operation) and PSUM (for PSUM operation) to the existing sub-array.

NOT Operation. NOT operation judiciously exploits the adjacent sub-arrays and the pre-charge operation. Figure 3.8 illustrates NOT operation. As shown in the figure, a COPY transistor (in blue) connects two sub-arrays. NOT operation is realized by copying the data of a bit line on a sub-array to the bit line on the adjacent sub-array and activating its S/A. In Figure 3.8 (Sensing), the bit line (BL) of sense amplifier S/A1 has 1. By turning on the COPY transistor, the contents of the bit line, i.e., 1 is copied to the bit line bar (BLB) of S/A0 which was originally pre-charged to 0.5 as shown in Figure 3.8 (data copying). Note that the EQ transistor (used to pre-charge both BL and BLB) is turned on. Thus, the contents of BLB of S/A0 is propagated to the BL of S/A0 as shown in the figure. Then, as Figure 3.8 (Pre-charging) shows, S/A1 pre-charges BLB of S/A0 via the COPY path. After that, when S/A0 is turned on, since BL (1) of S/A0 has higher voltage than BLB (0.5), S/A0 evaluates BLB as ‘0’. When this value is copied to the BL of S/A1 via COPY path, ‘0’ (the inversion of ‘1’) is stored in the cell of S/A1 as shown in the figure (Inverting). Note that the COPY transistor, used to connect the BL and BLB of adjacent sub-arrays, is similar to the transistor connecting short and long bit lines of the same sub-array in [65]. Our difference is that we utilize the transistor to copy data between two adjacent sub-arrays.

PSUM operation. The procedure for obtaining the partial sum with charge sharing is as follows. For simplicity, let us assume that we activate the row of sub-array1 in Figure 3.7 to fully charge/discharge the three bit lines. In reality, we first perform logic operations (AND, OR or XNOR). Then, we perform charge sharing to obtain the partial sum of the logic operation results. Thus, the row In this case, each of the three bit lines has a value of ‘1’, ‘0’, and ‘1’, respectively. After activation is completed, we turn off S/A while turning on PSUM switch. Then, charge sharing occurs between the bit lines connected to the PSUM switches. In the figure, the

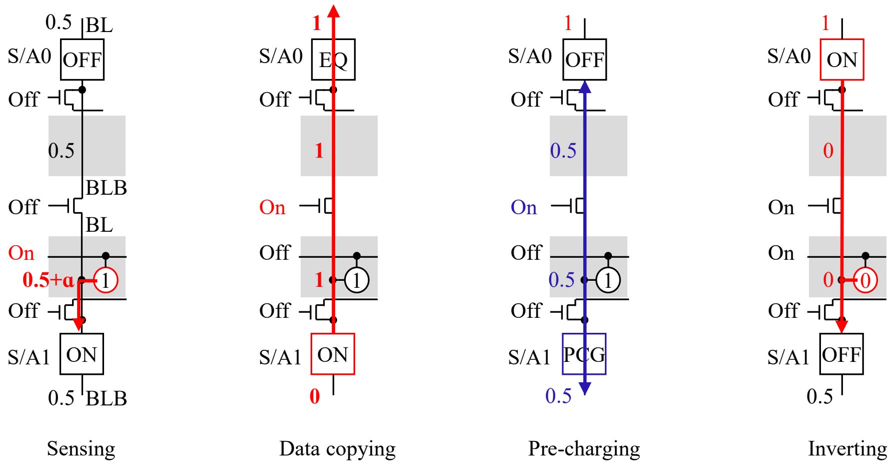


Figure 3.8 Operation example of NOT operation.

voltage of the three bit lines, connected by the PSUM switches, becomes 0.67 by change sharing. When we turns on S/A again, the bit line voltage of 0.67 is evaluated as logic ‘1’, as shown in Figure 3.9 (b). PSUM switch is required for each bit line, as shown in Figure 3.9 (b), to obtain the partial sum.

Figure 3.10 illustrates an accumulation for a dot product operation between two vectors of 64 bits. For simplicity, we use a smaller example than our real implementation. First, we perform bit-wise XNOR operation between row(A) and row(W). We explain the details of XNOR operation later in this section. After obtaining the results of XNOR operation, we run charge sharing based accumulation. As shown in the figure, we perform two steps of charge sharing based accumulation. It is due to the sensing margin of sense amplifier as will be explained below. In the first

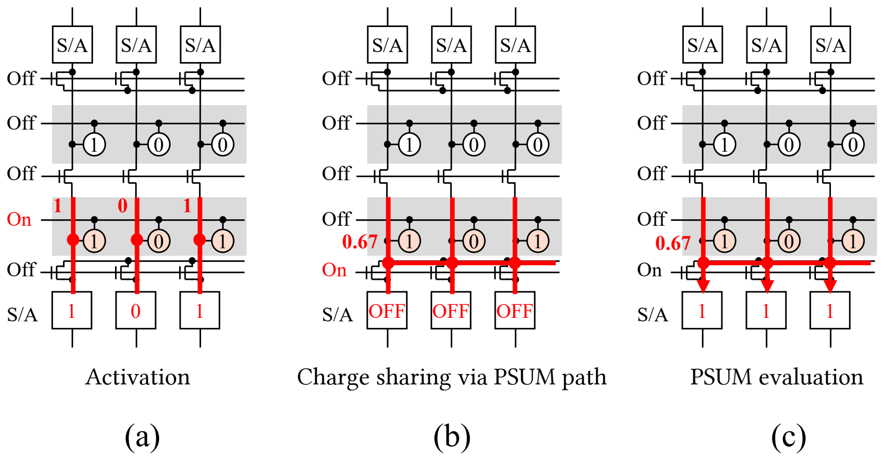
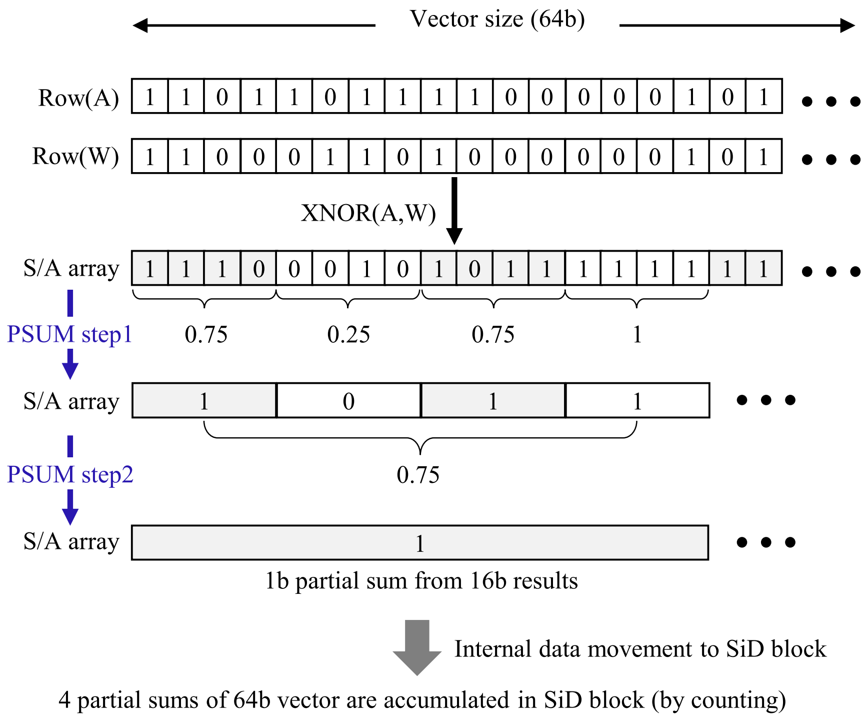


Figure 3.9 Operation example of PSUM operation.

step of accumulation, PSUM step 1 in the figure, we perform accumulation for each of four consecutive output bits. For instance, the accumulation of the first four bits, ‘1’, ‘1’, ‘1’, and ‘0’ produces 0.75 in the charge sharing based accumulation. The sense amplifier evaluates the results as ‘1’ or ‘0’ as shown in Figure 3.10. Then, we perform the second step of accumulation, PSUM step 2. In this case, the previously obtained partial sum results in PSUM step 1 are accumulated via the same charge sharing method. As the figure shows, four previous outputs, ‘1’, ‘0’, ‘1’, and ‘1’ are accumulated to produce 0.75V, which is finally evaluated by the S/A array to produce ‘1’.

In Figure 3.10, we perform each step of charge sharing based accumulation for a group of 4 bits. In our experiments, we utilize a group of 16 bits and call the number of bits psum_width. In our experiments,



Vector size (64b)
Row(A) 1 1 0 1 1 0 1 1 1 1 0 0 0 0 0 1 0 1 • • •
Row(W) 1 1 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 • • •
XNOR(A,W)
S/A array 1 1 1 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 • • •
PSUM step1
0.75
0.25
0.75
1
S/A array 1 0 1 1 • • •
PSUM step2
0.75
S/A array 1 • • •
1b partial sum from 16b results
Internal data movement to SiD block
4 partial sums of 64b vector are accumulated in SiD block (by counting)

Figure 3.10 Example of the proposed accumulation flow.

we obtained the psum_width considering the sensing margin of sense amplifier as follows. In case of VCC = 1 , the sensing margin of normal activation operation is 0.025 assuming the bit line capacitance is 20 times larger than the cell capacitance. In order to sense the charge sharing results, the charge sharing step size (=1/psum_width) must be at least twice the sensing margin. It is because the sense amplifier utilizes half VCORE, i.e., from 0.5 to 1 or 0. As a result, we obtained the maximum psum_width of 20.

In our experiments, we perform in-DRAM accumulation for the granularity of 128 bits via sensitivity analysis (as will be shown in Section IV). Thus, we divide 128 bit summation into two steps (psum_width = psum_width1 * psum_width2): the first step using psum_width1 (=16), and the second step using psum_width2 (=8). Each step takes tRC (=45ns) since the sense amplifier senses larger voltage difference than its original sensing margin

We realize operations other than basic operations (AND, OR, NOT) by a combination of basic operations. Figure 3.11 shows the operation flow of XNOR used for binary multiplication in BNN. The left column shows the operation flow of XNOR consisting of four logic operations. The middle column shows the operation flow of the OR operation, and the right column shows the flow that copies data to the row operator. Table 3.1 shows the latency and energy consumption of the logic operations. We estimated these values using the post-layout simulation result

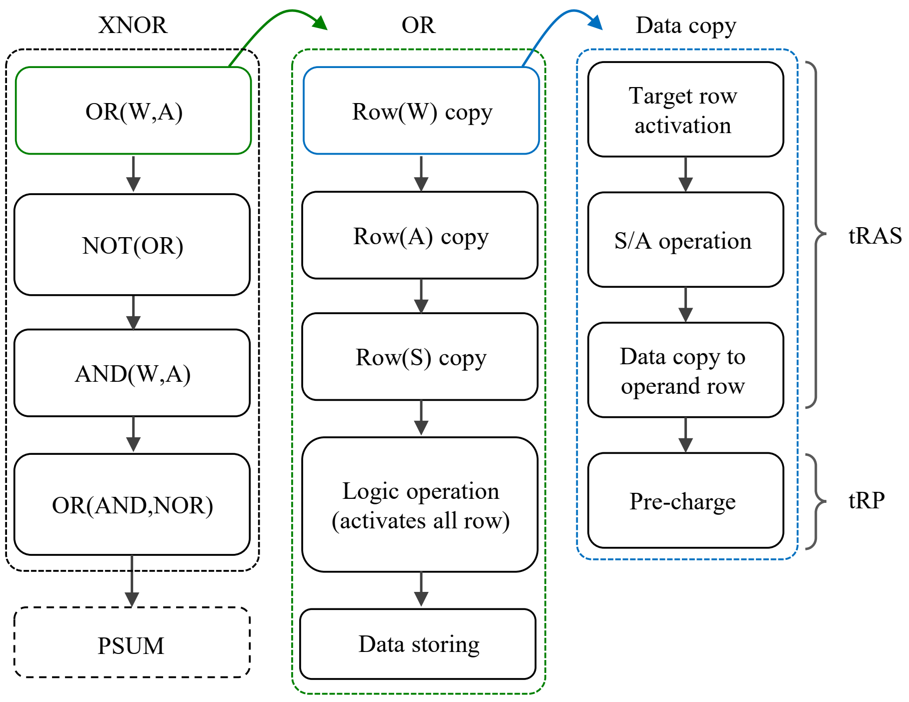


Figure 3.11 Operation flow of XNOR in proposed architecture.

Table 3.1 Latency and energy of operations in a sub-array

Operation	Latency	Energy
AND,OR	195ns	11.0nJ
NAND,NOR	270ns	19.2
XNOR	505ns	41.3nJ
NOT	135ns	8.2nJ
PSUM	90ns	5.4nJ

of 1x-nm DRAM and a DDR4 power calculator [13, 14].

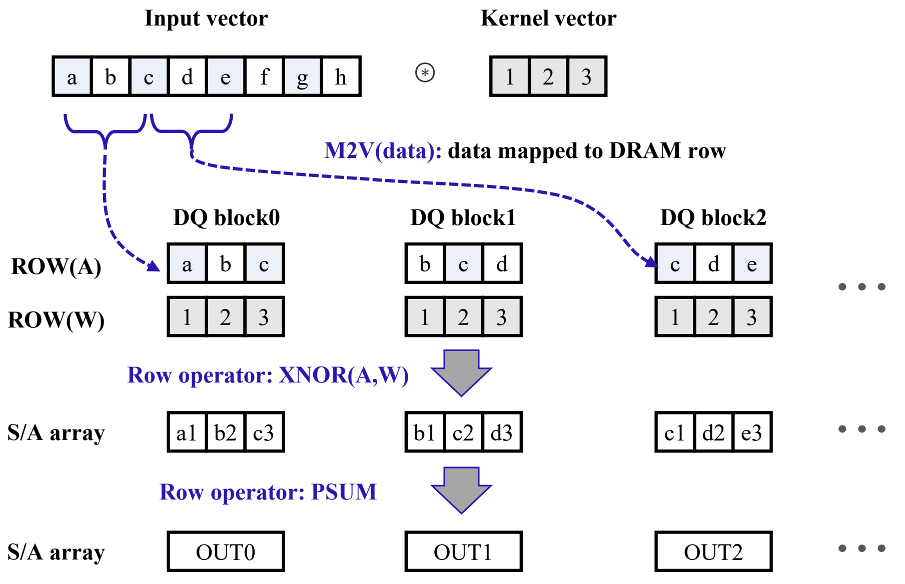
3.3.2 Convolutions on DRAM Chip

Figure 3.12 is an example of executing 1D-convolution in the proposed structure. In the 1D-convolution of the input vector and the kernel, the dot product of the partial input and kernel performed while the kernel vector moves one (stride=1) in the length direction of the input vector, and all results are accumulated. As shown in Figure 3.12 (a), the proposed structure executes the dot product in parallel in all DQ blocks of the sub-array. The result of 1D-convolution is obtained by accumulating the intermediate results (output in S/A array) in the SiD block.

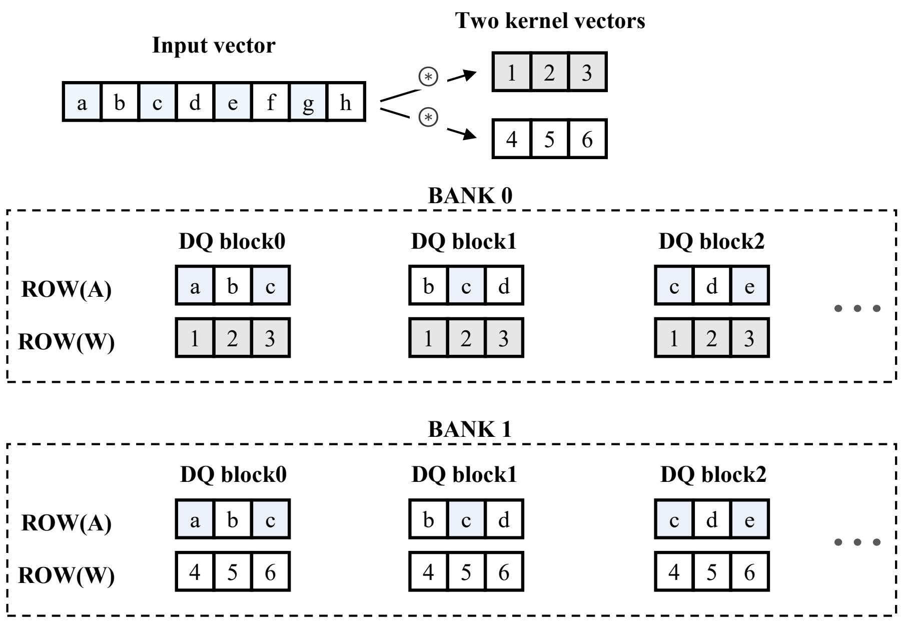
M2V maps and splits the input vector to the DQ block and stores the partial input vector. It is executed while receiving the input vector, and no additional data transmission or cycle is required. The M2V write operation is described in a later section. Kernel vector is stored in DRAM at the beginning of neural network execution, so it can be operated only by writing input. For parallel operation, the same kernel vector is stored in all DQ blocks.

Dot product is executed in Row operator. The row operator executes bit-wise multiplication using XNOR, and PSUM operation on the bit-wise multiplication result. All elements in the picture are binary values.

Figure 3.12 (b) is an example of a 1D-convolution of an input vector



(a)



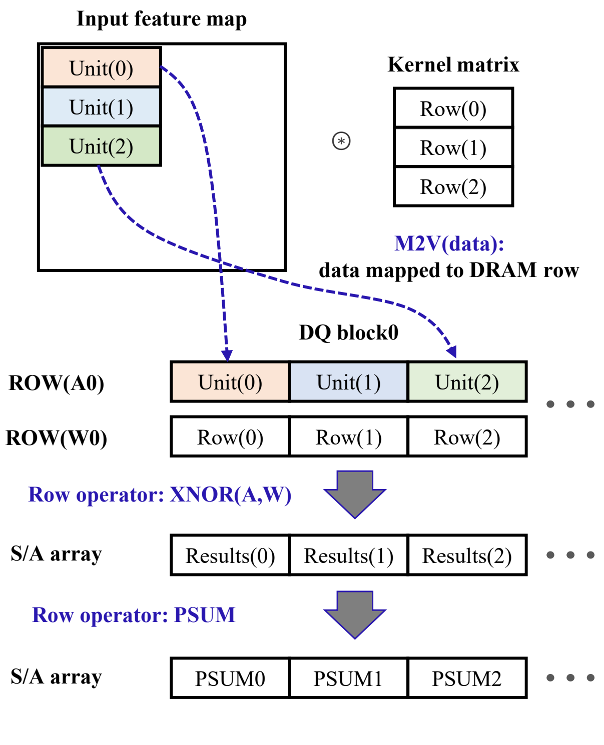
(b)

Figure 3.12 Example of 1D-convolution on DRAM chip.

with several kernels. It is the same as executing Figure 3.12 (a) in several banks at the same time. Different kernels are stored in the sub-array of each bank. At this time, the kernel stores the same address in each bank. In other words, except for the bank address, the address where the kernel running in parallel is stored is the same. Input is also stored in the same location in all banks. The data is stored by broadcasting to all banks. By broadcasting CMD, all banks can perform convolution for different kernels with the same control signal.

In the proposed structure, since there are 16 banks, 16 kernels can be operated simultaneously. When the number of kernels is greater than the number of banks, several kernels are stored in each sub-array, and the sub-array repeats convolution while changing the kernel. Furthermore, by applying sub-array level parallelism [66] to different sub-arrays of the same bank, it is possible to increase the number of sub-arrays operating at once.

Figure 3.13 is an example of a 2D-convolution operation. The row size in the kernel matrix is called unit, and the write sequence of M2V maps the input to the DQ block in unit size. When expanding to 3D-convolution, it is simply applied by increasing the unit as much as the the kernel matrix channel. As shown in Figure 3.13 (a), write the input feature map in order of unit(0), unit(1), and unit(2). The kernel is pre-stored in the same order as the input was written. Subsequent operations are performed in the same way as 1D-convolution.



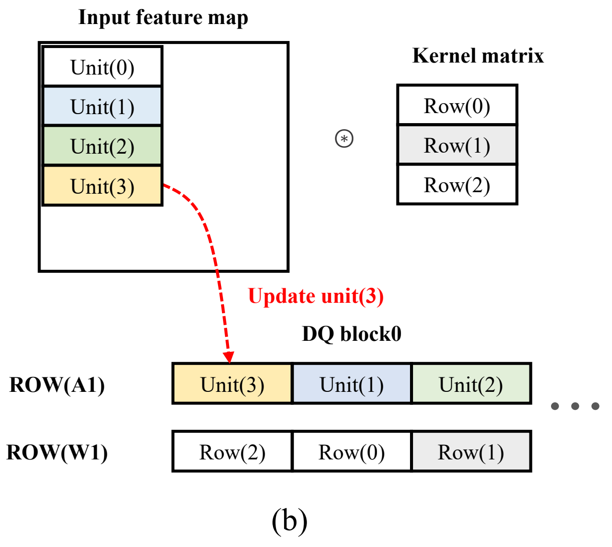


Figure 3.13 Example of 2D-convolution on DRAM chip.

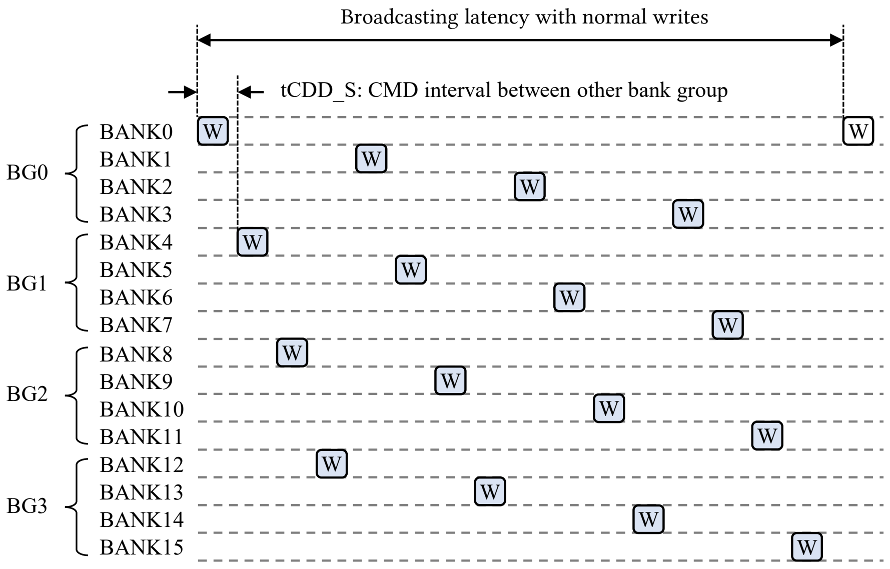
Figure 3.13 (b) is an example of stride and convolution in the height direction in Figure 3.13 (a). The stride in the width direction is executed simultaneously in the DQ block (see Figure 3.12 (a)), and the stride in the height direction is executed sequentially in the same sub-array (see Figure 3.13 (b)). It is an easy way to reuse the input stored in Row(A0) in Figure 3.13 (a). Since we assumed that the operation is performed after all input feature maps are stored in DRAM, the input of Figure 3.13 (a) is in Row(A0), and the input of Figure (b) is in Row(A1). To store Row(A1) of Figure 3.13 (b), copy Row(A0) to Row(A1). With the row clone technique, copying is possible during tRC without re-transmitting data. After the row copy, unit(3) is written to the unit(0) in Row(A1). Since unit(1) and unit(2) is already in Row(A1), off-chip data movement is reduced by about 67 %. For convolution, the location of the kernel unit must be the same as the input. The order of units is different between Row (W0) of Figure 3.13 (a) and Row (w1) of Figure 3.13 (b). Sub-array stores all kernels with different unit order. Since the sub-array has 1024 rows, it is possible to save the kernel.

3.4 Data Flow

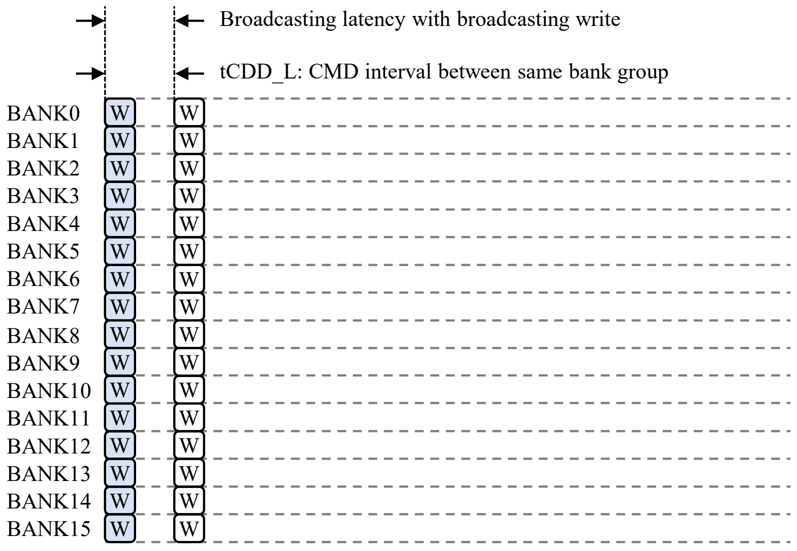
3.4.1 Input Broadcasting in DRAM

As shown in the Figure 3.12 (a), to run multiple kernels in parallel in all banks, input data must be written to all banks. Figure 3.14 (a) shows that DRAM writes data to all banks in Memory mode. All bank rows are activated, and write is repeated while changing only the bank address. This write sequence is a bank interleaving write. Figure 3.14 (b) shows the operation of simultaneously writing data to all banks in NN mode. The write latency of Figure 3.14 (a) is 40 ns, and the write latency of Figure 3.14 (b) is 5 ns, which is 87.5 % reduced. It significantly contributes not only to latency reduction but also to data movement power reduction. Because the off-chip data movement decreases to 1/16, and the internal data movement decreases to 1/4 compared to repeatedly writing the same data by the number of banks. Since GIOs are separated for each bank group, internal data movement is required once for every bank group.

The broadcasting method in the Figure 3.14 (b) is implemented by changing only the repeater control signals of CMD and DATA BUS. Figure 3.15 (a) is the data path when writing to BANK0 in Memory mode. Blueline is data BUS, redline is CMD BUS, the empty box is the turned-off repeater, and the filled box is the turned-on repeater. Figure 3.15 (a) turns on only the repeater connected from CMD and data bus to bank0. In



(a)


Broadcasting latency with broadcasting write
tCDD_L: CMD interval between same bank group
BANK0
BANK1
BANK2
BANK3
BANK4
BANK5
BANK6
BANK7
BANK8
BANK9
BANK10
BANK11
BANK12
BANK13
BANK14
BANK15
W

(b)

Figure 3.14 Examples of write the input to all banks, (a) bank-interleaving write, and (b) broadcasting write of the proposed architecture.

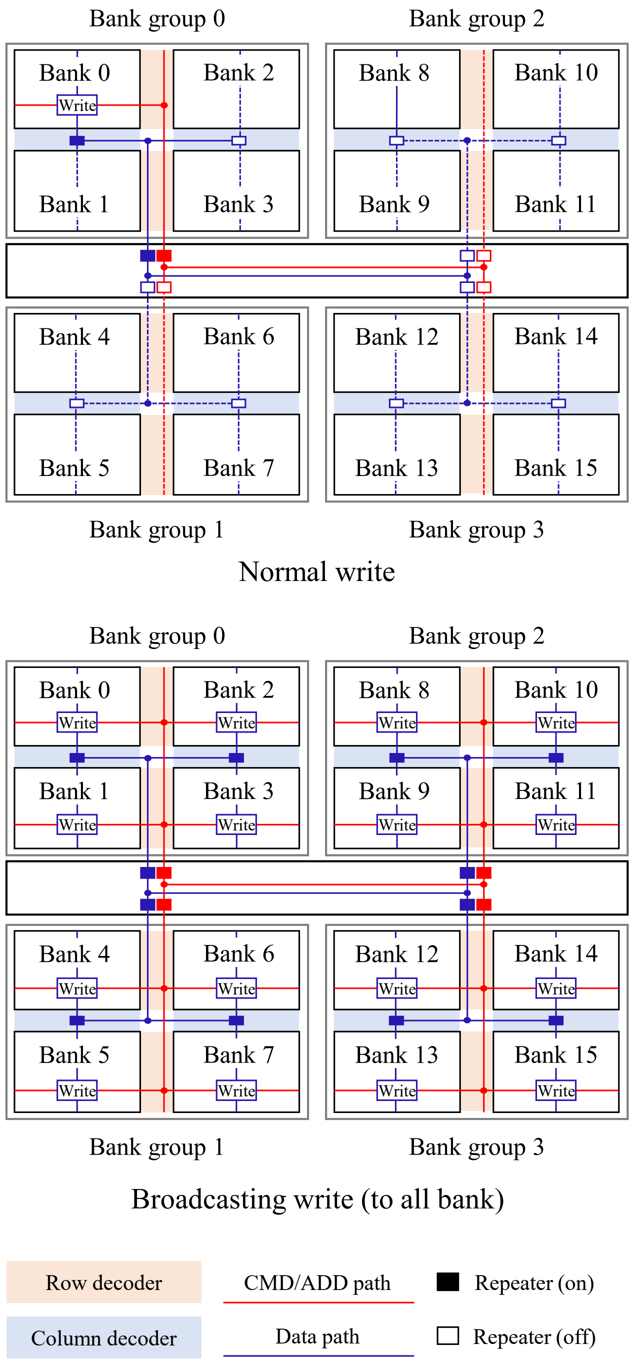


Figure 3.15 DATA and CMD BUS control for broadcasting.

the NN mode broadcasting in Figure 3.15 (b), the BUS's repeater control is changed. By activating the BUS control signal for each bank simultaneously, input data can be transmitted to all banks at the same time.

3.4.2 Input Data Movement With M2V

The Figure 3.16 shows an example of the operation of M2V. Figure 3.16 (a) shows the transmission of the data of the input feature map to DRAM. Since the DRAM chip has 8 DQ and the data length (BL) is 8, (H=1, W=8, C=8) is transferred to the DRAM chip in the input feature map of Figure 3.16 (a). As the data format is transmitted through the memory channel, the 1x1x8 (H, W, C) vector is transmitted as much as BL 8. M2V transposes and stores the transmitted vector. In other words, the vector is mapped to the BL in the memory channel, but M2V maps the vector to the DQ block and stores it.

Figure 3.16 (b) shows the write command sequence of NN mode. There are three commands for M2V write (M2V write BL8, M2V write BL4, M2V shift). M2V write BL8 (M2V_BL8) is a command to transmit input, as shown in Figure 3.16 (a). M2V write BC4 (M2V_BC4) is a command to transmit the data necessary for the kernel stride. For a 3x3 kernel, the convolution requires two more vectors. When transmitting two vectors with M2V BC4, the remaining data length is fixed to 1, and data is transmitted. (In the DDR4 I/O structure, off-chip data movement

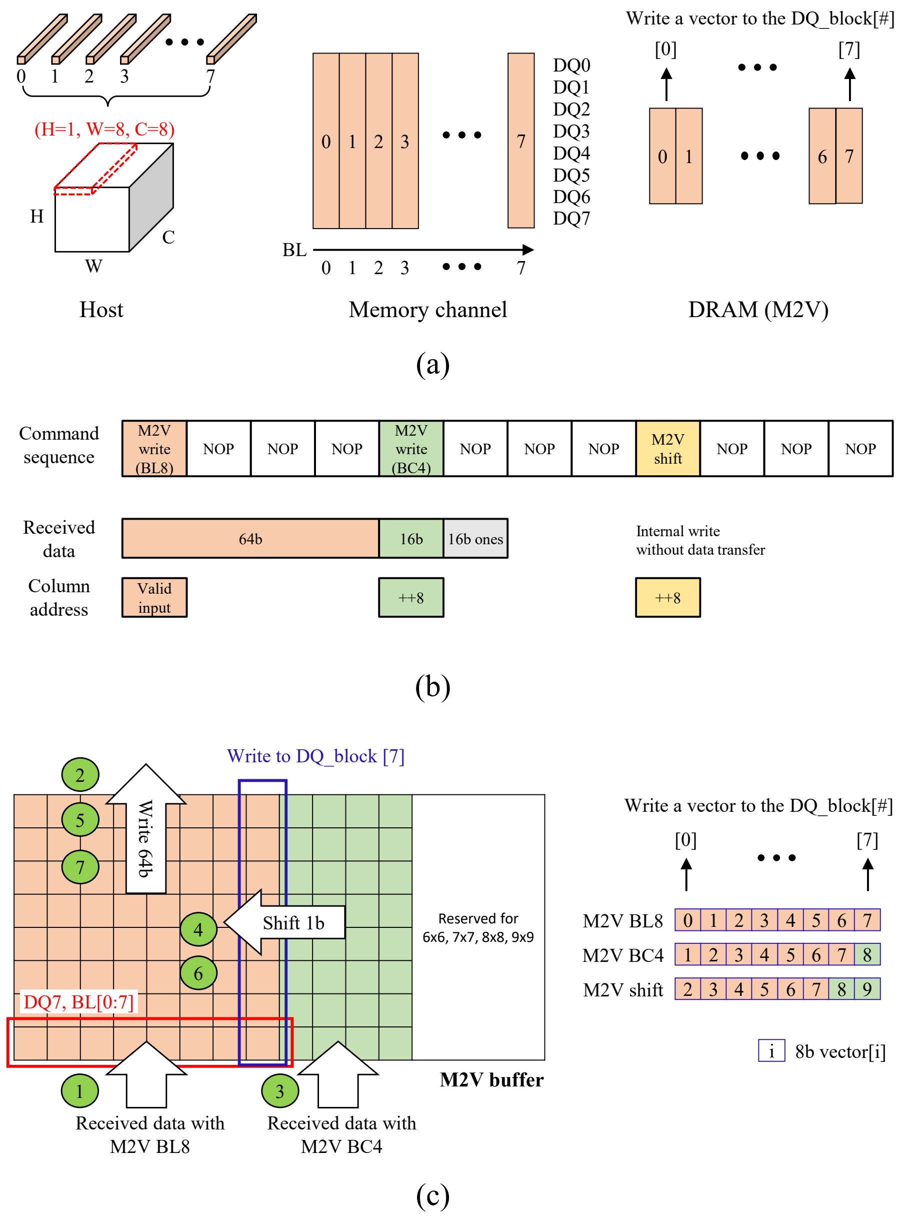


Figure 3.16 Example of M2V operation. (a) Input feature map transfer, (b) M2V write sequence in NN mode, and (c) M2V write operation with data reuse.

energy is not consumed to transmit data '1' continuously.) The proposed structure supports kernel from 1x1 to 5x5 and can be extended up to 9x9.

Figure 3.16 (c) shows the operation of M2V according to the command in Figure 3.16 (b). In the 3x3 convolution case, the M2V reuses the input feature map and stores it in the DQ block. The data received with the M2V_BL8 is stored in the M2V buffer and transmitted to the DQ block. The data received with the M2V_BC4 is stored in the M2V buffer, and the data in the M2V buffer is shift-left, then transmitted to the DQ block. The M2V_shift command shifts the M2V buffer data to the left without external data transmission. M2V stores the M2V buffer data to the DQ block. Without increasing the off-chip data movement, the M2V block applied im2col to the 3D-feature map and stored it as a vector in the DQ block.

3.4.3 Internal Data Movement With SiD

Figure 3.17 shows that SiD block accumulate eight partial sums of single DQ block. One SiD block accumulates eight DQ blocks in parallel. When accumulating with SiD, DRAM does not transmit data to the outside. NN mode uses the command SiD_iRD for internal read-only and the SiD_eRD command for transmit data to the outside after accumulation. As shown in the table, in NN mode, the read with auto pre-charge (RDA) command is changed to SiD command. When SiD_iRD command

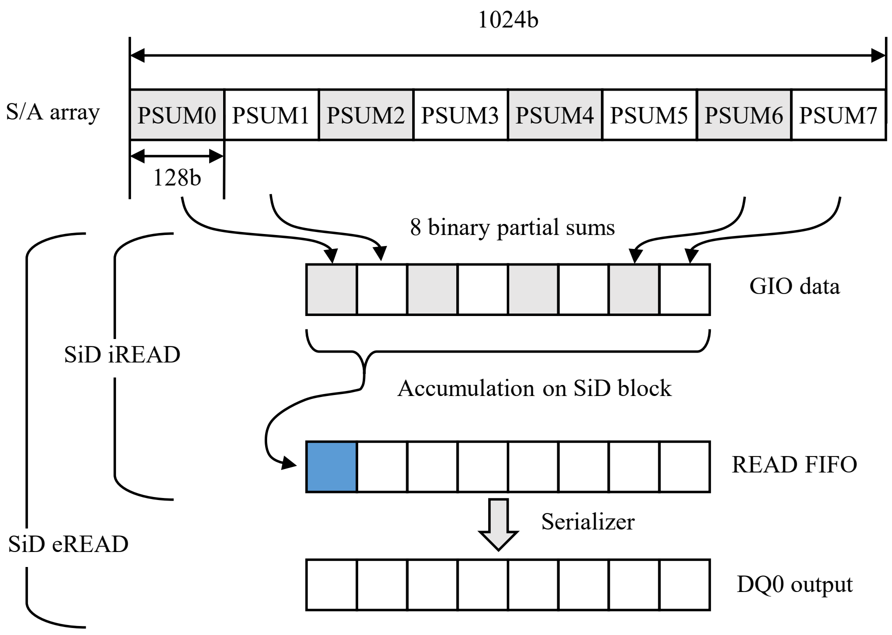
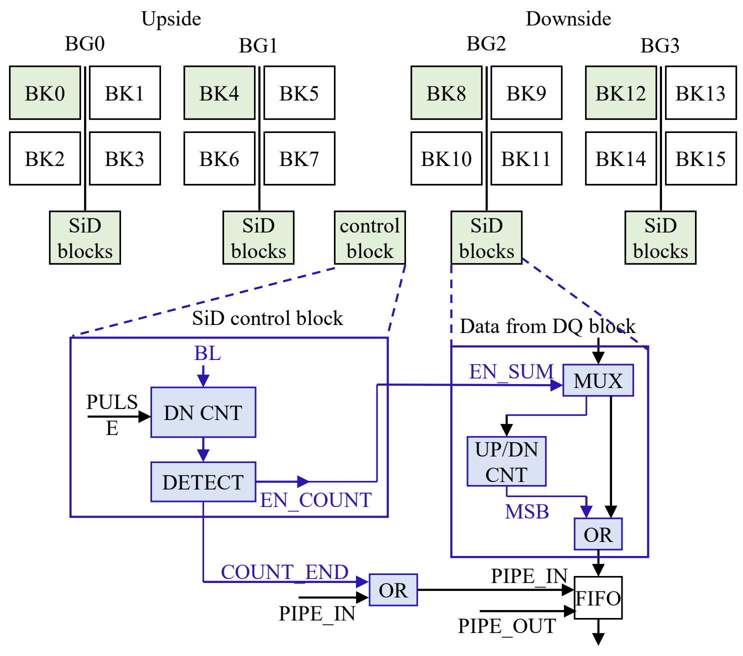


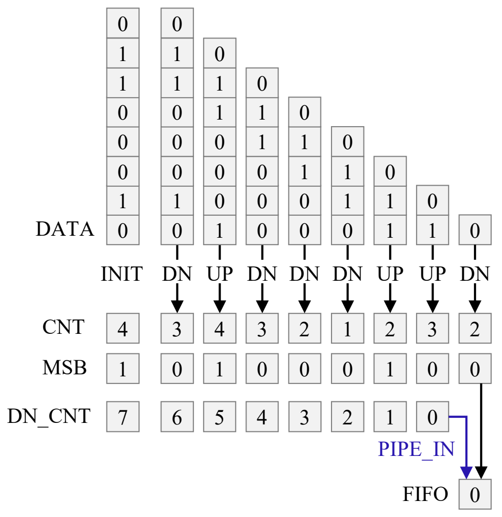
Figure 3.17 Example of SiD operation.

is input, eight binary partial sum data are read from DQ block, and SiD accumulates this 8b data. Accumulation results are stored to the read FIFO. Since the data length transmitted to the outside is 8b per DQ, SiD operation is repeated eight times to fill BL of the read FIFO.

Figure 3.18 is a SiD block diagram. One SiD block accumulates the data of 8 DQ blocks in parallel. Since each BG has separated GIO, SiD blocks are added to each BG to increase parallelism. The control block controls 4 SiD blocks. When SiD_iRD is input, the control block generates EN_SUM signal to accumulate (BL) as much as the data length transmitted to the SiD block. EN_SUM signal is a high pulse of BL length



(a)



(b)

Figure 3.18 SiD block diagram. (a) 4 SiD blocks on DRAM for parallel operation, and (b) accumulation example.

and is generated by using a down counter with a zero detector.

SiD accumulation result is a signed value using an up/down counter. The initial value of the counter is 0. If the input is ’0’, it counts as -1, and if ’1’, counts as +1. Note: In the proposed structure, the logic level ’0’ means network data’-1’. It operates with data ’-1’ and ’1’ (see figure logicalOperation). When accumulation is ended, EN_SUM goes low and COUNT_END signal is generated as a pulse. COUNT_END generates PIPE_IN signal through OR gate and stores the sign bit (MSB) of the accumulation result in the read FIFO.

Figure 3.18 (b) is an example of this accumulation. Eight binary partial sums are input, and the initial value of the 3bit up/down counter (CNT) is 4. At the circuit level, it is expressed as an unsigned value, so 4 is the same as a signed value ’+’0. Accumulation is executed until DN_CNT of the control block becomes 0, and MSB of CNT is stored in FIFO.

3.4.4 Data Partitioning for Parallel Operation

When applying a parallel operation to multiple DRAM chips, the input feature map should be partitioned according to off-chip data movement and DRAM memory structure. For example, in the rank of Figure 3.6, the data bus is separated between DRAM chips. When all DRAM chips convolution with the same input, redundant writes increase 7 times and

Table 3.2 Policy of input feature map partitioning

Name	Resource	Assign	DATA BUS	Priority	Partitioning order
Channel	1∼2	Dynamic	Separated	6	H, N, C
Rank	1∼4	Dynamic	Shared	5	H, N, C
Chip	8	Fixed	Separated	2	W, C, N, H, KN
BANK	16	Fixed	Shared	3	KN
DQ	8	Fixed	Separated	1	W
MAT	1∼4	Dynamic	Shared	4	N, KN, C

off-chip data movement power increases 7 times. Therefore, it is necessary to distinguish between a resource that shares DATA BUS and a resource that does not share. The operation latency is the same, but the input can be partitioned with the smallest off-chip data movement.

Table 3.2 summarizes the items required for input partitioning. Resource column is the number of resources that can be assigned, and Assign is an assignment method. For example, DQ, bank, and chip are physically fixed values in the DRAM module (x8 DDR4). DATA BUS is whether the BUS is shared. Priority is the order of improvement effect of data movement. Grouping order is a data dimension suitable for allocation in each resource. N, C, H, and W denote the batch size, channel, height, and width dimensions of the input feature map. KN is the number of kernels.

The grouping order determines what data is allocated to the resource. In order to increase the parallelism of the In-DRAM accelerator and reduce the off-chip data movement, we partition the inputs in the order of H/W resource priority. Since the data reuse method for the input matrix width direction (with M2V) and height direction (with Row copy) is different, we do not use a method of allocating H/W resources in the order of input dimensions. The grouping order reflects the data reuse pattern of the proposed structure. For example, since data in the input width direction is reused using M2V, it can be partitioned as a DQ resource. When partitioning with other resources, there is no data reuse effect, and there is no effect of data movement reduction using M2V. Since BUS is separated between chips of DRAM rank, it is possible to partitioning for all dimensions of input. However, to maintain the effect of row reuse in the input's height direction, the priority of height is low. Since banks are suitable for input broadcasting, only kernel partitioning is applied to banks.

Figure 3.19 shows an example in which inputs are allocated to DRAM according to the priority and partitioning order in Table 3.2. As shown in Figure 3.19 (a), the inputs allocated to the DRAM chip are displayed in color. Figure 3.19 (b) is the input feature map (batch size=1) of the VGG-9 conv2 layer. In order to allocate different data to 8 DRAM chips, it was divided into 4 in the width direction and 2 in the height direc-

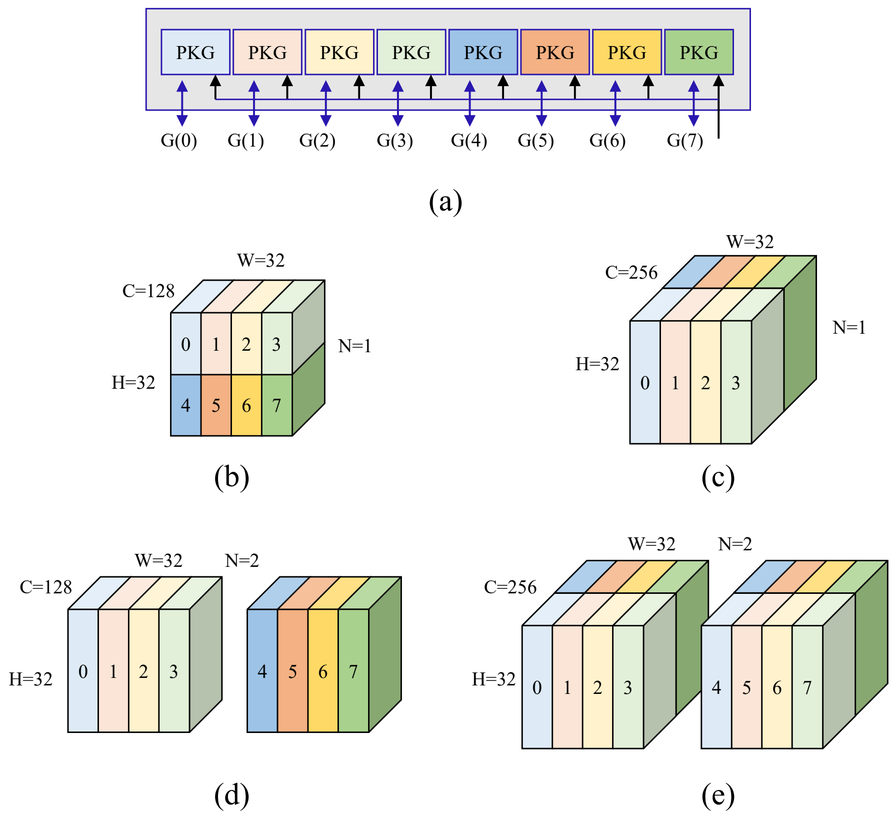


Figure 3.19 Input feature map partitioning for parallel operation.

tion. Figure 3.19 (c) shows the case where the channel length is doubled. Partitioning was performed in the width and channel directions. This is because row reuse in the height direction was considered. Figure 3.19 (d) is an example of batch size 2 in Figure 3.19 (b). At this time, instead of height, batch was partitioned. Figure 3.19 (e) is an example in which the channel is doubled in Figure 3.19 (d). Since the partitioning order of the channel is higher than that of batch, it is partitioned to execute batch 2 after batch 1.

3.5 Experiments

3.5.1 Performance Estimation

The base structure is 8Gb x8 DDR4 DRAM and the timing parameter of 3200 Mbps is used (e.g. tRAS = 35 ns, tRP = 15 ns, tRC = 50 ns). In previous experiments [67], we have confirmed that multi-row activation is possible at the same time without changing timing parameters. Table 3.1 shows latency and energy consumption which was estimated using the post-layout simulation result of 1x-nm DRAM and a DDR4 power calculator. We assumed a DIMM module as the base main memory where the in-DRAM accelerator runs. In previous work [68], the area overhead is estimated within 1.22 % of base DRAM chip due to the additional rows of Row operator, COPY and PSUM transistors in sub-arrays. In this dis-

Table 3.3 Performance of the proposed architecture

3200 8GB DIMM	Binary networks with CIFAR-10		
(x8 8Gb DDR4)	VGG-9	ResNet-14	MobileNetV2
Peak throughput (TOPS)	4.3	4.4	3.2
Average utilization (%)	93.6	94.6	67.6
Computation latency	282.5 µs	295.6 µs	49.7 µs
Data movement latency	130.0 µs	158.5 µs	260.6 µs
Latency per image	412.5 µs	454.1 µs	310.3 µs
Effective throughput (TOPS)	3.0	2.9	0.5

sertation, we copy the data row into two rows at a time when backing up data to improve data backup time. So, we reduced the vector dot product latency is reduced to 452 ps from [68]. The row operator's area overhead is increased two times, and the total area overhead is 1.9 %. Also, we assumed that DRAM row repair uses redundant rows provided in the same sub-array. And, functions other than convolution, such as batch normalization, activation function, max-pooling, etc., are assumed to be executed in the host processor.

Table 3.3 shows the performance of CIFAR-10 with binary VGG-9, ResNet-14, and MobileNetV2. In [68], MAC operation was counted as

1-OP to compare the performance between DRAM-based accelerators. In this dissertation, we counted the MAC operation as 2-OP to compare the DRAM-based accelerator's performance with the ASIC accelerator.

The peak throughput and utilization of VGG-9 and ResNet-14 is similar while MobileNetV2 gives lower throughput and utilization. Depthwise separable convolution of MobileNetV2 adversely affects throughput due to 3x3 depthwise (i.e., two-dimensional) convolution.

It is because 3x3 depthwise convolution yields 9b vectors of input activations in matrix (input)-matrix (kernel) multiplication of lowered convolution. In such a case, our charge sharing circuit, which performs charge sharing summation at the granularity of 16 bits, has only 9 active bit lines among 16, which reduces utilization and throughput.

Note that the data movement latency of VGG-9 and ResNet-14 are both less than computation latency. This is because the data movement size is reduced 130 times smaller. We reduced the redundant input data movement by using M2V and broadcasting write. We reduced the output data size to be transferred to the CPU by 128 times via the two-step accumulation in DRAM.

3.5.2 Configuration of In-DRAM Accelerator

We generate a binary partial sum with a DRAM sense amp, without ADC. The area overhead of the proposed architecture is small, but an ac-

Table 3.4 Network accuracy comparison with different psum_width

Binary network	Accuracy@CPU	psum_width			
		64	128	256	512
VGG-9	90.37	87.05	86.92	86.29	82.69
ResNet-14	88.40	83.69	82.99	82.25	79.38

curacy of the neural network drop occurs. Note that the larger psum_width (the granularity of in-DRAM accumulation) makes it possible to further reduce data transfer for intermediate results. Table 3.4 shows how we determined the psum_width of 128 as mentioned in Section 3.3.1.

Table 3.4 shows the results obtained by training 300 epochs from random initialization. All subsequent experiments use the same condition. We ran only convolution in DRAM since both networks have different numbers of FC layers. Table 3.4 shows that VGG-9 is more suitable than ResNet-14 for in-DRAM accelerators. The total computation of convolution layers is almost same, but VGG-9 has 50 % or more kernels than ResNet-14 in each conv layer. We use the VGG-9 in the following experiments.

As the table shows, as psum_width increases, accuracy is decreased. It is because the accumulation is performed in a premature way in case of large psum_width. The larger accumulation granularity, the more preci-

Table 3.5 Network accuracy with different number of ensembles

Binary network	Accuracy@CPU	Ensembles			
		1	2	4	8
VGG-9	90.37	86.81	88.11	89.05	89.52

sion error is incurred. According to Table 3.4, we set psum_width to 128, which offers a large psum_width at a tolerable loss of accuracy compared with the case that the BNNs run only on the CPU without further accuracy loss due to charge sharing-based in-DRAM accumulation. In the following sub-section, we recover the accuracy drop to 0.70 % from 3.56 %.

3.5.3 Improving the Accuracy of BNN

The proposed architecture loses the accuracy with binary partial sum. We improve the accuracy by adopting ensembles and increasing network size. Table 3.5 and Table 3.6 show how accuracy gets improved by adopting larger ensembles and larger kernels, respectively. We trained the ensemble networks with 80 % of training data randomly selected from the training set. The tables show the accuracy of running Conv2$\sim$FC2 layers of the network in DRAM (without first and last layer of the network).

In Table 3.5, when using 8 Ensemble networks, the accuracy is im-

Table 3.6 Network accuracy with different network size

Binary network	Vanilla model	Base kernel number			
	128	112	168	224	280
VGG-9	86.81	85.26	88.38	89.20	90.14
VGG-9 w/ imptW	87.80	86.77	88.79	89.67	

proved by 2.7 %. However, since it is about 0.4 % improvement per ensemble network, the efficiency is low. Due to computing energy, the ensemble network cannot apply to the in-DRAM accelerator.

Table 3.6 compares the accuracy according to the network size. Table shows the results of two cases, 'VGG-9 case' trained with a random initial value, and 'VGG-9 with important weight case (VGG-9 w/ imptW)' using the important weight of VGG-9 as an initial value. VGG network determines the number of kernels in Conv layer as a multiple of base kernel number (the number of Conv-1 kernels). In consideration of utilization, the number of kernels was increased to 168, 224, and 280. In the case of VGG-9, when the base kernel number is 280, the accuracy is improved to 90.14. The accuracy drop is very small at 0.23 %, but the network size increases by 2.9 times.

To reduce the computation overhead of VGG-9 case, we further improved the accuracy by training. In VGG-9 w/ imptW case, the weight of VGG-9 case is used instead of the random value for the initial value during training. In the BNN training process (for back-propagation), full precision weights are used. We use the weight larger than the threshold in VGG-9 case as an important weight. For other weights, a random value is used as an initial value. We set the threshold to 1. In VGG-9 w/imptW case, when the base kernel number is 224, the accuracy drop improves to 0.7 % and the network size increases by 2.1 times. Computation energy consumption is 34.4 % smaller than that of VGG-9 case. Based on the analyses shown in Tables 3.5 and Table 3.6, we chose the network having 224 kernels of Conv1 which offers the best accuracy for the comparison with the existing methods (shown in the following sub-section).

3.5.4 Comparison with the Existing Works

We compared the proposed architecture with Ambit and DRISA on the same sub-array characteristics. Since the original Ambit and DRISA utilize different sub-array characteristics, we call our modified ones also Ambit and DRISA. In case of DRISA, the sub-array size is halved from the original DRISA [21]. Therefore, the multiplication operation latency is halved in DRISA. However, both Ambit and DRISA do not have an in-DRAM accumulation solution, so they accumulate the intermediate results in CPU suffering from long data movement latency and high power consumption on off-chip DRAM accesses. Instead, Ambit and DRISA

Table 3.7 Data movement overhead comparison

Data movement overhead		Ambit	DRISA	Proposed
Input	Data size	4.8 MB	4.8 MB	52.7 KB
	Latency	196.6 µs	196.6 µs	4.2 µs
	Energy	368.4 µJ	368.4 µJ	73.5 µJ
Output	Data size	73.4 MB	73.4 MB	0.5 MB
	Latency	3.0 ms	3.0 ms	125.8 µs
	Energy	6.2 mJ	6.2 mJ	0.2 mJ

do not suffer from the accuracy degradation during accumulation.

Table 3.7 compares the data movement overhead of Ambit, DRISA, and Proposed architecture. The proposed architecture's input data movement size is about 93 times smaller, thanks to M2V and broadcasting write. Since Ambit and DRISA have to store the input vector to which im2col is applied, the data size is increased by the kernel size. Also, since inputs are stored in all banks by bank interleaving write, input data is repeatedly transmitted 16 times.The proposed architecture has 46.8 times faster latency and 5.0 times less energy consumption for input data movement. The difference in energy consumption is small because the broadcasting write also consumes internal write power to write to all banks.

In terms of the output data movement, the size of the intermediate results determines the output data movement overhead. It is because the PSUM operation reduces the intermediate results to 1/128. The proposed architecture has 23.8 times faster latency and 31.1 times less energy consumption.

Table 3.8 compares the two existing solutions (Ambit and DRISA) and two versions of our proposed method. Proposed (vanilla) represents the case of using the baseline binary network while proposed (accurate) represents the case of using larger networks (having 224 base kernel number as explained in Section 3.5.3) to improve accuracy. As described above, this is the result of executing the Conv and FC layers excluding the first and last layers with DRAM model. Functions other than convolution, such as batch normalization, activation function, max-pooling, etc., are assumed to be executed in the host processor, and the power and latency generated at this time are excluded.

The computation throughput of the proposed architecture is 4.3 TOPS, which is slower than DRISA (6.7 TOPS). The reason is that we compared the MAC operation latency of the proposed architecture with the XNOR latency of Ambit and DRISA. Ambit and DRISA only execute XNOR operations, but the proposed architecture includes the execution time of PSUM. In all items of Ambit and DRISA in the table, time and energy consumption for accumulation in the host processor are excluded (we de-

Table 3.8 Performance comparison

VGG-9 on CIFAR-10	Ambit	DRISA	Proposed (vanilla)	Proposed (accurate)
Area overhead	1 %	24 %	1.9 %	1.9 %
Accuracy	90.37	90.37	87.80	89.67
Computation throughput*	2.6 T	6.7 T	4.3 T	4.6 T
Computation latency*	236.9 µs	92.1 µs	282.5 µs	816.8 µs
Computation energy*	780.4 µJ	360.2 µJ	720.4 µJ	2.1 mJ
Computation power*	2.47 W	3.91 W	2.55 W	2.55 W
Data movement size	78.2 MB	78.2 MB	0.6 MB	1.0 MB
Data movement latency	3.2 ms	3.2 ms	0.1 ms	0.3 ms
Data movement energy	6.6 mJ	6.6 mJ	0.3 mJ	0.6 mJ
Latency per image*	3.4 ms	3.3 ms	0.4 ms	1.2 ms
Energy per image*	7.4 mJ	7.0 mJ	1.0 mJ	2.7 mJ
Throughput (OPS)*	179 G	187 G	3.0 T	3.2 T
Efficiency (OPS/W)*	83 G	88 G	1.2 T	1.4 T

note as * in the table).

Table 3.8 shows that our proposed method, proposed (accurate) offers 2.7 times (= 3.3 ms / 1.2 ms) lower latency per image and 2.6 times (=7.0 mJ / 2.7 mJ) smaller energy consumption at acceptable accuracy loss (0.7 %). Although the Proposed (accurate) case's computation energy and data movement energy are increased by 2.9 times and about 2 times, the total energy consumption is still 2.6 times smaller than that of DRISA. Such a gain comes mainly from the reduction in data movement between the in-DRAM accelerator and the host processor. When comparing only the computational latency, the Proposed (accurate) case is about 9 times slower than that of DRISA, but it should be noted that, as described above, the accumulation of DRISA is executed in the host processor, and the latency at this time is assumed to be 0. The throughput of Proposed (accurate) is 3.2 TOPS, and the efficiency is 1.4 TOPS/W. Compared to the existing solution, the effective throughput is more than 17.2 times higher, and the efficiency is more than 15.5 times better.

3.6 Discussion

3.6.1 Performance Comparison with ASIC Accelerators

Our proposed in-DRAM accelerator has 15 times better efficiency than the existing in-DRAM accelerator. As shown in Table 3.9, in-DRAM accelerators' throughput is higher even when compared to ASIC accelerators. In the table, throughput refers to peak performance, excluding data movement overhead. In particular, compared with TOPS/area term, the proposed architecture's performance is better than other ASIC accelerators except for [69]. Because the throughput is high, it seems that the proposed architecture is good in TOPS/W and TOPS/W/area terms. However, when comparing the computation power overhead by power/area, the proposed architecture's power is the largest. The efficiency with BNN accelerator should be more than 8 times better than 8b ASIC accelerator. So, power competitiveness is very low. For the proposed architecture, note that the network size is 2.1 times larger, and the computation energy consumption is 2.9 times consumed to compensate for the accuracy drop.

Tablee 3.10 compares the performance of the edge device and the proposed in-DRAM accelerator. The proposed architecture's efficiency is similar to that of edge TPU and is better than other edge devices. When considering the precision bit, the efficiency is significantly lower than

Table 3.9 Performance comparison with ASIC accelerator

	ASIC accelerators (presented on ISSCC'19)					Proposed
	[69]	[70]	[71]	[72]	[73]	(accurate)
Process (nm)	8	16	28	65	65	1x
Area (mm^2)	5.5	94.52	10.92	16	16	6.2*
Cores	1024	1024	512	1024	768	1M
VDD (V)	0.8	0.8	0.9	1.1	1.1	1.2
Precision (W,A)	(8,8)	(16,16)	(8,8)	(16,16)	(8,8)	(1,1)
Memory size (MB)	1.53	16	1.1	0.45	0.37	8192
Throughput (TOPS)	6.9	20.5	0.9	0.2	0.6	4.6
Power (W)	1.55	9.78	0.24	0.20	0.37	2.55
Efficiency (TOPS/W)	4.5	2.1	3.6	1.0	1.6	1.8
Power/area	0.28	0.10	0.02	0.01	0.02	0.41
TOPS/area	1.25	0.22	0.08	0.01	0.04	0.74
TOPS/W/area	0.81	0.02	0.33	0.06	0.1	0.29

Table 3.10 Performance comparison with Edge device

	Edge device				Proposed (accurate)
	Jetson nano	Jetson Tx1	Jetson Tx2	Edge TPU	
Cores	128	256	256	4096	1M
Precision (W,A)	(16,16)	(16,16)	(16,16)	(8,8)	(1,1)
Memory size (GB)	4	4	8	1	8
Throughput (TOPS)	0.1	1.0	1.3	4.0	4.6
Power (W)	10.0	10.0	7.5	2.0	2.6
Efficiency (TOPS/W)	0.1	0.1	0.2	2.0	1.8

that of edge TPU but better than other edge devices. Also, edge TPU has all functions integrated, and the proposed architecture only accelerates vector dot product operation. In this respect, the competitiveness of the proposed in-DRAM accelerators is very low.

Nevertheless, since the in-DRAM accelerator like the proposed architecture has the following advantages, further research is needed. Since it is added to the DRAM, there is no additional background power consumption. On the other hand, the standby power of Edge TPU is about 0.4W, which is as large as the operation power. The in-DRAM accelerator's cost overhead is small as the DRAM chip cost (about 40 \$) increases as much as the area overhead (1.9 %). The proposed architecture's cost overhead is about 0.8 \$, and the edge TPU module is 20.0 \$. Compared with TOPS/W/cost, the proposed structure is 2.25, which is 22 times better than edge TPU (0.1).

3.6.2 Challenges of The Proposed Architecture

Our proposed in-DRAM accelerator's efficiency is 16.5 times and 15.5 times better than that of Ambit and DRISA, respectively. In addition, as well as weight, input, and intermediate result data movement was also greatly improved. However, to put the in-DRAM accelerator into practical use, there are still challenges to be solved.

First, the proposed architecture consumes computation energy as maximum regardless of row utilization. It is because the activated row size and psum_width are physically fixed at the design stage. Designing to change the active row size affects the characteristics of the memory mode. Instead, it is better that add the PSUM path to increase row utilization by adjusting a psum_width.

Second, the computation overhead to improve the accuracy drop is large. While improving the accuracy with the larger Network, the computation energy consumption increased 2.9 times. Instead of increasing the network size, we need to improve the accuracy with the network topology and training method. It helps to improve performance even for all BNN accelerators. Also, we excluded the ADC by considering the area overhead, most importantly. To improve accuracy, consider adding a low-resolution ADC.

Third, the computation energy consumption itself using row activation is large. Despite the binary operation, the proposed architecture's MAC operation energy is large as 1.1 pJ/bit. The FP16 MAC operation energy of the ASIC accelerator (at 65 nm process) is 1.3 pJ [3], and simply converted into energy per bit, it is 0.08 pJ/bit. MAC operation energy of in-DRAM accelerator is 13.8 times larger. There is a need to reduce energy consumption in data backup and reduce computation energy consumption, such as using AND operation instead of XNOR.

Fourth, in addition to the vector dot product, functions such as max-pooling and batch normalization need to be integrated. When comparing the computation latency of several networks with the PyTorch profiler on the CPU without GPU acceleration, it was confirmed that the computation proportion of the Conv and FC layers gradually decreased. In Alexnet, VGG-16, and ResNet-34, the computation proportion of Conv and FC layers are 85.5 %, 87.5 %, and 74.9 %, respectively. However, the proportions of MobileNetv2, shufflenetv2, and MNASNet were small at 51.7 %, 53.9 %, and 50.4 %, respectively. The network topology has been proposed to reduce the computation overhead of the Conv and FC layers. Also, the computation proportion of max-pooling and batch normalization increased. Therefore, when the in-DRAM accelerator accelerates only the vector dot product, a performance bottleneck will occur in max-pooling and BN. Among DRAM-based accelerators, there is a structure that supports the end-to-end operation [27]. Although there is an area overhead, the advantages of this structure must be accommodated.

3.7 Conclusion

We proposed a novel computing-in-memory architecture based on DRAM. We focused on the two problems of in-DRAM accelerator, lack of NOT operation support and the data movement bottleneck due to the intermediate results to accumulate. Our proposed method, which is based on charge sharing circuit, judiciously exploits the existing DRAM architecture thereby incurring only a very small area overhead. Compared with the existing in-DRAM accelerators, on VGG-9 model for CIFAR-10, our proposed method offers 2.7 times lower latency per image and 2.6 times smaller energy consumption with acceptable accuracy loss (0.7 %). Effective throughput, including data movement latency, is 3.2 TOPS, and efficiency is 1.4 TOPS/W.

Chapter 4

Reducing DRAM Refresh Power Consumption by Runtime Profiling of Retention Time and Dual-row Activation

This work was published on MICPRO journal, Feb. 2020 [67]. The DRAM periodically performs refresh, data cannot be read or written during the refresh. It reduces the NN accelerator performance of data-intensive applications. Using the proposed refresh method, we can improve power consumption and performance degradation caused by DRAM refresh.

4.1 Introduction

The maximum density of a single DDR SDRAM chip, which is used for main memory, has increased 4X from 4Gb (DDR3) to 16Gb (DDR4) [4, 74]. Also, the number of DRAM chips on memory modules has also increased 2X from 64 EA (DDR3) to 128 EA (DDR4) [75]. Considering that the main memory often dominates total system power, e.g., up to 30% [59, 60], the increasing memory capacity makes the problem of high power consumption in main memory much more severe. In order to resolve the power issue, also called 'memory power wall' [60], it is imperative to devise novel solutions.

Memory power can be divided into operating and background power. Operating power is consumed by data read/write operation, so the power is not noticeably affected by memory density. However, the background power increases in proportion to memory capacity. It is because the background power is mostly incurred by refresh operation for holding data in the DRAM cells. Note that, among the background power, the leakage power has been reduced by power gating and fin-FET [61]. The standby power of the DRAM peripheral circuit has also been reduced by turning off some circuits during idle state [74, 76]. However, there is no appropriate method to reduce refresh power, although various methods are studied.

In this work, we focus on refresh power reduction. We analyze the existing refresh power improvement methods [2, 29–38, 77, 78] in three ways. 1) Temperature change: In the existing works [2,29,32,34–36], the temperature-dependent characteristics of DRAM cell has not been considered. 2) External support: Some methods could reduce refresh power only when operating system and memory controller support them, e.g., via data migration and turning off unused ranks [2, 30, 31, 33–36]. 3) Overhead of area/performance/low availability: Some methods have a substantial penalty in terms of area and/or speed [33, 36–38, 77, 78]. In particular, they often incur low system availability due to profiling during system boot up [2, 29, 34–36, 78].

In this chapter, we propose a novel method which resolves the problem of temperature changes and improves the refresh period in both active and idle states at a very small overhead of area and performance without requiring the external support. Our contribution is as follows.

- We propose a chip-level multi-rate refresh method which adjusts refresh period in both active and idle states according to the temperature dependent retention time characteristics of each DRAM chip.

- We propose a retention time profiling method which identifies the weak DRAM rows during DRAM idle time, hence avoiding low system availability.

- We present dual-row activation that improves retention time by simultaneously activating two rows (one weak and one low-cost additional row).

- We evaluate the proposed method experimentally by measuring real DRAM chip retention times. We improve the refresh period by 12.5 % compared with AVATAR and in-DRAM ECC, respectively. Under realistic system scenarios with SPEC 2006 benchmarks, we improve the energy-delay product by 19.7 %, 15.4 %, and 12.4 % compared with the baseline, AVATAR and in-DRAM ECC, respectively.

4.2 Background

DRAM cell consists of a capacitor to store 1-bit data and a switching transistor for access control (1T- 1C structure). One-bit data (0 or 1 represented by the amount of charge) is stored in the capacitor. To retain the data, DRAM periodically refreshes the cell before it is discharged below a critical level [63]. A cell with much shorter retention time than others is called a weak cell, and a row containing this cell is called a weak row [2]. Generally, each DRAM cell must be refreshed every 64 ms [4, 74]. During the refresh period (tREFW) of 64 ms, all rows must be refreshed by 8192 refresh commands each of which is issued every 7.8 µs, the refresh interval (tREFI).

Refresh is similar to performing pre-charge after activation without the read or write operation. Since DRAM cells have the same structure irrespective of type (DDR3, DDR4, LPDDR3, LPDDR4, and GDDR5), retention time depends only on process technology, operating voltage, and temperature. In our experiments, we measured retention time using 4Gb DDR3L DRAM chip samples.

DRAM refresh can be performed in auto-refresh (AREF) or self- refresh (SREF) mode. If a DRAM rank is active, AREF is used. Otherwise, SREF is used. The AREF refresh command is periodically generated by the memory controller, whereas the SREF refresh com- mand generated inside the DRAM after the controller issues the SREF mode entry. In SREF mode, only refresh operation is per- formed every 7.8 µs internally in DRAM. In both AREF and SREF modes, the row address to be refreshed is generated using the same counter on the DRAM chip [74]. To reduce DRAM background power, most peripheral circuits (particularly DRAM I/O), aside from the refresh logic block, are turned off during SREF mode. Thus, the DRAM refresh in SREF mode is not controlled by the memory controller.

4.3 Related Works

Hamamoto et al. presented a retention time model of DRAM cell where the temperature-dependent leakage behavior of cell transistor and capacitor is modeled [63]. Kim et al. showed that DRAM cells have a highly skewed distribution of retention time with a long tail due to a small number of weak cells [86]. Kim et al. reported variable retention time (VRT) where a cell can have multiple retention times [87]. Conventional solutions ensure DRAM retention time by applying a time margin during test and repair, e.g., 128 ms test for 64 ms refresh period [87,88]. Some weak DRAM rows are also replaced with repair rows during manufacturing

Table 4.1 Comparison of various related works to reduce refresh power

Category	Technique	Year	Modification			Retention time profiling			Overall	Support mode	
			OS	MC	DRAM	Profiling	Temp. variation	ECC	overhead	AREF	SREF
Considering the recent	Smart Refresh [30]	2007		✓	✓	-	-		Moderate		
DRAM row activations	TWW [32]	2017			✓	-	-		Small	✓	
refresh applied to the	PASR [79]	-	✓	✓	✓	-	-		Large		✓
memory cell used by	DIMMer [33]	2014	✓	✓	✓	-	-		Large		✓
application programs	SRA [29]	1998		✓	✓	-	-		Moderate	✓	✓
	ESKIMO [31]	2009	✓	✓	✓	-	-		Large	✓	✓
	EXTREME [31]	2018	✓	✓	✓	-	-		Large	✓	✓
Retention time-aware	VRA [29]	1998		✓	✓	-	-		Moderate	✓	✓
refresh	RAPID [34]	2006	✓	✓		-	-		Moderate		
	RAIDR [2]	2012		✓		Boot-up	Scaling		Small		
	AVATAR [35]	2015		✓		Online	Worst	✓	Large		
	TnE [80]	2014		✓		Online	-	✓	Large		
	Elaborate Refresh [81]	2018			✓	Boot-up	-		Moderate	✓	✓
	MCRDRAM [78]	2015		✓	✓	-	-		Very large	✓	✓
Increasing refresh period	Flikker [36]	2011	✓	✓	✓	-	-		Large	✓	✓
with error correction/	CAPB [82]	2019		✓	✓	Online	-	✓	Large	✓	
tolerance Self-refresh	In-DRAM ECC [38]	2017			✓	-	-		Large	✓	✓
power reduction	Temp-aware SREF [83]	1993			✓	-	-		Small		✓
	ESR [84]	2016			✓	-	-		Small		✓
	Long idle mode [85]	2019			✓	-	-		Small		✓
	Proposed [67]				✓	Online	Online		Small	✓	✓

testing to improve the margin [89]. ECC DIMM and post pack- age repair (PPR) have also been used [90]. However, at advanced technologies, these conventional approaches are expected to suffer from high cost due to the worse retention time [91,92].

Reiss et al. showed that more than half of the total memory re- sources in Google server are idle, i.e., in SREF mode [93]. Liu et al. reported, about 95% of memory resources are in idle state [34, 36]. Shin et al. showed that the memory scheduling and interleaving can place more DRAM chips in SREF mode [94]. Considering that most of DRAM chips can be in SREF mode, a practical solution of refresh power reduction must be applied to the SREF mode as well as the AREF mode.

We summarized the comparison between existing works and ours in Table 4.1. First, we group the existing works to five categories, three for refresh power reduction in active mode, one for error correction/tolerance and one for self-refresh power reduction. The first category covers methods which skip refreshing the DRAM rows which were recently activated. Smart Refresh and TWW delay the refresh operation of recently activated rows using a time-out counter in the DRAM module [30] or without such a counter [32]. Considering the trend of ever-increasing DRAM capacity, these methods have limited utility. They also lack in supporting SREF mode.

The second category methods refresh the memory cells used by application programs (at the granularity of row, bank or DIMM). In mobile

DRAM, PASR turns offrefresh operation in SREF mode at a granularity of DRAM bank [79]. DIMMer turns offthe power at the granularity of module [33]. In both methods, before turning offthe refresh operations of the target address space, valid data must be migrated to the memory space under refresh, which in- curs data migration overhead. In [29], SRA skips refresh operations for an empty row which does not store valid data. In [31], ESKIMO adds a flag reset function to SRA. In [95], EXTREME skips the re- fresh operation of unused rows. They add a page map table to the logic die of 3D-stacked DRAM, e.g., HBM, to refresh only the used pages. When applied to real systems, those works require system modification covering operating system and memory controller.

The third category is the most related with our proposed one since refresh operations can be applied at different rates based on the retention time information of each row, which is called retention time-aware refresh. In [29], VRA skips the row refresh command according to the retention time flag associated to the DRAM row. It is a stand-alone method in DRAM and supports both AREF and SREF refresh modes. In [34], RAPID increases the refresh period of the majority of rows (not weak ones). However, in [29, 34], there has been presented no profiling solution to measure the retention time of DRAM row. In [2], RAIDR reduces refresh operations by keeping the refresh period information of each row in the memory controller. RAIDR profiles retention time during system boot up possibly incurring the problem of low availability especially in case of large main memory. Moreover, it assumes the worst-case temperature and cannot be applied to SREF mode due to the necessity of memory controller support. In [35], AVATAR performs periodic profiling that exploits ECC-DIMM to identify weak cells. However, it has the same problems as RAIDR concerning temperature and SREF mode. In addition, it runs only on ECC-DIMM. In [80], TnE performs retention time-aware refresh on the memory area allocated by the application program. TnE has an advantage of demand scrubbing which profiles only the memory cells utilized by the application, not all the memory cells. TnE requires ECC to recover from bit errors during profiling while ours utilizes additional rows to backup and restore the contents of the row under profiling. In [81], Elaborate Refresh performs more fine-grained refreshes than AVATAR by storing the weak address in each DRAM chip. Elaborate Refresh adds weak group refresh (WGR) mode and refreshes weak group every 64 ms in WGR mode. They reduce the refresh operation of both AREF and SREF, because modified DRAM translates AREF command to WGR operation inside DRAM. RAPID, RADIR, AVATAR, TnE, and Elaborate Refresh assume the worst-case temperature in retention time thereby losing opportunities of further power reduction which is possible in a temperature-aware solution like ours. In [78], MCRDRAM activates multiple rows simultaneously to improve retention time and refresh period at a significant cost of DRAM capacity, e.g., 50% of the entire chip capacity, even when the retention time of only a single weak row needs to be improved.

In the fourth category, we can achieve longer refresh period by tolerating or correcting errors. In [36], Flikker reduces refresh operations while allowing errors on non-critical data with the support of applications, operating system and memory controller. Flikker can reduce refresh operations of SREF mode by equipping DRAM with the usage information. However, it cannot consider temperature change, i.e., assumes the worst-case temperature. In [82], CAPB skips the refresh operations of non-weak row during AREF mode by providing the refresh address counter in DRAM with the target row address to refresh. It also attempts to further reduce refresh operations by exploiting the fact that '0' data incur less bit errors and reducing refresh operations with the rows having abundant zero data. However, it does not consider temperature change, i.e., assumes the worst-case temperature. In-DRAM ECC achieves longer refresh period by correcting errors with ECC within the DRAM chip [38]. In-DRAM ECC reduces refresh operations by more than 75% [37,38,77]. However, it suffers from a significant area penalty ($>$6%) for the parity bit and long write latency (5X) due to read-before-write for parity calculation [77].

The fifth category includes works to reduce the power con- sumption of SREF mode only. Kagenishi et al. proposed controlling the refresh period of SREF mode according to temperature [83]. Oh et al. proposed ESR which, in SREF mode, reduces the word- line voltage (applied to the access transistor of DRAM cell) of non- activated rows in order to reduce leakage current thereby increasing the refresh period [84]. Pardeik et al. also proposed a method which controls the cell voltage to lengthen the refresh period in SREF mode according to SREF mode duration [85]. Like ours, these methods can be easily realized by modifying only the DRAM. How- ever, the voltage control can incur additional latency overhead due to voltage transition and associated DLL settling operation.

Note that our method can be applied together with the above- mentioned methods for further reduction in refresh power, which is left for future work.

4.4 Observations

Temperature varies during system runtime. Above all, the temperature is not usually controllable, e.g., cannot be set to the worst-case temperature for profiling, in real systems. However, the existing methods of multi-rate refresh and profiling lack in temperature consideration because they assume a fixed temperature in profiling and operation. In this section, we report our observations of temperature-related characteristics of retention time based on real DRAM chip measurements.

In order to measure retention time while varying temperature, we implemented a memory tester on an FPGA board and used a temperature controlled chamber as explained in Section 8.1 . We varied the temper-

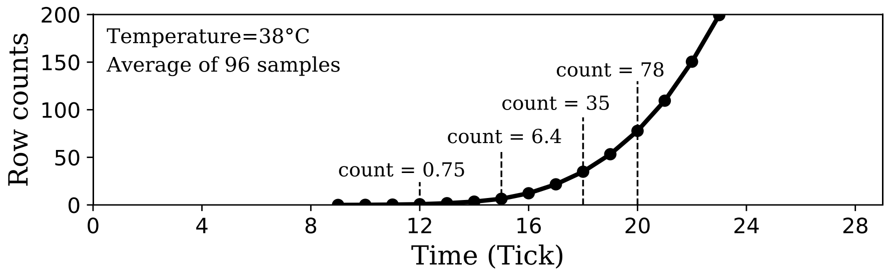
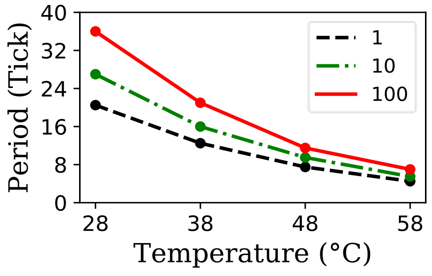


Figure 4.1 Cumulative distribution of retention time at 38 °C.

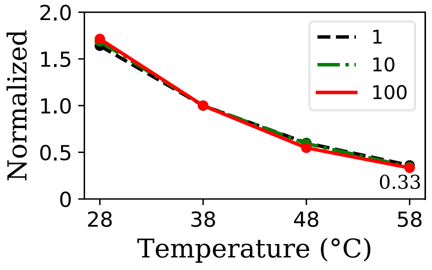
ature in a range of conventional desktop or server environment [96, 97], e.g., boot-up temperature = 28 °C and operating temperature = 38–58 °C. We measured 96 samples of recent 4Gb DDR3L DRAM chip at 28, 38, 48, and 58 °C.

Observation 1: Large gain in refresh period can be obtained by improving only a few weak rows

Figure 4.1 shows the cumulative distribution of per-row retention time averaged over 96 DRAM chips at 38 °C. The figure shows, refresh period can be significantly increased by improving only a small number of weak rows. For instance, average 78 rows have retention time smaller than or equal to 1280 ms (= 20 × 64 ms) while the other rows have much larger retention time. We represent the time axis at the granularity of 64 ms called a tick. Notably, the relative gain, i.e., increase in refresh period/number of improved rows, is large when improving a few weak rows. The relative gain gets smaller when improving more weak rows, e.g., relative gain gets reduced from 12 (for 1 row) to 0.26 (for 78 rows).



(a)

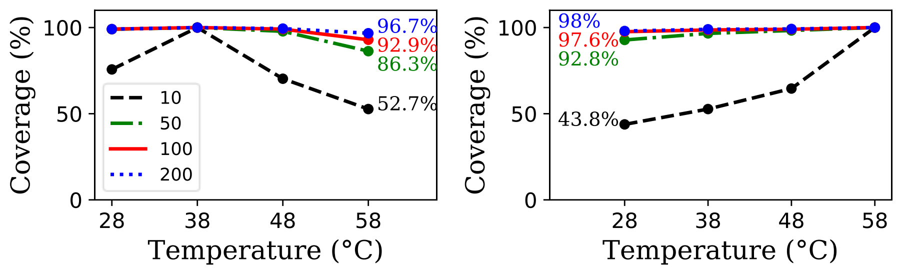


(b)

Figure 4.2 Refresh period (when improving weak rows) vs temperature as (a) measured and (b) normalized to 38 °C.

Observation 2: Relative improvement of refresh period inversely depends on temperature.

Figure 4.2 (a) shows that the temperature significantly affects the improvements in retention time. For example, improving 100 weak rows (red in the figure) increases refresh period up to 36 ticks (= 36 × 64 ms) at 28 °C but only 8 ticks at 58 °C. Figure 4.2 (b) shows the refresh period normalized to 38 °C (the data in Figure 4.2 (a) are normalized). The figure shows that the relative improvement of refresh period is a strong inverse function of temperature. Also, the function in Figure 4.2 (b) is practically the same across different numbers of improved rows, 1, 10 and 100 rows. It means that the trend is statistically consistent, regardless of the number of improved rows. In our proposed refresh method, we exploit this observation when extrapolating the refresh period from the measured temperature to a higher temperature level (Section 6).



(a) (b)

Figure 4.3 Refresh period (when improving weak rows) vs temperature as (a) measured and (b) normalized to 38 °C.

Observation 3: The weakest rows can change across temperature range.

Figure 4.3 shows how the weakest rows change across temperature. For instance, the graph of N = 10 shows how much the group of 10 weakest rows found at 38 °C covers 10 weakest rows found at other temperature levels. The figure shows, in the case of 58 °C, the coverage is 52.7%, which means the two temperature levels share average 5.27 weakest rows among 10. The key message of Fig 4.3 (a) is that tracking a small number, e.g., only one, of specific weakest rows can fail in capturing the minimum retention time of DRAM chip since different temperature levels can give different weakest rows. Figure 4.3 (b) shows that the coverage of weakest rows obtained at high temperature, 58 °C in this case, is better across temperature levels than that obtained at low temperature, 38 °C (Figure 4.3 (a)). According to this observation, it is clear that a group of weak cells, i.e., rows need to be monitored across different temperature levels for high confidence.

Typically, a weak row has a much smaller retention time than the other rows. We define a weak row, in terms of utility and reliability, as the one which, if dual-row activation is applied to it, offers improvement in refresh period as well as helps meet the constraint of manufacture test for reliability. In terms of utility in improving refresh period, when we apply dual-row activation to a row having small retention time, if we can increase the refresh period of the DRAM chip by at least 64 ms (in 38 °C), then the dual-row activation of the row is useful. In terms of reliability, the manufacture test typically requires, after repair, meeting a retention time of 128 ms for a DRAM chip at the worst-case temperature. Thus, after dual-row activation is applied to such rows having small retention time, we need to be able to satisfy the constraint of DRAM manufacture test like 128 ms. When both conditions are met, we call such rows weak rows and apply dual-row activation to them.

4.5 Solution overview

Fig 4.4 (a) shows the overall flow of refresh operation in our proposed scheme. Whenever a refresh command is issued (in SREF or AREF mode), the current temperature is checked by a thermometer in DRAM [4, 74]. If it is above the maximum temperature of the target range (38–58 °C

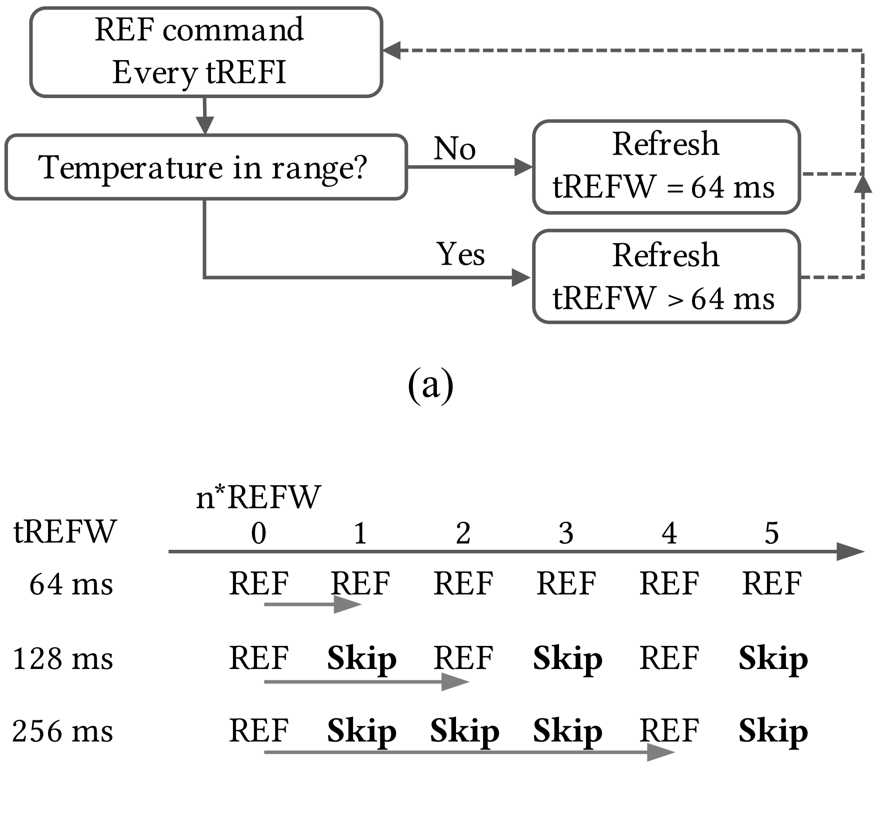


Figure 4.4 Our proposed refresh scheme: (a) mechanism and (b) examples of skipping refresh operations.

in our experiments) that our profiling method covers, then the proposed scheme performs refresh operations with 64 ms assuming the worst-case temperature. Otherwise, the proposed scheme refreshes with the refresh period (called target refresh period) determined by our proposed method which applies profiling (Section 6) and dual-row activation (Section 7

). Typically, the target refresh period is longer than 64 ms, e.g., average 4.5X longer at 58 °C in our experiments. Figure 4.4 (b) illustrates how refresh operations are skipped when the target refresh period is longer than 64 ms. For example, in order to perform refresh every 128 ms, the DRAM performs refresh once for every two refreshes. Note that since the proposed scheme skips refresh operation inside DRAM, it is applied to both AREF and SREF modes.

Note that each DRAM chip has its own target refresh period determined by the characteristics of each chip and obtained by our profiling method. Each DRAM device can have different characteristics (e.g., fast or slow corner due to inter-chip variations) and operating conditions (e.g., hot or low temperature). Our pro- posed method adjusts refresh period in a per-chip manner, which enables us to consider both chip characteristics and operating conditions. For instance, in the case of a DRAM chip at fast corner and hot temperature (with high leakage), the proposed profiling method may select more rows to apply dual-row activation while it will select less rows in the case of DRAM chip at slow corner and low temperature (with low leakage). This is the first work for the chip-level multi-rate refresh, and it allows us to further reduce the overhead of multi-rate refresh operations in the DRAM DIMM. The overhead of our refresh scheme is very small because it uses a thermometer inside the DRAM for temperature reading [4, 74] and a simple counter to keep track of skipping refresh operations, e.g., only a 2-bit global counter for a DRAM chip is needed in the target refresh period of maximum 256 ms.

The target refresh period is obtained by the proposed profiling method in two steps. First, the profiling scheme (Section 6) searches a minimum retention time of all the rows inside a chip at the current temperature. During the profiling, when the profiling method identifies weak rows, then, dual-row activation (Section 7) is applied to them to increase their retention time. Note that whenever the weak row is activated (for normal activation or re- fresh), dual-row activation is applied. Second, in order to account for temperature margin, it extrapolates the minimum retention time of the current temperature to the that of maximum temperature in the target range, 58 °C in our experiments. For in- stance, if the current temperature is 38 °C, then the minimum retention time is scaled by 1/3 to obtain the target refresh period (Observation 2).

Table 4.2 compares our proposed scheme with state-of-the-art methods, RAIDR, AVATAR, and in-DRAM ECC. Based on where the refresh control is executed, we classify them to memory-centric and controller-centric methods. Our proposed one is classified to the memory-centric method which is applied to both AREF and SREF modes. Note that the controller-centric method cannot be applied to SREF mode since it requires the memory controller to perform row-selective refresh operations. As the table shows, our proposed method determines the refresh period at a granularity of DRAM chip, rather than DRAM row in rank-level as RAIDR and AVATAR.

Table 4.2 Comparison with state-of-the-art methods

	Controller-centric method		Memory-centric method	
	RAIDR	AVATAR	In-DRAM ECC	Proposed
Refresh method	Row selective		Native (AREF,SREF)	
Refresh rate	Multi (row)	Multi (row)	Single	Multi (chip)
SREF support	No	No	Yes	Yes
Profiling	Needed	Needed	No	Needed
Online profiling	No	w/ ECC DIMM	-	Yes
Parallel profiling	No	No	-	Yes
Parallel overhead	Large	Large	No	No
Temp. variation	No	No	Yes	Yes(partial)
ECC DIMM	Optional	Essential	Optional	Optional
VRT correction	ECC	ECC	Default	ECC

Table 4.2 shows that our proposed scheme enables on-line profiling during system run while RAIDR allows for only boot-up time profiling and AVATAR requires ECC support for online profiling. Note that our proposed method allows parallel profiling by running multiple profiling operations in parallel, thereby the profiling time does not increase with memory capacity, which is not avail- able in the controller-centric method. As mentioned earlier, our scheme adjusts refresh period of the

target temperature range, not the entire temperature range, while RAIDR and AVATAR assume the worst-case temperature, thereby losing opportunities for further reduction in refresh period. All the methods require ECC (in-DRAM or DIMM-level one) for error correction of variable retention time (VRT) errors. In terms of reliability, even though it targets a very low bit error rate as explained in Section 7 , depending on the required reliability requirement, it needs to be utilized together with other solutions such as ECC DIMM, in order to address other error sources, e.g., soft error, which the proposed method does not cover.

Note that our proposed scheme does not require assistance from the memory controller while RAIDR and AVATAR require the memory controller control both profiling and multi-rate refresh operations.

4.6 Runtime profiling

4.6.1 Basic Operation

Figure 4.5 (a) shows the flow of retention time profiling. We perform parallel profiling while a DRAM chip is in idle state (SREF mode). For simplicity, the figure shows the case that we perform profiling one row (called target row) at a time. Note that one column in Figure 4.5 (b) represents a period of 64 ms (tREFW). As the figure shows, if the row counter in DRAM becomes equal to the target row address of retention

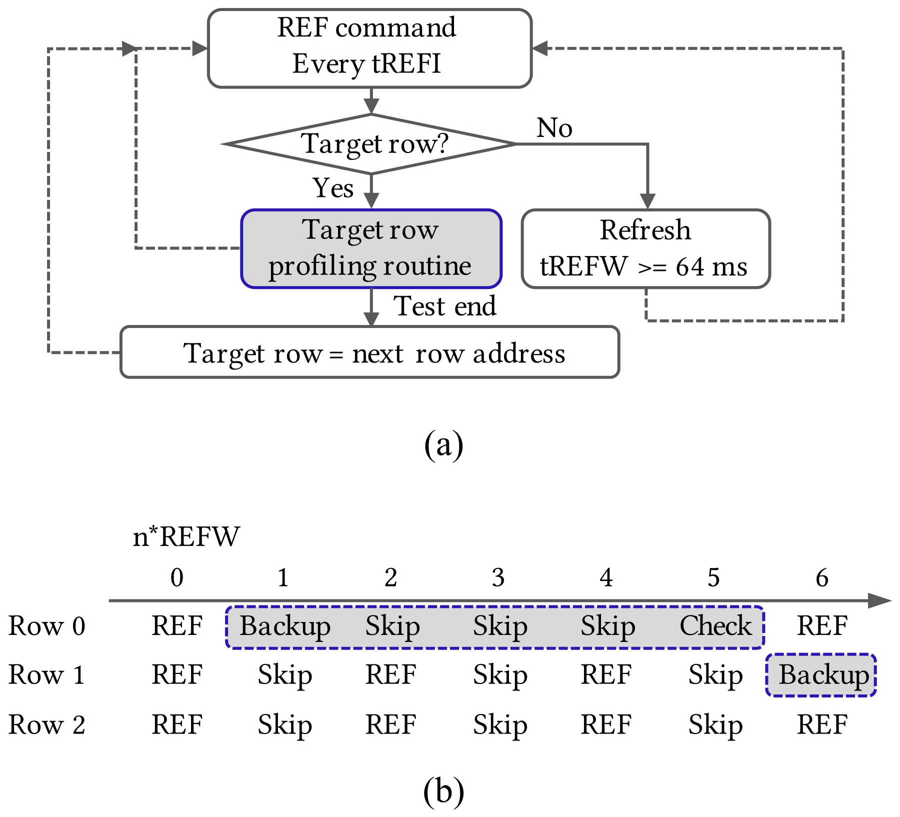


Figure 4.5 Profiling inside the DRAM: (a) profiling flow, (b) profiling routine examples.

time profiling, then its retention time is tested by a profiling routine which will be explained later. After finishing the profiling of the target row, the address of the target row is incremented for the profiling of the next row.

Figure 4.5 (b) illustrates the profiling routine. It consists of three states, data backup, refresh skip(s) and error check. Profiling a tar- get row is to skip its refresh for the target retention time, e.g., 256 ms. Note that, as illustrated in the figure, the other rows not under profiling are refreshed in the target refresh period, e.g., at a refresh period of 128 ms. Since retention time profiling is per- formed in idle state, it can be interrupted by SREF exit. In such a case, the profiling of the remaining rows resumes in the next SREF mode. The contents of the target row are restored during SREF exit operation, e.g., DLL initialization. Thus, the latency of data restoration can be hidden.

After completing the profiling for the current target retention time on all the rows of the DRAM chip, if there is no error, i.e., retention error in error check, then we increase by 64 ms the target retention time and perform profiling for all the rows. This procedure continues until there is any retention error found in the error check. The current target retention time without error that represents the minimum retention time of the DRAM chip. Then, as described in Section 5 , we obtain the target refresh period by extrapolating the minimum retention time of the current temperature to that of the maximum temperature, 58 °C.

4.6.2 Profiling Multiple Rows in Parallel

It is important to reduce the total profiling time, especially, in order to promptly adapt to temperature change. As mentioned previously, our method performs parallel profiling, where all sub- arrays of each bank perform profiling in parallel while sharing the error check logic per bank. Note that, in the parallel profiling, the target row is determined by the row index of the sub-array, not the entire row address.

4Gb DDR3 DRAM chip has 8 banks and each bank consists of 64 sub-arrays. Thus, we can profile 512 (= 64 × 8) rows in a DRAM chip in parallel. The profiling of the entire chip requires 1024 profiling routines (one per row) since the sub-array consists of 1024 rows. Note that the runtime of the pro- posed parallel profiling, i.e., the number of profiling routines depends on the number of rows in a sub-array, not the memory capacity, e.g., the number of DRAM chips in the module.

4.6.3 Temperature, Data Backup and Error Check

When realizing the basic idea of retention time profiling in idle state, there are two issues to be resolved, temperature and time for data backup and test. First, while profiling, the temperature may change. When the temperature increases, the retention time gets reduced, and data can be lost. Thus, if an error is detected, then we reduce by 64 ms the target retention time and continue to perform profiling with the new, shorter

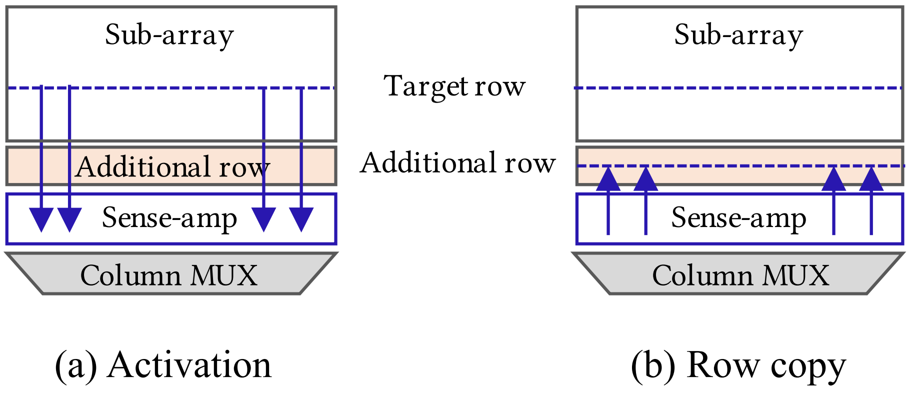


Figure 4.6 Backup on a sub-array: (a) row data copied to the sense amplifier array, and (b) backed up to an additional row.

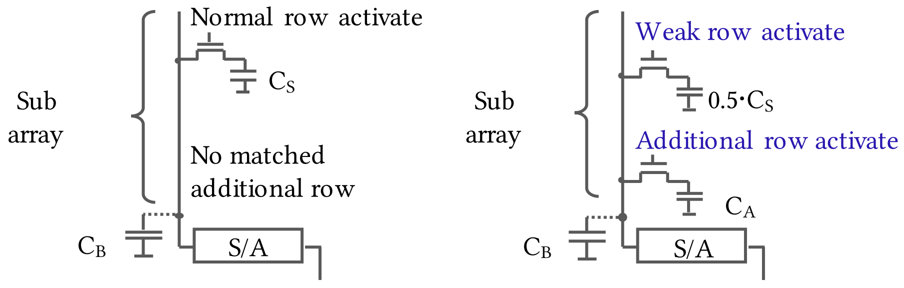
target retention time. Second, the data backup and error check require data movement, which incurs the overhead of runtime and energy consumption. For instance, it takes 0.7 µs to read an entire row from one bank to another in DRAM. Its latency is determined by tRCD + tCL + 128 tCCD, where tRCD, tCL and tCCD are RAS-to-CAS delay, CAS latency, and CAS-to-CAS delay, respectively. Hence, in case of DDR3 1600 (clock period = 1.25 ns, CL = 11, RCD = 11, and CCD = 4), the latency is 0.7 µs. A data backup operation needs one read and two writes (write original data to backup storage and write test pattern to a target row) of the entire target row while the error check requires one read. In order to reduce the backup overhead, we propose utilizing an additional row in each sub-array as the backup storage using the RowClone technique [98].

Figure 4.6 illustrates the backup operation. When the target row is activated, its contents loaded into sense-amplifier (S/A) array. Then, the additional row is activated, and the data on the bit-line are copied to the additional row. The latency of the row-level copy operation is less than 40 ns [98]. Adding an additional row for each sub-array incurs a very small area overhead (approximately 0.1%, assuming 1024 rows and an additional row per sub-array). Note that we equip each sub-array with multiple additional rows and utilize one of them for data backup during profiling. We exploit the additional rows mainly to improve the retention time of weak row as will be explained in Section 7 .

For the error check, the contents of the target row are read to the error check logic. The area overhead of the error check logic is small (0.05%, where the area of one XOR gate is 24 m^2) because only 64 XOR + OR gates are required for each bank to compare the contents of the target row and the test pattern at a granularity of 64 bits. The runtime of error check operation, mostly determined by the read latency of the entire row, is 0.74 µs which can be hidden by tREFI (7.8 µs).

4.7 Dual-row Activation

Figure 4.7 illustrates dual-row activation which utilizes an additional row to keep the same contents of a weak row. When accessing the weak row, we activate both the weak and additional rows. Figure 4.7 (a) shows the



(a) Normal row access (b) Weak row access

Figure 4.7 Dual-row activation.

case of normal row activation. When a cell is activated, charge sharing between the bit-line and the activated cell makes voltage difference (detected by S/A) which is proportional to C_S / (C_B + C_S), where C_S and C_B are the cell and bit-line capacitance, respectively. In general, C_B is much (4X–10X) larger than C_S .

Figure 4.7 (b) illustrates a case where C_S is reduced by half in a weak cell. In such a case, the voltage difference detected by S/A is halved, which increases the risk of a read error. In this case, we activate both the weak row and the additional row containing the same contents of the weak row. Thus, the difference becomes proportional to $(0.5 \cdot C_S + C_A)/(C_B + 0.5 \cdot C_S + C_A)$, where C_A is the capacitance of additional cell. Typically, $C_A = C_S$. Thus, the additional row increases the voltage difference thereby avoiding the error.

Dual-row activation is not new [78] . In our work, we propose a low-cost realization of dual-row activation in terms of additional row and weak row information. Our proposal to reduce the cost is to equip each sub-array with additional rows to improve the re- liability of weak rows in the sub-array. The number of additional rows is determined by the reliability requirement. We aim at improving up to 100 weak rows in a DRAM chip, which is much more challenging than the existing PPR, e.g., 1 or 2 row repairs. We expect such a high-reliability level is necessary for more advanced technology.

We target the reliability requirement, 10E-12 in terms of the bit error rate (BER) [99]. Considering this requirement, we need to add 8 additional rows per sub-array to cover 100 weak rows in a DRAM chip. The total number of additional rows per DRAM chip is 100 which was obtained to meet the constraints of reliability and utility in our definition of weak row. Our area analysis shows that additional 8 rows per sub-array incur a very small area ($<$0.8%) and power ($<$0.02%) overhead.

In our implementation, the dual-row activation method works with the DRAM repair block in parallel. Figure 4.8 shows the flow of row activation and Figure 4.9 its realization. In Figure 4.9 , the shaded blocks and bold arrows correspond to the new circuit of the proposed idea. Specifically, the gray blocks are used for profiling and blue ones for dual-row activation.

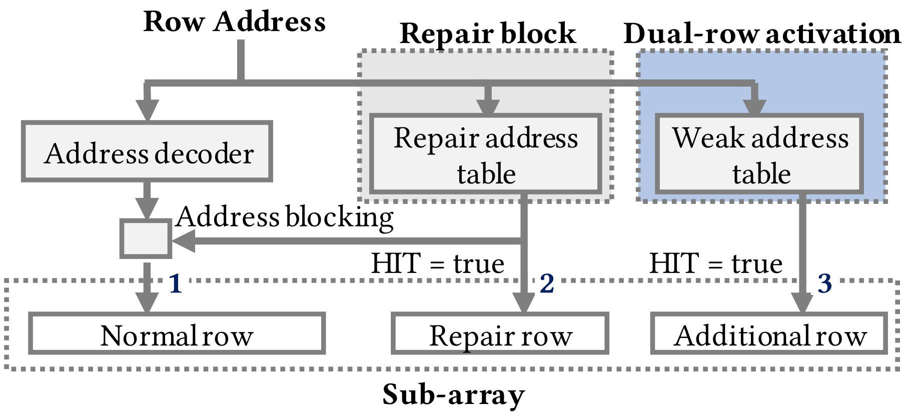


Figure 4.8 Flow of row activation in the proposed method.

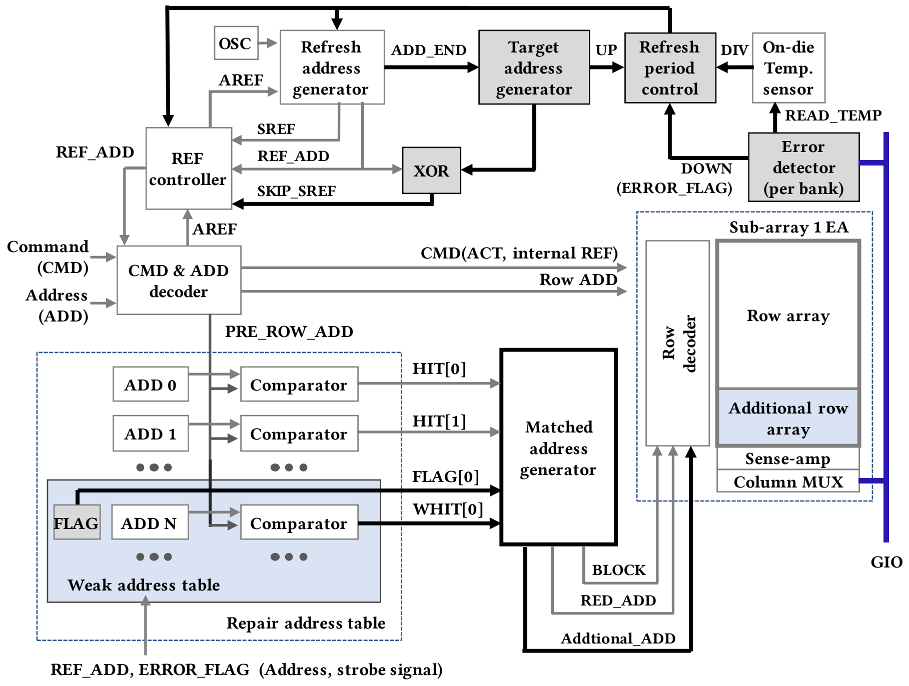


Figure 4.9 Block diagram of proposed method.

As the figures show, when an input row address (Row Address in Figure 4.8 and PRE_ROW_ADD in Figure 4.9) is given for row activation, we consult two tables, repair address table and weak address table, which contains the address of repaired rows and of weak rows, respectively. If the input row address is found in the repair address table, then we activate the repair row address (arrow 2 in Figure 4.8 and HIT[] signal in Figure 4.9) and block an output of the ad- dress decoder (BLOCK signal in Fig 4.9) to prevent the activation of the original row (which is repaired by the repair row). If not, we perform an activation for the input row address (arrow 1 in Figure 4.8). In addition, if the input row address is found in the weak address table, then we simultaneously activate the additional row (arrow 3 in Figure 4.8 and WHIT[] and Additional_ADD signals in Figure 4.9). We can implement dual-row activation method to DRAM within 0.9% area (includes additional array penalty, 0.8%) since we only add blocks for the weak address table to the repair block and do not change the existing blocks.

4.8 Experiments

We evaluate our proposed method in three ways: real measurements, a statistical model and an architecture simulator. In real measurements, we evaluate the improved retention time by dual-row activation. Based on the measured results, we show that different refresh periods can be set across DRAM chips in a temperature-changing condition. Using the architecture model, we compare the existing methods and ours in various memory configurations.

4.8.1 Experimental Setup

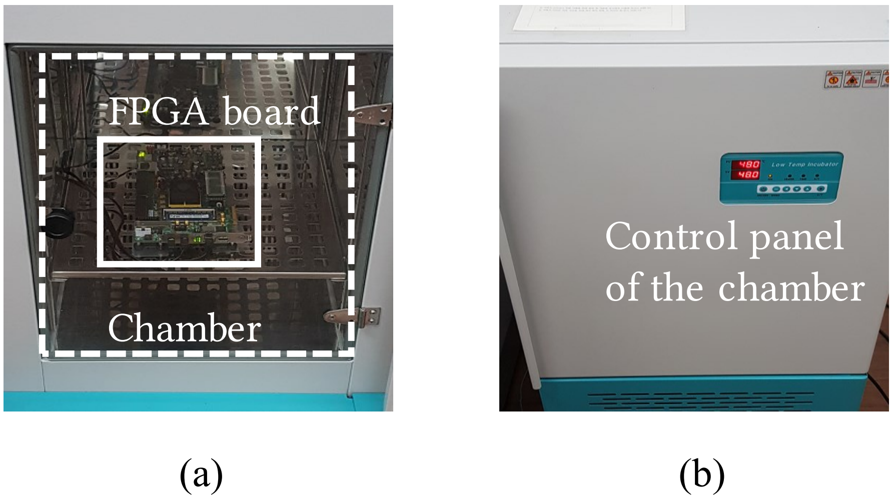


Figure 4.10 Measurement environment. (a) FPGA board in the chamber, and (b) temperature control panel of the chamber.

We used an in-house memory tester on an FPGA board to measure the retention time of DRAM chips. The board is equipped with a Xilinx VC707 and 4Gb DDR3L DRAM. We tested 96 DRAM chips (12 ranks) of recent technology from a single company. The DRAM chips are equipped with a dual-row activation function for testing purposes only. The parameters of the DRAM chips used in our experiments are very close to those used in [76]. Retention time was measured at 28 °C, 38 °C (default), 48 °C and 58 °C using a temperature-controlled chamber (accuracy of $\pm$ 1 °C) as shown in Figure 4.10. We set the temperature condition according to the existing works [96,97] . Lee et al. [96] reports that the temperature of server and desktop ranges between 34C and 50C during 2-hour operation. El-Sayed [97] reports that memory throughput is significantly reduced at higher than 58 °C. Considering these, we set the range of operating temperature to 38 °C–58 °C. To be exact, the boot-up temperature is the room temperature, 28 °C and the typical/worst temperature is 38 °C/58 °C. Figure 4.1, 4.2,and 4.3 show the measurement results we obtained from this setting.

We developed a statistical model based on a hidden Markov model (HMM) [100] of retention time based on our measurements. We utilize the HMM to evaluate the target refresh period of each DRAM chip under the proposed method (profiling with dual-row activation) in a temperature-changing condition. We utilize 8 additional rows per sub-array for dual-row activation and improve (up to) 100 weak rows utilizing dual-row activation and weak address table.

In order to evaluate our proposed method during system run, we developed a simulation model of AVATAR, in-DRAM ECC and proposed method, on Ramulator [101]. We compare these methods on Ramulator and DRAMPower [102]. Table 4.3 shows the key parameters used in our simulation experiments. As will be shown in the following sub-section (in Figure 4.13), our proposed method improves by 4.5X refresh period at the typical range of operating temperature (below 58 °C) which is by 12.5% better than AVATAR (4X) and in-DRAM ECC (4X). The period of AVATAR, 4X is extrapolated from 3.6X at 60 °C [35]. The refresh period of in-DRAM ECC is fixed to 4X during the manufacturing process [38]. In terms of memory technology, AVATAR used older technology than the one used in our experiments. Considering the fact that the retention time gets shorter in the newer technology, our gain of 4.5X improvement can be considered much better than 4X in AVATAR. Therefore, we assume the refresh period of all methods to be 4 times the baseline, i.e., 256 ms (= 4 × 64 ms). The other common parameters are based on 4Gb DDR3L datasheet [76].

We used Ramulator with CPU trace mode. In the CPU trace mode, Ramulator directly reads instruction traces and simulates a simplified model of a ”core” that generates memory requests to the DRAM subsystem. Table 4.4 shows the architectural parameters of CPU and memory subsystem. To be specific, the system is equipped with two channels and each channel consists of two 8GB (4Gb x8 2 ranks) DDR3L 1600 Mbps DIMMs, i.e., total 8 ranks. We used the CPU traces of SPEC 2006 provided in Ramulator [101, 103].

Table 4.3 Key simulation parameters

	DDR3	AVATAR	In-DRAM ECC	Proposed
Refresh command	AREF	(ACT-PRE)	AREF	AREF
Refresh period (tick)	1	4	4	4
Power overhead (%)	No	No	3	0.02
Latency overhead	No	No	WR_CCD (20ns)	No
			tRFC (+100ns)	
			CL,CWL (+2ns)	

Table 4.4 System configuration for simulation

CPU	3.2 GHz (in-order), 1-core
Cache (L1, L2, L3)	Density (32kB, 256kB, 8 MB)
	Latency (4, 12, 31), all 8-way set-associative
Memory controller	32/32-entry read/write request queues, FR-FCFS
System memory	32GB (2DPC, 2CH)
DRAM	Four 8GB (4Gb x8 2rank)
(nCL-nRCD-nRP)	DDR3L 1600k (11–11–11)

4.8.2 Refresh Period Improvement

In this section, we utilize the measurement data of real DRAM chips and our statistical model based on the measurement data of real DRAM chips. We did neither utilize benchmark programs nor any architecture simulation model since we assume that the characteristics of refresh period is not affected by program characteristics.

Figure 4.11 shows the measured retention time (cumulative distribution) of two cases at 38 °C, before ('baseline') and after the dual- row activation is adopted ('dual-row activation'). Note that dual-row activation without profiling would not offer meaningful improvements because it requires profiling to select which rows to perform dual-row activation. Our proposed method gives the better distribution (the lower, the better) than the baseline. The improvement comes from dual-row activation. It gives 11% (9 to 10 ticks) improvement of minimum retention time, so we can expect the target refresh period of each chip can be further improved in a similar manner. However, the retention time improvement is smaller than expected (up to 20 ticks as shown in Figure 4.2 (a)). We think it is mainly because the two activation are not well synchronized in the current DRAM chips (supporting dual-row activation), the improvement of which is left for future work.

Using the statistical HMM based on the real measured result, we evaluate the target refresh period in a temperature changing condition. Fig-

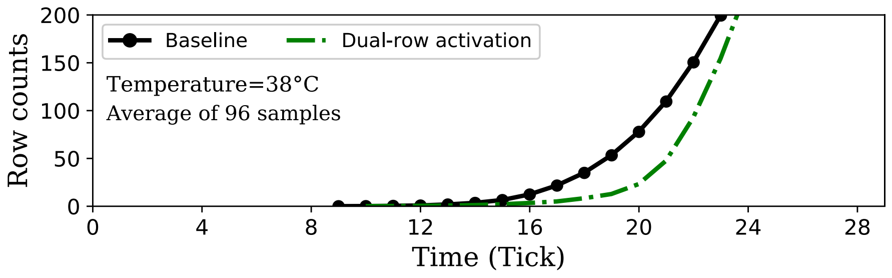
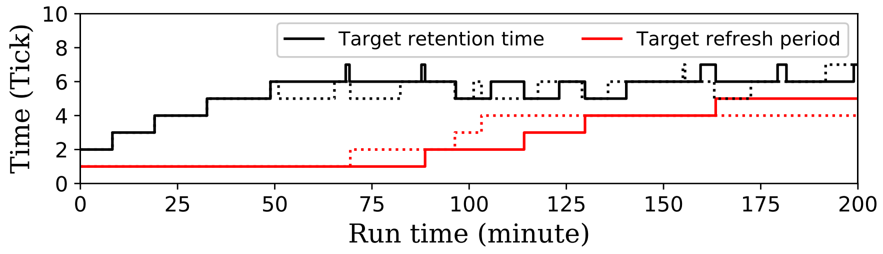


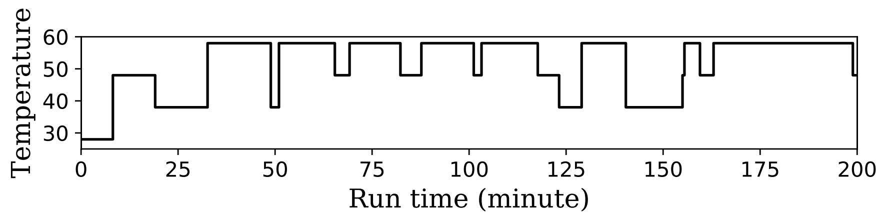
Figure 4.11 Retention time distribution where dual-row activation was applied to 100 weak rows (measurements).

ure 4.12 illustrates how the proposed method of profiling and dual-row activation (Section 4.6 and 4.7) improves refresh period on different DRAM chips. The x-axis represents the profiling time accumulated in SREF mode. Figure 4.12 (a) shows the profiling operation on two different DRAM chips on a rank. Target refresh period improved to 5 and 4 ticks for denoted as solid and dashed lines, respectively. Figure 4.12 (b) shows how temperature changes over time in the experiment while Figure 4.12 (c) gives the number of weak rows found in the profiling. The figure shows that only a few weak rows need to be covered by our method.

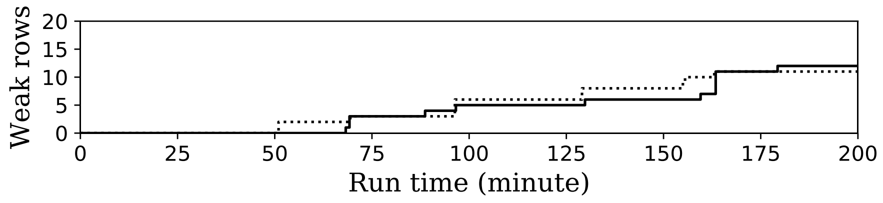
Figure 4.13 shows the average target refresh period we obtained by running the simulations 10 times on the HMM. The figure shows that the integrated method of proposed profiling and dual-row activation (denoted as 'Integrated' in the figure) offers 4.5X to 22.1X improvements in refresh period in the temperature range of 28 °C and 58 °C. The figure also



(a)



(b)



(c)

Figure 4.12 Simulation results: (a) profiling and refreshing different DRAM chips on a rank, (b) temperature changes during profiling, and (c) the number of weak rows found in each DRAM chip.

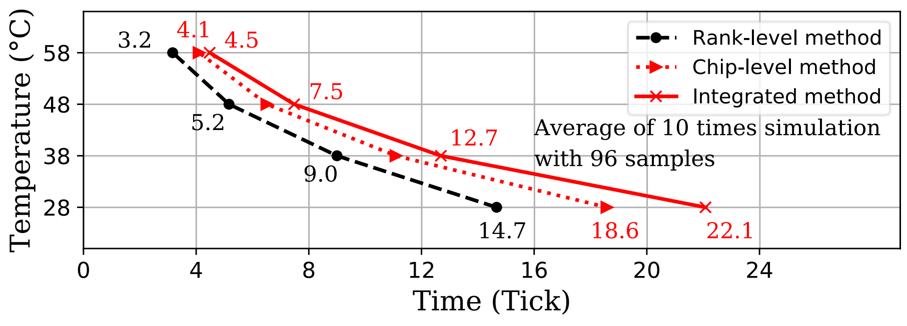


Figure 4.13 Average value of target refresh period (simulated 10 times).

shows the refresh period of rank-level and chip-level refresh methods (Section 4.6 only) which adjust re- fresh period based on the proposed profiling method at a granularity of DRAM rank and chip, respectively. As the figure shows, the more fine-grained management of refresh period, i.e. our proposed chip-level and integrated method, can give more improvements in refresh period than the coarse-grained rank-level solution.

4.8.3 Power Reduction

We evaluate the refresh power on Ramulator and DRAMPower with the simulation model of AVATAR, in-DRAM ECC and the proposed method. We assume that the retention time profiling is already done for AVATAR. Thus, the power consumption of AVATAR due to profiling is not included in the power consumption.

Table 4.5 gives a comparison in terms of power consumption, performance, and energy-delay-product (EDP). We obtain EDP by running 23 workloads of SPEC2006 with Ramulator and DRAMPower. The table shows the overall results (in geometric mean). We give more detailed results in Figure 4.14.

Table 4.5 shows that AVATAR, in-DRAM ECC and the proposed integrated method reduce the AREF power consumption by 20%, 16%, and 19% compared with the baseline of 64 ms refresh (DDR3 in the table). The reduction comes from 4X longer refresh period. AVATAR has no power overhead of DRAM, so its power consumption is the lowest in AREF mode. In-DRAM ECC and our proposed method have additional power consumption due to ECC and dual-row activation operation, respectively.

In terms of SREF power, the proposed method gives the largest reduction of 23.4% due to the (4X) increased refresh period. AVATAR does not reduce the power consumption since it re- quires the memory controller to perform multi-rate refresh and the memory controller operation is not available in SREF mode. Thus, AVATAR needs to adopt the conventional refresh period of 64 ms in SREF mode. Note that the SREF power of the proposed method includes the power consumption due to profiling, 0.88 mW as shown in Table 4.5, because we turn on the proposed profiling method continuously to check the retention time variation mostly due to temperature.

As Table 4.5 shows, we simulate three memory configurations, where

Table 4.5 Comparison of power, performance, and EDP (normalized to the ddr3 baseline case)

	DDR3	AVATAR	In-DRAM ECC	Proposed
Profiling power of 8 ranks (mW)	N/A	0.32	N/A	0.88
Area overhead in DRAM (%)	N/A	Controller support	>6.0%	<1.05%
AREF power of 8 ranks (mW)	259.8	208.5	217.4	210.6
SREF power of 8 ranks (mW)	141.1	141.1	109.4	108.2
Baseline case (4-active-rank)				
Power (normalized)	1.00	0.88	0.86	0.84
Speed (normalized)	1.00	1.01	0.95	1.00
EDP (normalized)	1.00	0.87	0.95	0.84
Energy-friendly case (1-active-rank)				
Power (normalized)	0.70	0.67	0.57	0.56
Speed (normalized)	0.96	0.97	0.91	0.96
EDP (normalized)	0.75	0.71	0.69	0.60
Performance-friendly case (8-active-rank)				
Power (normalized)	1.29	1.05	1.12	1.09
Speed (normalized)	1.02	1.03	0.98	1.02
EDP (normalized)	1.23	0.99	1.17	1.04

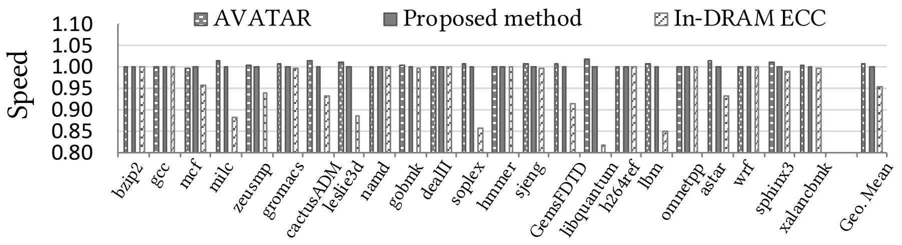
1, 4, 8 ranks of the total 8 ranks are operated. In the baseline condition (called 4-active-rank case), assuming that only half of DRAM chips operate as reported in [93], only 4 ranks of total 8 ranks run, and the rest are in SREF mode. All values are normalized to the DDR3 baseline. We obtain the minimum energy consumption when only 1 rank operates while the other 7 are in SREF mode, and the best performance when all the 8 ranks operate. We call these cases energy-friendly (1-active-rank) and performance-friendly (8-active-rank) cases, respectively. The performance-friendly case has a poor EDP because it consumes by 29% more power while giving only 2% performance gain compared with the DDR3 baseline. In the case of the performance-friendly case, ours improves EDP by 15.1% compared with the baseline, while AVATAR and in-DRAM ECC improves 19.0% and 4.6%, respectively. According to [93], this case rarely occurs in real systems.

Our proposed method offers the largest improvements in both the baseline (4-active-rank) and energy-friendly cases. In the case of the baseline rank configuration, ours improves EDP by 16.1%, 4.0%, and 11.2% compared with the baseline DDR3, AVATAR, and in-DRAM ECC, respectively.

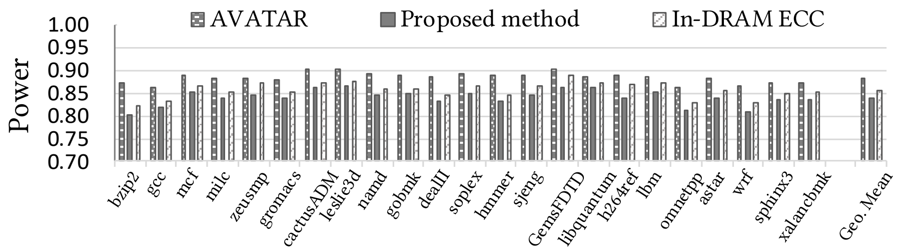
Our method gives larger improvements in the energy-friendly case that only one rank operates while providing 19.7%, 15.4%, and 12.4% better EDP than the other three methods. As will be shown later in this section, all the improvements of the proposed method come from the refresh power because no methods significantly improve the speed compared with the DDR3 baseline. Our proposed method outperforms AVATAR by the largest margin. It is because the energy-friendly case has the largest number of ranks in SREF mode and ours reduces the refresh power in SREF mode. In-DRAM ECC can also reduce the refresh power due to the same reason as ours. However, the power benefit of in-DRAM ECC is offset by the slower speed due to the poor write performance, which gives worse EDP than our proposed method.

Figure 4.14 shows the details on the results of the 4-active-rank case reported in Table 4.5. In Figure 4.14 (a), in-DRAM ECC gives a performance drop of average 5% (in geometric mean) while AVATAR and the proposed method do not lose performance. The performance loss is due to the additional read operation required for every write to generate a new parity. The error correction granularity of data, where in-DRAM ECC is applied, is 128 bytes while the access granularity (i.e., the amount of the read/write data) is 64 bytes. Thus, in case of write operation, in order to produce a new parity, the existing 64 byte data of the 128 bytes (which is not modified by the new write operation) needs to be read from the memory, which incurs additional latency and finally increases write latency, especially, for consecutive writes, i.e., WR_CCD on Table 4.3 [77]. Figure 4.14 (a) shows AVATAR has a slight (1%) speed gain due to the reduction in tRFC [35].

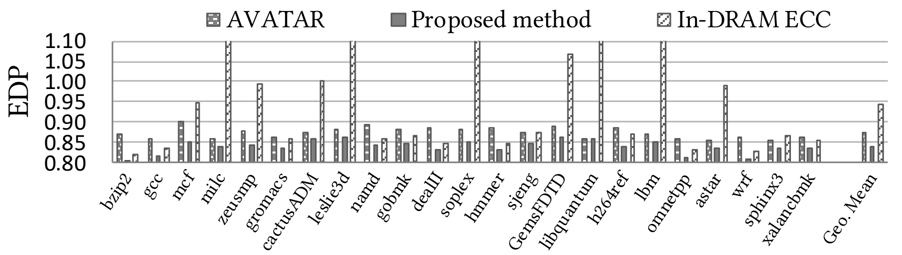
Figure 4.14 (b) shows that AVATAR, in-DRAM ECC and our pro-



(a)



(b)



(c)

Figure 4.14 Performance and power consumption on the baseline case.

posed method improve power consumption by 12%, 14%, and 16%, respectively. AVATAR reduces power consumption only in AREF mode as shown in Table 4.5. On the contrary, in-DRAM ECC and our pro- posed method can reduce power consumption in both AREF and SREF modes thereby offering more power reduction than AVATAR. Figure 4.14 (c) compares the EDP of the three methods. As the figure shows, our proposed method gives the best EDP (0.84), and the gain of 3.5% and 11.6% from AVATAR (0.87) and in-DRAM ECC (0.95), respectively.

We also performed a comparison on 11 benchmark programs in SPEC 2017 and obtained a similar trend, e.g., ours gives 18% gain in EDP while AVATAR and in-DRAM ECC give 4% and 10%, respectively. Due to the space limit and the completeness of comparison, we provide only the comparison on SPEC 2006.

4.9 Conclusion

In order to reduce the power consumption of DRAM main memory, we propose a novel scheme of chip-level multi-rate re- fresh. It is based on runtime profiling which runs in idle state and determines the minimum refresh period of each DRAM chip in a temperature-aware manner. Our proposed method reduces refresh rate, thereby refresh power, in both AREF and SREF modes without the support of memory controller. Our method offers further reduction in refresh period by improving weak rows with a low-cost dual-row activation method.

We evaluated the refresh period based on the statistical model developed with real measurements. Our proposed method gives 4.5X improvement in refresh period for the typical range of operating temperature (below 58 °C) which is by 12.5% better than AVATAR and in-DRAM ECC. We also evaluated performance and power consumption in various memory configurations using Ramulator and DRAMPower. The proposed method consistently outperforms AVATAR and in-DRAM ECC. Especially, in the energy-friendly case, our method improves EDP by 19.7%, 15.4%, and 12.4% compared with the baseline DDR3, AVATAR and in-DRAM ECC, respectively.

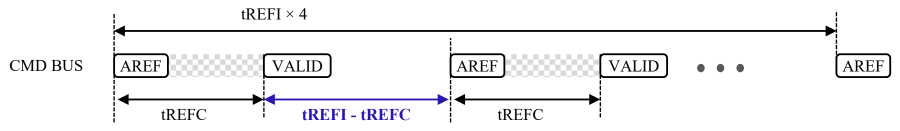
Chapter 5

System Integration

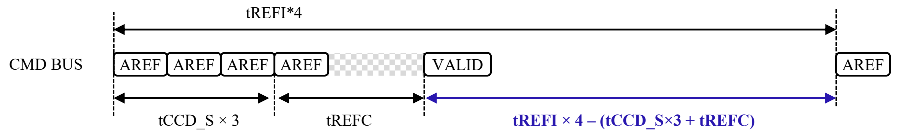
5.1 Integrate The Proposed Methods

The proposed accelerator is based on 8Gb DDR4. The refresh overhead of 8Gb DDR4 is about 4.5 % of throughput loss and about 13.6 % of background power. Figure 5.1 (a) shows the refresh schedule for the normal case. AREF is scheduled every tREFI. And after AREF is input, refresh is performed inside DRAM during tREFC. Due to the tREFC for refresh every tREFI, refresh affects the throughput. Since data access is not possible in DRAM while a refresh is being executed, the performance of not only DRAM-based accelerators but also all accelerators operating data-intensively is reduced. Therefore, the proposed refresh method needs to be integrated not only in DRAM-based accelerators but also in NN accelerators. And, as shown in the Figure 2.2, as the density increases, the refresh overhead is proportional to the DRAM density. For example, the overhead of 16Gb DDR4 increases twice.

The accelerator and refresh method proposed in this dissertation are



(a)



(b)

Figure 5.1 Refresh sequence. (a) Refresh schedule for the normal case and (b) for the integration of the proposed method.

orthogonal to each other, so they can be integrated into one DRAM chip. Among the DRAM basic commands ACT, PRE, RD/WT, and REF, the proposed accelerator was implemented using ACT, PRE, and RD/WT, and the proposed refresh method corresponds to REF and SREF. Therefore, each method's blocks are separate, and there is no need to modify the circuits to integrate them into one chip. An additional row is added to the sub-array and used for data backup and dual-row activation in both methods, so the additional row is also separated. In-DRAM accelerators need four additional rows to store weights, activations, AND offsets, and OR offsets. Since dual-row activation requires eight additional rows, a total of 12 additional rows are used. The refresh method's dual-row activation works only when the normal row is activated, so it does not affect

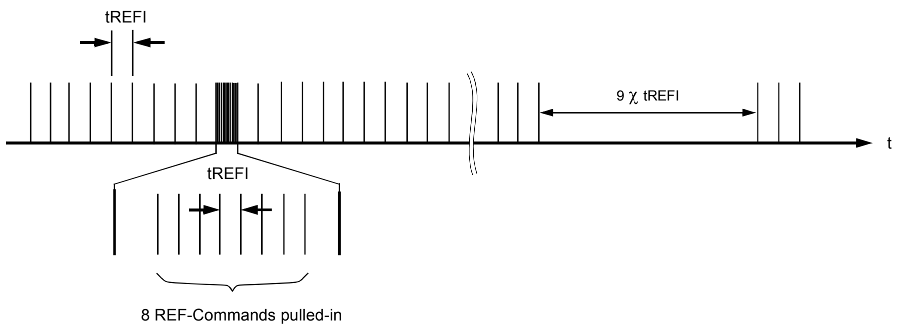


Figure 5.2 Pulling-in Refresh Commands [4].

the accelerator activating only additional rows for calculation.

To improve throughput loss due to refresh, execute refresh as shown in the Figure 5.1 (b). Execute AREF 4 times at intervals of tCCD_S, wait for tRFC, and execute another command. And it repeats at tREFI*4 intervals. When the proposed dual-row activation is applied, the refresh period increases by an average of 4.5 times, as shown in Figure 4.13. The refresh is scheduled in consideration of this characteristic. In general, DRAM performs refresh at tREFI intervals. However, to effectively perform scheduling and switching between tasks, postponing and pulling the AREF input are allowed. As shown in Figure 5.2, maximum of 8 AREF commands can be entered during the tREFI interval. In the proposed method, the refresh period of the DRAM is changed in integer units. For example, when the refresh period is increased four times, the refresh is increased by executing the refresh once the AREF command is input four times to the DRAM chip. Therefore, after inputting the 4th

AREF at tCCD_S interval, as shown in the figure, schedule the tREFC interval.

When the in-DRAM accelerator and refresh method of this dissertation are integrated into 8Gb DDR4, the throughput loss is improved by about 3.3% and the background power is reduced by about 10.0%. With the proposed refresh method, the accelerator throughput is 3.2 TOPS, and energy per image is 3.6 mJ. Efficiency is 1.0 TOPS/W.

5.2 Software Stack

Although research on customized accelerators is actively being conducted, not many accelerators are actually integrated into the system, except for the accelerator implemented based on FPGA. The reason is that a software stack between the deep learning framework (e.g., PyTorch [104] , Tensorflow [105]) and the accelerator is required to run various neural network models on the accelerator. Since this is redundant engineering work, developing a software stack to support accelerators in the framework at the academic level of accelerator research is beyond the research scope. At the industry level, the hardware vendors have released specially optimized libraries for neural network computations (e.g., cuDNN for NVIDIA GPU [106]). It is supported by all frameworks. However, since these libraries are hardware dependent, they cannot be used in other customized accelerators.

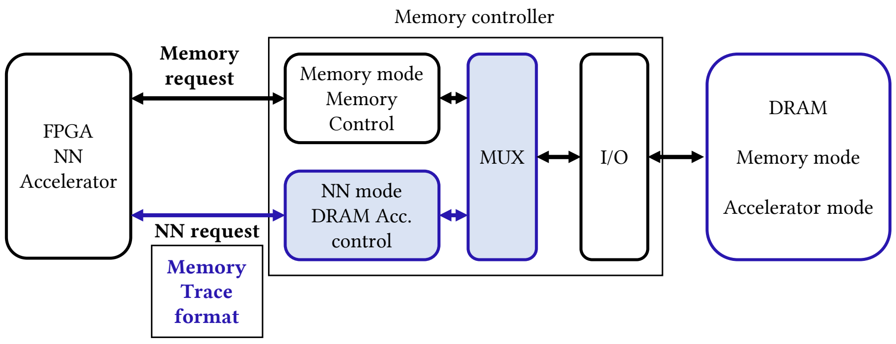
Table 5.1 The comparison of DL compilers [1]

	TVM	Glow	XLA
Developer	Apache	Facebook	Google
Framework support	tensorflow/tflite/keras pytorch/caffe2 mxnet/coreml/darknet	pytorch/caffe2 tensorflowlite	Use tensorflow interface
Supported devices	CPU/GPU/ARM FPGA/Customized (use VTA)	CPU/GPU Customized	CPU/GPU/TPU Customized
Programming	Python/C++	Python/C++	Python/C++
Training support	No (under developing)	Yes (Limitted)	Yes

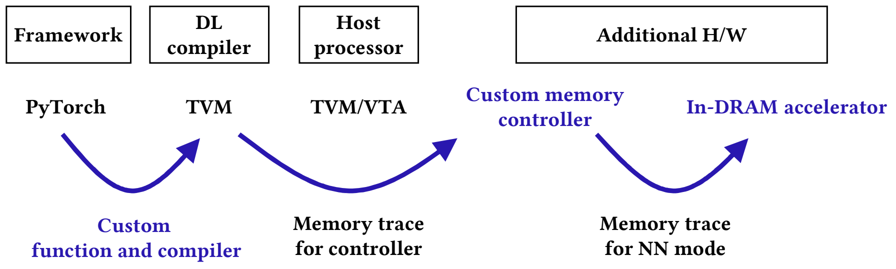
Recently, to effectively use customized accelerators in the DL framework, deep learning compilers that compile the model definition of the deep learning framework into optimized code for custom hardware have been proposed [1]. Popular DL compilers are TVM [107], Tensor Comprehension [108], Glow [109], nGraph [110], and XLA [111]. Table 5.1 is a comparison of those supporting DL framework PyTorch and tensorflow among these DL compilers. TVM and Glow support both PyTorch and tensorflow, but XLA only supports tensorflow. TVM supports accelerators that operate stand-alone, and Glow supports executing specific functions in customized accelerators. All DL compilers support Python and C++ languages and training support well only in XLA.

When comparing the currently developed DL compiler, TVM is the most suitable to use the in-DRAM accelerator in the DL framework. In order to operate the in-DRAM accelerator in Memory mode and NN mode, memory controller support is required. Also, since it only accelerates the vector dot product operation, a host processor is needed to execute other operations. TVM compiler supports custom accelerator implemented in FPGA, and officially provides TVM/VTA. Therefore, if we modify the TVM/VTA accelerator's memory controller, we can provide an environment that supports the in-DRAM accelerator.

The Figure 5.3 (a) is an example of system integration of our proposed in-DRAM accelerator. Since there is no in-DRAM accelerator chip, this is a summary of the system integration blocks. The block marked



(a)



(b)

Figure 5.3 (a) Block diagrams for system integration, and (b) software stack.

with a blue line is for NN mode, and the block painted blue is verified in Chapter 4 [67]. Because our proposed accelerator supports only vector dot product or bit-wise logical operation, the host processor is required. In Figure 5.3, it is assumed that the host processor uses the FPGA NN accelerator (TVM/VTA) and the DL compiler uses the TVM. Our proposed accelerator should control memory access with memory mode and NN mode. For easy implementation, put a memory controller corresponding to each mode, as shown in the Figure. In other words, the memory controller has two internal-controller blocks with output MUX.

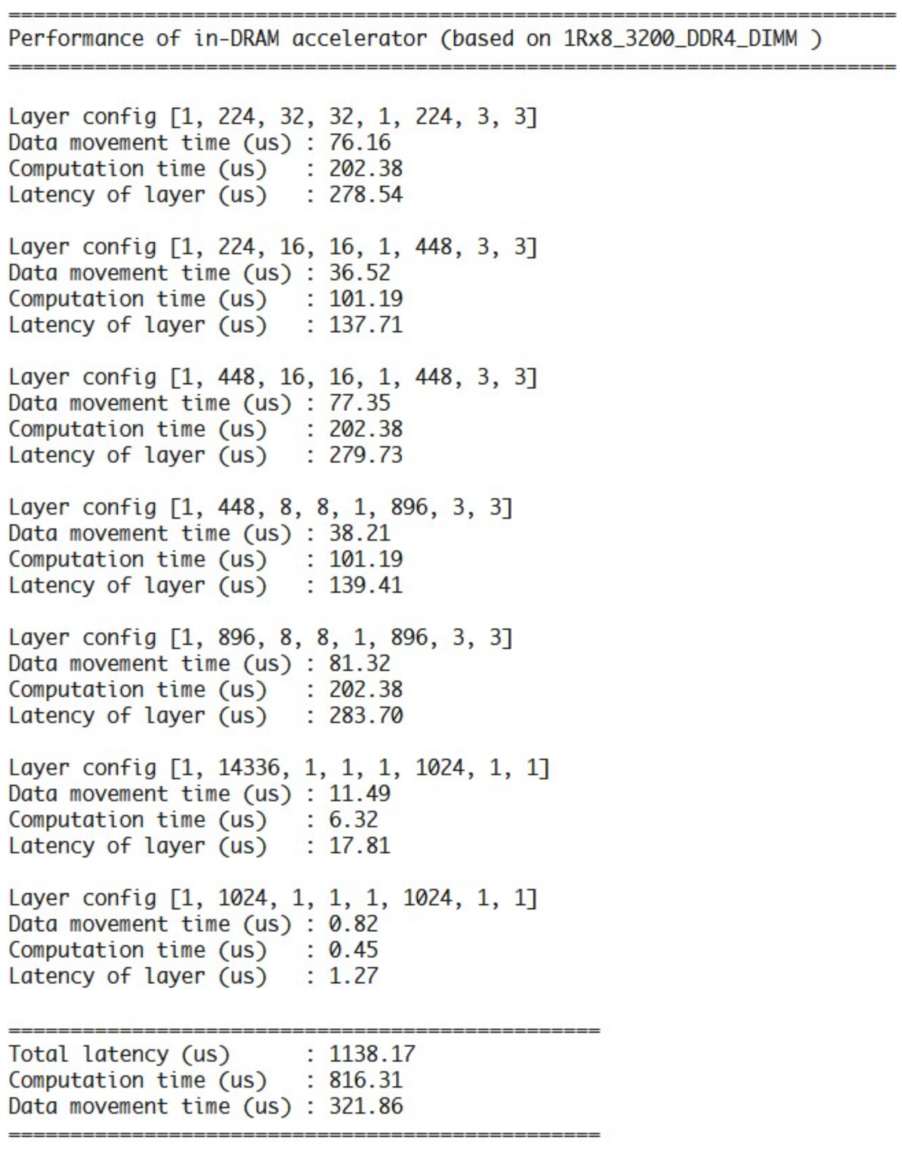
In the experiment of Chapter 4, we implemented a memory controller on an FPGA board and measured the characteristics of DRAM. As shown Figure 5.3 (a), a normal memory controller and a custom memory controller are used. A custom memory controller is used when measuring DRAM characteristics. DRAM can be controlled at a low-level also in the application level. The memory controller's operation has been verified in an actual environment. Since there is no real in-DRAM accelerator chip, implementing an NN mode memory controller is out of the current research focus. However, as described above, it is clear that FPGA board can support in-DRAM accelerator by implementing NN mode memory controller, as shown in the figure.

Unlike the code executed in the host processor, the in-DRAM accelerator's code cannot be compiled through the DL compiler because it is a memory command trace format. Instead, as shown in Figure 5.3 (b), we will convert the convolution or fully-connected layer of the PyTorch model into a custom function. Using a custom compiler, we compile the custom function to a memory write (or read) sequence in binary data format for the NN mode memory controller. We called this a memory trace for controller. When the memory trace for controller transferred to the NN mode memory controller, the memory controller generates a memory trace for NN mode. In the memory controller, logical address with the memory trace for controller is mapped to the physical address with memory trace for NN mode. The memory trace for NN mode is a low-level DRAM commands sequence to control the in-DRAM accelerator to execute the neural network. Command trace is generated in consideration of off-chip data movement and DRAM structure, as described in the data flow of Chapter 3. The memory write sequence becomes the operation sequence of the in-DRAM accelerator. After the operation, read the Memory. Operations not supported by the in-DRAM accelerator are executed by the host processor.

We added the performance simulation function of the in-DRAM accelerator to PyTorch. When running a neural network in PyTorch, the simulation program detects the layer which can be executed in in-DRAM accelerator, and it reports the computation latency and data movement latency when running in the proposed in-DRAM accelerator. Figure 5.4 is an example of simulation results of VGG-9 with CIFAR-10 in the 1Rx8 3200 DDR4 DRAM based accelerator. It reports the result for each layer executed in the in-DRAM accelerator, and the sum of latency of the executed layers.

Simulation reflects the operation of Chapter 3. Computation latency results from data being mapped to DRAM and executed according to the feature map size. Data movement latency is the sum of input data movement latency (M2V write) and output data movement latency (SiD read). The simulation result is the result of reflecting the DRAM configuration and timing parameters. Since it can be executed by changing the DRAM configuration, it is also possible to evaluate in-DRAM accelerators' performance in Memory other than DDR4 DRAM.

Since it is a simulation, we can check the in-DRAM accelerator's performance regardless of the current hardware that the neural network is running on. At the stage of co-designing the In-DRAM accelerator hardware and the network suitable for it, it is advantageous to predict the performance in advance. We can develop a network suitable for in-DRAM accelerators or review hardware changes to fit the neural networks. It can also be used to compare performance when run on different hardware.



```
=========================================================================
Performance of in-DRAM accelerator (based on 1Rx8_3200_DDR4_DIMM )
=========================================================================

Layer config [1, 224, 32, 32, 1, 224, 3, 3]
Data movement time (us) : 76.16
Computation time (us)   : 202.38
Latency of layer (us)   : 278.54

Layer config [1, 224, 16, 16, 1, 448, 3, 3]
Data movement time (us) : 36.52
Computation time (us)   : 101.19
Latency of layer (us)   : 137.71

Layer config [1, 448, 16, 16, 1, 448, 3, 3]
Data movement time (us) : 77.35
Computation time (us)   : 202.38
Latency of layer (us)   : 279.73

Layer config [1, 448, 8, 8, 1, 896, 3, 3]
Data movement time (us) : 38.21
Computation time (us)   : 101.19
Latency of layer (us)   : 139.41

Layer config [1, 896, 8, 8, 1, 896, 3, 3]
Data movement time (us) : 81.32
Computation time (us)   : 202.38
Latency of layer (us)   : 283.70

Layer config [1, 14336, 1, 1, 1, 1024, 1, 1]
Data movement time (us) : 11.49
Computation time (us)   : 6.32
Latency of layer (us)   : 17.81

Layer config [1, 1024, 1, 1, 1, 1024, 1, 1]
Data movement time (us) : 0.82
Computation time (us)   : 0.45
Latency of layer (us)   : 1.27

==================================================
Total latency (us)      : 1138.17
Computation time (us)   : 816.31
Data movement time (us) : 321.86
==================================================
```

Figure 5.4 An example of simulation results of VGG-9 with CIFAR-10 in the 1Rx8 3200 DDR4 DRAM based accelerator.

Chapter 6

Conclusion

In this dissertation, I present 1) a DRAM-based accelerator architecture and 2) a DRAM refresh method to improve performance reduction due to DRAM refresh. Both methods are orthogonal, so can be integrated into the DRAM chip and operate independently.

First, we proposed a DRAM-based accelerator architecture capable of massive and large vector dot product operation. In the field of CNN accelerators to which BNN can be applied, a computing-in-memory (CIM) structure that utilizes a cell-array structure of Memory for vector dot product operation is being actively studied. Since DRAM stores all the neural network data, it is advantageous to reduce the amount of data transfer. The proposed architecture operates by utilizing the basic operation of the DRAM.

The second method is to reduce the performance degradation and power consumption caused by DRAM refresh. Since the DRAM cannot read and write data while performing a periodic refresh, system performance decreases. The proposed refresh method tests the refresh characteristics inside the DRAM chip during self-refresh and increases the refresh cycle according to the characteristics. Since it operates independently inside DRAM, it can be applied to all systems using DRAM and is the same for deep neural network accelerators.

When the in-DRAM accelerator and refresh method of this dissertation are integrated into 8Gb DDR4, the throughput loss is improved by about 3.3% and the background power is reduced by about 10.0%. With the proposed refresh method, the accelerator throughput is 3.2 TOPS, and energy per image is 3.6 mJ. Efficiency is 1.0 TOPS/W.

We surveyed system integration with a software stack to use the in-DRAM accelerator in the DL framework. As a result, it is expected to control in-DRAM accelerators with the memory controller implementation method verified in the previous experiment. Also, we have added the performance simulation function of in-DRAM accelerator to PyTorch. When running a neural network in PyTorch, it reports the computation latency and data movement latency occurring in the layer running in the in-DRAM accelerator. It is a significant advantage to predict the performance when running in hardware while co-designing the network.

Bibliography

[1] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, L. Gan, G. Yang, and D. Qian, "The deep learning compiler: A comprehensive survey," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 708–727, 2020.

[2] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 1–12, 2012.

[3] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "Rana: Towards efficient neural acceleration with refresh-optimized embedded dram," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 340–352, IEEE, 2018.

[4] "DDR3 SDRAM specification JESD79-4B." 2017.

[5] Y. Jia, *Learning semantic image representations at a large scale*. PhD thesis, UC Berkeley, 2014.

[6] A. Biswas and A. P. Chandrakasan, "Conv-ram: An energy-efficient sram with embedded convolution computation for low-

power cnn-based machine learning applications," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 488–490, IEEE, 2018.

[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 Sigplan Notices*, vol. 49, pp. 269–284, ACM, 2014.

[8] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, IEEE Computer Society, 2014.

[9] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[10] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 243–254, IEEE, 2016.

[11] L. S. Standard, "Jedec jesd209-4," 2015.

[12] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Neurostream: Scalable and energy efficient deep learning with smart memory cubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 2, pp. 420–434, 2017.

[13] "DDR4 Power Calculator 4.0." `https://www.micron.com/~/media/documents/products/power-calculator/ddr4_power_calc.xlsm`. Accessed: 2019-07-18.

[14] "8Gb DDR4 SDRAM." `https://www.skhynix.com/products.view.do?vseq=2658&cseq=73`. Accessed: 2020-10-28.

[15] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, pp. 4107–4115, 2016.

[16] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[17] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural net-

works," in *European conference on computer vision*, pp. 525–542, Springer, 2016.

[18] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition*, p. 107281, 2020.

[19] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 383–396, IEEE, 2018.

[20] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 273–287, IEEE, 2017.

[21] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 288–301, IEEE, 2017.

[22] S. Li, A. O. Glova, X. Hu, P. Gu, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Scope: A stochastic computing en-

gine for dram-based in-situ accelerator.," in *MICRO*, pp. 696–709, 2018.

[23] M. Imani, S. Gupta, and T. Rosing, "Ultra-efficient processing in-memory for data intensive applications," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2017.

[24] Y. Kim, H. Kim, D. Ahn, and J.-J. Kim, "Input-splitting of large neural networks for power-efficient accelerator with resistive crossbar memory array," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 1–6, 2018.

[25] S. K. Gonugondla, M. Kang, Y. Kim, M. Helm, S. Eilert, and N. Shanbhag, "Energy-efficient deep in-memory architecture for nand flash memories," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2018.

[26] H. Shin, D. Kim, E. Park, S. Park, Y. Park, and S. Yoo, "Mcdram: Low latency and energy-efficient matrix computations in dram," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2613–2622, 2018.

[27] S. Cho, H. Choi, E. Park, H. Shin, and S. Yoo, "Mcdram v2: In-dynamic random access memory systolic array accelerator to

address the large model problem in deep neural networks on the edge," *IEEE Access*, vol. 8, pp. 135223–135243, 2020.

[28] G. Santoro, G. Turvani, and M. Graziano, "New logic-in-memory paradigms: An architectural and technological perspective," *Micromachines*, vol. 10, no. 6, p. 368, 2019.

[29] T. Ohsawa, K. Kai, and K. Murakami, "Optimizing the dram refresh count for merged dram/logic lsis," in *Proceedings. 1998 International Symposium on Low Power Electronics and Design (IEEE Cat. No. 98TH8379)*, pp. 82–87, IEEE, 1998.

[30] M. Ghosh and H.-H. S. Lee, "Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3d die-stacked drams," in *40th Annual IEEE/ACM international symposium on microarchitecture (MICRO 2007)*, pp. 134–145, IEEE, 2007.

[31] C. Isen and L. John, "Eskimo-energy savings using semantic knowledge of inconsequential memory occupancy for dram subsystem," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 337–346, IEEE, 2009.

[32] H. H. Shin, H. Seo, B. Lee, J. Kim, and E.-Y. Chung, "Timing window wiper: A new scheme for reducing refresh power of dram," in

2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 133–138, IEEE, 2017.

[33] D. Zhang, M. Ehsan, M. Ferdman, and R. Sion, “Dimmer: A case for turning off dimms in clouds,” in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 1–8, 2014.

[34] R. K. Venkatesan, S. Herr, and E. Rotenberg, “Retention-aware placement in dram (rapid): Software methods for quasi-non-volatile dram,” in *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.*, pp. 155–165, IEEE, 2006.

[35] M. K. Qureshi, D.-H. Kim, S. Khan, P. J. Nair, and O. Mutlu, “Avatar: A variable-retention-time (vrt) aware refresh for dram systems,” in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 427–437, IEEE, 2015.

[36] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, “Flikker: saving dram refresh-power through critical data partitioning,” in *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, pp. 213–224, 2011.

[37] S.-H. Kim, W.-O. Lee, J.-H. Kim, S.-S. Lee, S.-Y. Hwang, C.-I. Kim, T.-W. Kwon, B.-S. Han, S.-K. Cho, D.-H. Kim, *et al.*, "A low power and highly reliable 400mbps mobile ddr sdram with on-chip distributed ecc," in *2007 IEEE Asian Solid-State Circuits Conference*, pp. 34–37, IEEE, 2007.

[38] N. Kwak, S.-H. Kim, K. H. Lee, C.-K. Baek, M. S. Jang, Y. Joo, S.-H. Lee, W. Y. Lee, E. Lee, D. Han, *et al.*, "23.3 a 4.8 gb/s/pin 2gb lpddr4 sdram with sub-100μa self-refresh current for iot applications," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 392–393, IEEE, 2017.

[39] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

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

[42] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[43] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[44] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[45] H. Wu, “Nvidia low precision inference ongpu,” *GPU Technology Conference*, 2019.

[46] J. Song, Y. Cho, J.-S. Park, J.-W. Jang, S. Lee, J.-H. Song, J.-G. Lee, and I. Kang, “7.1 an 11.5 tops/w 1024-mac butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile soc,” in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 130–132, IEEE, 2019.

[47] M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in *2014 IEEE International Solid-State Circuits Con-*

ference Digest of Technical Papers (ISSCC), pp. 10–14, IEEE, 2014.

[48] 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," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[49] H. Kim, Y. Kim, and J.-J. Kim, "In-memory batch-normalization for resistive memory based binary neural network hardware," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pp. 645–650, 2019.

[50] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 802–815, IEEE, 2019.

[51] W.-H. Chen, K.-X. Li, W.-Y. Lin, K.-H. Hsu, P.-Y. Li, C.-H. Yang, C.-X. Xue, E.-Y. Yang, Y.-K. Chen, Y.-S. Chang, *et al.*, "A 65nm 1mb nonvolatile computing-in-memory reram macro with sub-16ns multiply-and-accumulate for binary dnn ai edge processors," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 494–496, IEEE, 2018.

[52] T. F. Wu, H. Li, P.-C. Huang, A. Rahimi, J. M. Rabaey, H.-S. P. Wong, M. M. Shulaker, and S. Mitra, “Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study,” in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 492–494, IEEE, 2018.

[53] H. Jang, J. Kim, J.-E. Jo, J. Lee, and J. Kim, “Mnnfast: A fast and scalable system architecture for memory-augmented neural networks,” in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 250–263, 2019.

[54] Y. Kim, H. Kim, and J.-J. Kim, “Neural network-hardware co-design for scalable rram-based bnn accelerators,” *arXiv preprint arXiv:1811.02187*, 2018.

[55] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, “Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks,” in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 236–249, 2019.

[56] R. Mochida, K. Kouno, Y. Hayata, M. Nakayama, T. Ono, H. Suwa, R. Yasuhara, K. Katayama, T. Mikawa, and Y. Gohou, “A 4m synapses integrated analog reram based 66.5 tops/w neural-network processor with cell current controlled writing and flexible

network architecture," in *2018 IEEE Symposium on VLSI Technology*, pp. 175–176, IEEE, 2018.

[57] S. Mittal, "A survey of reram-based architectures for processing-in-memory and neural networks," *Machine learning and knowledge extraction*, vol. 1, no. 1, pp. 75–114, 2019.

[58] S. Mittal, "A survey of soft-error mitigation techniques for non-volatile memories," *Computers*, vol. 6, no. 1, p. 8, 2017.

[59] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie, "Half-dram: A high-bandwidth and low-power dram architecture from the rethinking of fine-grained activation," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 349–360, IEEE, 2014.

[60] S. Mittal, "A survey of architectural techniques for dram power management," *International Journal of High Performance Systems Architecture*, vol. 4, no. 2, pp. 110–119, 2012.

[61] S.-W. Park, S.-J. Hong, J.-W. Kim, J.-G. Jeong, K.-D. Yoo, S.-C. Moon, H.-C. Sohn, N.-J. Kwak, Y.-S. Cho, S.-J. Baek, *et al.*, "Highly scalable saddle-fin (s-fin) transistor for sub-50nm dram technology," in *2006 Symposium on VLSI Technology, 2006. Digest of Technical Papers.*, pp. 32–33, IEEE, 2006.

[62] 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 *2008 IEEE International Test Conference*, pp. 1–7, IEEE, 2008.

[63] T. Hamamoto, S. Sugiura, and S. Sawada, "On the retention time distribution of dynamic random access memory (dram)," *IEEE Transactions on Electron devices*, vol. 45, no. 6, pp. 1300–1309, 1998.

[64] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, *et al.*, "Rowclone: fast and energy-efficient in-dram bulk data copy and initialization," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 185–197, 2013.

[65] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-latency dram: A low latency and low cost dram architecture," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 615–626, IEEE, 2013.

[66] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (salp) in dram," in *2012*

39th Annual International Symposium on Computer Architecture (ISCA), pp. 368–379, IEEE, 2012.

[67] H. Choi, D. Hong, J. Lee, and S. Yoo, “Reducing dram refresh power consumption by runtime profiling of retention time and dual-row activation,” *Microprocessors and Microsystems*, vol. 72, p. 102942, 2020.

[68] H. Choi, Y. Lee, J.-J. Kim, and S. Yoo, “A novel in-dram accelerator architecture for binary neural network,” in *2020 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, pp. 1–3, IEEE, 2020.

[69] J. Song, Y. Cho, J.-S. Park, J.-W. Jang, S. Lee, J.-H. Song, J.-G. Lee, and I. Kang, “7.1 an 11.5 tops/w 1024-mac butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile soc,” in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 130–132, IEEE, 2019.

[70] Y. Yamada, T. Sano, Y. Tanabe, Y. Ishigaki, S. Hosoda, F. Hyuga, A. Moriya, R. Hada, A. Masuda, M. Uchiyama, *et al.*, “7.2 a 20.5 tops and 217.3 gops/mm 2 multicore soc with dnn accelerator and image signal processor complying with iso26262 for automotive applications,” in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 132–134, IEEE, 2019.

[71] Z. Li, Y. Chen, L. Gong, L. Liu, D. Sylvester, D. Blaauw, and H.-S. Kim, "An 879gops 243mw 80fps vga fully visual cnn-slam processor for wide-range autonomous exploration," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 134–136, IEEE, 2019.

[72] C. Kim, S. Kang, D. Shin, S. Choi, Y. Kim, and H.-J. Yoo, "A 2.1 tflops/w mobile deep rl accelerator with transposable pe array and experience compression," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 136–138, IEEE, 2019.

[73] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "7.7 lnpu: A 25.3 tflops/w sparse deep-neural-network learning processor with fine-grained mixed precision of fp8-fp16," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 142–144, IEEE, 2019.

[74] "DDR3 SDRAM specification JESD79-3F." 2010.

[75] "Micron DRAM Modules." `https://www.micron.com/products/dram-modules`. Accessed: 2019-07-18.

[76] "4Gb DDR3L SDRAM Data sheet." `http://www.skhynix.com/eng/support/technicalSupport.jsp`. Accessed: 2020-10-28.

[77] S. Cha, O. Seongil, H. Shin, S. Hwang, K. Park, S. J. Jang, J. S. Choi, G. Y. Jin, Y. H. Son, H. Cho, *et al.*, "Defect analysis and cost-effective resilience architecture for future dram devices," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 61–72, IEEE, 2017.

[78] J. Choi, W. Shin, J. Jang, J. Suh, Y. Kwon, Y. Moon, and L.-S. Kim, "Multiple clone row dram: A low latency and area optimized dram," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3S, pp. 223–234, 2015.

[79] "LPDDR3 SDRAM Specification JESD209-3C." 2015.

[80] Y. Han, Y. Wang, H. Li, and X. Li, "Data-aware dram refresh to squeeze the margin of retention time in hybrid memory cube," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 295–300, IEEE, 2014.

[81] H. Seol, W. Shin, J. Jang, J. Choi, H. Lee, and L.-S. Kim, "Elaborate refresh: a fine granularity retention management for deep submicron drams," *IEEE Transactions on Computers*, vol. 67, no. 10, pp. 1403–1415, 2018.

[82] S. Wang, M. N. Bojnordi, X. Guo, and E. Ipek, "Content aware refresh: Exploiting the asymmetry of dram retention errors to reduce

the refresh frequency of less vulnerable data," *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 362–374, 2018.

[83] Y. Kagenishi, H. Hirano, A. Shibayama, H. Kotani, N. Moriwaki, M. Kojima, and T. Sumi, "Low power self refresh mode dram with temperature detecting circuit," in *Symp. VLSI Circuits Dig. Tech. Papers*, pp. 43–44, 1993.

[84] B. Oh, N. Abeyratne, J. Ahn, R. G. Dreslinski, and T. Mudge, "Enhancing dram self-refresh for idle power reduction," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 254–259, 2016.

[85] M. D. Pardeik, E. R. Cordero, A. Raychaudhuri, and D. B. C. Vidyapoornachary, "Implementing dram refresh power optimization during long idle mode," May 28 2019. US Patent 10,304,501.

[86] K. Kim and J. Lee, "A new investigation of data retention time in truly nanoscaled drams," *IEEE Electron Device Letters*, vol. 30, no. 8, pp. 846–848, 2009.

[87] H. Kim, B. Oh, Y. Son, K. Kim, S.-Y. Cha, J.-G. Jeong, S.-J. Hong, and H. Shin, "Study of trap models related to the variable retention time phenomenon in dram," *IEEE transactions on electron devices*, vol. 58, no. 6, pp. 1643–1648, 2011.

[88] Y. Wang, Y.-H. Han, C. Wang, H. Li, and X. Li, "Retention-aware dram assembly and repair for future fgr memories," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 5, pp. 705–718, 2016.

[89] M. Chang, J. Lin, C.-S. Lai, R.-D. Chang, S. N. Shih, M.-Y. Wang, and P.-I. Lee, "Si-h bond breaking induced retention degradation during packaging process of 256 mbit drams with negative wordline bias," *IEEE transactions on electron devices*, vol. 52, no. 4, pp. 484–491, 2005.

[90] "DDR4 Device Operation." `http://www.skhynix.com/eng/support/technicalSupport.jsp`. Accessed: 2020-10-28.

[91] S. Hong, "Memory technology trend and future challenges," in *2010 International Electron Devices Meeting*, pp. 12–4, IEEE, 2010.

[92] S.-K. Park, "Technology scaling challenge and future prospects of dram and nand flash memory," in *2015 IEEE International Memory Workshop (IMW)*, pp. 1–4, IEEE, 2015.

[93] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*, pp. 1–13, 2012.

[94] H. Huang, K. G. Shin, C. Lefurgy, and T. Keller, "Improving energy efficiency by making dram less randomly accessed," in *Proceedings of the 2005 international symposium on Low power electronics and design*, pp. 393–398, 2005.

[95] H. H. Shin, Y. M. Park, D. Choi, B. J. Kim, D.-H. Cho, and E.-Y. Chung, "Extreme: Exploiting page table for reducing refresh power of 3d-stacked dram memory," *IEEE Transactions on Computers*, vol. 67, no. 1, pp. 32–44, 2017.

[96] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-latency dram: Optimizing dram timing for the common-case," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 489–501, IEEE, 2015.

[97] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder, "Temperature management in data centers: why some (might) like it hot," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, pp. 163–174, 2012.

[98] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch,

et al., “Rowclone: fast and energy-efficient in-dram bulk data copy and initialization,” in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 185–197, 2013.

[99] S. Chaudhuri, J. A. McCall, and J. H. Salmon, “Proposal for ber based specifications for ddr4,” in *19th Topical Meeting on Electrical Performance of Electronic Packaging and Systems*, pp. 121–124, IEEE, 2010.

[100] L. Rabiner and B. Juang, “An introduction to hidden markov models,” *ieee assp magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[101] Y. Kim, W. Yang, and O. Mutlu, “Ramulator: A fast and extensible dram simulator,” *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[102] K. Chandrasekar, C. Weis, B. Akesson, N. Wehn, and K. Goossens, “Towards variation-aware system-level power estimation of drams: an empirical approach,” in *Proceedings of the 50th Annual Design Automation Conference*, pp. 1–8, 2013.

[103] “SPEC CPU2006.” `http://www.spec.org/cpu2006`.

[104] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, “Pytorch: An

imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, pp. 8026–8037, 2019.

[105] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.

[106] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[107] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, *et al.*, "{TVM}: An automated end-to-end optimizing compiler for deep learning," in *13th Symposium on Operating Systems Design and Implementation (18)*, pp. 578–594, 2018.

[108] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. DeVito, W. S. Moses, S. Verdoolaege, A. Adams, and A. Cohen, "Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions," *arXiv preprint arXiv:1802.04730*, 2018.

[109] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, R. Levenstein, *et al.*, “Glow: Graph lowering compiler techniques for neural networks,” *arXiv preprint arXiv:1805.00907*, 2018.

[110] S. Cyphers, A. K. Bansal, A. Bhiwandiwalla, J. Bobba, M. Brookhart, A. Chakraborty, W. Constable, C. Convey, L. Cook, O. Kanawi, *et al.*, “Intel ngraph: An intermediate representation, compiler, and executor for deep learning,” *arXiv preprint arXiv:1801.08058*, 2018.

[111] C. Leary and T. Wang, “Xla: Tensorflow, compiled,” *TensorFlow Dev Summit*, 2017.

국문초록

컨볼루셔널 뉴럴 네트워크 (CNN) 어플리케이션에서는, 대부분의 연산이 컨볼루션 레이어와 풀리-커넥티드 레이어에서 발생하는 곱셈과 누적 연산이다. 게이트-로직 레벨에서는, 대량의 벡터 내적으로 실행되며, 입력과 커널 벡터들을 반복해서 사용하여 연산한다. 딥 뉴럴 네트워크 연산에는 범용 연산 유닛보다, 단순한 연산이 가능한 작은 연산 유닛을 대량으로 사용하는 것이 적합하다. 가속기의 성능이 일정 이상 높아지면, 가속기의 성능은 연산에 필요한 데이터 전송에 의해 제한된다. 메모리에서 데이터를 오프-칩으로 전송할 때의 에너지 소모가, 연산 유닛에서 연산에 사용되는 에너지의 수백배로 크다. 또한 연산기의 성능은 초당 수백 기가~수 테라-연산이 가능하지만, 메모리의 데이터 전송은 초당 수십 기가 바이트이다.

데이터 전송에 의한 파워와 성능 문제를 동시에 해결하는 방법은, 전송되는 데이터 크기를 줄이는 것이다. 알고리즘 중에서는 네트워크의 데이터를 양자화하여, 낮은 정밀도로 데이터를 표현하는 방법이 널리 사용된다. 이진 뉴럴 네트워크(BNN)는 정밀도를 1비트까지 극단적으로 낮춘다. 16비트 정밀도보다 네트워크의 정확도가 낮아지는 문제가 있지만, 다양한 연구를 통해 정확도가 지속적으로 개선되고 있다. 또한 구조적으로는, 전송된 데이터를 재사용하여 동일한 데이터의 반복적인 전송을 줄이는 방법이 있다. 위의 두 가지 방법은 추론 과정에서

별도의 연산 없이 적용 가능하여 가속기에서 널리 적용되고 있다.

본 논문에서는, DRAM 기반의 가속기 구조를 제안하고, DRAM refresh에 의한 성능 감소를 개선하는 기술을 제안하였다. 두 방법은 하나의 DRAM 칩으로 집적 가능하며, 독립적으로 구동 가능하다.

첫번째는 대량의 벡터 내적 연산이 가능한 DRAM 기반 가속기에 대한 연구이다. BNN을 적용할 수 있는 CNN가속기 분야에서, 메모리의 셀-어레이 구조를 벡터 내적 연산에 활용하는 컴퓨팅-인-메모리 (CIM) 구조가 활발히 연구되고 있다. 특히, DRAM에는 뉴럴 네트워크의 모든 데이터가 있기 때문에, 데이터 전송량의 감소에 유리하다. 우리는 DRAM 셀-어레이의 구조를 바꾸지 않고, DRAM의 기본 동작을 활용하여 연산하는 방법을 제안하였다.

두번째는 DRAM 리프레쉬 주기를 늘려서 성능 열화와 파워 소모를 개선하는 방법이다. DRAM이 리프레쉬를 실행할 때마다, 데이터를 읽고 쓸 수 없기 때문에 시스템 혹은 가속기의 성능 감소가 발생한다. DRAM 칩 내부에서 DRAM의 리프레쉬 특성을 테스트하고, 리프레쉬 주기를 늘리는 방법을 제안하였다. DRAM 내부에서 독립적으로 동작하기 때문에 DRAM을 사용하는 모든 시스템에 적용 가능하며, 딥 뉴럴 네트워크 가속기에서도 동일하다.

또한, 제안된 가속기를 PyTorch와 같이 널리 사용되는 딥러닝 프레임 워크에서도 쉽게 사용할 수 있도록, 소프트웨어 스택을 비롯한 system integration 방법을 조사하였다. 결과적으로, 기존의 TVM compiler와 FPGA로 구현하는 TVM/VTA 가속기에, DRAM refresh 실험에서 검증된 메모리 컨트롤러와 커스텀 컴파일러를 추가하면 in-DRAM

가속기를 제어할 수 있을 것으로 기대된다. 이에 더하여, in-DRAM 가속기와 뉴럴 네트워크의 설계 단계에서 성능을 예측할 수 있도록, 시뮬레이션 기능을 PyTorch에 추가하였다. PyTorch에서 신경망을 실행할 때, DRAM 가속기에서 실행되는 계층에서 발생하는 계산 대기 시간 및 데이터 이동 시간을 확인할 수 있다.

주요어: 뉴럴네트워크, 메모리 병목현상, 컨볼루션 뉴럴네트워크, 이진 뉴럴 네트워크, 컴퓨팅-인-메모리, 디램-기반의 가속기, 디램 리프레쉬
학번: 2016-34620