An Approximate Memory Architecture for Energy Saving in Deep Learning Applications

Duy Thanh Nguyen, Nguyen Huy Hung, Hyun Kim, *Member, IEEE*, and Hyuk-Jae Lee, *Member, IEEE*

***Abstract*— DRAM devices require periodic refresh operations to preserve data integrity. Slowing down the refresh rate can reduce the energy consumption; however, it may cause a loss of data stored in the DRAM cell. This paper proposes a new memory architecture of soft approximation for deep learning applications, which reduces the refresh energy consumption while maintaining accuracy and high performance. Utilizing the error-tolerant property of deep learning applications, the proposed memory architecture avoids the accuracy drop caused by data loss by flexibly controlling the refresh operation for different bits, depending on their criticality. For data storage, the approximate DRAM architecture reorganizes the data so that these data are sorted according to their bit significance. Critical bits are stored in more frequently refreshed devices while non-critical bits are stored in less frequently refreshed devices. In addition, for further reduction of the DRAM energy consumption, this paper combines hard approximation, which reduces the number of accesses to DRAM, with soft approximation. Simulation results show that the refresh energy consumption is reduced by 69.71%, and the total energy consumption is reduced by 26.0 % for the hybrid memory with a negligible drop in both training and testing phases on state-of-the-art deep networks.**

***Index Terms*— Approximate DRAM, deep learning, energy-efficient, row-level refresh, transposed memory.**

I. Introduction

DEEP learning applications have been widely used in various fields. For achieving high performance in terms of accuracy, they require large amounts of computation with excessive data traffic from the memory [1]. Furthermore, with the increase in the number of layers in neural networks for obtaining better performance, there has been an increase in the complexity of deep learning algorithms. For instance, AlexNet [2], which was proposed in 2012, uses 8 layers, whereas ResNet [3], which was proposed in 2015, is implemented with 152 layers. Accordingly, memory bandwidth has become one of the biggest bottlenecks for speeding up deep learning applications [1]. The increased memory bandwidth results in significant memory energy consumption. Therefore, extensive research efforts have been undertaken to reduce the number of memory accesses [1], [4], [5].

In addition to the energy consumed by the data access operation, a large portion of the energy consumption is due to the refresh operation, which involves periodically reading out the data and then written back into the same memory cells. This refresh operation is necessary because DRAM cells cannot retain the stored data permanently, and is performed for all cells in a DRAM, irrespective of whether they store significant data or not. Therefore, this refresh operation results in significant energy consumption even though certain DRAM cells do not store the data that are accessed by an active process in the processor. Furthermore, as the DRAM density increases, the energy consumed by the refresh operation also increases. The ratio of the refresh energy consumption to the total energy consumption of the DRAM increases in proportion to the density of the DRAM [6]. In future 64 Gb DRAM devices, the refresh operation is expected to account for up to 50% of the total energy consumption [6]. Therefore, the refresh energy consumption should be considered as one of the most critical parameters in computing system design.

Various techniques have been proposed to reduce the energy consumption in memory access and refresh operations [6]–[13]. In previous studies, [12] and [13], several cache compression algorithms were presented for reducing the memory accesses in order to achieve memory energy saving and system performance enhancement. However, these compression methods are sophisticated and require non-trivial cache block management. On the other hand, in three studies, [6]–[8], the operating system (OS) is requested by default to figure out the retention time information for each DRAM row, and a DRAM controller uses this information to selectively perform refresh operation for each row. These techniques significantly reduce the refresh energy consumption by skipping unnecessary refresh commands. However, with increasing memory sizes, it is neither cost-effective nor scalable to store the retention time information of all the rows of the DRAM. Moreover, profiling the retention time information of all DRAM cells requires a significant amount of effort, and incorrect results may be obtained, as it has to deal with Variable Retention Time and Data Pattern Dependencies [14].

Manuscript received July 17, 2019; revised November 15, 2019 and December 12, 2019; accepted December 12, 2019. Date of publication January 13, 2020; date of current version May 1, 2020. This work was supported in part by the Samsung Research Funding Center, Samsung Electronics, under Project SRFC-IT1602-03. This article was recommended by Associate Editor P. K. Meher. *(Corresponding author: Hyun Kim.)*

Duy Thanh Nguyen, Nguyen Huy Hung, and Hyuk-Jae Lee are with the Inter-University Semiconductor Research Center, Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea (e-mail: thanhnd@capp.snu.ac.kr; hungnh@capp.snu.ac.kr; hyuk_jae_lee@capp.snu.ac.kr).

Hyun Kim is with the Research Center for Electrical and Information Technology, Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, South Korea (e-mail: hyunkim@seoultech.ac.kr).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2019.2962516

Two studies, [9] and [10], proposed another approach to reduce the energy consumption by allowing the possibility of occurrence of a small amount of errors in DRAM cells. In [9], Flikker provides a software solution for the approximate memory by partitioning data into critical and non-critical data. It leverages the Partial-Array Self-Refresh Mode (PASR) [15], which supports different refresh rates for the different sections of the same DRAM bank. However, since all the bits of data are unprotected from errors, in the case of floating data, an erroneous most significant bit (MSB) may change the data to very large value, which causes overflow in computation. For providing a finer-grained refresh control than [9], in [10], Sparkk uses varying refresh periods for different bits based on their importance. However, it lacks in software support and does not provide energy measurement and performance evaluation based on architectural simulation. The concept of bit criticality is derived from approximate computing, and has been applied to SRAM cell design in order to optimize the energy consumption in video applications [16], [17]. Based on the bit importance analysis, the study in [11] proposed a precision-aware DRAM restore scheduling approach. This approach maps important data bits to fast row segments and unimportant data bits to slower row segments. The purpose of this mapping is to reduce the restore time in order to obtain improved performance at a low application error rate. This method requires the OS to store the restore profiling results for all the row segments in the DRAM. However, the problem with this method is that the DRAM restores characteristic changes randomly during DRAM operation [14].

To overcome the shortcomings of the previous studies, this paper proposes an approximate memory for deep learning applications with finer-grained refresh control. It is widely known that deep learning applications are error tolerant. To leverage this property, this paper proposes approximate memory architectures, which allow different error probabilities for different bits of data. Based on the analysis of the significance of each bit, the proposed architectures refresh data at bit granularity in order to reduce the refresh frequency for insignificant bits considerably. As a result, the refresh energy consumption is significantly reduced while the non-critical bits of data may cause errors due to the delayed refresh operations. In addition, to further reduce the energy consumption by memory accesses while maintaining accuracy and high performance, this paper proposes a new memory architecture based on hard approximation for deep learning applications. Hard approximation flexibly reduces the memory data words to reduce the memory bandwidth and the corresponding energy consumption. Different from low-bit quantization (such as 4-bit or 8-bit quantization), hard approximation does not require fine-tuning or re-training to preserve the accuracy of networks at the inference phase. Finally, a combination of soft approximation (i.e., slowing down refresh operations) and hard approximation (i.e., non-critical bit truncation) is presented. In a system that requires a large memory space and high memory bandwidth, a combined approach efficiently facilitates the reduction of both the static energy and dynamic energy, which cannot be solved by soft approximation or hard approximation alone. In order to evaluate the proposed approximate memory, this paper evaluates various aspects of the proposed approximate memory, such as energy breakdown, system performance, and accuracy degradation in some state-of-the-art deep convolutional neural networks (CNN). The simulation results with these CNNs in both the inference and training phases show that the proposed approximate memory can significantly reduce the refresh energy consumption by up to 69.71% with negligible degradation in accuracy. Accordingly, the total energy consumption for the proposed hybrid memory is reduced by up to 26.0%. These results show that the proposed memory with soft approximation and hard approximation is highly suitable for deep learning applications.

In particular, compared to the precedent research in [18], this paper has several improvements. In [18], the concepts of the transposed memory and row-level refresh algorithm are presented and the refresh power savings on CNN inference phase are derived mathematically. However, it lacks the detailed implementation of the bit transposed unit in the memory controller and the performance evaluation. Hence, there are no energy breakdowns and system performance of the transposed memory. On the other hand, this paper presents the design of the bit transposed unit, partitions the data of deep network applications into approximate data and non-approximate data, and runs multi-core architectural simulation for various design configurations. Moreover, this paper proposes a combination of soft approximation (i.e., slowing down the refreshment) and hard approximation (i.e., non-critical data truncation) to further reduce the energy consumption without performance loss. It also provides the system performance and energy breakdowns regarding the area/energy overhead of the bit transposed unit. Furthermore, the performance of the proposed scheme compared to the reduced precision version and how the proposed scheme effects the inference phase of sparse deep networks are thoroughly discussed. In addition, an inference test on the real approximate DRAM is conducted to verify the practicality of the bit injection model used in simulation. Finally, a training example using the approximate memory is added to see whether the approximate memory can be applied on training phase.

The rest of this paper is organized as follows. Section II presents the transposed approximate memory architecture, which is partly presented in [18]. Section III describes the proposed combination of hard and soft approximation schemes. In Section IV, the design of the bit transpose unit, which supports the approximate memory, is presented. Sections V and VI describe the evaluation methodology and the experimental results. Finally, the conclusions are provided in Section VII.

II. Approximate Memory Design

A. Memory Architecture

The main idea of an approximate memory is that data are stored in a DRAM in a different manner from that of the conventional way, as shown in Fig. 1. Fig. 1(a) shows the conventional storage scheme whereas Fig. 1(b) shows the proposed approximate memory scheme. In Fig. 1(a), data[0] = 11001010 is stored at the memory location addressed 0 while

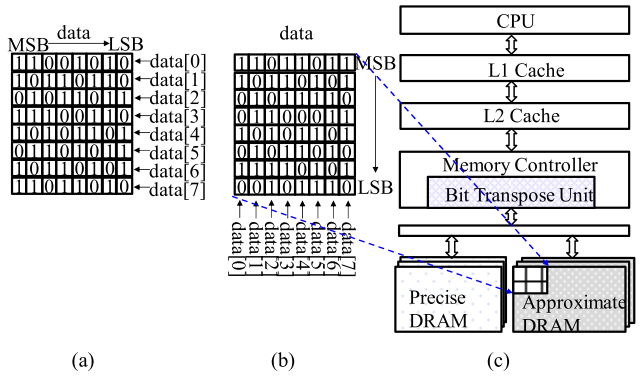

Fig. 1. Approximate memory architecture. (a) Conventional storage scheme. (b) Approximate storage scheme. (c) Hybrid architecture to support approximate memory.

data [1] = 10110110 is stored at address 1 and so on. As shown in Fig. 1(b), the approximate memory stores the data in a transposed manner such that all the MSBs from data[0] to data[7] are stored in the memory location at address 0. The second MSBs of data[0], data[1],…, data[7] are all stored at address 1, and so on. When address 0 is accessed by a host processor, only the MSBs are delivered to the processor.

One of the main goals of this data distribution is to give a different refresh rate (i.e., soft approximation) and truncate some data (i.e., hard approximation) depending on the significance of the row. In the example shown in Fig. 1(b), the data stored at address 0 are the MSBs that are, in general, the most important bits of data. Therefore, any occurrence of errors in these data may severely affect the outcome of computation using these data. On the other hand, the data at address 7 consist of the least significant bit (LSB) of data. Because these data may be relatively insignificant, a loss of these data may not badly affect the result. In a deep learning application for object classification, for example, the accuracy of the classification may not be significantly degraded. In this case, the refresh period at address 7 may be prolonged, which can reduce the energy consumption for refreshment, or the data at address may be truncated, which can reduce the energy consumption for data access. In this manner, the new data distribution shown in Fig. 1(b) allows the approximation to be controlled depending on the significance of data. In other words, for less significant data, the energy consumption is reduced by prolonged refresh period or data truncation, whereas for significant data, the data are preserved and the normal refresh period is maintained to avoid any occurrence of errors. The soft and hard approximations are explained in detail in Section III.

As mentioned above, for both soft and hard approximations, the transposed data storage shown in Fig. 1(b) requires the DRAM to be accessed in a blocked manner. To access one byte of data[0], a host needs to request the data stored from addresses 0 to 7. This means that the host needs eight memory accesses to access a single byte of data. Therefore, this memory architecture is inefficient when a single byte of data is accessed. On the other hand, this inefficiency can be avoided if a block of data is accessed together. For example, when a block of data from data[0] to data[7] is accessed together, eight data accesses are necessary for the eight bytes of data. Therefore, no unnecessary data request is required in this case. In order to support the data storage scheme shown in Fig. 1(b), a computer system requires a memory controller that converts the data format when it fetches the data from the memory and delivers them to the host. For this, a memory controller requires a hardware unit, called the "bit transpose unit" (shown in Fig. 1(c)), which is responsible for the data format conversion. The bit transpose unit is explained in detail in Section IV.

TABLE I
MEMORY FOOTPRINT PROFILING

Deep Networks	AlexNet	VGGNet	GoogLeNet
Proportion of approximate pages	99.78%	99.85%	99.67%

B. System Architecture Support for Approximate DRAM

The memory architecture shown in Fig. 1(c) is dedicated to deep networks computing systems. To leverage this approximate memory in a real system, a hybrid architecture is required. In the hybrid architecture, critical data such as instruction codes, integers, and non-approximate data are stored in the precise memory. On the other hand, non-critical data for the deep neural network (DNN) are stored in the approximate memory. The instruction set architecture (ISA), OS, and compiler support for systems with approximate memory are well described in [9] and [19]. For the experiments in this study, a custom memory allocator is built to allocate the approximate data to the approximate memory. Similar to that in [9], the critical data partitioning is done by the programmer. As mentioned above, in deep learning applications, if slight errors occur in the weights and feature maps, the performance might not degrade significantly. Therefore, they are categorized as non-critical data and can be stored in the approximate memory. This study uses the aforementioned memory allocator to allocate these data to the approximate memory. The memory footprint profiling results of some deep networks are listed in Table I, and they show that the portion of critical data in the total memory access is very small. Therefore, the refresh energy can be considerably reduced by using the approximate memory. However, the partitioning of application data potentially affects locality and parallelism. In this study, to enhance parallelism, a hybrid memory system may contain multiple channels or multiple ranks, out of which some are precise devices while the others are approximate devices, as depicted in Fig. 1(c).

III. PROPOSED SOFT AND HARD APPROXIMATIONS

In order to implement the approximate memory, the single precision floating-point format (i.e., FP32) is studied further in this paper, as it is the most popular format used in machine learning, especially for GPU computation [20]. It should be noted that each single precision floating-point data has 1 sign bit, 8 exponent bits, and 23 mantissa bits.

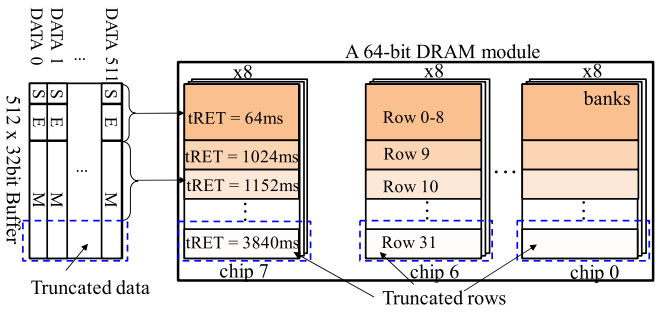

Fig. 2. Approximate memory with refresh control and data truncation.

A. Soft Approximation: Row-Level Refresh Control for Deep Learning Applications

The significance of each bit is important for designing the approximate memory. The sign bit is obviously the most critical bit. The bit errors in the exponent bits may result in a relatively large amount of error. For instance, a bit error occurring in the LSB of the exponent bits may double the value of the data. On the other hand, the bit errors at the MSB of the mantissa bits cause a relative small change in value. Therefore, in the single precision floating point data, (1) the sign bit and the 8 exponent bits are very critical and should be protected from bit error, because errors in these bits may significantly degrade the performance of the deep learning applications, (2) the mantissa bits are relatively non-critical as they may not affect the performance badly. For simplicity, only 32 rows per bank of a DRAM device are displayed in Fig. 2. As explained above, there are 9 significant bits (i.e., 1 sign bit + 8 exponent bits), which are refreshed at the normal rate (tRET = 64 ms) and stored in rows 0 to 8, respectively. The 23 mantissa bits are less important, and therefore they are stored in the approximate rows from 9 to 31. These rows are refreshed with prolonged periods. The period increases as the row number increases, which implies that the less significant bits are stored in rows with higher error probabilities. The refresh period (in milliseconds) increases linearly as the row number increases, as follows:

$$RP(n) = \begin{cases} 64 & for\ 0 \le n \le 8 \\ (n-9) * incr + offset & for\ 9 \le n \le 31 \end{cases} \quad (1)$$

where *RP(n)* represents the refresh period of the *n-th* row and the two parameters, *incr* and *offset*, are chosen experimentally. It should be noted that the *RP(n)* must be a multiple of 64 ms so that the proposed row-level refresh algorithm can be applied.

The adjustment of the refresh rate requires a slight change in the logics inside the DRAM device. For this, there are 23 additional counters for storing the current tRET round for 23 refresh rates (each tRET round is 64 ms) and additional logics for deciding whether the current row needs to be refreshed or not so that the DRAM device skips the refresh operation for the rows that are approximated. Since the maximum bit width of round counters is 6 bits, the additional gate count for the round counter is at most $6 \times 10 \times 23 = 1380$ gates, assuming that each bit needs 10 gates. The proposed row-level refresh algorithm is illustrated in Fig. 3. In the normal refresh operation of a DRAM device, a memory controller issues 8K Auto-Refresh commands. For a DRAM device having 8K rows per bank, one Auto-Refresh command refreshes a single row per bank in a refresh interval tREFI = 64 ms/8K = 7.8 μs. The proposed row-level refresh operation differs from the original Auto-Refresh in that there are additional logics, as shown in the shaded boxes and diamonds in Fig. 3. For (offset, incr) = (1024 ms, 128 ms), the operation of additional logics to decide which row needs to be refreshed can be explained as follows: Rows 0 - 8 are always refreshed regardless of the current tRET round. Row 9 (whose refresh period is 1024 ms or 16× tRET) is only refreshed when the tRET round is 15. Similarly, rows 10, 11, …, and 31 are refreshed when tRET is 17, 19, …, and 59, respectively. While the DRAM device skips the refresh operation for approximate rows, the external memory controller still sends 8K refresh commands periodically at the normal rate for every tRET round.

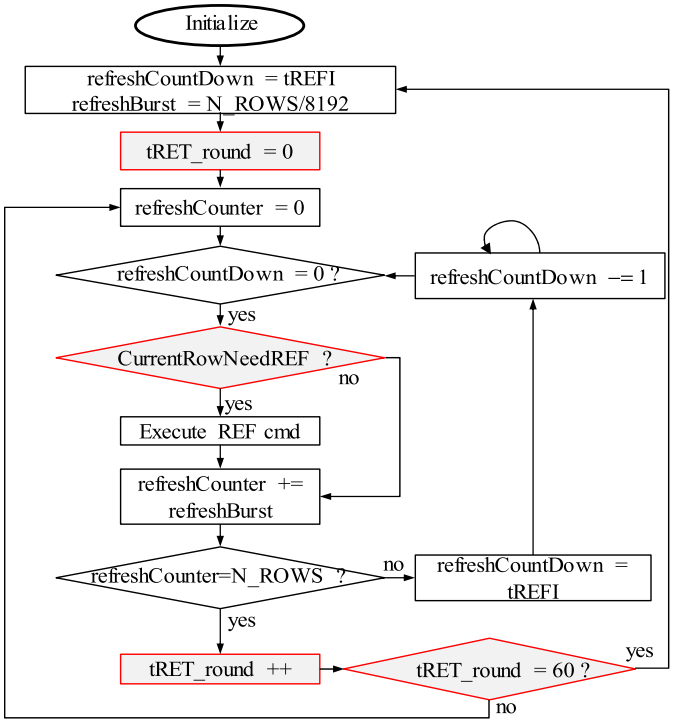

Fig. 3. Row-level refresh algorithm.

For applying the proposed row-level algorithm for soft approximation, some buffers are required for supporting the bit transpose, as shown in Fig. 2. The size of the buffer in the transposed unit is calculated as follows. To avoid additional data transfer, 64 bits of data are transferred together as a block. In order to support the data transfer, the transpose unit uses an additional buffer for temporarily storing the data to be transposed. Assuming that the burst length of a DRAM is eight (BL = 8) and the data width is 64 bits, a single data access to a DRAM transfers 512 bits. It is worth mentioning that all the data accessed by a single burst are stored in the same row in a single bank of a DRAM device. For the data to be stored into different rows according to their significance,

those 512 bits of data should represent the same significance of 512 different data items. To access all 32 bits of a data, 32 DRAM access commands should be issued. Therefore, the access of a single data item requires 32 DRAM commands which result in the data transfer of size 512×32 bits. This implies that the size of the buffer should be also 512×32 bits.

It should be noted that 32 commands to a DRAM demand a long latency. However, deep networks are normally composed of basic elements such as convolutional layer, pooling layer, and so on. Hence, these networks access data in a predictable pattern and consequently, the temporal locality of the buffer is leveraged to avoid performance degradation.

B. Hard Approximation: Block Data Truncation for Deep Learning Applications

The soft approximation can efficiently control the refresh energy consumption, but it cannot control the dynamic energy consumption due to memory data accesses. On the other hand, hard approximation enables a flexible control of the size of the memory data words, and thus reduces the dynamic energy consumption efficiently. If the size of the memory data words is reduced, both the memory bandwidth and the resulting energy consumption are reduced. Such hard approximation can be performed by data truncation. As shown in the above section, the mantissa bits of data are relatively non-critical, and are hence good candidates for truncation. Similar to soft approximation, insignificant data (i.e., LSBs of mantissa bits) can be truncated considering the trade-off between accuracy and energy saving, as shown in the blue boxes of Fig. 2. In terms of memory bandwidth and dynamic energy reduction, block data truncation helps in reducing the number of commands needed to access a block of data. For example, when 4 LSBs out of 32 bits of data are truncated, a 512×32-bit block is reduced to a 512×28-bit block. As a result, only 28 commands, instead of 32, are needed to access this block from the approximate memory. Thus, the memory bandwidth is reduced by 12.5%. It is noteworthy that the energy saving by hard approximation is as large as that by soft approximation because the dynamic energy dominates the total energy consumption for memory-intensive workloads [21].

The advantage of truncation design is that the same row-level refresh algorithm shown in Fig. 3 can be applied. This means that it can achieve the same amount of refresh energy reduction when the memory bandwidth and the corresponding dynamic energy are reduced. In other words, by increasing the refresh period, the energy saving can also be achieved for the data with intermediate importance, for which truncation cannot be applied. In the case of rows that are subjected to truncation, hard approximation can achieve additional refresh energy reduction since the refreshment of the rows containing truncated data can be turned off completely, although the refresh energy of these rows is relatively small (in case of no truncation) due to the increased refresh period.

C. Combination of Hard and Soft Approximation

As mentioned in the previous subsection, soft and hard approximations can be applied together for achieving higher energy savings. The first priority of the combination of these two approximation schemes is accuracy preservation. Hence, the goal of this subsection is to show how to combine hard and soft approximations for maximizing the energy reduction while preserving the accuracy of deep learning applications. Because the performance of hard approximation is the upper bound of the performance of the combined approximation design, the level of hard approximation is taken into account first. Experiments are performed for evaluating the effect of hard approximation by data truncation in DNNs. Pre-trained CNN models from Caffe library [22] are used in these experiments. The weights of CNNs are truncated by several bits, and then used for ImageNet classification [23]. Table II shows that the loss of accuracy due to data truncation varies for different CNNs. It should be noted that the accuracy in this table is normalized to the floating point model. The number of truncated bits increases until the error rate is recognizable, and 8-bit and 16-bit truncations are represented as examples in Table II. This table shows that data can be truncated by 18 bits with negligible loss of accuracy (i.e., within 1% loss) in GoogLeNet [24] and ResNet [3], whereas in VGGNet [25], data can be truncated even further (i.e., 20 bits) with very small loss of accuracy. Therefore, the truncation level can be larger than 16 bits on all networks while maintaining high accuracy. In that case, the size of a single-buffer reduces by half. The design of the bit transpose unit is presented in detail in Section IV. After hard approximation with the transposed buffer design, the soft approximation level can be optimized to maximize the refresh energy reduction without any change in accuracy.

TABLE II
NORMALIZED ACCURACY (%) OF CNNS WITH BIT TRUNCATION

Number of truncations	GoogLeNet	VGGNet	ResNet-152
8	100	100	100
16	99.84	100	99.8
17	100.16	99.92	100.45
18	99.19	99.83	100
19	98.2	99.66	96.71
20	94.54	99.16	83.51
21	78.49	94.4	41.08

From the normalized accuracy results in Table II, it is noteworthy that the combination of hard and soft approximation is necessary. Some bits in the LSBs can be truncated without compromising accuracy as shown in Table II. However, for the bits between the MSBs and LSBs, it is necessary to obtain power reduction effect through soft-approximation because they cause too much accuracy drop when they are truncated. Soft-approximation is based on the stochastic method and consequently, it has much less impact on the performance degradation than deterministic hard-approximation. For example, in Table II, 16-bits can be truncated with almost no performance degradation, but 21-bits truncation results in significant performance degradation on all networks. However, as shown in the later section (i.e., evaluation section), even if 16-bits are truncated and soft-approximation is additionally applied

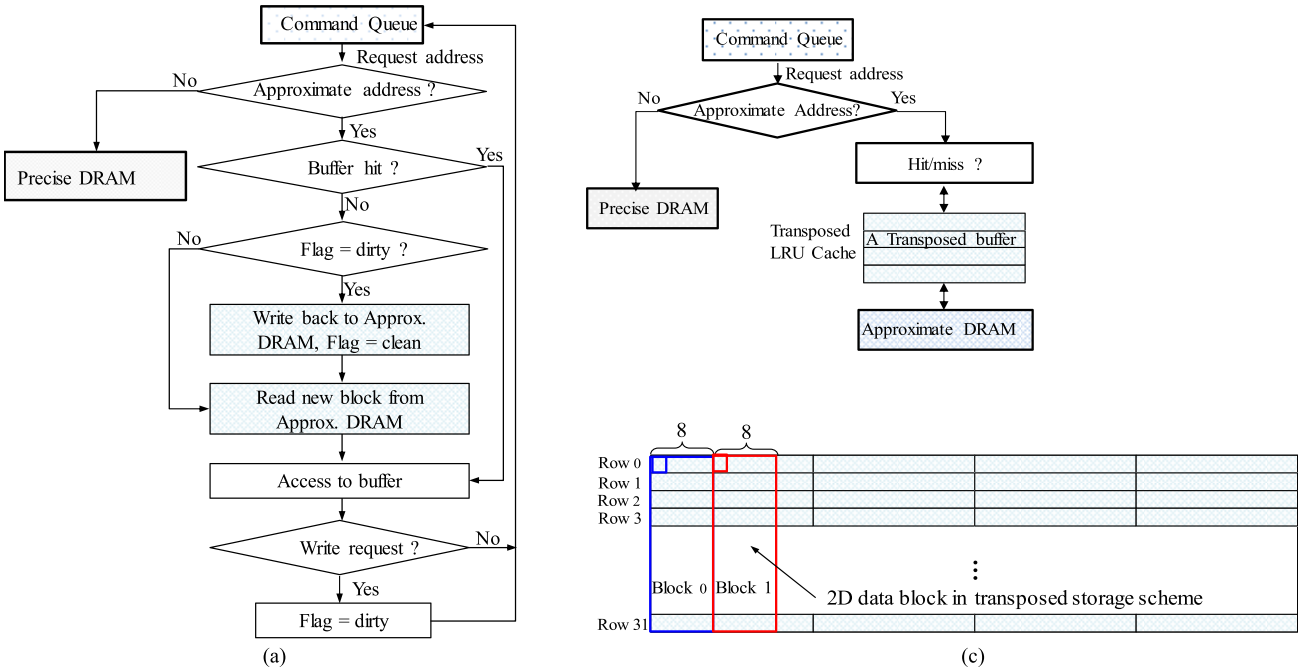

Fig. 4. Design of bit transpose units in the memory controller. (a) Single buffer design. (b) Multi-buffer design. (c) 2-D Block data in approximate DRAM.

to the rest of the mantissa bits (i.e., 7-bits), the performance degradation is negligible. It should be noted that if all of mantissa bits (i.e., 23 bits) are truncated, the performance degradation will be very severe. The appropriate level of soft approximation is detailed in Section VI.

IV. Design of a Memory Controller for the Proposed Approximate Memory

A. Design of the Bit Transpose Unit

This study proposes two kinds of bit-transpose unit designs, as shown in Fig. 4: (a) a single buffer design and (b) a multi-buffer design. The multi-buffer design is an extension of the single buffer design. The operation of the single buffer mode is described in Fig. 4(a). It operates like a large line of cache with a special hit or miss condition. When there is a read or write request from the cache to the command queue, the memory controller checks whether the address corresponds to an approximate space or a precise space. If the requested address is in the precise space, the memory controller sends access commands as in the conventional design. On the other hand, if the requested address is in the approximate space, the memory controller accesses the approximate DRAM. Firstly, the buffer hit/miss condition is checked. The hit/miss condition is explained in detail in Section IV-B. If the hit condition is satisfied, the memory controller just needs to access the buffer and return the data. In the case of buffer miss, a new block containing the requested data needs to be loaded to the buffer. Similar to a cache, the buffer includes a flag bit indicating whether this block is dirty or not. The block is dirty if it has been modified by the memory controller and has not yet been written in the approximate DRAM. If the buffer is miss and dirty, the block is flushed back to the approximate memory. Then, a new block is loaded to the buffer and the flag bit is cleared. On the other hand, if the buffer is miss and not dirty, the memory controller only needs to load the new block containing the requested data to the buffer. Finally, the memory controller accesses the buffer and returns the data. If the access is a write operation, the flag bit is set to dirty. It is noteworthy that it requires 32 commands to load a buffer from the approximate DRAM on a miss. This may cause long latency, and hence system performance degradation. However, thanks to the high access locality of CNNs, the design of the bit transpose unit leverages this property to alleviate the effect of long-latency block accesses.

To further improve the hit rate of the transposed buffer when running the CNN on multi-core system, this paper proposes a multi-buffer design as depicted in Fig. 4(b). The multi-buffer design operates as a Least Recently Used (LRU) transposed cache. Each cache line is a bit transposed buffer described in Fig. 4(a). When applying hard approximation, the size of the LRU transposed cache can be reduced 2×, thereby saving standby energy and area.

B. Data Mapping to Approximate Memory and Hiss/Miss Condition

This section discusses in detail how the transposed storage scheme is applied to the proposed approximate memory, and when an access request from the cache is considered as a hit or miss. This is critical for the correctness of the design of the memory controller with bit transpose unit. Fig. 4(c) shows how 512×32 bit blocks are stored in the approximate storage scheme. A DIMM consists of 8 DRAM devices, and therefore, each DRAM device stores 2,048 bits. Inside each DRAM device, a single byte of data are stored in 8 subarrays of a single bank. Therefore, each subarray stores 256 bits. The 256-bit data are stored in 32 rows depending on the significance of each bit. Thus, the 256-bit data are stored as shown in Fig. 4(c) such that each block size is 8×32. In this

TABLE III
SYSTEM CONFIGURATION

Parameters	Values
Processor	8 out-of-order cores 2.5 GHz, 8-issue per core, 128 instruction queue, 64 ROB size, ×86 ISA support
L1I, L1D cache	16 KB per core, 4-way associative
L2 cache	4 MB shared cache, 4-bank, 16-way associative
Memory Controller	Dual channels, open page, FR-FCFS, 64-entry queues per rank
DRAM	8 GB DDR3-1600 using 16 Gb ×16 Micron devices

figure, only the 32 first rows of a DRAM device are shown assuming that the DRAM device has 1024 columns per row.

The entire block is accessed together, and therefore, the address of the first element of the block can represent the entire block. When an access request is generated from the cache with the address denoted by *req_address*, the first element of the block including *req_address* has its address equal to *((req_address/8) & (~0xFF))*. The address is denoted by *req_addr_first* hereafter. Suppose that the buffer stores the data block of which the address of the first element is *req_addr_first*. Then, the condition of buffer hit is given as follows:

$$req\_addr\_first \leq req\_address \leq req\_addr\_first + 255 \tag{2}$$

In the case of buffer miss, *req_address* is converted to (*row_req_addr, col_req_addr*) which represents the top-left element of the data block. This conversion is carried out by a memory controller which generates 32 DRAM access commands in sequence to access the entire block. This address satisfies the following condition:

$$row\_req\_addr \times 1024 + col\_req\_addr \times 32 = req\_addr\_first \tag{3}$$

where *row_req_addr* and *col_req_addr* are multiples of 32 and 8, respectively. Therefore, (*row_req_addr, col_req_addr*) that satisfies (3) is obtained from the following equations:

$$\begin{aligned} row\_req\_addr &= 32 \times floor(req\_addr\_first/(32 \times 1024)) \\ col\_req\_addr &= req\_addr\_first/32 - 32 \times row\_req\_addr \end{aligned} \tag{4}$$

After deciding the row address and column address using (4), the memory controller sends 32 commands with the corresponding address to access a 512×32-bit data block that contains the requested data.

V. EVALUATION METHODOLOGY

A. Architectural Simulation Environment

To estimate the performance and energy consumption of the proposed system on deep learning applications, a cycle-accurate memory simulator called DRAMSim2 [26] is integrated into the event-driven McSimA+ simulator [27] based on Pin [28] binary instrumentation for achieving fast simulation and detailed results of the micro-architecture and DRAM memory subsystem [29]. The system configuration is described in Table III. The system has 8 out-of-order processor cores sharing the L2 cache, and each core has its own L1 instruction cache and data cache. A single memory channel is connected to an 8 GB Micron DDR3-1600 memory module consisting of ×16 16 Gb devices.

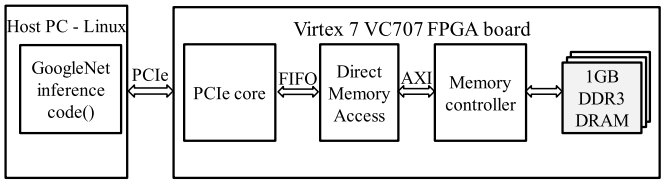

Fig. 5. Hardware platform for CNN testing on the real approximate DRAM.

The bit transpose unit is implemented inside of the memory controller of McSimA+. The memory controller of McSimA+ works as an interface between the two simulators. Its role is to send read/write requests to the memory controller of DRAMSim2. It should be noted that the access scheduling is done in the memory controller of DRAMSim2. The Verilog model of bit transpose unit is used to estimate the gate count with Synopsys Design Compiler. The bit transpose unit requires 16,384 bits resulting in a gate count of 243K gates. It consumes 0.918 mW of standby leakage power. According to [30], it consumes 1 pJ to access a 32-bit register file. Therefore, one 64-byte cache line access to the buffer consumes 16 pJ and one buffer write back or buffer load spends $16 \times 32 = 512$ pJ. These numbers of energy consumption are used to estimate the dynamic energy overhead of the bit transpose unit. It is noteworthy that the energy overhead of the bit transpose unit is added to non-refresh energy of approximate DRAM in Section VI.

B. Simulation of Deep Learning Inference and Training Phase With Bit Error Injection

The most recent research in [31] has shown that the retention time is not equal for all DRAM cells. Instead, most cells have high retention time (called strong cells) while only a few cells (called leaky cells) have low retention time. This research also shows that DDR3 DRAM cells can hold their values longer than 64 ms. Based on the bit error probability given in [31], the error rates for the custom refresh periods are linearly interpolated. For each bit in the approximate data, it is injected with an error having a corresponding error probability depending on its criticality.

To verify the practicality of the proposed DRAM error injection model, a platform is built as shown in Fig. 5. The memory controller in FPGA chip is customized to study the behavior of real approximate DRAM. The FPGA board runs at 200MHz and DDR3 DRAM module operates at 800 MHz. The role of the approximate DRAM on the FPGA board is to inject errors into the CNN model at run time. For each test, the CNN model is temporarily saved in the approximate DRAM for a refresh period, then copied back to the host PC to run the classification. The similar performances of the deep networks when using the real approximate DRAM and bit error injection are presented in Fig. 6. The experimental results show that the performance of the real approximate DRAM is

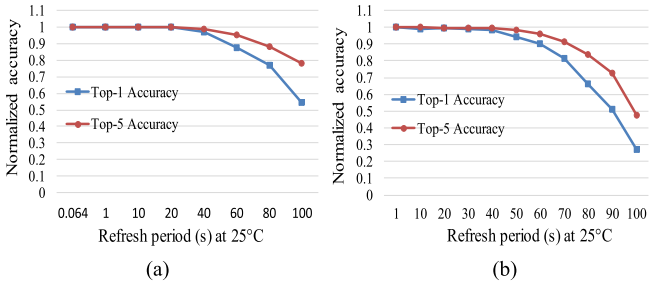

Fig. 6. Comparison of CNN test results on real approximate DRAM and bit injection simulation. (a) FPGA test. (b) Bit injection simulation.

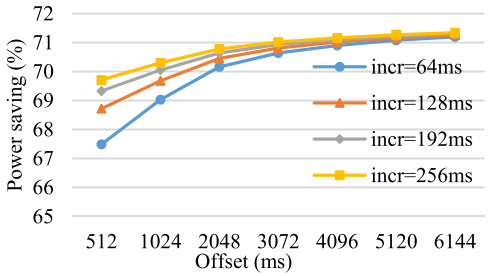

Fig. 7. Refresh power reduction with regard to (*offset, incr*).

slightly better than that of the error injection. This is because the error probabilities used in the bit injection are the worst-case values measured in the real approximate DRAM [31]. At room temperature (i.e., 25 °C), the refresh period can be prolonged to tens of seconds with very small performance loss.

This study uses the pre-trained model from the Caffe library [22] of deep networks, such as GoogLeNet [24] and VGGNet [25] for analyzing the accuracy effect of the approximate memory. In this paper, it is assumed that the CNN models are stored in the proposed approximate memory. For inference phase simulation, a set of 10,000 test images from ImageNet 2012 dataset [23] is used to measure the accuracy of the CNNs using the proposed approximate memory. The inference accuracy of each test is presented as Top-1 accuracy and Top-5 accuracy normalized to the FP32 without using memory approximation. The experiments are conducted with various (*offset, incr*) values under different working temperatures. In each test, the DRAM error map is randomly generated to simulate the time-dependent of DRAM bit error. Each test result is presented by a mean value and a standard deviation. For training phase simulation, a custom deep network, which includes three convolution layers and one fully-connected layer, is trained using Cifar-10 dataset [32]. In training large networks, such as GoogLeNet and VGGNet, the use of a huge dataset, e.g., ImageNet, consumes substantial amounts of time. On this account, the Cifar-10 dataset is selected in this study for training and testing, and the network is trained on the Caffe library using multiple step size for about 140 epochs.

VI. Evaluation

A. Performance and Refresh Energy Reduction

The DRAM energy consumption is calculated by following the method described in [33]. The refresh energy reduction is measured by calculating the portion of refresh commands that are skipped. For the normal working condition, the refresh period is 64 ms. Therefore, the reduction of refresh energy is derived mathematically from the following equation:

$$Psave = 1 - \frac{9 + \sum_{n=0}^{22} 64/(n * incr + offset)}{32} = 0.71875 - \sum_{n=0}^{22} \frac{2}{(n * incr + offset)} \quad (5)$$

where *n, incr,* and *offset* are the parameters used in (1). Fig. 7 shows the refresh energy reduction with respect to these parameters. The refresh energy reduction is almost saturated at about 71% when the offset is 4,096 ms. Thus, for the approximate device, the refresh period needs not be longer than 4,096 ms.

Large convolutional (CONV) layers (with 2.36 million parameters) and large fully-connected (FC) layers (with 75.5 million parameters) of VGGNet are selected for studying the effect of the proposed design on system performance. Figs. 8 and 9 show the results of multi-core simulation of the CONV layer and the FC layer, respectively. The baseline represents the results of the inference programs running on the conventional memory, whereas the others represent the results of the programs running on the proposed approximate memory with varying numbers of transposed buffers. The notation 1 BTU+16 represents a single buffer with 16-bit truncation (hard approximation). Experiments with a 16-bit fixed point model are also presented. As shown in Fig. 8(a), the single buffer (1 BTU) or double buffer (2 BTU) results in a low hit rate, thereby slowing down the system. On the other hand, as the number of transposed buffers increases (such as 4 BTU, 8 BTU, and 16 BTU), the hit rate also increases. Consequently, there is no instruction per cycle (IPC) loss, in comparison to the baseline. The hard approximation (16-bit truncation) and the 16-bit fixed point (int16) do not have significant impacts on the IPC, because in convolutional computations the portion of the DRAM accesses is relatively small compared to that of the computations. It is noteworthy that the hit rate is saturated when the number of buffers becomes larger than 16. Fig. 8(b) illustrates the DRAM energy consumption of the proposed design. One major factor that the effective energy reduction of the proposed design can bring is the refresh energy saving (soft approximation), as the DRAM dynamic energy is relatively small due to the high hit rate. The simulation results show that the design of the 16BTU+16 achieves 17% and 12.8% energy savings compared to the baseline and int16, respectively. It should be noted that one buffer access to the approximate DRAM causes 32 row misses. This potentially increases the energy overhead and degrades the performance. However, owing to small miss rate of the bit transpose unit, the proposed scheme does not degrade the system performance or energy efficiency.

Next, the impact of the approximate memory on the multi-core FC layer is illustrated in Fig. 9. In FC computations, the amount of memory accesses is considerably large (hundreds of MBs), compared to the CONV computations. Therefore, the IPC is very sensitive to the hit rate of the transposed cache,

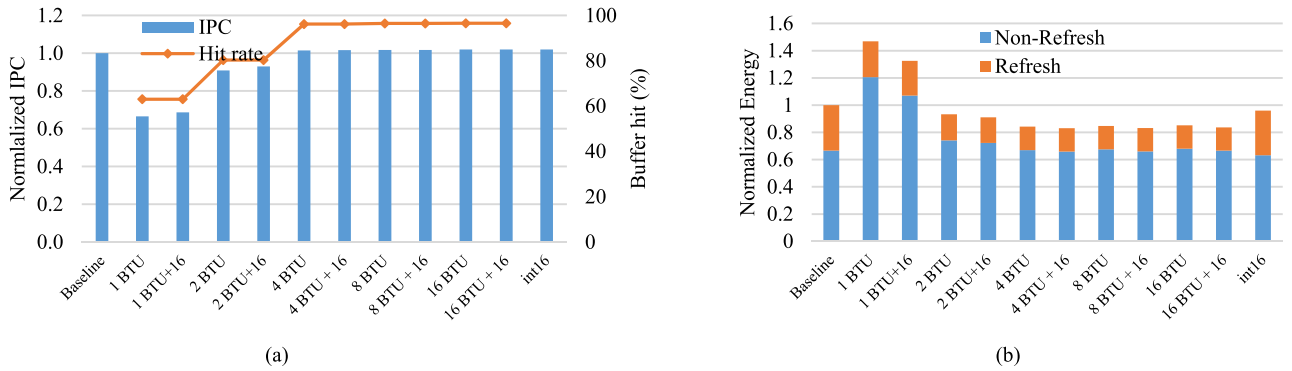

Fig. 8. 8-core simulation of CONV layers: (a) System performance. (b) DRAM Energy + Energy overhead of buffers.

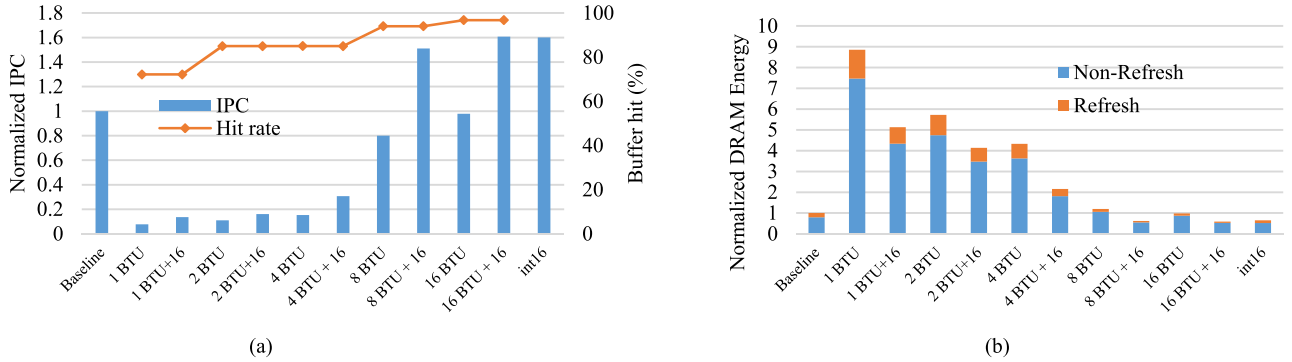

Fig. 9. 8-core simulation of FC layers: (a) System performance. (b) DRAM Energy + Energy overhead of buffers.

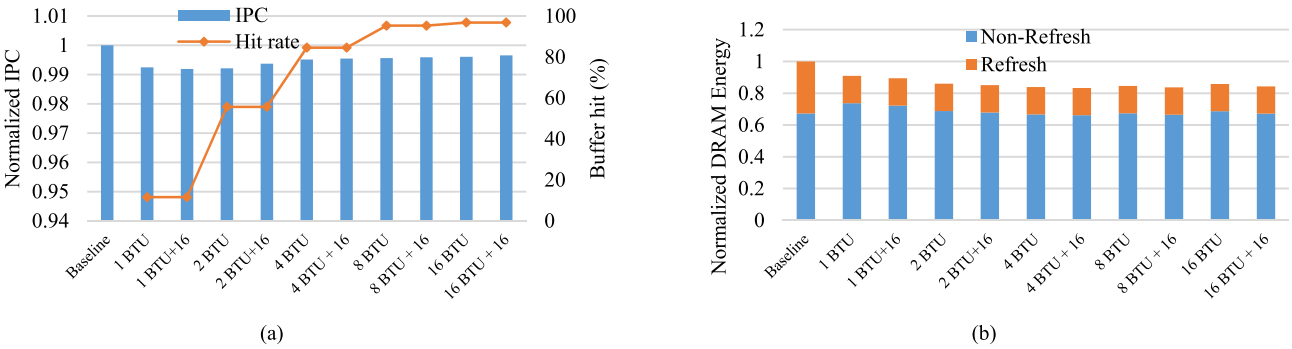

Fig. 10. 8-core simulation of sparse CONV layers: (a) System performance. (b) DRAM Energy + Energy overhead of buffers.

as shown in Fig. 9(a). The designs with 1, 2, and 4 buffers (i.e., 1 BTU, 2 BTU, and 4 BTU) degrade the IPC dramatically. On the other hand, when the number of buffers increases to 16 (16 BTU), there is almost no loss of IPC, even though hard approximation is not applied. It should be noted that similar to the CONV layer, the buffer hit rate gets saturated when the number of buffers is larger than 16. However, unlike the CONV layer, the IPC is significantly boosted up when hard approximation is applied. For example, the 16 BTU+16 design achieves 1.6× higher IPC compared to the baseline, owing to hard approximation. The breakdown of DRAM energy for the FC layer is depicted in Fig. 9(b). It can be seen that for the FC layer, the refresh energy accounts for a small portion of the total energy. The designs of 1, 2, and 4 BTUs cause a large energy overhead because of excessive DRAM accesses. However, the 16 BTU + 16 performs the best, saving 41% of total DRAM energy, compared to the baseline, due to both refresh skipping and memory access reduction. Compared to the 16-bit fixed point results, 16BTU+16 is still more energy efficient (approximately 1.1×).

In recent years, there have been extensive influential studies on sparse CNNs [34], [35] majorly preoccupied with reducing the computation, memory accesses, and energy consumption of the CNNs. Therefore, the next experiments are conducted for the purpose of studying the effect of the proposed approximate memory on the sparse CNN inference. According to [34], it is assumed that the sparsity of the CONV and FC layers is 60% and 94%, respectively. To represent the sparse matrices, a block memory is used to store non-zero weights, while another block memory is used to store the coordinates of non-zero weights. Weight blocks are stored in the approximate memory, while index blocks are stored in the precise memory. Figs. 10 and 11 demonstrate the practicality of applying the proposed scheme to the sparse CNN inference. The baseline is

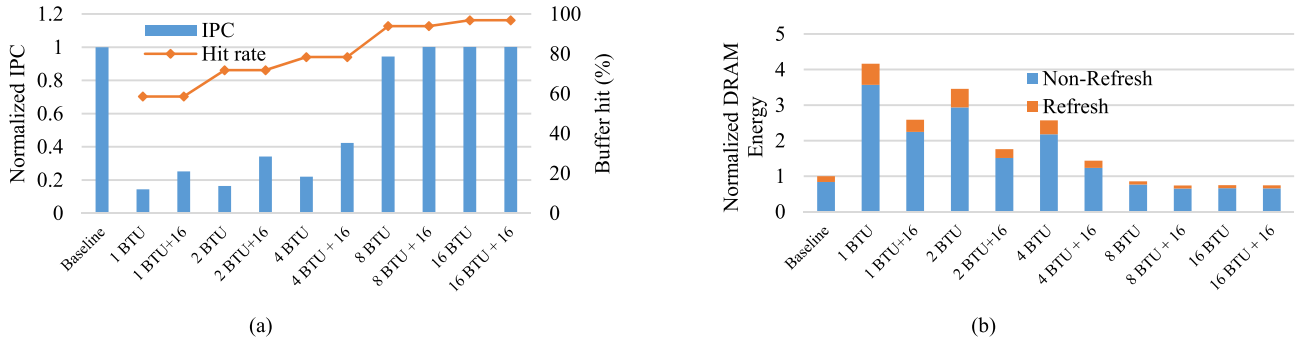

Fig. 11. 8-core simulation of sparse FC layers: (a) System performance. (b) DRAM Energy + Energy overhead of buffers.

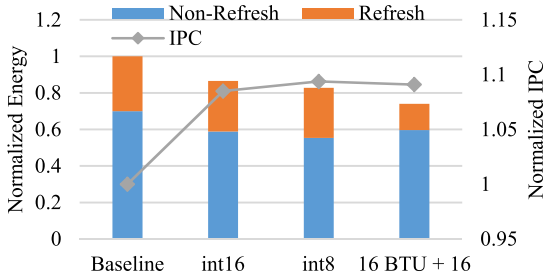

Fig. 12. 8-core simulation results of VGGNet.

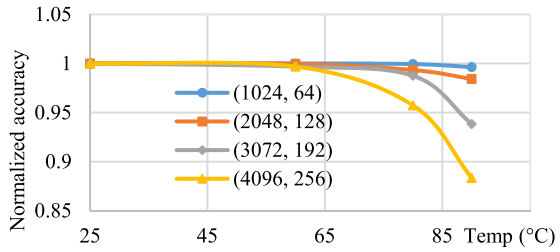

Fig. 13. Accuracy drop of GoogLeNet with regard to (*offset*, *incr*) and temperature.

the simulation result of sparse CNN layer on the conventional memory. Others are the simulation results of sparse CNN layer on various proposed schemes. In Fig. 10, the designs incorporating 4, 8, and 16 transposed buffers (4 BTU, 8 BTU, and 16 BTU) record almost no loss of IPC while reducing the total DRAM energy by 15.6%–16.7%. It should be noted that the dynamic energy saving achieved by applying hard approximation is relatively small, but the refresh energy saving is relatively critical. On the other hand, similar to the dense CNN inference, the sparse FC layer is more sensitive to buffer hit rate than the sparse CONV layer, as shown in Fig. 11(a). It achieves a high buffer hit rate when the number of buffers is 8 or 16. Because the sparse FC has aggressively eliminated parameters, the 16-bit truncation does not bring much benefit to the 16BTU design, compared to the dense CNN; however, the 16BTU+16 design still saves 26.3% of the total DRAM energy, as shown in Fig. 11(b).

To evaluate the performance of the proposed scheme on a whole network, the entire VGGNet, GoogleNet are simulated. As shown in Fig. 12, the proposed scheme for VGGNet is compared with an 8-bit fixed point (int8), 16-bit fixed point (int16), and the baseline running on the conventional memory. The IPC of the proposed scheme is similar to that of the fixed point formats, and 9% higher than the baseline. Although not shown in the figure, the IPC of the proposed scheme on GoogleNet is also 4.1% higher than that of the baseline. It should be noted that the number of instructions in the proposed scheme and that in the baseline are the same because the proposed scheme does not change the inference code of CNNs. Therefore, the comparison result of throughput between the proposed scheme and baseline is the same as that of IPC. Moreover, in the proposed scheme, non-refresh energy and refresh energy reduce the total energy by 10.4% and 15.6%, respectively. Overall, the proposed scheme saves 26.0% of the DRAM energy consumed by the baseline. Compared to the 8-bit fixed point network, the proposed scheme consumes 10.6% less energy. It is noteworthy that the energy overhead of the 16BTU+16 design accounts for 1.8% of the total energy of the proposed scheme.

Regarding the area overhead of the proposed scheme, the gate counts of supporting logics and transposed buffers are 2.59K and 243K, respectively. As a result, the area overhead of the proposed scheme with 16BTU+16 design is 0.93 mm^2 at TSMC 40nm technology. This overhead leads to a slight increase in power consumption, but the dynamic/refresh power reduction effect on the DRAM device through the proposed scheme is much greater, resulting in a significant reduction in overall system power. It should be noted that the proposed approximate DRAM structure only requires the small change in the memory controller with small hardware overhead mentioned above while keeping DRAM devices unchanged.

B. Simulation of Classification With Bit Error Injection

The experiments in [18] show that the accuracy changes of VGGNet and GoogLeNet caused by the delayed refresh are similar. Therefore, in this subsection, GoogLeNet is used to evaluate the accuracy performance of the proposed scheme. The accuracy drop of GoogLeNet with various combinations of (*offset, incr*) using the proposed approximate memory is shown in Fig. 13, in which the horizontal axis represents the operating temperature. At temperatures lower than 60 °C, the accuracy loss is negligible with all (*offset, incr*) pairs. However, the accuracy drop is relatively significant for temperatures higher than 60 °C.

Accordingly, the next simulation is conducted with the temperature set higher than 60 °C. The accuracy drops of GoogLeNet at high temperatures (i.e., 80 and 90 °C) are

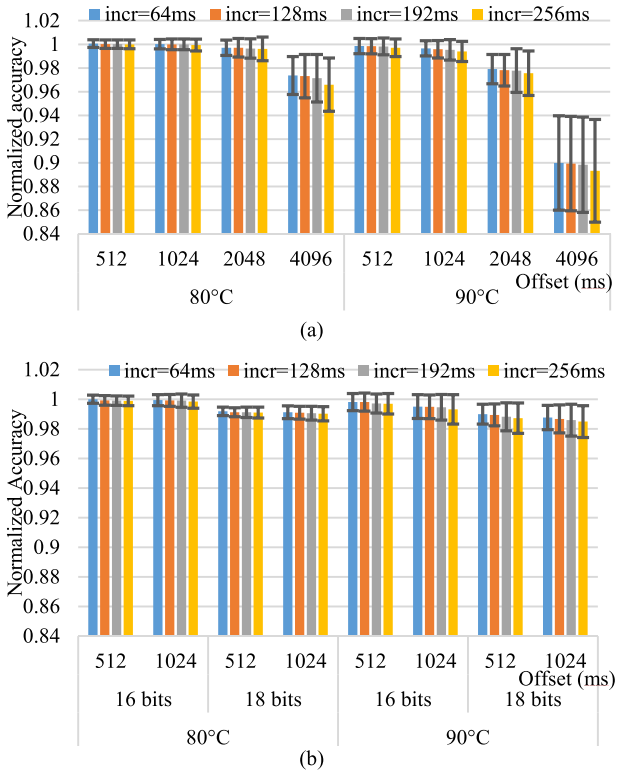

Fig. 14. Accuracy loss with regard to (*offset*, *incr*) at high temperatures. (a) Accuracy with no truncation. (b) Accuracy with truncation of 16 and 18 bits.

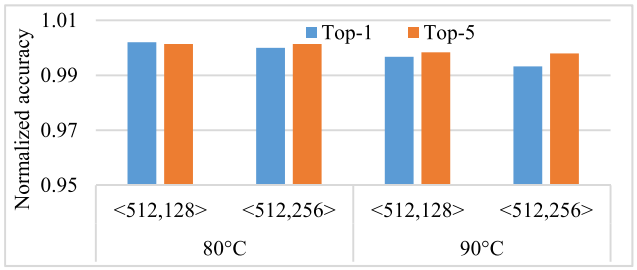

Fig. 15. Normalized accuracy of the proposed scheme on the compressed AlexNet.

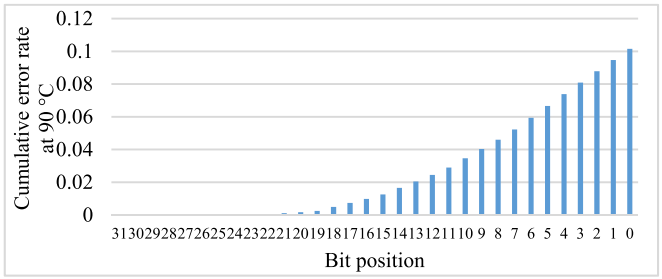

Fig. 16. Cumulative error rate of the proposed scheme for each bit of data at 90 °C.

shown in Fig. 14. The columns and error bars denote the mean values and standard deviations, respectively, of the tests for each (*offset, incr*) pairs. Fig. 14 (a) shows the accuracy drop with respect to various (*offset, incr*) without bit truncation. As shown in the figure, offsets of 2048 ms and 4096 ms degrade the accuracy significantly, especially at extremely high temperature. However, offsets of 512 ms and 1024 ms can preserve the accuracy well at any working temperature. For example, the accuracies of both (512, 256) and (1024, 256) are 99.71% and 99.4% with a standard deviation of 0.71% and 0.85%, respectively, at 90 °C. Because only the offsets of 512 ms and 1024 ms preserve the accuracy of the soft approximation, the next experiment shows the accuracy effect of combining hard and soft approximations with these offset values. Fig. 14 (b) illustrates the accuracy loss of hard approximation (16-bit and 18-bit truncation) in combination with soft approximation for the remaining MSBs. It should be noted that the level of bit truncation (i.e., 16-bit and 18-bit truncation) is determined by the analysis given in Section III-C. As can be seen in this figure, at 90 °C, an offset of 1024 ms causes a large standard deviation. For example, (1024, 64) achieves a high mean accuracy of 99.5 % with 16-bit truncation. However, the standard deviation is 0.8 %, which prevents the accuracy from satisfying the constraint (i.e., within 1% accuracy loss). On the other hand, an offset of 512 ms preserves the accuracy well with a small standard deviation for both temperatures. In addition, in the case of the offset of 512 ms, the increment parameter does not affect the accuracy significantly. Therefore, to achieve maximum refresh energy saving while sacrificing a small loss of accuracy, (512, 256) is selected because it can save 69.71% of the refresh energy (compared to conventional DRAM refreshing with a 64-ms refresh period), while maintaining a very high accuracy of 99.7% with a standard deviation of 0.69% at even very high operating temperatures such as 90 °C.

Further experiments are performed to show the compatibility of the proposed scheme with CNN compression techniques. The deep compression technique in [34] can compress AlexNet model by 27×. The proposed scheme is applied to this compressed AlexNet model to evaluate the compatibility of the proposed scheme with state-of-the-art network compression techniques. For this purpose, the compressed AlexNet model is truncated to 16-bits for hard approximation, and errors are injected to this truncated network with various (*offset, incr*) pairs at high temperatures, 80 °C and 90 °C, for soft approximation. Fig. 15 shows the performance of the proposed scheme on the compressed network. The experimental results show that both of (512, 128) and (512,256) pairs cause less than 0.7% of Top-1 accuracy loss at any working temperature. These results prove that the proposed scheme is well compatible with deep compression techniques.

It is noteworthy that although approximate DRAM can be efficiently used for only short lifetime data due to the error propagation, the proposed approximate DRAM structure is very suitable for DNN processing because data for DNNs are updated very frequently. For example, RANA [36] and DaDianNao [37] also use approximate embedded DRAM (eDRAM) as an internal buffer for CNN accelerator and thus, all the intermediate data are stored in eDRAM. This architecture can be implemented because the inference phase of DNN normally takes very short time (i.e., less than tens of miliseconds in AI dedicated accelerators) so new data are loaded before a new refresh cycle. Therefore,

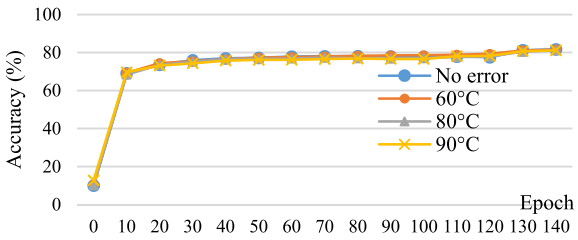

Fig. 17. Training CNN with (*offset, incr)* = (512, 256) and 16-bit truncation with regard to temperatures.

TABLE IV
TESTING ACCURACY OF THESE ABOVE MODELS

Trained w/o error	Trained at 60 °C	Trained at 80 °C	Trained at 90 °C
81.86%	81.36%	81.32%	81.14%

the accumulated errors of data can be kept small. Fig. 16 shows the cumulative error rates for each bit of data with (512,256) pairs at 90 °C. The experimental results show that although the error accumulates over time, the LSBs approximated by soft approximation can tolerate errors according to their criticality. In other words, in the proposed method, less important data have higher error rates. Therefore, thanks to the memory structure considering data significance and the fast new data load in DNN processing, the proposed structure can support DNN effectively.

C. Training CNN With Approximate Memory

As analyzed in the previous subsection, the operating option with (512, 256) performs best in terms of both error robustness and energy efficiency. Therefore, the simulation of the training phase with error injection is conducted with (offset, incr) = (512, 256). The training results are shown in Fig. 17, and the testing results are listed in Table IV. These results show that the training is also error robust and there is almost no performance loss (only about 0.7%) at even very high operating temperatures such as 90 °C in testing the models trained with the approximate memory. Training using the approximate memory is believed to be a potential way to avoid overfitting and to improve the stability of the network. The previous research in [38] injects the Gaussian noise to the input images for training the GoogLeNet to obtain a more robust model, whereas that in [39] identifies a very powerful technique, named Dropout, to avoid overfitting when training deep networks. However, further research is needed to acquire more insight into the regularization characteristic and error robustness of training with approximate memory.

D. Comparison With the Related Works

This study attempts a combined approach that greatly reduces the DRAM energy consumption by reducing both dynamic and refresh energy. Previous research efforts on the approximate memory through the refresh control are presented in [8]–[10], and [36].

The studies in [8]–[10] are applied for the embedded system setting. Sparkk in [10] maps different bits of data to

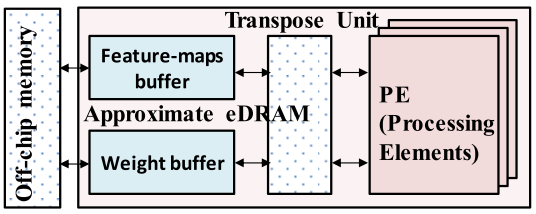

Fig. 18. Block diagram of an AI accelerator with the approximate memory.

different approximate DRAM chips based on their importance. However, since all bits of data are unprotected from errors, it works only for integer data (i.e., image pixels). Moreover, there are not enough implementation details or evaluations of energy consumption and architectural performance. In [9], the experiments show a slight reduction of 20–25% self-refresh power in the idle mode only. Another software technique proposed in [8] divides approximate DRAM into bins with different error rates and the data are allocated to a bin based on their criticality. At ideal room temperature, this scheme works well by reducing almost refresh energy. However, at higher temperature, the size of accurate bin (*qbin0*) reduces significantly. Moreover, profiling the retention time information of all DRAM cells once at the starting up could cause incorrect results, as it has to deal with Variable Retention Time and Data Pattern Dependencies [14]. It should be noted that wrong bin allocation could damage the program.

There have been many works, [37] and [40], that propose different methods to reuse data on chips for reducing DRAM access. As the DRAM access decreases by techniques for reusing data, the dynamic power decreases, so the portion of the refresh power relatively increases. Refresh operation should always operate regardless of DRAM access and therefore, the proposed scheme is compatible with many state-of-the-art technologies that reduce DRAM access and as a result, the effectiveness of the proposed scheme becomes more important. It should be noted that the effect from reducing DRAM access is similar to that from increasing the capacity of DRAM because the DRAM area that is not accessed increases. The research in [6] shows that the refresh power consumption accounts for up to 50% of the total memory power consumption as the size and density of DRAM increase. Moreover, DRAM power consumption has proven to be significant in both specific acceleration systems and general purpose systems. In the famous AI accelerator, DaDianNao [37], eDRAM buffer accounts for 38.3% of the power consumption in the overall system. In addition, in a general commercial server system with 8-core processors and 128 GB of memory in [41], DRAM power accounts for 41% of the power consumption in the overall system.

In addition, the proposed method brings some advantages to the dedicated AI chips. Similar to previous works, RANA [36] and DaDianNao [37], the feature-maps buffer and weight buffer of the accelerator are implemented by the eDRAM as shown in Fig. 18. The research in [36] shows that the refresh power consumption of the buffer accounts for 36.9% of the total power consumption of the accelerator. Therefore, the proposed row-level refresh algorithm can be applied to

these buffers to further save the power consumption. In addition, in AI accelerators, data are normally represented as fixed-point 16-bit (int16). It is noteworthy that fixed-point number is more error tolerant than floating point number. Hence, all bits of data in AI accelerators can be efficiently approximate depending on their criticality. Different from general purpose processors, the AI accelerator deterministically accesses to memory block by block in continuous addresses. Therefore, a single transpose buffer of 16×16 bits is sufficient without changing the eDRAM access latency owing to the high hit rate of the transpose buffer. Moreover, the study in [36] saves the refresh energy of the DRAM significantly while applying the same refresh period to all bits of data. Therefore, it is possible that the proposed refresh scheme further reduces the refresh power consumption with minimal hardware overhead over dedicated AI accelerators.

Finally, with regard to CNN compression and quantization techniques, there have been many previous works aimed at reducing the model size of CNNs [34], [42]–[45]. Unlike existing compression and quantization methods, the proposed technique has the advantage that it does not require any extra effort (i.e., fine-tuning, retraining, pruning, etc.), which causes additional power consumption, to preserve accuracy of neural networks. In other words, the original floating point model can be applied directly without any extra task. Moreover, existing quantization methods could reduce the bits for computation effectively, but could not finely control the bits from memory because data placement in memory follows the conventional way that is only suitable for standardized data precision, such as INT8, FP16, and FP32. On the other hand, the proposed memory structure uses the transposed method, which achieves the fine power saving effect on the memory bandwidth (i.e., dynamic power) as well as the refresh operation (i.e., static power) for various data precisions while maintaining the power saving effect in computing.

VII. Conclusion

This paper presents an approximate memory architecture that can aggressively save both the refresh energy and dynamic energy consumptions by applying the combination of soft approximation (i.e., row-level refresh control) and hard approximation (i.e., block data truncation) to non-critical data while preserving the accuracy of error-resilient applications such as DNNs. The experiments show that the approximate memory architecture is exceptionally energy-efficient in both training and testing phases.

References

[1] Z. Deng, C. Xu, Q. Ci, and P. Faraboschi, *Reduced-Precision Memory Value Approximation for Deep Learning*. Accessed: Jan. 3, 2020. [Online]. Available: http://www.labs.hpe.com/techreports/2015/HPL-2015-100.html

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 1, 2012, pp. 1097–1105.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," Dec. 2015, *arXiv:1512.03385*. [Online]. Available: https://arxiv.org/abs/1512.03385

[4] M. Courbariaux, J. David, and Y. Bengio, "Training deep neural networks with low precision multiplications," Dec. 2014, *arXiv:1412.7024*. [Online]. Available: https://arxiv.org/abs/1412.7024

[5] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

[6] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," in *Proc. 39th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2012, pp. 1–12.

[7] I. Bhati, Z. Chishti, S.-L. Lu, and B. Jacob, "Flexible auto-refresh: Enabling scalable and energy-efficient DRAM refresh reductions," in *Proc. 42nd Annu. Int. Symp. Comput. Archit. ISCA*, 2015, pp. 235–246.

[8] A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan, "Quality configurable approximate DRAM," *IEEE Trans. Comput.*, vol. 66, no. 7, pp. 1172–1187, Jul. 2017.

[9] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," in *Proc. 16th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2011, pp. 213–224.

[10] J. Lucas, M. Alvarez-Mesa, M. Andersch, and B. Juurlink, "Sparkk: Quality-scalable approximate storage in DRAM," *Memory Forum*, pp. 1–6, Jun. 2014.

[11] X. Zhang, Y. Zhang, B. R. Childers, and J. Yang, "DrMP: Mixed precision–aware DRAM for high performance approximate and precise computing," in *Proc. 26th Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, Sep. 2017, pp. 1–11.

[12] G. Pekhimnko *et al.*, "Linearly compressed pages: A low-complexity, low-latency main memory compression framework," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2013, pp. 1–13.

[13] S. Sardashti and D. A. Wood, "Decoupled compressed cache: Exploiting spatial locality for energy-optimized compressed caching," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2013, pp. 1–12.

[14] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms," in *Proc. 40th Annu. Int. Symp. Comput. Archit. ISCA*, 2013, pp. 60–71.

[15] Micron Technology, Inc. *Partial Array Self Refresh (PASR) TN*. Accessed: Jan. 3, 2020. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/technicalnote/dram/e0597e10.pdf

[16] I. J. Chang, D. Mohapatra, and K. Roy, "A priority-based 6T/8T hybrid SRAM architecture for aggressive voltage scaling in video applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 2, pp. 101–112, Feb. 2011.

[17] H. Kim, I. J. Chang, and H.-J. Lee, "Optimal selection of SRAM bit-cell size for power reduction in video compression," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 431–443, Sep. 2018.

[18] D. T. Nguyen, H. Kim, H.-J. Lee, and I.-J. Chang, "An approximate memory architecture for a reduction of refresh power consumption in deep learning applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.

[19] A. Samson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in *Proc. 32nd ACM SIGPLAN Conf. Program. Lang. Des. Implement.*, 2011, pp. 164–174.

[20] N. Whitehead and A. Fit-Florea. *Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs*. Accessed: Jan. 3, 2020. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.231.215

[21] H. Zheng and Z. Zhu, "Power and performance trade-off in contemporary DRAM system design for multicore processor," *IEEE Trans. Comput.*, vol. 59, no. 8, pp. 1033–1046, Aug. 2010.

[22] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, and J. Long, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimed.*, 2014, pp. 675–678.

[23] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[24] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, Sep. 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[26] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan. 2011.

[27] J. H. Ahn, S. Li, S. O, and N. P. Jouppi, "McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2013, pp. 74–85.

[28] C. K. Luk *et al.*, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proc. ACM SIGPLAN Conf. Program. Language Des. Implement.*, 2005, pp. 190–200.

[29] K. Bick, D. T. Nguyen, H.-J. Lee, and H. Kim, "Fast and accurate memory simulation by integrating DRAMSim2 into McSimA+," *MDPI Electron.*, vol. 7, no. 8, p. 152, 2018.

[30] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1135–1143.

[31] M. Jung, D. M. Mathew, C. C. Rheinlander, C. Weis, and N. Wehn, "A platform to analyze DDR3 DRAM's power and retention time," *IEEE Des. Test.*, vol. 34, no. 4, pp. 52–59, Aug. 2017.

[32] A. Krizhevsky, V. Nair, and G. Hinton. *The Cifar-10 Dataset*. Accessed: Jan. 3, 2020. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[33] *Calculating Memory System Power for DDR3*, Micron Technol., Boise, ID, USA, 2007.

[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *arXiv:1510.00149*. Oct. 2015, [Online]. Available: https://arxiv.org/abs/1510.00149

[35] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural networks," Feb. 2016, *arXiv:1602.01528*. [Online]. Available: https://arxiv.org/abs/1602.01528

[36] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "RANA: Towards efficient neural acceleration with refresh-optimized embedded DRAM," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 1–13.

[37] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. Int. Sump. Microarchitecture*, 2014, pp. 1–14.

[38] S. Zheng, Y. Song, T. Leung, and I. Goodfellow, "Improving the robustness of deep neural networks via stability training," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016.

[39] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

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

[41] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *IEEE Comput.*, vol. 36, no. 12, pp. 39–48, Dec. 2003.

[42] D. T. Nguyen, T. N. Nguyen, H. Kim, and H.-J. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1861–1873, Aug. 2019.

[43] M. Restegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," Mar. 2016, *arXiv:1603.05279*. [Online]. Available: https://arxiv.org/abs/1603.05279

[44] K. Wang, Z. Lin, Y. Liu, J. Lin, and S. Han, "HAQ: Hardware aware quantization with mixed precision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 8612–8620.

[45] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," Feb. 2017, *arXiv:1702.03044*. [Online]. Available: https://arxiv.org/abs/1702.03044

Duy Thanh Nguyen received the B.S. degree in electrical engineering from the Hanoi University of Science and Technology, Hanoi, Vietnam, and the M.S. degree in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2011 and 2014, respectively, where he is currently pursuing the Ph.D. degree in electrical and computer engineering.

His research interests include computer architecture, memory system, and SoC design for computer vision applications.

Nguyen Huy Hung received the B.S. degree in electrical engineering from the Hanoi University of Science and Technology, Hanoi, Vietnam, in 2012, and the M.S. degree in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2018.

His research interests include computer vision, deep learning, and GPU implementation for computer vision applications.

Hyun Kim (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2009, 2011, and 2015, respectively. From 2015 to 2018, he was a BK Assistant Professor with the BK21 Creative Research Engineer Development for IT, Seoul National University. In 2018, he joined the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, where he is currently an Assistant Professor. His research interests include algorithms, computer architecture, memory, and SoC design for low-complexity multimedia applications and deep neural networks.

Hyuk-Jae Lee (Member, IEEE) received the B.S. and M.S. degrees in electronics engineering from Seoul National University, South Korea, in 1987 and 1989, respectively, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 1996. From 1996 to 1998, he was with the Faculty of the Department of Computer Science, Louisiana Tech University, Ruston, USA. From 1998 to 2001, he was a Senior Component Design Engineer with the Server and Workstation Chipset Division, Intel Corporation, Hillsboro, OR, USA. In 2001, he joined the School of Electrical Engineering and Computer Science, Seoul National University, where he is currently a Professor. He is the Founder of Mamurian Design, Inc., a fabless SoC design house for multimedia applications. His research interests include computer architecture and SoC design for multimedia applications.