IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 11, 2020, accepted July 13, 2020, date of publication July 22, 2020, date of current version August 3, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3011265

McDRAM v2: In-Dynamic Random Access Memory Systolic Array Accelerator to Address the Large Model Problem in Deep Neural Networks on the Edge

SEUNGHWAN CHO[1], (Graduate Student Member, IEEE), HAERANG CHOI[1], (Graduate Student Member, IEEE), EUNHYEOK PARK[2], (Member, IEEE), HYUNSUNG SHIN[3], (Member, IEEE), AND SUNGJOO YOO[1,4], (Senior Member, IEEE)

[1]Department of Computer Science and Engineering, Seoul National University, Seoul 08826, South Korea
[2]Inter-University Semiconductor Research Center (ISRC), Seoul National University, Seoul 08826, South Korea
[3]Memory Division, Samsung Electronics, Hwaseong 18448, South Korea
[4]Neural Processing Research Center (NPRC), Seoul National University, Seoul 08826, South Korea

Corresponding author: Seunghwan Cho (seunghwan.cho21@gmail.com)

This work was supported in part by the Samsung Electronics and in part by the National Research Foundation of Korea under Grant NRF-2016M3A7B4909604 and Grant NRF-2016M3C4A7952587.

ABSTRACT The energy efficiency of accelerating hundreds of MB-large deep neural networks (DNNs) in a mobile environment is less than that of a server-class big chip accelerator because of the limited power budget, silicon area, and smaller buffer size of static random access memory associated with mobile systems. To address this challenge and provide powerful computing capability for processing large DNN models in power/resource-limited mobile systems, we propose McDRAM v2, which is a novel in-dynamic random access memory (DRAM) systolic array accelerator architecture. McDRAM v2 makes the best use of large in-DRAM bandwidths for accelerating various DNN applications. It can handle large DNN models without off-chip memory accesses, in a fast and efficient manner, by exposing the large DRAM capacity and large in-DRAM bandwidth directly to an input systolic array of a processing element matrix. Additionally, it maximizes data reuse using a systolic multiply–accumulate (MAC) structure. The proposed architecture maximizes the utilization of large-scale MAC units by judiciously exploiting the DRAM's internal bus and buffer structure. An evaluation of large DNN models in the fields of image classification, natural language processing, and recommendation systems shows that it achieves 1.7 times tera operations per second (TOPS), 3.7 times TOPS/watt, and 8.6 times TOPS/mm^2 improvements over a state-of-the–art mobile graphics processing unit accelerator, and 4.1 times better energy efficiency over a state-of-the-art server-class accelerator. Moreover, it incurs a minimal overhead, i.e., a 9.7% increase in area, and uses less than 4.4 W of peak operating power.

INDEX TERMS Accelerator, convolutional neural network, deep neural network, dynamic random access memory, edge inference, multi-layer perceptron, natural language processing, neural processing unit, recommendation system, transformer.

I. INTRODUCTION

The sizes of deep neural network (DNN)-based models in the fields of image recognition, natural language processing (NLP), and recommendation systems have been growing rapidly. In NLP tasks, massive language models such as bidirectional encoder representations from transformers (BERT) [1], Megatron-LM [2], and Turing-NLG [3] are being continuously introduced. The number of parameters used in the three aforementioned models is 340 million, 8.3 billion, and 17 billion, respectively. In the field of recommendation systems, the model sizes of embedding lookup tables have

The associate editor coordinating the review of this manuscript and approving it for publication was Xu Chen.

reached the order of tens or hundreds of MBs in mobile systems [4], and those of industry production-level recommendation systems running on cloud have also increased to the order of hundreds of GBs [5]–[9]. The number of parameters in convolutional neural networks (CNNs) such as ResNet [10], Inception [11], NasNet [12], and EfficientNet [13], which are widely used in image classification tasks, ranges from a few millions to hundreds of millions.

The inference stage of massive deep learning (DL) models commonly involves model compression using techniques such as quantization. Recent studies on quantization show that an inference using low-bit integer precision can provide sufficient accuracy, comparable to that acquired using a higher-bit floating point precision. As a result, numerous DNN accelerators support low-bit integer precision inference, taking advantage of the reduced memory footprint. However, the large memory footprint often far exceeds the size of the corresponding on-chip static random access memory (SRAM) buffer of the graphics processing unit/neural processing unit (GPU/ NPU) and continues to be one of the largest challenges in the acceleration of inferences of large neural networks.

A server-class accelerator, with an abundant power budget, adopts a large on-chip (tens of MBs) SRAM buffer, thereby enabling the maximization of ops/byte via buffering activation and weight data loaded from a dynamic random access memory (DRAM) cell. Furthermore, a server-class GPU/NPU is usually equipped with an energy-efficient and high-bandwidth DRAM (e.g., GDDR6 [14] and HBM2 [15]) to attain higher tera operations per second (TOPS)/W for the accelerator. However, the constant storage and retrieval of data by large neural networks is one of the limiting factors that has a dominant impact on the energy efficiency of DNN accelerators in servers.

In contrast, in a mobile environment, which has tight constraints regarding the power and silicon area budget, it is difficult for a mobile DNN accelerator to adopt either a large on-chip SRAM or an energy-efficient DRAM interface. When executing large DNN models with more than 100 million parameters, the mobile accelerator, equipped with a small on-chip SRAM memory (several MBs in the case of a commonly used mobile application processor) and having a small bandwidth of the conventional DRAM interface, e.g., the low-power double data rate synchronous dynamic random access memory (LPDDR4) [16], provides low energy efficiency. As shown in Fig. 1, mobile GPU/NPU accelerators, such as the Jetson AGX Xavier [17], Edge TPU [18], and McDRAM v1 [19], lag far behind the high-performing server-class DNN accelerators, with more than 10 TOPS of computing performance. This is true not only with respect to performance, but also with regard to energy/area efficiency when inference is performed using the ResNet-50 [10] and BERT base [1], where the numbers of parameters are 26 million and 108 million, respectively.

Recently, several studies on in-DRAM computation have been proposed to overcome the aforementioned limitation in

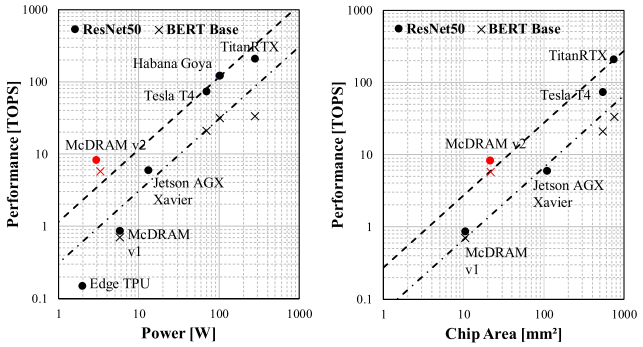

FIGURE 1. Energy efficiency and area efficiency of graphics processing units (GPUs) and neural processing units (NPUs). Power measurement excludes host CPU and host system power. Detailed conditions for comparisons are presented in Section VIII.A and VIII.B.

mobile accelerators; however, they have little efficacy in a real system, for the following three reasons.

The first reason is the low computing capability of the in-DRAM accelerator designs that have been proposed. Most of these accelerators have a low computing capability because they primarily focus on minimizing the silicon area overhead. McDRAMv1 [19], which exploits the internal mobile DRAM bandwidth, is expected to perform 1 TOPS when executing matrix multiplication, and the 3D-stacked hybrid memory cube (HMC) DRAM [20]-based Neurocube [21] and TETRIS [22] show less than hundreds of GFLOPS, i.e., far less than the performance of accelerators [23], [24] based on the HBM2 DRAM [15], a similar 3D-stacked DRAM type.

The second reason concerns area efficiency. DRISA [25], an in-DRAM DNN accelerator design supporting matrix multiplication composed of binary weight and fixed-point activation data, uses a considerable amount of area overhead to increase the computing performance.

The third reason is that some in-DRAM accelerators are not optimized for matrix-matrix multiplication, e.g., McDRAM v1 [19] and DRISA [25]. In these architectures, the number of reuses of data loaded from the DRAM cell is too low. When performing computation with a large amount of data reuse opportunities, especially in matrix-matrix multiplication, the energy efficiency of these architectures decreases owing to the frequent memory cell accesses. Given that many recent large neural network models are based on a transformer [28] or a convolution, both of which primarily involve matrix-matrix multiplications, in-DRAM computation which focuses primarily on matrix-vector multiplication limits the scope of application within a single-batched recurrent neural network (RNN)/multi-layer perceptron (MLP).

In this study, we address the aforementioned three challenges by proposing a DNN inference accelerator design that can be used on the DRAM subsystem, to achieve high performance and high energy/area efficiency. Our key contributions are summarized as follows:

- We propose an energy-efficient systolic array architecture for an in-DRAM accelerator to enable

matrix-matrix multiplication with a low area cost and to make the best use of the in-DRAM memory bandwidth. To the best of our knowledge, this work is the first architectural solution that adopts a 2D systolic array structure accelerator in a DRAM cell die for a DNN.
- We propose a novel dataflow for the judicious utilization of the existing in-DRAM bus to provide the systolic array accelerator with input data for matrix-matrix multiplication. The proposed dataflow enables the system to fully utilize the potential performance of the in-DRAM systolic array accelerator by exploiting the in-DRAM bandwidth, with which the weight data are accessed, the weight data reuse using a systolic multiply–accumulate (MAC) structure, and ultra-low precision.
- We extensively evaluate large DNN models with hundreds of MBs of parameters and embedding tables in language processing, recommendation system, and image classification tasks.
- We demonstrate that our proposed in-DRAM systolic array accelerator incurs only 9.7% area overhead and consumes less than 4.4 W of peak operating power. It offers 3.7 times better energy efficiency than the state-of-the-art commercial solutions such as the NVIDIA mobile GPU Jetson AGX Xavier [17], which consumes an operating power of 15 W. We also compare McDRAM v2 with the state-of-the-art server/mobile GPU and NPU solutions for various DNN workloads and demonstrate the effectiveness of our approach.

II. BACKGROUND

A. TARGET WORKLOADS: LARGE MODEL ON MOBILE DEVICES

Among the large modern DL applications that run in mobile environments, we consider workloads in the fields of image classification, NLP, and recommender systems for the evaluation of various DNN accelerators.

We selected the CNNs running on a mobile system as the image classification model or the backbone network for object detection, segmentation, and tracking. The following models were used: VGG19 [26], GoogleNet [11], and ResNet-50 [10], with 144 million, 7 million, and 26 million parameters, respectively. Furthermore, regarding the dataset, the ImageNet classification dataset [27] was used.

In a mobile device, the language processing model plays a key role in various applications, such as automatic speech recognition, chatbots, and virtual assistants. As the transformer [28]-based language models are being widely used as the de-facto standards in the recent years, we selected the BERT base and BERT large [1], which are 12-layer and 24-layer transformer models with 108 million and 334 million parameters, respectively. Among the total parameters of the BERT base and BERT large models, 24 million and 31 million parameters comprise word embedding tables, respectively.

With the recent increase in the awareness of privacy concerns [4], running recommendation systems on the edge has garnered attention. We selected the MLP-based NCF [29] (a form used in MLPerf [30]) and the transformer-based BERT4Rec [31] with the Movie-Lens 20 million dataset [32] (and its two expanded versions through fractal expansions [33] used in MLPerf [30]). In the case of NCF [29], the three workload configurations have 32 million, 180 million, and 590 million parameters, respectively; for all the three cases, 99 percent of the parameters are used for embedding tables. In the case of BERT4Rec [31], the three configurations have 10 million, 113 million, and 222 million parameters, respectively, and the parameters for embedding tables are 7 million, 110 million, and 219 million parameters, respectively.

The number of parameter of most of models used in this study exceeds tens of millions, which means they cannot fit in the on-chip SRAM buffer/cache of GPU/NPU of a conventional server/mobile device. Thus, when executing these models, they have to read the weight each time new input activation data are provided. This leads to low energy efficiency as a result of several off-chip DRAM accesses, as is demonstrated in this study.

B. LATEST QUANTIZATION RESULTS

Quantization is one of the most effective compression techniques used for reducing the weight and activation bit-width of a neural network model, thereby resulting in reduced model size and execution cost in terms of energy, time, and silicon area. The general matrix multiply (GEMM) operation is most commonly used to perform inferences for DNNs. Although CPUs and GPUs generally process this GEMM operation using floating-point 32-bit precision without applying a quantization technique, previous studies have proposed numerous methods for quantizing DNN models for a variety of tasks in a range of domains, and have successfully attained a nearly lossless inference model, using less than 8-bit integer (int8) precision.

For CNNs, inferences with int8 precision has been widely adopted in an increasing number of real-world GPUs [17], [34]/NPUs [35], [37], as int8 precision has been shown to be sufficient for GEMM operation in the inference of various CNNs in recent studies. It has also been found that sub-8-bit quantization of the inference of a CNN does not lead to significant accuracy loss. For example, with the ImageNet classification dataset [27], [38] reports an activation 4-bit and weight 4-bit quantization of an ResNet-50 [10] with a 0.2% top-1 accuracy loss compared to the case of FP32, in which an activation 2-bit and weight 2-bit quantization was done and a 3.2% top-1 accuracy loss was obtained.

Among the recent studies concerned with the quantization on BERT [1], a model widely used in the field of NLP, [39] presented the state-of-the-art results. For example, in tasks of sentiment classification, natural language inference, named entity recognition, and machine reading comprehension with benchmarks of the general language understanding

evaluation (GLUE) [40], Stanford Question Answering Dataset (SQuAD) [41], and CoNLL-03 [42], the activation 8-bit and weight 4-bit quantization precision yielded a comparable performance, without any significant quality degradation in the BERT model [1].

NCF [29], a traditional recommendation algorithm, implemented according to MLPerf [30] on the MovieLens 20 million dataset [32], has been shown to be quantized using int8 precision with a minimal loss of 0.5% in the hit rate [43].

In this work, to minimize the model's storage and memory access cost and to maximize the computing capability per unit silicon area, we aim to evaluate our in-DRAM acceleration using aggressive quantization inference models. For this reason, our proposed acceleration supports input precision over a range of int2 to int8.

III. MOTIVATION

A. NEURAL PROCESSING UNIT (NPU)'S LARGE MODEL PROBLEM IN MOBILE ENVIRONMENT

With regard to running a large neural network-based model on a GPU and NPU, the Roofline Model [44], which was used in our study for analyzing the performance bottlenecks, is a simple performance model, where the accelerator's real performance is determined by the machine's theoretical peak performance, peak bandwidth, and arithmetic intensity. Although the Roofline Model [44], as a simplified model, does not consider all the detailed features of the accelerators, it offers insights into designing a better NPU when running a large model that does not fit in the on-chip SRAM buffer/cache of the accelerator. To achieve higher performance within the tightly constrained power and silicon area budget of a mobile environment, these three factors have to be maximized; however, they encounter limitations for the following reasons.

1) MACHINE PEAK BANDWIDTH

When a DL model with a parameter size greater than the size of the on-chip SRAM buffer/cache (tens of MBs) is being processed, the DRAM bandwidth determines the peak performance of the memory-bound apps, as shown in Fig. 2. A high-bandwidth DRAM, which is bandwidth-optimized but has a high level of power/area consumption, provides the accelerator with hundreds of GB/s of memory bandwidth, but consumes more than 10 W of operating power. Thus, it is unsuitable for adoption in mobile systems with power budgets less than 10 W. Most power-limited mobile systems are equipped with low-power LPDDR4 DRAM [16] providing only tens of GB/s of bandwidth; consequently, adopting such a DRAM in a mobile system is a major challenge encountered when obtaining a high performance. Several highly energy-efficient NPU designs [45]–[47] proposed in the literature are not suitable for running large DNN models, because in these designs, all the parameters of the model are assumed to fit in the on-chip SRAM buffer.

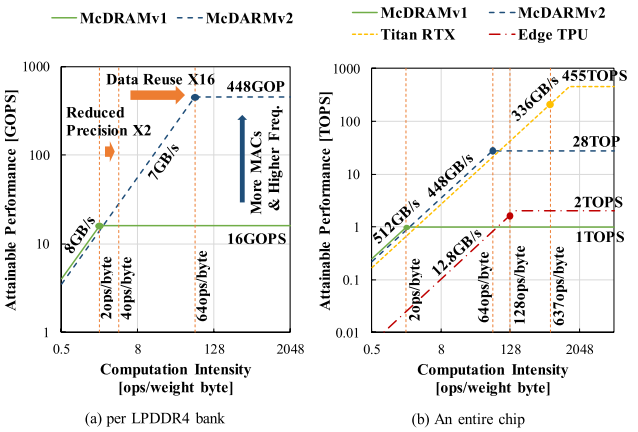

FIGURE 2. Peak performance analysis using the Roofline Model [44] for several GPUs and NPUs. For McDRAM v1 [19] and Edge TPU [18], int8 precision is used for general matrix multiply (GEMM) operations. For McDRAM v2 and Titan RTX's Tensor Cores [34], int4 precision is used. In the case of Titan RTX's Tensor Cores, we assume the number of data reuses, when loaded from DRAM, is 160, which is acquired by executing a 4096 x 4096 dimension GEMM on Titan RTX and inferring from the attained performance and memory bandwidth used.

2) COMPUTATIONAL INTENSITY

The quantization and reuse of weight data loaded from the DRAM cell can increase the computational intensity. High levels of power consumption and silicon area overhead are incurred when a large on-chip SRAM buffer/register file/cache is adopted or when a broadcasting bus or systolic array structure is employed, with which large-scale (e.g., more than 128 $\times$ 128) processing units can share data to maximize the number of data reuses. The server-class Titan RTX GPU [34], which is the fastest GPU at int4 precision, has approximately 30 MB of on-chip SRAM cache/register file/shared memory and a complicated datapath architecture for maximizing data reuse. Thus, it can obtain more than 500 ops/weight byte when it runs int4 matrix-matrix multiplications with dimensions of 4096. In contrast, within the context of power/area constrained mobile systems, Edge TPU [18], which is favorable for data reuse as it has 64 x 64 systolic array-based processing units and supports int8 quantization, can only acquire a maximum computational intensity of 128 ops/weight byte.

B. INTERNAL BANDWIDTH FOR COMPUTATION IN DRAM

The DRAM chip package used in mobile devices can provide data with an internal bandwidth ten of times larger than that of the off-chip bandwidth. The x64 4ch. LPDDR4 DRAM [16] package used in this study offers 32 GB/s of off-chip bandwidth, but 512 GB/s of internal bandwidth, when read/write is performed across all the DRAM internal banks. It is this massive internal bandwidth that is the most important characteristic when the accelerator is processing a large DNN application. In the following paragraph, we briefly describe the DRAM basic operation, along with the manner in which a DRAM chip is organized and the ways in which it could offer a large internal bandwidth to internal processing

units. We also present the major drawbacks of the previous in-DRAM acceleration solutions.

The x64 LPDDR4 DRAM [16] package is composed of eight DRAM dies, each of which has eight DRAM banks that share a data bus and I/O interface. As the data bus and I/O interface are shared across all the banks in each DRAM die, a read/write operation can be performed from/to a single DRAM bank at a time. Each bank consists of 16 burst length (BL) blocks; 16-bit data are read in parallel from each BL block for a single read command, and 256-bit data are read from a DRAM cell to local bit-line sense amps (BLSAs) as an entire bank. The DRAM has a 256-bit-wide internal data path from the BLSAs to the DQ buffers, and we assume this internal data path is split into the Path0 (cell–MAC) bus and Path1 bus (MAC–serializer-deserializer (SERDES)), via an intermediate buffer between them.

The 256-bit read data move parallel via the path0 bus to the intermediate buffer and then they move parallel through the path1 bus to the DQ output buffer. At the DRAM's output driver, the SERDES converts the 256-bit-wide data into 16 bursts of 16-bit-wide-data; then, this 16-bit data can be provided to the 16 DQ pins toward off-chip at every rising and falling edge of the 2133-MHz clock.

Although each bank can be read at a maximum speed of 8 GB/s (32 GB/s for a whole 4ch x64 LPDDR4 DRAM [16] package), if we assume that the path0 bus is not shared among banks and dedicated to all the banks, a concurrent all-bank burst read from the DRAM cell via the path0 bus to the intermediate buffer is possible, resulting in a maximum of 64 GB/s of internal bandwidth per die, and 512 GB/s for the entire LPDDR4 package.

McDRAM v1 [19] is a proposed in-DRAM acceleration architecture for enabling vector-matrix multiplication at 8-bit MAC precision; the internal bandwidth of the 4ch x64 LPDDR4 DRAM was fully utilized to provide weight data to the MAC units, and the off-chip bandwidth was fully utilized to provide activation data via the shared bus and I/O interface by way of broadcasting. However, for the 2 MAC units per BL block located in the column decoder region, the data reuse count was limited to 1; consequently, the benefit of exploiting the considerable internal bandwidth is substantially limited in terms of the computing performance, showing a peak performance of only 1 TOPS in the case of McDRAM v1 [19]. In addition, McDRAM v1 [19] had to process all computations except for int8 vector-matrix multiplication at an off-chip CPU.

C. OUR GOAL: BALANCED MOBILE NPU FOR LARGE DEEP NEURAL NETWORKs (DNNs): PERFORMANCE AND POWER/AREA OVERHEAD

As shown in the previous section, density/energy-optimized LPDDR4 DRAM [16] can offer data to its in-DRAM computational units with hundreds of GBs of internal bandwidth, similar to that of a bandwidth-optimized GDDR6 [14] or HBM2 [15] DRAM, which are mostly used in server- class accelerators. These findings suggest that an LPDDR4-based in-DRAM accelerator can offer high performance, energy efficiency, and area efficiency within a power budget of 10 W, even when DNN applications with over 100 million parameters are executed. One of the main challenges faced by the proposed in-DRAM accelerators [19], [25] with regard to performance, energy efficiency, and area efficiency is the limited computational intensity resulting from low levels of data reuse, despite the massive internal bandwidth. In this study, we propose an in-DRAM accelerator which can not only fully exploit the large DRAM capacity and internal bandwidth as in previous cases, but can also address the issue of low computation intensity, thereby resulting in a high-performing DNN accelerator with minimized power/silicon area overheads. In the following section, we will present the key ideas of our proposed in-DRAM accelerator design.

IV. OVERVIEW

In the Roofline Model [44], computation intensity is determined by two factors: the data reuse count and input precision of the computing units. Based on our deep understanding of the cell, bus, and buffer structure of the DRAM, we propose McDRAM v2, a high-performing and high-energy efficient in-DRAM DNN inference accelerator, to address the issue of low-computation-intensity. The proposed design aims to (1) fully enjoy the DRAM's large in-DRAM bandwidth and large capacity, which is several hundred times larger than that of the conventional on-chip SRAM, directly expose them to the in-DRAM processing units, and (2) minimize the number of power-consuming off-chip accesses. Some notable features of the McDRAM v2 design are described as follows.

First, the McDRAM v2 adopts the matrix MAC units with its input/output buffer connected in a systolic manner to increase the reuse count of data fetched from the DRAM cell. Although many DNN accelerators use systolic arrays in their buffer structure to provide massive amounts of processing units with the data, no other previous studies, to the best of our knowledge, have presented an architectural solution for the adoption of a 2D systolic array structure inside the DRAM cell die for large DNNs. Adopting a systolic array buffer structure increases the reuse count of data loaded from the DRAM cell, thereby reducing the burden on the memory bandwidth by a significant amount when large DNNs are employed.

Second, McDRAM v2 supports four input precision modes, namely int2-int4, int4-int4, int4-int8, and int8-int8. Consequently, it can adapt to the lowest possible quantization level for a range of DNN workloads. As mentioned in Section II.B, the lowest possible quantization level of various DNN workloads without significant quality loss varies across (1) domains of DL, (2) DNN models within the same domain, and even (3) layers within the same DNN model. Therefore, supporting multiple input precision modes in a single accelerator design provides an excellent opportunity for maximizing the computational intensity.

Finally, McDRAM v2 is designed based on the idea that all the weight and activation data are internally read and almost

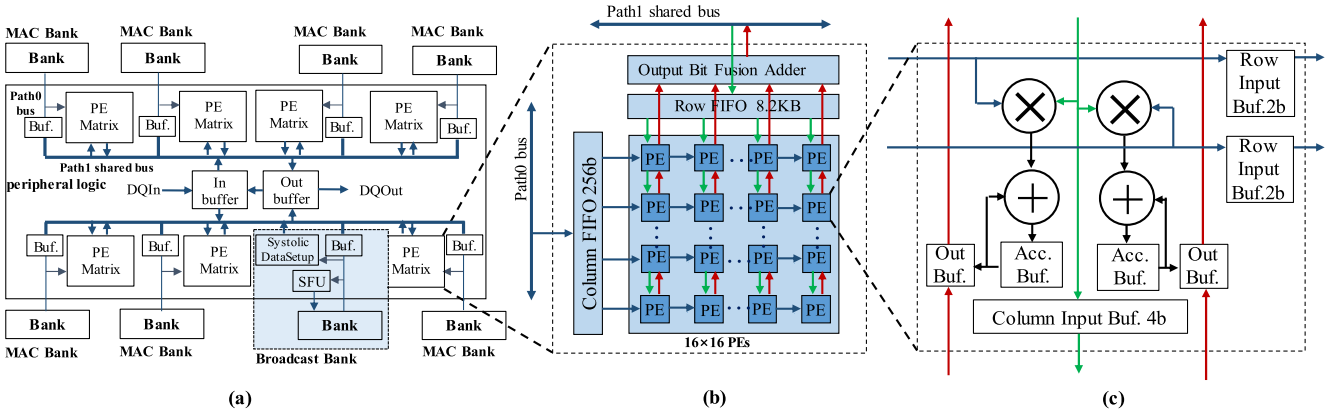

FIGURE 3. (a) Overall architecture of a single die of McDRAM v2, implemented on low-power double data rate synchronous dynamic random access memory (LPDDR4) die with additional logics for McDRAM v2; (b) systolic array-based "PE matrix"; (c) processing element (PE).

all the processing is executed within the in-DRAM boundary. The accelerator supports in-DRAM operation of not only matrix multiplication, but also the rest of the operations, including encoding/decoding of quantization, batch normalization, and embedding lookup/manipulation. Thus, it can exclude most of off-chip DRAM accesses, thereby making it possible to achieve high-level energy efficiency. Otherwise, it must send data to an off-chip CPU in the case of large DL workloads. Given that accessing a large embedding table located in an off-chip memory is one of the main challenges affecting the performance and energy efficiency of the accelerator when running a recommendation system [9], [48], McDRAM v2's internal embedding lookup and processing offer an excellent opportunity to obtain better performance and energy efficiency as compared to the other NPUs, which require sending/receiving data to/from an off-chip processor when processing embedding layers. Our proposed design of the accelerator can handle DNN applications of up to 1 GB of weight, embedding table, and intermediate activation space.

V. MCDRAM V2 ACCELERATOR DESIGN

Fig. 3 shows the proposed architecture of McDRAM v2 implemented on the LPDDR4 DRAM [16]. Most of the modifications are concentrated in the DRAM's peripheral area. At first, as shown in Fig. 3(a), among the eight banks per die, a bank is assigned as a "Broadcast Bank", from which the activation data are read and sent to the processing units. The rest of the banks are assigned as "MAC Banks," where the weight data are stored. Each MAC Bank has a single PE matrix with a 2D input/output systolic array buffer, and each Broadcast Bank has a systolic data setup unit and special function unit (SFU). During a read/write of McDRAM v2, the path1 bus can be shared between the banks, and the path0 bus is dedicated to each bank. For each bank to have their own path0 bus, we use additional silicon area for an additional bus interconnection, as adjacent banks share the path0 bus in the commodity DRAM.

A. MICROARCHITECTURE

In this section, the datapath structure of McDRAM v2 is described in detail. Additional logics for McDRAM v2 from the LPDDR4 DRAM [16] operate at 1 GHz, which is why we place these additional logic circuits in the DRAM's peripheral area, where a fast transistor is already available.

1) PROCESSING ELEMENT

As shown in Fig. 3(c), a single processing element (PE) is equipped with two integer multipliers, whose input precision is a 2bit-4bit pair, and two int16 precision accumulators. It has two horizontally connected 2-bit-wide input buffers, a vertically connected 4-bit-wide input buffer, and two vertically connected 16-bit-wide output buffers. It can perform two 2-bit/4-bit input MACs per cycle.

2) SYSTOLIC ARRAY-BASED PE MATRIX FOR MATRIX MULTIPLICATION

We adopt matrix placement for the PEs; the input/output buffer constitutes a systolic array, so that the 2-bit weight data can be moved horizontally, and 4-bit input activation data can be moved vertically in a systolic manner. Hereafter, we refer to this systolic array-based PE matrix as the "PE matrix." Once the input data are used in a PE, these data are sent to the next adjacent PE for the computation of the next cycle; thus, the input data reuse count is increased to the number of dimensions of the PE matrix. The PE matrix performs matrix multiplication in the output stationary manner, where the output of an inner product in a matrix multiplication is accumulated in a single PE, and the input data flow horizontally and vertically. Each PE matrix has an 8.2 kB row first-in-first-out (FIFO) buffer and a 256-b column FIFO buffer, for buffering activation and weight data from the Broadcast Bank and MAC Bank, respectively.

After the output computation is complete, the output data are moved vertically in the systolic manner, the same as in the input case, but in the opposite direction. The paths for the systolic input buffer and the output buffer are entirely

separate, which enables the concurrent input and output of data.

The features that distinguish McDRAM v2 from other famous systolic array-based accelerators, such as TPU [37], are as follows. The design of McDRAM v2 involves the adaptation of the systolic array, using an approach tailored for the in-DRAM context. First, it is important to note that the PE matrix dimensions for McDRAM v2 are not as large as those for other accelerators. Large dimensions can be advantageous in providing increased reuse counts in a systolic design; however, we adopt smaller dimensions to fit it within the bandwidth capacity of a single DRAM bank. Second, we implement multiple PE matrices and assign them to the MAC Banks in a per-bank manner. This is different from other systolic array-based NPUs, where a small number of large-dimension PE matrices are used.

As shown in Fig. 3(a), multiple PE matrices per MAC Bank are located between the dedicated path0 bus and shared path1 bus, to fully utilize both the large in-DRAM bandwidths from the DRAM bank via the path0 bus, and the shared broadcasting bandwidth from the DQ buffers via the path1 bus. This provides three advantages, which are as follows: (1) the input systolic array can be provided with weight data with a full in-DRAM bandwidth, (2) the MAC units can operate at frequencies higher (1 GHz) than 250 MHz when the units are near the cell area, and (3) the additional silicon area incurred by modifying the DRAM lies within the DRAM's peripheral area; therefore, the area of the DRAM cell is not affected, thereby minimizing the silicon area overhead incurred by the additional logic in McDRAM v2.

A single McDRAM v2 package is composed of eight dies. A single McDRAM v2 die can contain 1, 2, 4, or 7 PE matrices, depending on the configuration. When all the dies are operating at full PE utilization and each die has 7 PE matrices, the expected peak performance for the package is as shown in Table 1. When the number of PE matrices per die increases, the performance, power consumption, and silicon area overhead increase, and PE utilization decreases. To attain better power efficiency, this tradeoff is examined in detail in the experimental results section.

TABLE 1. MCDRAM v2 maximum performance by multiply–accumulate (MAC) precision.

	Int8-int8	int8-int4	int4-int4	int4-int2
# MACs/package	3584	7168	14336	28672
# dies/package	8	8	8	8
# MAC banks/die	7	7	7	7
# PE matrix/MAC bank	1	1	1	1
# MACs/PE matrix	64	128	256	512
MAC operation frequency	1 GHz	1 GHz	1 GHz	1 GHz
Max TOPS	7 TOPS	14 TOPS	28 TOPS	56 TOPS

3) OUTPUT BIT FUSION ADDER TREE

Each PE performs the MAC operations in an int2-int4 input precision, unlike the one supported in McDRAM v2 (int2-int4, int4-int4, int4-int8, int8-int8). To bridge the gap, we propose an output bit fusion adder tree for converting the MAC output of the PE into a matrix-matrix multiplication output of a PE matrix, immediately before the output data leave the PE Matrix toward the Broadcast Bank. As depicted in Fig. 4, the output data from the PE's systolic output data buffer are processed serially by a series of adders and bit-shifters and the complete output data are obtained with the intended precision for the matrix multiplication. The PE matrix contains eight output bit fusion adder trees. Once processed with an output bit fusion adder tree, the output data are concatenated in a granularity of 256 bits and then sent via the path1 bus.

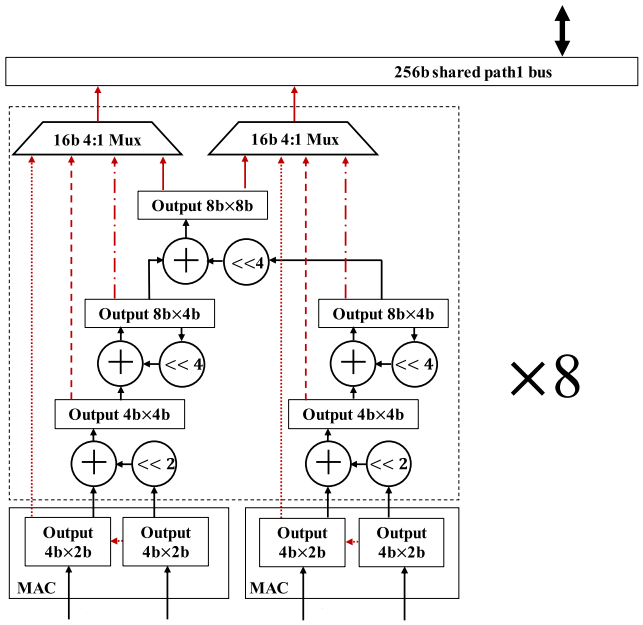

FIGURE 4. Structure of the output bit fusion adder tree.

4) SYSTOLIC DATA SETUP UNIT

Although the activation data are stored in the form of a rectangular matrix, the input systolic array of the PE matrix requires that this activation be rearranged for a systolic data flow in the output stationary manner. In the case of weight data, rearrangement can be performed during compilation, and the resulting data can then be stored in the MAC Bank. For this rearrangement, we propose a systolic data setup unit consisting of multiple shift registers sharing a parallel input. If the PE matrix dimensions are 4×4, the unit has four 4-bit-wide shift registers with a depth of 4. At first, as shown in Fig. 5, rectangular-shaped activation vectors are alternately loaded to a shift register through a parallel data input bus; then, the rearranged activation data are sent out via the serial data output port by four clock signals, with a single cycle of serial delay.

5) MULTILANE SPECIAL FUNCTION UNIT

McDRAM v2 supports on-chip processing of activation functions (ReLu, glue, sigmoid), (de-)quantization, embedding layers, normalization layers, softmax layers, and batch

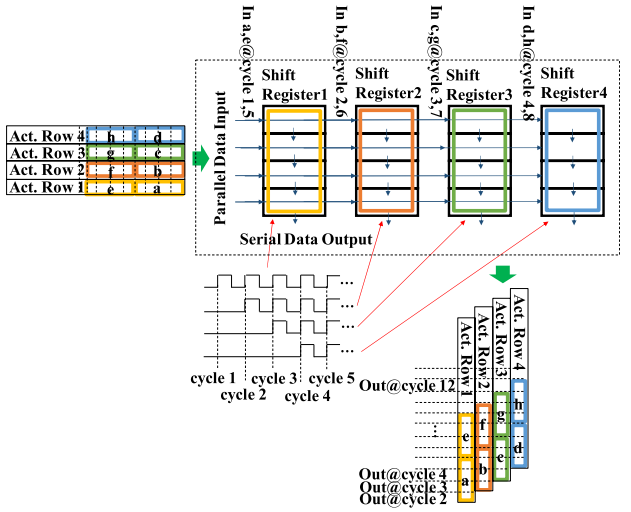

FIGURE 5. Activation processing in the systolic data setup unit.

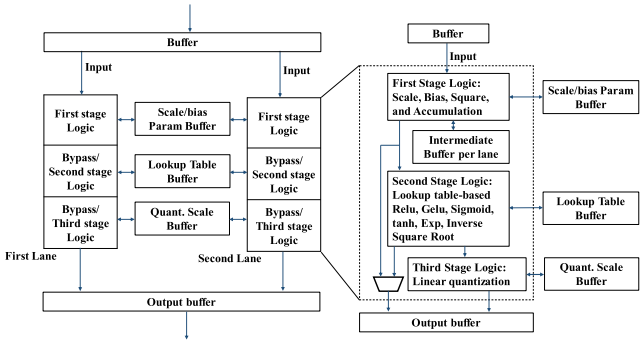

FIGURE 6. Functional description of multilane special function unit.

norm layers with int16 precision. To support these functions, McDRAM v2 is equipped with one three-stage multi-lane SFU per die, as shown in Fig. 6. Two lanes of SFU logic conduct scale/bias/average/rounding for vectors, element-wise addition/multiplication between vectors, and lookup table-based activation functions/math operations. The first-stage logic performs dequantization, scaling, and bias, and has an intermediate buffer for element-wise and average operations. The second-stage logic uses lookup tables to perform activation functions and math operations. The third-stage logic performs linear quantization, with an output precision that can be int4, int8, or int16. The output of each stage can be directly sent to the output buffer, thereby making various combinations of functions possible in the SFU.

B. MCDRAM V2 OPERATIONS

In this section, we describe the major processing operations of McDRAM v2, which are as follows: matrix multiplication, output saving, and embedding lookup and manipulation, along with its commands.

1) MATRIX MULTIPLICATION

Fig. 7 illustrates an example of a matrix-matrix multiplication that McDRAM v2 can perform. In the figure, for the sake of simplicity, we assume that there are 4 × 4 PEs in each PE matrix (for the experiments, we choose 16 × 16 PEs in each PE matrix). The weight matrix is stored in the MAC Bank, and the activation matrix is stored in the Broadcast Bank; both types of data are fed into the PE matrix near the MAC Bank. Each PE matrix performs matrix multiplication between the activation matrix from the Broadcast Bank and the weight sub-matrix from the MAC Bank. The multiple PE matrices in the die operate in a lock-step manner. As can be seen from the figure, a large matrix can be split into multiple rounds to fit the dimensions of the PE matrix and its count in the die.

The multiple rounds of operation are conducted as follows. As shown in Fig. 7, one out of four PE matrices performs the matrix multiplication between four broadcast activation rows (rows 1–4) and four weight columns (round 1). In this operation, the four broadcast activation rows are buffered in an 8.2 kB row FIFO for all the four PE matrices. Next, each MAC matrix loads another set of four weight columns (columns 17–20 in the case of PE matrix 1) from the MAC Bank cell, loads four activation rows from the row FIFO, and then performs matrix multiplication between them (round 2). The activations stored in the row FIFO of the PE matrix are reused in round 2. In round 3, new activations (rows 5–8) are loaded into each PE matrix, and the weight data are loaded from the MAC bank cell. The matrix multiplication of this round is performed similarly to round 1.

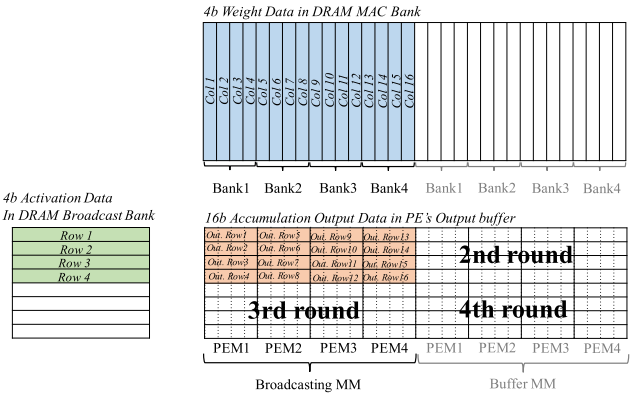

FIGURE 7. Examples of matrix multiplication, running on McDRAM v2. In this figure, we assume the number of MAC banks and PE matrices in the die to be 4 and 4, respectively, for the sake of simplicity without any loss of generality.

Fig. 8 illustrates the dataflow for each operation. The matrix multiplication can be conducted using two McDRAM v2 commands. The commands to be used depends on whether the activation has already buffered in the row FIFO of the PE matrix. The commands are as follows: Broadcasting_MM (when not in the row FIFO) or Buffer_MM (when buffered in the row FIFO) commands. Matrix multiplication is performed by executing a series of Broadcasting_MM commands or Buffer_MM commands.

a: BROADCASTING_MM

When the activation is not in the row FIFO in the PE matrix (rounds 1 and 3 in Fig. 7), the Broadcasting_MM command conducts the matrix multiplication.

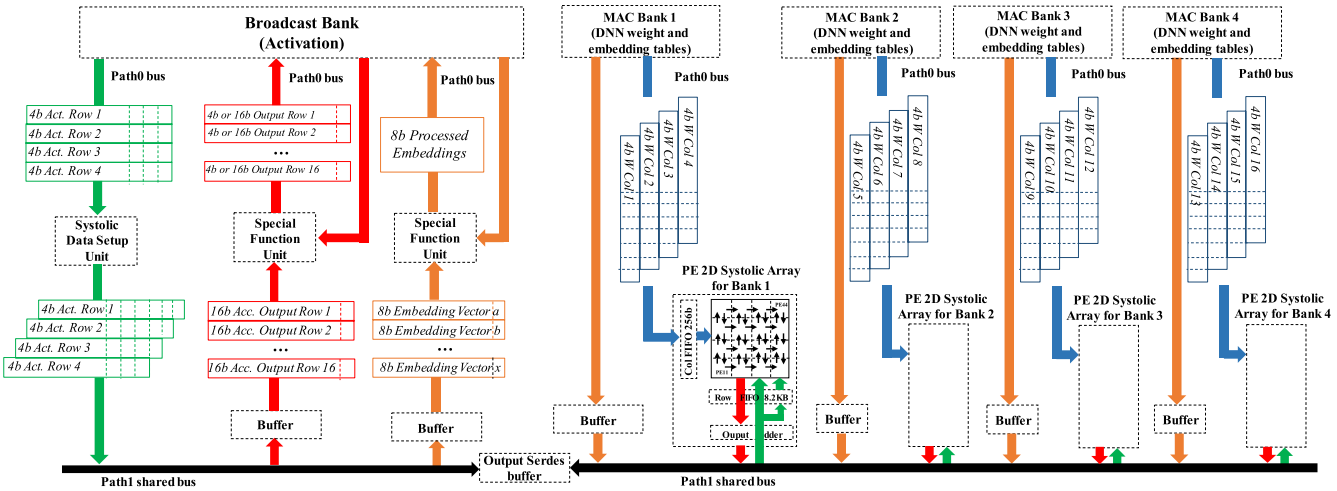

FIGURE 8. McDRAM v2 dataflow for matrix multiplication (blue for weight, green for activations), embedding lookup & manipulation (orange), and output saving (red). To match the dimension, in this figure, we assume the numbers of MAC banks and PE matrices are 4 and 4 per die, respectively, for the sake of simplicity without loss of generality.

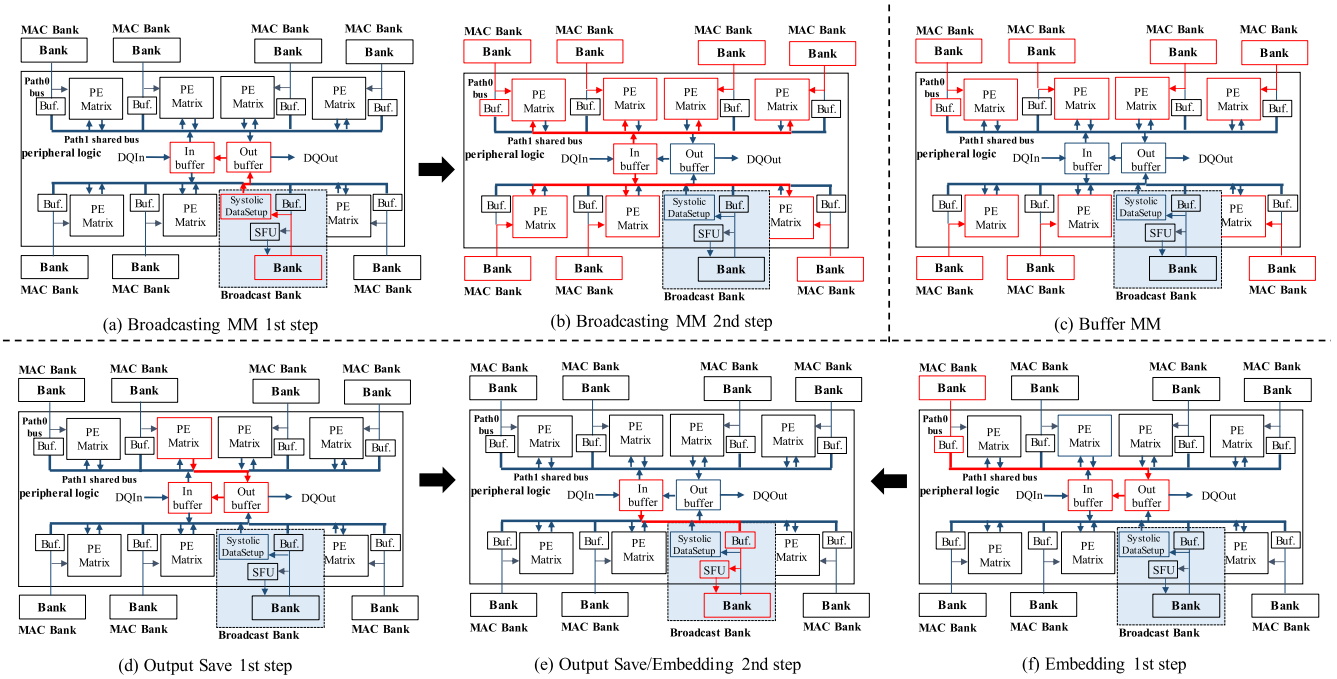

FIGURE 9. Occupancy of each component of McDRAM v2 for each step of operation: broadcasting/buffer matrix multiplication, embedding lookup & manipulation, and output saving. In this figure, we assume the numbers of MAC banks and PE matrices are 7 and 7 per die, respectively.

In the first step (Fig. 9(a)) of processing this command, 256 bits of activation data are read from the Broadcast Bank, moved via the path0 bus, and fed into the systolic data setup unit, where activation data rearrangement is performed, as described in Section V.A.4. Subsequently, these activation data are moved through the shared path1 bus to the output DQ buffer in the SERDES circuit in the I/O interface logic, and then copied into the input DQ buffer.

In the second step (Fig. 9(b)), these activation data are transferred via the shared path1 bus and broadcast vertically into the input systolic array of all the PE matrices. These activation data are also buffered in the row FIFO. Simultaneously, 256 bits of weight data are read from the MAC Bank cell via the dedicated path0 bus, into the horizontal input systolic array of each PE matrix. With 256 bits of both activation and weight data prepared in the input systolic array, the command performs operations on the MAC four times, and the input systolic array shifts across all the PEs in all the PE matrices. Each PE conducts MAC operations for the inner product in the output stationary manner and simultaneously shifts the input activation and weight data into the adjacent PEs in a horizontal and vertical manner using the input systolic array.

As can be observed in Fig. 9 (a) and (b) and considering that the minimum period of a command is determined based on the occupancy of the components related to the particular command, the occupancy of the shared path1 bus during both step1 and step2 is used to determine the minimal interval for the operation. In each step, the shared path1 bus is used in the opposite direction. During the burst read of the LPDDR4 DRAM, it takes RL $\times$ t_{CK} + t_{DQSCK} + t_{DQSQ} from the last rising edge of the clock of the read command for the first valid data to be ready in the DQ pins. As the internal data path consists of path0 (cell–MAC) and path1 (MAC–SERDES), we assume that a single burst read command occupies the path1 bus for 4 ns. Therefore, we assume that the Broadcasting_MM command has a minimum interval of 8 ns for a series of issues.

b: BUFFER_MM

When the activation data are already buffered in the row FIFO in the PE matrix (rounds 2 and 4 in Fig. 7), the Buffer_MM command conducts the matrix multiplication. The way the weight data are fed into the input systolic array of the PE matrix is the same as in the case of Broadcast_MM. Activation data are loaded from the row FIFO located in the PE matrix and fed into the input systolic array. The command performs four iterations of operations for all of the PEs and systolic arrays in all the PE matrices.

As can be seen from Fig. 9(c), the burst read of the DRAM has an interval of 8 t_{CK} (approximately 4 ns), which is the minimum interval required for the operation. As the Buffer_MM commands do not occupy the path1 bus, they can be issued concurrently with an Output_Save command, as described below.

2) OUTPUT SAVING

With the **Output_Save** commands, the output of matrix multiplication in a PE's systolic output buffer is saved back to the Broadcast Bank cells. This command comprises two steps, as described in Fig. 9(d) and (e). As observed in Fig. 8, only one PE matrix output can be saved at a time (the flow of data is indicated by red arrows).

In the first step, 16 bits of output data are shifted from the output systolic array and fed into an output bit fusion adder tree, where the partial output data are converted into the complete output of the matrix multiplication with the intended input precision. Then, the output data are concatenated into a granularity of 256 bits and moved via the path1 bus to the DQ buffer in the I/O SERDES logic.

In the second step, these data are loaded again to the path1 bus, but in the opposite direction, i.e., toward the DRAM cell. These output data then reach the SFU and undergo post-processing, such as scaling/biasing, activation functions, and (de)quantization. The processed data are then stored in the Broadcast Bank cell.

As depicted in Fig. 9(d) and (e), each step uses the path1 bus and this is why the interval of this command is 8 ns, which is the same as that in Broadcasting_MM.

3) EMBEDDING LOOKUP AND MANIPULATION

The processing of the embedding layers, found in NLP and recommendation systems, is illustrated in the Fig. 10. Unlike a Conv/FC layer, which is based on matrix multiplication, the embedding layer is processed in two phases.

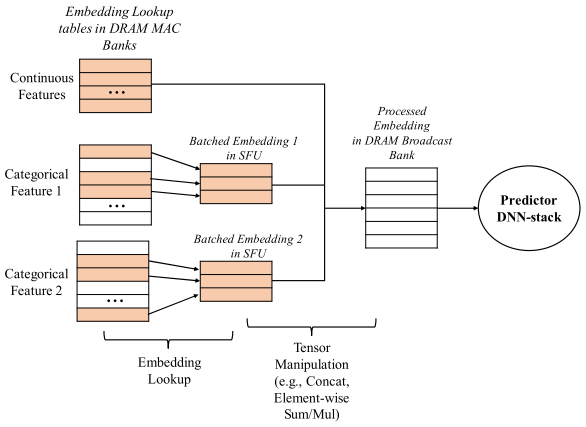

FIGURE 10. Examples of embedding lookup and manipulation running on McDRAM v2.

In the first embedding lookup phase, the embedding vectors are fetched by the DRAM random access from large embedding tables containing user/item/word embedding, to form a batched tensor of sparse embedding. The dataflow of this phase is indicated by orange arrows in Fig. 8; the embedding vectors are fetched from the embedding tables stored in the MAC Banks and moved via the DQ SERDES input/output buffers into the SFU. When conducting an embedding lookup, to exploit the DRAM's bank-level parallelism, the embedding vector is split at a granularity of 256 bits, i.e., the smallest size of a single LPDDR4 DRAM read operation, and the split chunks are stored across multiple banks. In this manner, both the row activation latency (t_{RCD}, from row activation to read/write) and pre-charge latency (t_{RP}, from pre-charge to row activation) can be hidden. When the embedding vector size is smaller than 256 bits, multiple copies of the embedding tables are stored across the banks.

In the second phase, the batched tensor of sparse embedding vectors loaded at the SFU undergoes tensor manipulations, such as concatenation and element-wise addition/multiplication/averaging, at SFU, to become an input vector of the following DNN stack.

The processing of the embedding layers is conducted using the **Data_Load_to_SFU** and **Run_SFU** commands, whose descriptions are listed in Table 2. As shown in Fig. 9(e) and (f), for the same reason as the Broadcasting_MM/Output_ Save commands, the combination of the two commands for embedding processing uses 8 ns as a minimum period.

C. MCDRAM V2 COMMANDS

We add ten additional commands for McDRAM v2 to the conventional LPDDR4 [16] commands. Broadcasting_MM,

TABLE 2. MCDRAM v2 commands.

Name	Description
Broadcasting_MM	Broadcasts activations from Broadcast Bank, internally reads weights from nearby MAC Bank, and then performs a matrix multiplication (Sec V.B.1)
Buffer_MM	Loads activations from row FIFO, internally reads weights from nearby MAC Bank, and performs a matrix multiplication (V.B.1)
Output_Save	Systolically pops out accumulation output from a single output systolic array, performs output bit fusion reduction, and moves this to SFU (V.B.2)
Data_Load_to_SFU	Loads data from the DRAM bank to SFU (V.B.3)
Run_SFU	Runs SFU (V.B.3)
Set_SFU	Sets register of SFU
Internal_Data_ Movement	Moves data from Broadcast Bank via systolic data setup unit to MAC Bank for operation of batched MM layer (V.C)
Clear_MM	Clears PEs
Clear_buffer	Clears buffer of PE matrix
Clear_SFU	Clears SFU

Buffer_MM, Data_Load_to_SFU, Run_SFU, and Output_Save commands are used for operations described in the previous section. The Internal_Data_Movement command is introduced for executing batched matrix-matrix multiplications found in the transformer network, and it performs data movement internally between banks, without off-chip access. Using this command, the activation data can be loaded from the Broadcast Bank via the systolic data setup unit to the MAC Bank. Table 2 describes the functions of the McDRAM v2 commands.

VI. SYSTEM INTEGRATION

In this section, we discuss the method of integrating McDRAM v2 into a current computing system. The McDRAM v2 accelerator is an in-memory computer architecture designed to be used as an LPDDR4-based McDRAM v2 package in mobile systems such as smart phones, IoT devices, autonomous vehicles, or mobile development boards. Implementation of McDRAM v2 is based on the LPDDR4 DRAM and is compatible with the control/address/data bus interface of its JEDEC specification [16]. In addition, as presented in Table 2, the memory controller of the accelerator is modified to support 10 additional McDRAM v2 command extensions for accelerating DNN applications. This accelerator has two operation modes: the memory mode, where it operates as a normal LPDDR4 DRAM, and the accelerator mode. In the following sections, we present the experimental results when it is running in the accelerator mode.

McDRAM v2 is controlled by the McDRAM v2 memory controller vis command/address (C/A) pins, as a completely passive device. The McDRAM v2 memory controller is modified from a LPDDR4 memory controller to support 10 additional McDRAM v2 commands and handle requests for the execution of DNN applications from host processors. A detailed modification of the LPDDR4 memory controller and host processor for supporting McDRAM v2 is out of the scope of the present study and is therefore reserved for future work.

The procedure for McDRAM v2 to perform DNN acceleration is presented as follows.

1) The McDRAM v2 compiler creates memory request streams and a host binary for McDRAM v2 from the input data and the DNN model extracted from a DL framework (PyTorch, in this study) (Fig. 11.①). The memory request consists of a layer-wise model specification and addresses of the location of the weight and activation data, along with the configurations of the McDRAM v2 hardware.
2) The generated McDRAM v2 memory request streams are stored in the DRAM cell along with the weight and activation data, using conventional DRAM write operations (Fig. 11.②). Like most memory management unit-less NPUs, McDRAM v2 uses the physical address when accessing the DRAM cell.
3) To initiate DNN acceleration, the user runs the host binary, which loads the McDRAM v2 memory request streams from the DRAM cell to the memory request queue of the memory controller (Fig. 11.③).
4) The McDRAM v2 memory request is loaded in the request queue and then fed into the McDRAM v2 command generator, where it is converted into McDRAM v2 commands. The generated commands are saved in the command queue (Fig. 11 ④).
5) The McDRAM v2 command scheduler schedules the McDRAM v2 commands in proper order and timing, while meeting all the time constraints of the LPDDR4 DRAM (Fig. 11.⑤).
6) Next, the memory controller sends commands to the McDRAM v2 device and runs commands in it (Fig. 11.⑥).

VII. EVALUATION METHODOLOGY

A. WORKLOADS

For evaluation, we used the target workloads and datasets mentioned in Sec II.A. For the ResNet-50 [10], BERT base/large [1], NCF [29], and BERT4Rec [31], it is assumed that the McDRAM v2 is used in an energy-efficient server environment. When comparing McDRAM v2 with mobile accelerators, we used the VGG19 [26], GoogleNet [11], and ResNet-50 [10]. We applied quantization only to the input data of the GEMM operations, and the remaining operations were executed with higher computing precisions (fixed-point 16-bit in the case of McDRAM v2). The quantization levels used in the experiment across all the target DNN workloads were guided by previous works on quantization and determined such that quality loss in the DNN applications was insignificant. We applied suitable quantization levels to each DNN workload: w4a4 for VGG19 [26] and GoogleNet [11], w2a4 for ResNet [10], w4a8 for BERT base/large [1], and w8a8 for NCF [29] and BERT4Rec [31]. In the case of ResNet [10], we performed a w2a4-quantization of

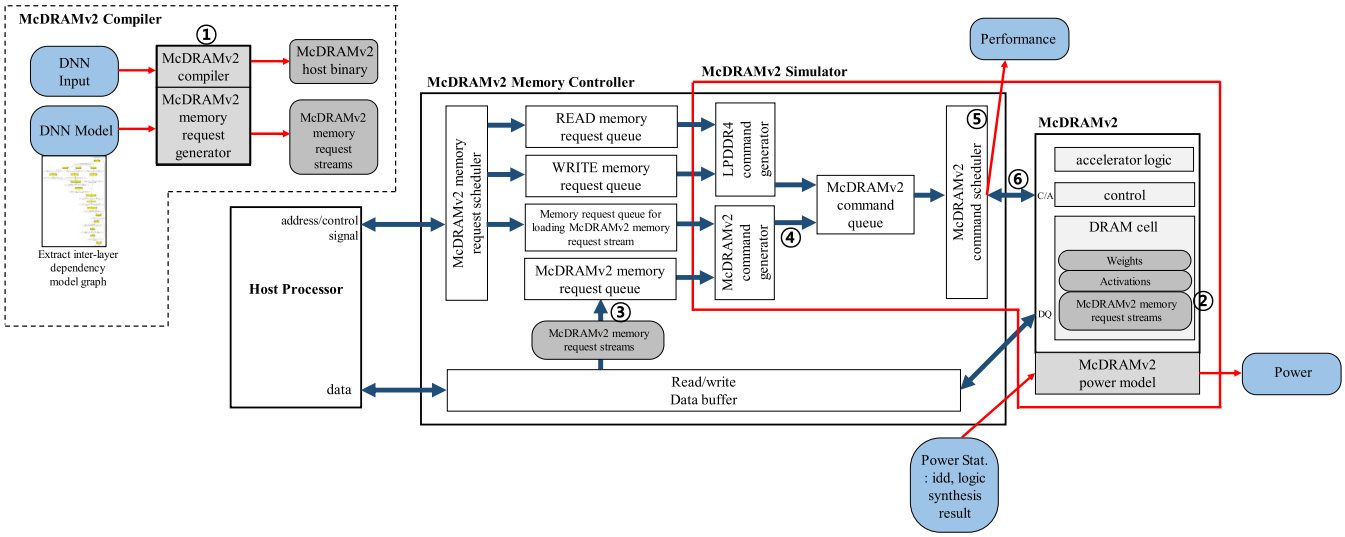

FIGURE 11. Proposed system integration.

ResNet-18/50, resulting in 1.1% and 0.6% top-1 accuracy losses, respectively. For all the neural networks used in the experiment, the first and last layers were quantized with int8 or int4 precision, while ensuring negligible accuracy loss.

B. BASELINE GPUS AND NPUS

We conducted extensive evaluation of the state-of-the-art GPUs and NPUs, McDRAM v1 [19], and McDRAM v2 based on various DNN benchmarks and compared them in terms of performance, energy efficiency, and silicon area efficiency. For the comparison of the server-class GPUs, NPUs, and McDRAM v2, DNN applications (with more than 100 million parameters from the domains of NLP and recommendation systems usually executed in the server system) were used to demonstrate the effectiveness when processing the inference of the large DNN models in McDRAM v2.

1) SERVER CLASS

In this study, we considered a server-class DNN accelerator, as it can perform the inference of a more-than-100-million-parameter model at its own speed, without any significant performance drops. We selected the NVIDIA Titan RTX, Tesla T4 [34], Habana Goya [35], McDRAM v1 [19], and McDRAM v2 and conducted an inference performance/cost comparison between them. NVIDIA Titan RTX and Tesla T4 are based on the Turing GPU microarchitecture and equipped with the Turing Tensor Cores, which can execute int4 or int8 GEMM operation with tremendous speed [34]. We used CUTLASS [51], a collection of CUDA C++ template abstracts for dense linear algebra, to perform GEMM operation for both GPUs, where the input precisions to the Tensor Cores were determined as the lowest possible precisions. In the cases of Habana Goya [35] and McDRAM v1 [19], we used an input precision of int8.

2) MOBILE CLASS

We selected the Edge TPU [18], NVIDIA Jetson AGX Xavier [17], McDRAM v1 [19], and McDRAM v2 as the mobile DNN accelerators and conducted performance/cost analysis over the inference of CNNs. Edge TPU [18] can only contain int8 precision data; thus, all the data in Edge TPU [18] are int8 data. For NVIDIA Jetson AGX Xavier [17] and McDRAM v1 [19], int8 input precision is used for the generalized matrix-vector multiplication (GEMV) and GEMM operations. We used the 15 W mode for the NVIDIA Jetson AGX Xavier because the inference benchmark [17] shows that the 15 W mode is far more energy efficient than the 30 W MAX-N mode, and the main focus of this study is energy efficiency, rather than performance.
McDRAM v2 uses the most suitable input precision mode between w2a4, w4a4, w4a8, and w8a8.

C. MCDRAM V2 SIMULATION INFRASTRUCTURE

To evaluate the performance with respect to the energy/silicon area overhead, we built a McDRAM v2 simulator and a McDRAM v2 memory request generator. The McDRAM v2 simulator is an in-house command-driven cycle-accurate simulation model, equipped with performance counters and a power model. The input data of the power model are the power results from logic implementation and DRAM's I_{DD} specification. In the simulator, we also implement a McDRAM v2 command generator and command scheduler, from which the performance of McDRAM v2 can be obtained as a result of command scheduling.

To estimate the power/silicon area overhead for the additional McDRAM v2 components, we designed a manual chip layout for a MAC unit, using Samsung 20-nm DRAM technology. The remaining components were synthesized, placed, routed, and estimated using a Synopsys Design Compiler,

IC Compiler, and Prime Time PX with Samsung System LSI 65-nm logic technology. Then, the estimated results were converted to match the ones obtained using Samsung 20-nm DRAM technology, based on the power/ area ratio between them. To estimate the power and area of the original LPDDR4 DRAM circuit in McDRAM v2, we used the power/area data of a Samsung 20-nm LPDDR4 DRAM chip.

A single McDRAM v2 die can be set in one of the following four configurations: 1, 2, 4, or 7 PE matrices per die. Among these configurations, we selected the 4 PE matrix version of McDRAM v2 as the baseline configuration, as it is the most energy efficient as well as the second most efficient with regard to inference performance. Table 3 presents the architectural parameters of McDRAM v2 in this configuration. An extensive evaluation of performance/energy/area/ PE utilization, as the number of PE matrix per die varies, is presented in Sections VIII.D and VIII.E.

TABLE 3. Baseline McDRAM v2 configuration.

Name	Description
McDRAM v2 Package Capacity	8 GB
Number of McDRAM v2 dies	8
McDRAM v2 Die Capacity	1 GB
Number of PE matrices per die	4 PEM (1, 2, 4, or 7 is possible)
Number of PEs per PE matrix	16 x 16
Input buffer size of systolic array	8.232 kB / PE matrix, 263.4 kB / McDRAM v2 package
Single PE compute capability	Two (int2, int4) input (int16) accumulation MACs per cycle
MAC operation frequency	1 GHz
Max matrix multiplication TOPS	32/16/8/4 TOPS for w2a4/w4a4/w4a8/w8a8 input precision modes
Internal memory bandwidth	512 GB/s
off-chip bus width	x 64
off-chip data rate	4266 MT/s
Process technology	20 nm

VIII. EXPERIMENTAL RESULTS

A. ENERGY EFFICIENCY

Fig. 12 shows the energy efficiency of server-class accelerators, McDRAM v1 [19], and McDRAM v2 while performing inference for large DNN workloads. The power measurement comprises the power consumption of the accelerator's processor and memory but excludes the power consumption of the host CPU and host memory. For GPUs and Habana Goya [35], the entire power of the accelerator card is used; for McDRAM v1 [19] and McDRAM v2, the DRAM package power is used to obtain the power efficiency. Despite the fact that McDRAM v2 consumes approximately 16 to 56 times less power than Tesla T4 [34], Habana Goya [35], and Titan RTX [34], it performs more-than-100 million-parameter DNNs at the best energy efficiency. For the inference of ResNet-50 [10] and BERT base [1], McDRAM v2 achieves 2.4 times and 5.6 times better TOPS/W when compared with the best server-class accelerator, i.e., Habana Goya [35]. For 5 large DNN workloads in the domains of image classification, language modeling, and recommendation systems, McDRAM v2 attains 4.1 times and 9.3 times better TOPS/W than Tesla T4 [34] and Titan RTX [34], respectively. This is accounted for by the fact that (1) it can achieve as wide a memory bandwidth as that of a GDDR6 [14] memory, used in server-class GPU, but without GDDR6 DRAM [14], (2) it scarcely uses any off-chip data movements in conducting even a large DNN, and (3) it can use lower computational precision, once the lower precision has been guaranteed to provide a negligible quality loss.

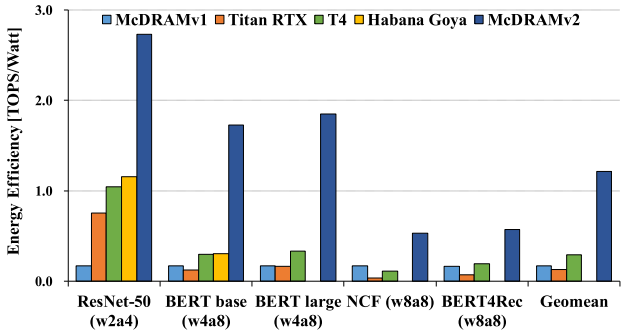

FIGURE 12. Inference energy efficiency of the server-class GPUs/NPU, McDRAM v1, and McDRAM v2 over large DNNs. The batch size is 128, except for NCF, whose batch size is 1024. Maximum sequence lengths for BERT and BERT4Rec [31] are 128 and 200, respectively. This batch size and maximum sequence length are used for the results of the rest of the paper, unless otherwise stated. Results for Habana Goya are available only for ResNet-50 and BERT base from its white paper [36].

Fig. 13 shows the inference energy efficiency of the mobile accelerators Edge TPU [18], Jetson AGX Xavier [17], McDRAM v1 [19], and McDRAM v2 over VGG19 [26], GoogleNet [11], and ResNet-50 [10]. For all the mobile accelerators used in the experiments, the board power was used for the comparison of energy efficiency in TOPS/W. For McDRAM v1 [19] and McDRAM v2, we assumed McDRAM v1 [19] or McDRAM v2 to replace the LPDDR4

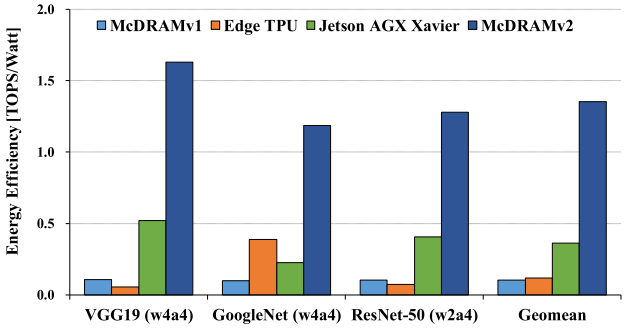

FIGURE 13. Inference energy efficiency of the mobile class GPU/NPU, McDRAMv1, and McDRAMv2 over CNNs.

DRAM of a Jetson AGX Xavier [17] and estimated the board power accordingly. Consequently, McDRAM v2 achieved 14.3, 11.7, and 3.7 times better energy efficiency in TOPS/W as compared to McDRAM v1 [19], Edge TPU [18], and Jetson AGX Xavier [17], respectively. These results were attributed to McDRAM v2's capability to handle lower precision and the absence of off-chip accesses when running CNNs.

Regarding the memory-intensive models of the BERT base and BERT large running in mobile environments, based on the benchmark results [49], McDRAM v2 with 4 PEM per die attains 3.2 and 3.7 times better energy efficiency than the Jetson AGX Xavier for the BERT base and BERT large, respectively, thereby suggesting the reliable energy efficiency of the McDRAM v2 DNN accelerator when performing the inference of a large and memory-intensive DNN model.

Table 4 presents the peak performance of all accelerators used in the experiments. The measured utilization of the arithmetic units of the accelerators was analyzed across several DNN workloads on servers and mobile environments, as follows.

TABLE 4. Peak performance of GPUs and NPUs.

Name	Peak performance
McDRAM v1 [19]	int8: 1 TOP
Titan RTX Tensor Cores [34]	int4: 455 TOPS, int8: 228 TOPS
Tesla T4 Tensor Cores [34]	int4: 260 TOPS, int8: 130 TOPS
Habana Goya [35]	int8: n/a
Edge TPU [18]	int8: 4 TOPS
Jetson AGX Xavier [17]	int8: 34TOPS
McDRAM v2 baseline (4 PE matrix)	int2/int4: 32 TOPS, int4/int4: 16 TOPS, int4/int8: 8 TOPS, int8/int8: 4 TOPS

Fig. 14 shows the measured TOPS/peak TOPS values of server-class GPUs/NPUs, McDRAM v1 [19], and McDRAMV v2 when processing large DNNs such as ResNet-50 [10], BERT base/large [1], NCF [29], and BERT4Rec [31]. The Titan RTX and Tesla T4 [34] attain 15% and 14% arithmetic unit utilizations, respectively, which is significantly lesser than that achieved by McDRAM v2 (44%). Considering the models' parameter sizes (ResNet-50 [10] has the smallest parameter size at 26 million, and the others have more than 100 million parameters) and the GPU's on-chip SRAM size, it is reasonable for Titan RTX and Tesla T4 to achieve 45% and 28% utilization on ResNet-50, respectively. For the rest of the large models, Titan RTX and Tesla T4 show 10% and 12% utilization, respectively.

The performance of McDRAM v2 is affected not by the parameter size of the DNN models, but by the length of the inner products during matrix multiplication. Fig. 15 shows

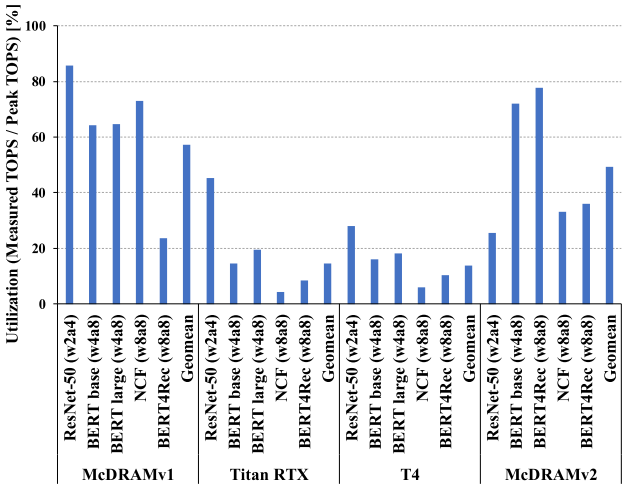

FIGURE 14. Measured performance over peak performance of server-class GPUs, McDRAMv1 [19], and McDRAMv2 over DNNs. Habana Goya data [35] is not presented because the peak performance of Habana Goya [35] is not available.

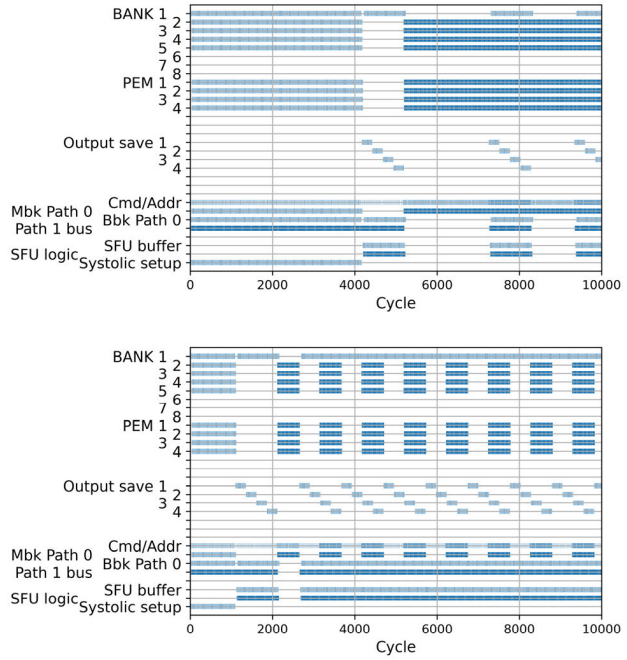

FIGURE 15. Resource scheduling of McDRAM v2 when matrix multiplication is performed. The upper (lower) one is for the matrix multiplication when the dimension of the inner product is 1024 (256) in the inference of the BERT large (NCF).

that McDRAM v2 can completely hide the time for saving the output behind the MAC computation time when the length of the inner products is 1024 in the inference of the BERT large and can partially hide the time for saving the output behind the MAC computation time when the length of the inner products is 256 in the inference of the NCF. In cases of BERT (base and large) [1], where the lengths of the inner products are sufficiently long (768 and 1024, respectively), McDRAM v2 achieves 75% utilization. In contrast, in the other three apps with small dimensions, 31% utilization is

achieved. This is explained by the fact that the apps must have sufficiently large inner product dimensions to completely hide the output saving latency with the Buffer_MM operation (which can be issued concurrently). In McDRAM v2, activation post-processing can be integrated with the output saving operation, without any off-chip accesses.

McDRAM v2 can process the ResNet-50 [10], BERT base, and BERT large [1] at 997 frames per second, 258 sentences per second, and 72 sentences per second, respectively, with power budgets of under 5 W, thereby providing the opportunity for large DNN applications to run on mobile environments in real time.

As shown in Fig. 16, the Edge TPU [18], Jetson AGX Xavier [17], McDRAM v1 [19], and McDRAM v2 attained 6%, 15%, 87%, and 46% arithmetic unit utilization, respectively. McDRAM v1 [19] shows the best utilization among them but exhibited the lowest peak TOPS. Although McDRAM v2 has a lower peak performance than Jetson AGX Xavier [17], the former achieves 1.7 times better TOPS than the latter [17] owing to the former's superior utilization capability.

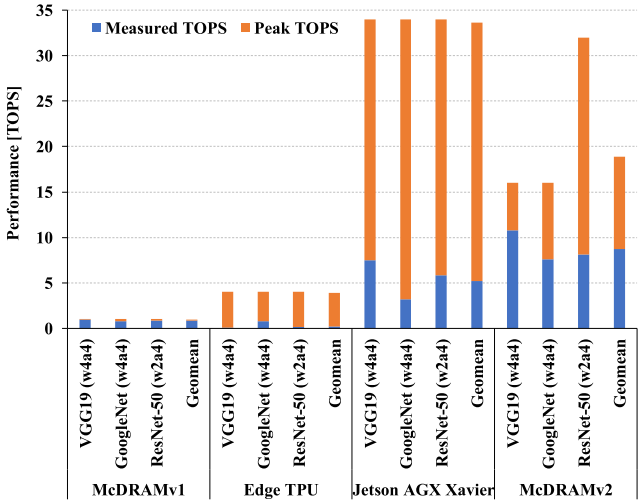

FIGURE 16. Peak and inference performance of the mobile class GPU/NPU, McDRAMv1, and McDRAMv2 over CNNs.

B. AREA EFFICIENCY

Fig. 17 show a comparison of the silicon area efficiency between server-class accelerators with McDRAM v1 [19] and McDRAM v2. For the Turing Architecture-based GPU Titan RTX and Tesla T4 [34], area of the entire GPU die is used. The Turing GPUs contain not only Tensor Cores for the computation of integers, but also 4608/2560 compute unified device architecture (CUDA) cores for floating-point arithmetic, thereby leading to disadvantages as compared to other accelerators. We assume that the compensation factor for the Titan RTX and Tesla T4 could be at most 2, as a large portion of the area is available for the on-chip SRAM, which both the tensor and CUDA cores utilize. Additionally, 8-bit quantization for DNNs essentially requires higher

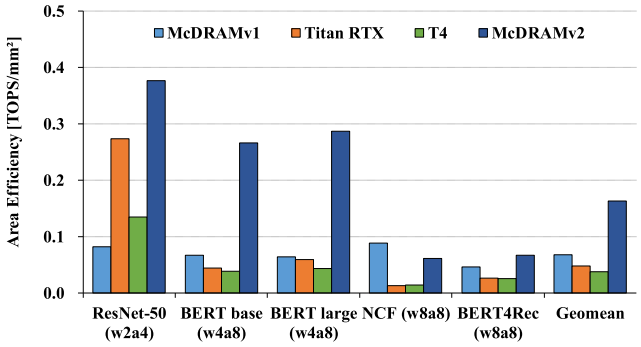

FIGURE 17. Inference area efficiency of the server-class GPUs/NPU, McDRAM v1 [19], and McDRAM v2 over large DNNs.

precision in computations other than GEMMs, which can lead to faster processing in Turing GPUs for operations other than GEMMs. Furthermore, in the GPU silicon area, the GDDR6 DRAM's area is not considered. For McDRAM v1 [19] and McDRAM v2, the entire additional McDRAM overhead area is used.

McDRAM v2 achieves 2.4 times, 3.4 times, and 4.3 times better silicon area efficiency than McDRAM v1 [19], Titan RTX, and Tesla T4 [34], respectively. Titan RTX's ResNet-50 [10] inference performance is from MLPerf's Open Inference-0.5v [30]. Notably, the Titan RTX on the ResNet-50 [10] attains far better performance, which is attributed to two reasons. First, only in ResNet-50 [10] does Titan RTX [34] use int4 compute precision with the Tensor Cores. The second reason is that the GPU's on-chip SRAM capacity (28.6 MB of l1/l2 cache, shared memory, and register file) can contain the entire set of parameters of ResNet-50. In this case, the Tensor Cores can exploit the much larger memory bandwidth of the on-chip SRAM, as compared with the case where most of the parameters come from the DRAM.

Fig. 18 shows the silicon area efficiency of mobile accelerators when performing CNNs: VGG19 [26], GoogleNet [11], and ResNet-50 [10]. For McDRAM v1 [19] and McDRAM v2, the entire additional McDRAM overhead area is used. For Jetson AGX Xavier [17], only the area of the GPU and MM Engine in DLA is used, out of the entire SoC die. The mobile GPU's area used in this comparison does not

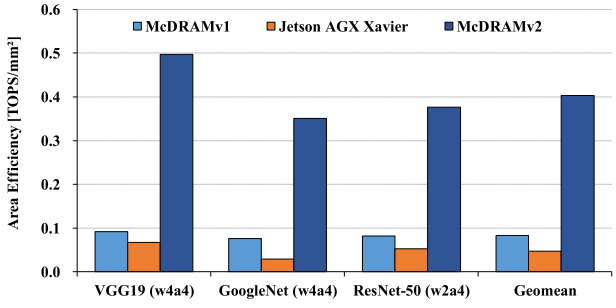

FIGURE 18. Inference area efficiency of the mobile class GPU, McDRAM v1, and McDRAM v2 over CNNs.

include the LPDDR4 DRAM [16] die area. Because of this comparison, it can be concluded that McDRAM v2 achieves 4.9 times and 8.6 times better silicon area efficiency than McDRAM v1 [19] and Jetson AGX Xavier [17], respectively.

C. PERFORMANCE OF EMBEDDING LOOKUP

Fig. 19 shows the runtime for the embedding layers of BERT base/large [1], NCF [29], and BERT4Rec [31]. As described in Section V.B.3, using McDRAM v2's parallel access and pooling operation for embedding tables, the runtime for executing the embedding layers, which is known to take a large portion of the total runtime in the case of NLP models and recommendation systems [8], [9], can be reduced. In the experiment, the desktop CPU is an Intel core-i7 5930K; the mobile CPU and GPU are from the NVIDIA Jetson AGX Xavier [17]. The mobile 8-Core ARM v8.2 64-Bit CPU is equipped with an 8 MB L2 cache and 4 MB L3 cache, and the mobile 512-core Volta GPU includes the Tensor Cores and deep learning accelerator (DLA) for integer GEMM. We measured the inference runtimes of the embedding layers across the accelerators using PyTorch [50]; the datasets used are described in Section II.A. In terms of the inference performance, McDRAM v2 processes the embedding layers 15.7 times, 1.57 times, and 2.04 times faster than the mobile GPU, Titan RTX, and Tesla T4 [34], respectively. McDRAM v2 can efficiently handle embedding tables whose size can fit in a single die capacity of 1 GB.

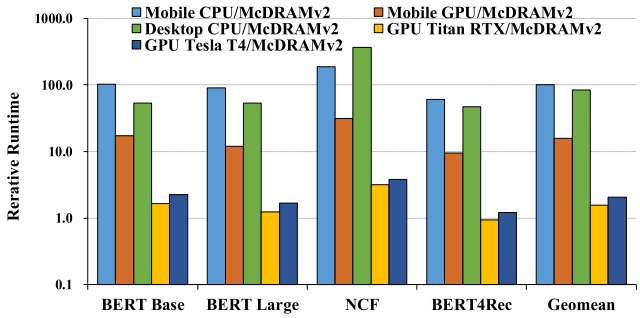

FIGURE 19. Throughput of various hardware for embedding lookup.

The Titan RTX and Tesla T4 [34] are quipped with bandwidth-optimized but loose-latency GDDR6 DRAMs [14], whose bandwidths are 672 GB/s and 320 GB/s, respectively. In contrast, McDRAM v2 uses 64 GB/s of internal bandwidth when it fetches embedding vectors during embedding lookup. It is generally known that the random access performance of a graphic DRAM is far inferior to its sequential performance. McDRAM v2 shows better performance for embedding layers as compared to GPUs and mobile GPUs because McDRAM v2 can effectively hide the latency incurred during a random access to DRAM banks and benefit from a shorter length of data movement without off-chip accesses, as described in Section V.B.3. However, GPUs and mobile GPUs cannot mitigate the overheads incurred by random accesses from embedding lookups.

D. AREA OVERHEAD

McDRAM v2 is implemented on a Samsung 20-nm 4ch LPDDR4 8 GB LPDDR4 DRAM base. As shown in Fig. 20, in cases of 1, 2, 4, and 7 PE matrices per die, the silicon area overheads, which are based on the original LPDDR4 silicon die area, are 3.2%, 5.4%, 9.7%, and 16.1%, respectively. The area overhead contains the area for the PE matrix, row/column FIFO buffer, SFU, systolic data setup unit, and additional interconnections for McDRAM v2. Regarding the area overhead, the PE matrix, which contains the PEs and input/output systolic array, incurs the highest cost, followed by the FIFO buffer in the PE matrix, but, in both the cases, the portion varies from case to case. An additional area for more interconnections is available for assigning a dedicated path0 bus and estimated based on the word line (WL)/bit line (BL)/active pitch of the LPDDR4 DRAM.

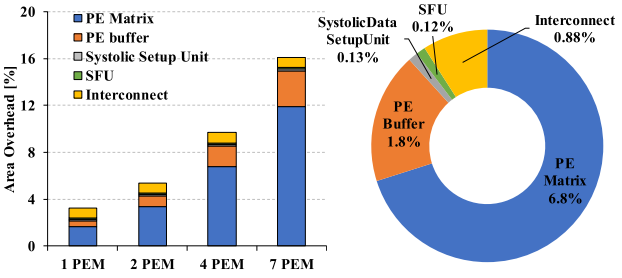

FIGURE 20. Left graph shows the breakdown of area overhead of McDRAM v2 when the number of PE matrices per die is 1, 2, 4, and 7. Right graph unfolds the area breakdown in the case of 4 PE matrices per die.

E. EVALUATION OF MCDRAM V2 WITH VARIOUS CONFIGURATIONS

Fig. 21 shows the performance, average power, energy efficiency, and PE utilization of the inference of McDRAM v2 for various combinations of the number of PE matrices and PE computing precision levels. The first and last layers use int8 precision in all the cases.

As the PE matrix count per die increases from 1 to 7, the TOPS increases for all the configurations, yet from 4 to 7, the margin shows diminishing returns. In addition, the average power increases as the number of PE matrices increases for all cases, but also with diminishing returns from 1 to 7. Consequently, McDRAM v2 with 4 PE matrices per die is considered to be the optimum configuration. As for the PE utilization, a lower number of PE matrices per die yields better results.

When we vary the input precision of the PEs, a lower precision shows better performance and energy efficiency in the inference of a DNN. This is attributed to the fact that lower compute precision has an effect of having more MAC units due to McDRAM v2's intrinsic multi-bit support structure. Regarding power consumption, no relationship was found between the average power and computing precision. In terms of PE utilization, a higher precision was found to lead to better results. This is because a higher computing precision means

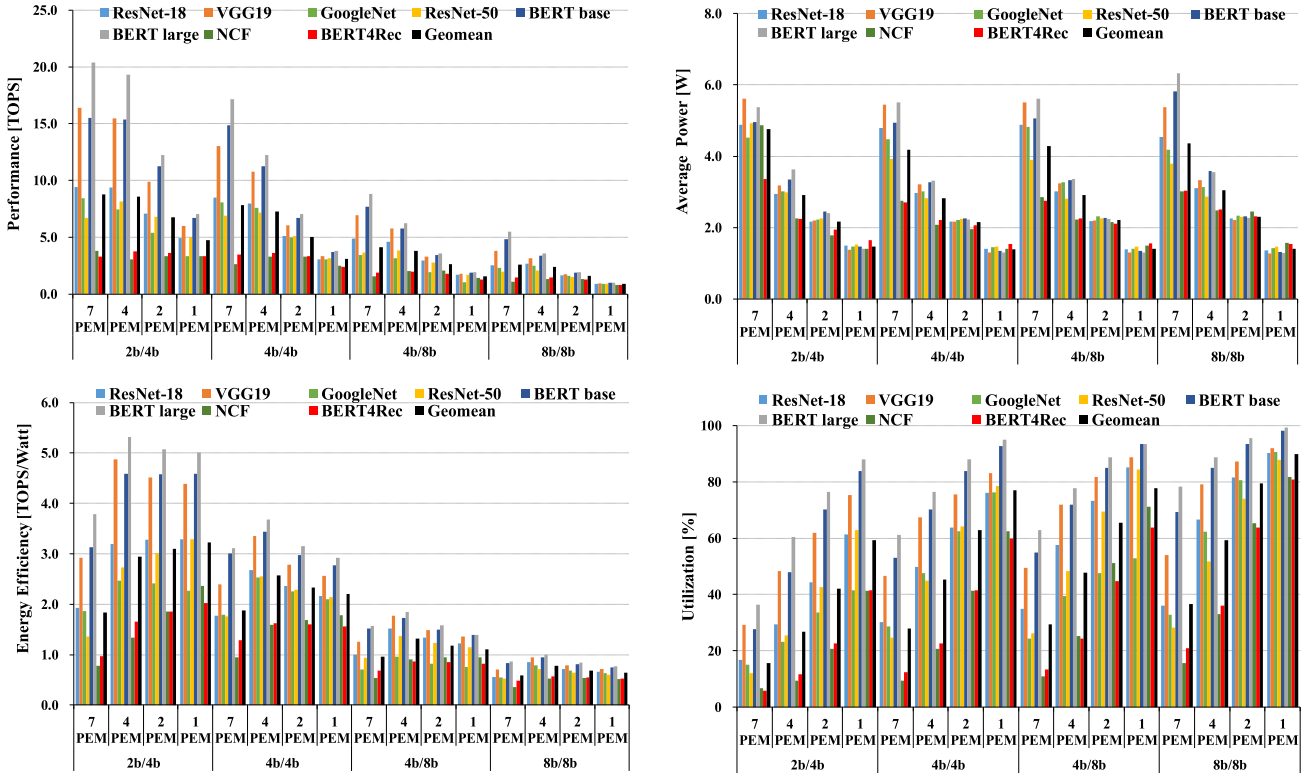

FIGURE 21. Performance, power consumption, energy efficiency, and PE utilization with varying numbers of PE matrices and varying computing precisions of McDRAM v2.

that smaller matrix tile dimensions need to be processed by McDRAM v2 at a time, and a smaller tile size is usually advantageous as it helps in finding parallelism when matrix multiplications of various dimensions need to be performed.

In addition to bank-level scalability, McDRAM v2 can support die- and package-level scalabilities, if the host processor and the host memory controller of the system adopting McDRAM v2 support them. In the case of die- and package-level scalabilities, more dies and more packages of McDRAM v2 can perform inference in larger batch sizes, as a single batch is designed to run only within a single die without any off-chip accesses. As for the 2D systolic array dimension scalability, the dimensions of the 16 x 16 systolic array of the int4-int4 input mode fits the combination of the LPDDR4's single bank read bandwidth (8 GB/s) and the MAC unit's input precision to maximize the inference performance. Therefore, 2D systolic array dimensions larger than the 16 x 16 systolic array of the int4-int4 mode do not result in performance improvement. McDRAM v2 can therefore support a lower int2-int4 input mode, which effectively increases the dimensions of the PE matrix by two times and can run twice as many MAC operations than with the int4-int4 input mode within the same period of time.

F. OPERATING POWER ANALYSIS

We measured the peak operating power consumption of the McDRAM v2 package for various operating scenarios by varying the number of PE matrices from 1 to 7 (Fig. 22). Five operating scenarios were considered: broadcasting matrix multiplication, buffer matrix multiplication, output saving, embedding lookup, and a combination of buffer matrix multiplication and output saving, from Section V.B. Regardless of the scenario, the combination of buffer matrix multiplication and output saving consumed the highest peak operating power among the five cases. For 1, 2, 4, and 7 PE matrices per die, McDRAM v2 consumed peak powers of 2.2 W, 3.1 W, 4.4 W, and 7.9 W, respectively.

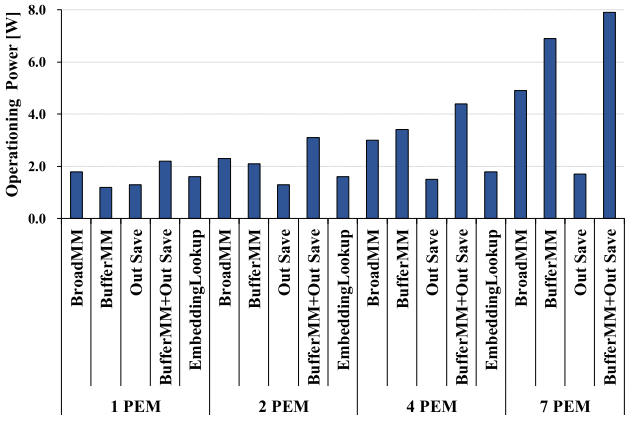

FIGURE 22. Operating power consumption of McDRAM v2 under various operating scenarios with the number of PE matrices ranging from 1 to 7.

G. COMPARISON WITH MCDRAM V1

Although McDRAM v1 [19] and McDRAM v2 are both based on an x64 4ch 8 GB LPDDR4 DRAM [16] and exploit maximum internal bandwidth, they have fundamental differences. A comparison between them is summarized in Table 5.

TABLE 5. Baseline McDRAM v2 configuration.

Name	McDRAM v1 [19] (our implementation)	McDRAM v2 with 2 PEM per die
Base DRAM	LPDDR4 DRAM x64 4ch 64 Gb, 4266 Mbps	LPDDR4 DRAM x64 4ch 64 Gb, 4266 Mbps
Peak internal bandwidth	512 Gb/s	512 Gb/s
#active banks operating simultaneously for weight read	8	2
Peak weight bandwidth	512 GB/s	128 GB/s
# weight reuse	1	16, 16, 8, and 8 for w2/a4, w4/a4, w4/a8, and w8/a8 input precision
# MACs	int8/int8: 2048, MACs (32 MACs / bank)	int2/int4: 8192 MACs (2 MACs / PE, 16 x 16 PE / bank)
Supported compute precision	mult: int8 Add: int16	mult: int2/int4, int4/int4, int4/int8, int8/int8 Add: int16
MAC operation frequency	250 MHz	1 GHz
Input buffer size	16 kB	132 kB
Peak TOPS	1 TOPS	16, 8, 4, and 2 TOPS for int2/int4, int4/int4, int4/int8, and int8/int8 input precision modes
Average TOPS	0.77 TOPS	3.61 TOPS, (1.59 TOPS for int8)
Peak Power	5.8 W	3.1 W
Average Power	4.52 W	2.29 W (2.31 W for int8)
Process technology	20 nm	20 nm
Area overhead	4.7%	5.4%

First, each has different placements of processing units and different data paths. McDRAM v1 places two MAC units at each BL block of the DRAM bank, whereas McDRAM v2 adopts a matrix of processing units whose input/output buffers constitute a 2D-systolic array. The latter leads to many more processing units within the same in-DRAM bandwidth budget (internal single bank read bandwidth (8 GB/s)) and a much larger reuse count (from 1 to 8 in the int8 mode) of input data read from the DRAM cell. Also, McDRAM v2 supports multiple computing precision values for its matrix multiplication, but McDRAM v1 can only perform int8 matrix-vector multiplication.

Second, the two accelerators differ in the locations of the MACs. In McDRAM v1, the MAC is located in the column address decoder area, close to the DRAM cell matrix; however, in McDRAM v2, a 2D matrix of MACs is located in the DRAM peripheral area, where a faster transistor is available. The DRAM column decoder area has a transistor operating at 250 MHz, but the DRAM peripheral area has a faster transistor operating at 1 GHz. Thus, within a similar area overhead constraint, McDRAM v2 has a smaller number of MACs than McDRAM v1 and shows better peak performance (2048 MACs vs. 1024 MACs and 1 TOPS vs. 2 TOPS).

Third, McDRAM v2 can perform almost all operations of a DNN, including pre- and post- processing of the activation data, within the in-DRAM die boundary, without incurring the overhead of off-chip accesses. Also, McDRAM v2 performs activation post-processing fused with output saving operation minimizing DRAM read accesses. In contrast, McDRAM v1 [19] requires that activation data be sent to the off-chip host CPU to prepare the input activation data before all the computing operations and to perform post-processing such as activation functions (e.g., sigmoid, tanh) and (de-)quantization, and then to be broadcast back into it.

Fourth, McDRAM v1 cannot hide the time for saving output behind the MAC computation time, but McDRAM v2 can hide the time for saving output behind the MAC computation time, once the lengths of the inner product are sufficiently long, as shown in Fig. 15.

As a result of implementing these two different in-DRAM accelerators, McDRAM v2 shows 2.1 times better performance and 4.0 times better energy efficiency for the inference of DNN workloads of ResNet-18/50 [10], VGG19 [26], GoogleNet [11], BERT base/large [1], NCF [29], and BERT4Rec [31], when compared with two PE matrix configurations with similar area overheads and with the same int8 input precision.

IX. RELATED WORK

A. NEAR-MEMORY PROCESSING DNN ACCELERATOR

When the accelerator is processing large DNN models, one of the most prominent bottlenecks is the memory bandwidth [37]. To overcome this challenge, near-memory processing (NMP) processors [19], [21], [22], [48] have been proposed to reduce the length of the data path between a processing unit and a memory cell as well as to exploit the large in-memory bandwidth.

HMC [20] is a type of 3D-stacked DRAM. In the HMC, the DRAM dies and logic dies are vertically integrated using through-silicon vias. Neurocube [21] and TETRIS [22] place the DNN processing functions in the logic die of the HMC and store the data in vaults in the DRAM die. In this way, the accelerator designs can reduce power-hungry off-chip accesses as well as the data load latency by shortening the length through which the data moves. However, the internal memory bandwidth they use is the same as an off-chip bandwidth.

TensorDIMM [48] enables the processing of a recommendation system with more than hundreds of GBs of embedding tables, possibly within the context of multiple GPUs and interconnects, without accessing a host CPU and host memory. It comprises custom DDR4 DIMMs located in a GPU interconnect as a single independent node, and this custom DIMM is equipped with NMP cores to perform simple embedding vector manipulations on the DIMM board.

McDRAM v1 [19] is an LPDDR4-based DNN accelerator. In this accelerator, 32 MAC units per DRAM bank are located in the column decoder area and acceleration of the GEMV operations is done using both the massive in-DRAM bandwidth obtained by a concurrent all-bank read operation and the broadcasting of data, which is done via the shared bus and I/O interface.

In addition, various in-DRAM architectures [25], [52] can support binary networks; however, it is still not possible to quantize into the binary level without a significant quality loss when quantizing large DNN models such as the convolution-based ResNet [10], transformer-based BERT language model [1], and NCF recommendation system [29] over large datasets. DRISA [25] adds simple Boolean logic circuits near the bit line and the local sense amp of the DRAM and runs binary CNNs, but with a prohibitive area overhead of 91% of the base LPDDR4 area.

B. LOW-PRECISION INTEGER INFERENCE NEURAL NETWORK ACCELERATOR

Low-precision quantization of DNN inference is one of the most important optimization techniques that can be used for efficient DNN acceleration. In a variety of DNN domains, such as image classification, NLP, and recommendation systems, numerous quantization methods [38], [39], [43], [53]–[56] have been proposed to show that the input data of GEMM operations in the DNN inference can only be quantized to int8 or int4 with a negligible quality loss. However, to fully support this, the NPU has to process the remaining computations with higher precision. This means that the NPU has to be equipped with special hardware for processing operations, e.g., quantization encoding/decoding, and manipulation of intermediate activation/embedding vectors with high precision, or that such processing must be performed in an external processor, such as the host CPU.

There are numerous NPU designs [21], [22], [45], [46], [57]–[61] proposed in academia and industry that use a computing precision of floating-point 16 or fixed-point 16, both of which are sufficient for processing a DNN without quality loss; however, these designs do not support low-precision integer arithmetic computations. In contrast, there are several accelerator designs that support low-precision integer arithmetic GEMM operations, as well as higher-precision computations for the remaining operations. The TPU v1 [37], NVIDIA Turing Titan RTX/Tesla T4 [34], Habana Goya [35], NVIDIA Jetson AGX Xavier [17], Tesla's NPU in full self-driving chip [62], and OLAccel [47] are some examples of such designs.

What distinguishes McDRAM v2 from accelerators supporting low-precision integer arithmetic is that it processes large DNN applications with more than hundreds of parameters, within the power budget of the mobile environment (within 7 W, 5 W, 3.5 W, or 2.5 W, depending on the configuration) but with better efficiency, as is demonstrated in this study.

To overcome the aforementioned challenge, McDRAM v2 is motivated by some architectural ideas from cutting-edge accelerator designs, for example: (1) the systolic array architecture from TPU [37], but with several different details (such as modified matrix dimensions to fit in the DRAM context, different accumulation methods, multi-precision support, and so forth) and (2) exploiting the large in-DRAM bandwidth from McDRAM v1 [19], but with an entirely new architecture with a different datapath, different locations of processing units in the DRAM, and several other architectural changes.

X. CONCLUSION

To accelerate a large DNN model in a mobile environment, this study presents a cost/benefit-balanced architecture for in-DRAM computations that makes the best use of the systolic array on a DRAM die. Our proposed McDRAMv2 architecture exploits the large in-DRAM bandwidth, thereby boosting the utilization of large-scale computing units in the systolic array accelerator. McDRAMv2 also supports efficient in-DRAM processing of embedding lookups, without off-chip DRAM accesses. We provide extensive evaluations over large DNN models with hundreds of MBs of parameters and embedding tables in natural language processing, recommendation systems, and image classification. With most of the data movements and computations confined within the energy-optimized LPDDR4-based McDRAM v2 die, McDRAM v2 achieves 1.7 times TOPS, 3.7 times TOPS/W, and 8.6 times TOPS/mm^2 improvement over a state-of-the–art mobile GPU accelerator, and 4.1 times better energy efficiency over a state-of-the-art server-class accelerator, yet incurs a minimal overhead of a 9.7% area increase, and requires less than 4.4 W of peak operating power.

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*. [Online]. Available: http://arxiv.org/abs/1810.04805

[2] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019, *arXiv:1909.08053*. [Online]. Available: http://arxiv.org/abs/1909.08053

[3] *Turing-NLG: A 17-Billion-Parameter Language Model by Microsoft*. Accessed: May 1, 2020. [Online]. Available: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft

[4] H. Koepke. (Jun. 2019). *Recommender Models in Create ML*. [Online]. Available: https://devstreaming-cdn.apple.com/videos/wwdc/2019/427quw6yo5agrbit/427/427_training_recommendation_models_in_create_ml.pdf?dl=1

[5] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for YouTube recommendations," in *Proc. 10th ACM Conf. Recommender Syst.*, Sep. 2016, pp. 191–198.

[6] Z. Zhao, L. Hong, L. Wei, J. Chen, A. Nath, S. Andrews, A. Kumthekar, M. Sathiamoorthy, X. Yi, and E. Chi, "Recommending what video to watch next: A multitask ranking system," in *Proc. 13th ACM Conf. Recommender Syst.*, Sep. 2019, pp. 43–51.

[7] H. T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "Wide & Deep Learning for Recommender Sys-tems," in *Proc. 1st Workshop Deep Learn. Recommender Syst.*, Boston, MA, USA, 2016, pp. 7–10.

[8] M. Naumov *et al.*, "Deep learning recommendation model for personalization and recommendation systems," 2019, *arXiv:1906.00091*. [Online]. Available: http://arxiv.org/abs/1906.00091
[9] U. Gupta, S. Hsia, V. Saraph, X. Wang, B. Reagen, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "DeepRecSys: A system for optimizing end-to-end at-scale neural recommendation inference," 2020, *arXiv:2001.02772*. [Online]. Available: http://arxiv.org/abs/2001.02772
[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
[12] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.
[13] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, Long Beach, CA, USA, 2019, pp. 6105–6114.
[14] *Graphics Double Data Rate (GDDR6) SGRAM Standard*, Standard JEDEC Standard JESD250B, 2018.
[15] *High Bandwidth Memory (HBM) DRAM*, Standard JEDEC Standard JESD235A, 2015.
[16] *Low Power Double Data Rate 4 (LPDDR4)*, Standard JEDEC Standard JESD209-4, 2014.
[17] *Jetson AGX Xavier: Deep Learning Inference Benchmarks*, Accessed: May 1, 2020. [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-dl-inference-benchmarks
[18] *Edge TPU Performance Benchmarks*. Accessed: May 1, 2020. [Online]. Available: https://coral.ai/docs/edgetpu/benchmarks/
[19] H. Shin, D. Kim, E. Park, S. Park, Y. Park, and S. Yoo, "McDRAM: Low latency and energy-efficient matrix computations in DRAM," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2613–2622, Nov. 2018.
[20] *Hybrid Memory Cube Specification 2.1*. Accessed: May 1, 2020. [Online]. Available: https://www.Nuvation.com/sites/default/files/Nuvation-Engineering-Images/Articles/FPGAs-and-HMC/HMC-30G-VSR_HMCC_Specification.pdf
[21] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 380–392.
[22] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and efficient neural network acceleration with 3D memory," in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Xi'an, China, 2017, pp. 751–764.
[23] *System Architecture*. Accessed: May 1, 2020. [Online]. Available: https://cloud.google.com/tpu/docs/system-architecture
[24] *Nvidia Tesla V100 Gpu Architecture*. Accessed: May 1, 2020. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf
[25] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-based reconfigurable *in-situ* accelerator," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture MICRO*, 2017, pp. 288–301.
[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, San Diego, CA, USA, 2015, pp. 1–14.
[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.
[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, Long Beach, CA, USA, 2017, pp. 5998–6008.
[29] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. S. Chua, "Neural collaborative filtering," in *Proc. Int. Conf. World Wide Web*, Perth, WA, Australia, 2017, pp. 173–182.
[30] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang, G.-Y. Wei, and C.-J. Wu, "MLPerf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, Mar. 2020.
[31] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manage.*, Nov. 2019, pp. 1441–1450.
[32] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 1–19, Jan. 2016.
[33] F. Belletti, K. Lakshmanan, W. Krichene, Y.-F. Chen, and J. Anderson, "Scalable realistic recommendation datasets through fractal expansions," 2019, *arXiv:1901.08910*. [Online]. Available: http://arxiv.org/abs/1901.08910
[34] *Nvidia Turing GPU Architecture*, Accessed: May 1, 2020. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf
[35] E. Medina and E. Dagan, "Habana labs purpose-built AI inference and training processor architectures: Scaling AI training systems using standard Ethernet with gaudi processor," *IEEE Micro*, vol. 40, no. 2, pp. 17–24, Mar. 2020.
[36] (Aug. 2019). *Goya Inference Platform White Paper*. Accessed: May 1, 2020. [Online]. Available: https://habana.ai/wp-content/uploads/pdf/habana_labs_goya_whitepaper.pdf
[37] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.
[38] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," 2019, *arXiv:1902.08153*. [Online]. Available: http://arxiv.org/abs/1902.08153
[39] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q-BERT: Hessian based ultra low precision quantization of BERT," 2019, *arXiv:1909.05840*. [Online]. Available: http://arxiv.org/abs/1909.05840
[40] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proc. EMNLP Workshop BlackboxNLP, Analyzing Interpreting Neural Netw. NLP*, 2018, pp. 1–3.
[41] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2016, pp. 2383–2392.
[42] E. F. T. K. Sang and H. Dejean, "Introduction to the CoNLL-2001 shared task: Clause identification," Jul. 2001, *arXiv:cs/0107016*. [Online]. Available: https://arxiv.org/abs/cs/0107016
[43] Y. Nahshan, "Loss aware post-training quantization," Mar. 2020, *arXiv:1911.07190*. [Online]. Available: https://arxiv.org/abs/1911.07190
[44] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.
[45] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, Toronto, ON, Canada, 2017, pp. 27–40.
[46] 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 *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.
[47] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 688–698.
[48] Y. Kwon, Y. Lee, and M. Rhu, "TensorDIMM: A practical near-memory processing architecture for embeddings and tensor operations in deep learning," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 740–753.
[49] *Bringing Cloud-Native Agility to Edge AI Devices with the NVIDIA Jetson Xavier NX Developer Kit*. Accessed: Jun. 5, 2020. [Online]. Available: https://devblogs.nvidia.com/bringing-cloud-native-agility-to-edge-ai-with-jetson-xavier-nx/
[50] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, 2019, pp. 8024–8035.
[51] *CUTLASS 2.1*, Accessed: May 1, 2020. [Online]. Available: https://github.com/NVIDIA/cutlass
[52] C. Sudarshan, J. Lappas, M. M. Ghaffar, V. Rybalkin, C. Weis, M. Jung, and N. Wehn, "An in-DRAM neural network processing engine," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[53] E. Park, D. Kim, S. Yoo, and P. Vajda, "Precision highway for ultra low-precision quantization," 2018, *arXiv:1812.09818*. [Online]. Available: http://arxiv.org/abs/1812.09818
[54] E. Park, S. Yoo, and P. Vadja, "Value-aware quantization for training and inference of neural networks," in *Proc. Conf. Eur. Conf. Comput. Vis.*, Munich, Germany, Sep. 2018, pp. 580–595.
[55] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4350–4359.
[56] D. Zhang J. Yang, D. Ye, G. Hu, "LQ-Nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. Conf. Eur. Conf. Comput. Vis.*, Munich, Germany, Sep. 2018, pp. 365–382.
[57] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 367–379.
[58] T. Luo, S. Liu, L. Li, Y. Wang, S. Zhang, T. Chen, Z. Xu, O. Temam, and Y. Chen, "DaDianNao: A neural network supercomputer," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 73–88, Jan. 2017.
[59] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.
[60] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 58–70.
[61] D. Shin, J. Lee, J. Lee, J. Lee, and H.-J. Yoo, "DNPU: An energy-efficient deep-learning processor with heterogeneous multi-core architecture," *IEEE Micro*, vol. 38, no. 5, pp. 85–93, Sep. 2018.
[62] E. Talpes, D. D. Sarma, G. Venkataramanan, P. Bannon, B. McGee, B. Floering, A. Jalote, C. Hsiong, S. Arora, A. Gorti, and G. S. Sachdev, "Compute solution for Tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, Mar. 2020.

SEUNGHWAN CHO (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from POSTECH, Pohang, South Korea, in 2014 and 2015, respectively. He is currently pursuing the Ph.D. degree in computer science and engineering with Seoul National University, Seoul, South Korea. His research interests include energy-efficient hardware accelerators, hardware-software co-optimization of deep neural networks, and near-memory processing architecture.

HAERANG CHOI (Graduate Student Member, IEEE) received the B.S. and M.S. degrees from Hanyang University, Seoul, South Korea, in 2005 and 2007, respectively. He is currently pursuing the Ph.D. degree in computer science and engineering with Seoul National University, Seoul, South Korea. From 2007 to 2016, he designed and developed DRAM circuits at SK Hynix. His research interests include memory-domain architecture.

EUNHYEOK PARK (Member, IEEE) received the B.S. degree in physics and electrical engineering and the M.S. degree in electrical engineering from POSTECH, Pohang, South Korea, in 2014 and 2015, respectively, and the Ph.D. degree in computer science and engineering from Seoul National University, Seoul, South Korea, in 2020. He is currently a Postdoctoral Researcher with the Inter-University Semiconductor Research Center, Seoul National University, Seoul. His research interests include energy-efficient hardware accelerators and hardware-considering optimization of deep neural networks, especially on quantization.

HYUNSUNG SHIN (Member, IEEE) received the B.S. degree in electronics engineering from Hanyang University, Seoul, South Korea, in 2007, and the M.S. degree in computer science and engineering from Seoul National University, Seoul, in 2018. He is currently a Senior Research and Development Engineer with the Memory Division, Samsung Electronics. His main research interests include memory systems and analysis of memory fault phenomena.

SUNGJOO YOO (Senior Member, IEEE) received the Ph.D. degree from Seoul National University, Seoul, South Korea, in 2000. From 2000 to 2004, he was a Researcher with the TIMA Laboratory, Grenoble, France. From 2004 to 2008, he led a System-Level Design Team, System LSI, Samsung Electronics, Hwaseong, South Korea, as a Principal Engineer. From 2008 to 2015, he was an Associate Professor with the Pohang University of science and technology, Pohang, South Korea. In 2015, he joined Seoul National University, where he is currently a Full Professor. His current research interests include memory sub-systems and software/hardware co-design of deep neural networks.