저작자표시-비영리-변경금지 2.0 대한민국

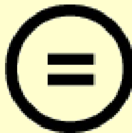
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학박사 학위논문

DRAM-based Processing-in-Memory Microarchitectures for Memory-intensive Machine Learning Applications

메모리 집약적 기계학습 응용프로그램을 위한 디램 기반 프로세싱 인 메모리 마이크로아키텍처

2022년 2월

서울대학교 융합과학기술대학원
융합과학부 지능형융합시스템전공
김 병 호

DRAM-based Processing-in-Memory Microarchitectures for Memory-intensive Machine Learning Applications

지도교수 안 정 호

이 논문을 공학박사 학위논문으로 제출함
2022년 1월

서울대학교 융합과학기술대학원

융합과학부 지능형융합시스템전공

김 병 호

김병호의 공학박사 학위논문을 인준함
2021년 12월

위 원 장:	이 원 종	(인)
부위원장:	안 정 호	(인)
위 원:	김 동 준	(인)
위 원:	유 민 수	(인)
위 원:	이 석 한	(인)

Abstract

DRAM-based Processing-in-Memory Microarchitectures for Memory-intensive Machine Learning Applications

Byeongho Kim
Intelligence Systems
Department of Transdisciplinary Studies
The Graduate School
Seoul National University

Recently, as research on neural networks has gained significant traction, a number of memory-intensive neural network models such as recurrent neural network (RNN) models and recommendation models are introduced to process various tasks. RNN models and recommendation models spend most of their execution time processing matrix-vector multiplication (MV-mul) and processing embedding layers, respectively. A fundamental primitive of embedding layers, tensor gather-and-reduction (GnR), gathers embedding vectors and then reduces them to a new embedding vector. Because the matrices in RNNs and the embedding tables in recommendation models have poor

reusability and the ever-increasing sizes of the matrices and the embedding tables become too large to fit in the on-chip storage of devices, the performance and energy efficiency of MV-mul and GnR are determined by those of main-memory DRAM. Therefore, computing these operations within DRAM draws significant attention.

In this dissertation, we first propose a main-memory architecture called MViD, which performs MV-mul by placing MAC units inside DRAM banks. For higher computational efficiency, we use a sparse matrix format and exploit quantization. Because of the limited power budget for DRAM devices, we implement the MAC units only on a portion of the DRAM banks. We architect MViD to slow down or pause MV-mul for concurrently processing memory requests from processors while satisfying the limited power budget. Our results show that MViD provides 7.2× higher throughput compared to the baseline system with four DRAM ranks (performing MV-mul in a chip-multiprocessor) while running inference of Deep Speech 2 with a memory-intensive workload.

Then we propose TRiM, an NDP architecture for accelerating recommendation systems. Based on the observation that the DRAM datapath has a hierarchical tree structure, TRiM augments the DRAM datapath with "in-DRAM" reduction units at the DDR4/5 rank/bank-group/bank level. We modify the interface of DRAM to provide commands effectively to multiple reduction units running in parallel. We also propose a host-side architecture with hot embedding-vector replication to alleviate the load imbalance that arises across the reduction units. An optimal TRiM design based on DDR5 achieves up to a 7.7× and 3.9× speedup and reduces by 55% and 50% the energy con-

sumption of the embedding vector gather and reduction over the baseline and the state-of-the-art NDP architecture with minimal area overhead equivalent to 2.66% of DRAM chips.

keywords: Processing-in-Memory architecture, Near-data processing, In-DRAM processing, Memory microarchitecture, Memory-intensive
student number: 2017-22676

Contents

Abstract i

Contents iv

List of Tables vii

List of Figures viii

1 Introduction 1

1.1 Accelerating RNNs on Edge 3

1.2 Accelerating Recommendation Model 5

1.3 Research Contributions 8

1.4 Outline 9

2 Background 11

2.1 Memory-intensive Machine Learning Applications 11

2.2 DRAM Organization and Operations 13

3 MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks 18

3.1 Background and Motivation 18

3.1.1 Energy-efficient RNN Mobile Inference 18
3.1.2 How to Improve the Energy Efficiency and Bandwidth of DRAM Accesses in MV-mul 21
3.2 MV-mul in DRAM 23
3.2.1 Exploiting Quantization and Sparsity in RNN's Matrix Elements 23
3.2.2 The Operation Sequence of MV-mul in DRAM 27
3.2.3 Concurrently Serving Requests from Processors and Performing MV-mul in DRAM 32
3.2.4 Put It All Together: MViD Architecture 37
3.2.5 Additional Optimization Schemes 38
3.3 Evaluation 39
3.3.1 Power/Area/Timing Analysis 39
3.3.2 Performance/Energy Evaluation 42
3.4 Discussion 48

4 TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory 51

4.1 Prior NDP architectures for accelerating Tensor Gather-and-Reduction 51
4.1.1 Tensor Gather-and-Reduction in RecSys 51
4.1.2 Prior NDP accelerators for GnR 52
4.1.3 Quantitative Analysis 56
4.1.4 Additional Schemes for Accelerating GnR 58
4.2 Tensor Reduction in Memory 58

4.2.1 Basic Concept for TRiM 59
4.2.2 How to Provision C/A Bandwidth 62
4.2.3 Exploring NDP Unit Placement 65
4.2.4 TRiM-G Organization and Operations 68
4.2.5 Host-side Architecture for TRiM 70
4.2.6 Schemes for Improving Reliability 75
4.3 Experimental Setup 76
4.4 Evaluation 77
4.4.1 Performance and Energy Efficiency 79
4.4.2 Sensitivity Study of Hot-entry Replication 82
4.4.3 Design Overhead 82
4.5 Discussion 83

5 Discussion 86

6 Related work 89

7 Conclusion 92

REFERENCES 94

국문초록 117

List of Tables

3.1 Energy/area/timing of MViD components. 39
3.2 Default simulation parameters. 42
4.1 Timing/energy parameters of 16Gb DDR5-4800 ×8 DRAM chips and NDP units. 76

List of Figures

1.1 Simplified architecture of a RecSys model. 6

2.1 Roofline model for four arithmetic operations. 12

2.2 Simplified DRAM datapath and module organization of DDR5 DRAM 14

3.1 The FLOP and execution time breakdown of Deep Speech 2 . . . 19

3.2 The aggregate size of weight matrices and word error rate (WER) of representative ASR models over time 19

3.3 Distribution of the distance between two adjacent non-zero values (NZs) in the matrices of GRU layers (75% sparsity) in Deep Speech 2. 24

3.4 An exemplary sparse matrix and its representation using the delta encoding format. 24

3.5 The storage requirement for a matrix with different index bits when using delta encoding format. 26

3.6 The relative size of the tested matrices encoded with various sparse matrix formats normalized to the size of the dense format. . . . 26

3.7 Mapping of the weight matrix to DRAM reads and pipeline stages for Single-Row per Read. 30
3.8 The MViD architecture. 37
3.9 The breakdown of LPDDR4 peak power and MViD power consumption. 40
3.10 The impacts of MViD on DS2 in performance and energy-delay product (EDP). 44
3.11 The impacts of MViD when DS2 is running solely or with non-MV-mul workloads. 45
3.12 The impacts of bank partitioning when DS2 is running with non-MV-mul workloads (mix-high). 48
3.13 The impacts of one-rank MViD on MV-mul by changing the size of the weight matrix. 49

4.1 Exemplar GnR in the baseline and the evaluated NDP architectures. 53
4.2 Speedup and DRAM energy breakdown of baseline system and the state-of-the-art NDP accelerators when performing GnR. . 54
4.3 High-level overview of TRiM-G/B. 59
4.4 Various methods for transferring C-instr to the memory node in the TRiM architecture. 62
4.5 C/A bandwidth requirement to utilize all memory nodes for each TRiM architecture and bandwidth provision for each method of transferring C-instr. 64

4.6 Heatmaps showing the speedup of TRiM-R/G/B over baseline system. 66
4.7 Overall architecture of TRiM-G. 69
4.8 Distribution of the load imbalance ratio. 71
4.9 Execution flow of lookup request distribution through hot-entry replication. 71
4.10 An overall execution flow of the lookup requests on the host-side. 73
4.11 The GnR speedup of the TRiM architectures. 78
4.12 GnR speedup and relative DRAM energy consumption of the baseline system and the TRiM architectures, and energy consumption breakdown of TRiM-G. 78
4.13 Speedup of TRiM-G over the baseline system with various p_{hot} and N_{GnR} values. 81

Chapter 1

Introduction

Deep neural networks (DNNs) are widely used due to their significant impact [36]. A wide range of tasks can be handled by DNNs, from image classification [47] and speech recognition [8] to personalized recommendations [123] and graph processing [70]. To handle these tasks, various DNN models are used, such as convolutional neural network (CNN) [71], recurrent neural network (RNN) [19], transformer-based models [114], and recommendation models (RMs) [90].

The computational characteristic of a neural network model is determined by that of the layers composing the model. Convolutional (CONV) layer and attention layer from transformer have high reuse of data which is stored in on-chip storage of a system [15, 53]. These layers show compute-intensive property with high operational intensity, which represents a large amount of computation compared to the number of off-chip accesses. By contrast, the activa-

---

This chapter is based on [65, 66, 99].

©2020 IEEE. Reprinted, with permission, from Byeongho Kim, Jongwook Chung, Eojin Lee, Wonkyung Jung, Sunjung Lee, Jaewan Choi, Jaehyun Park, Minbok Wi, Sukhan Lee, and Jung Ho Ahn, "MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks," IEEE Transactions on Computers, April 2020.

"TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory" ©2021 by Byeongho Kim, Jaehyun Park, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1145/3466752.3480080.

"TRiM: Tensor Reduction in Memory" ©2020 by Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1109/LCA.2020.3042805.

tion layer (e.g., ReLU), batch normalization layer, and embedding layer show memory-intensive property with low data locality, exhibiting low operational intensity [61, 63].

The computational characteristic of a layer may vary depending on the use case of the network model. For example, when models composed of RNN layers or FC layers are running with a small batch size for real-time inference, the layers show memory-intensive behavior as weight matrices are hardly reused. However, when a large batch is used for training, the layers exhibit compute-intensive characteristics as input data and weight matrices are highly reused [33].

Due to the memory wall problem [119], the increase in main-memory bandwidth is relatively low compared to that in the computational power of the microprocessor within the same period [85]. Therefore, the performance improvement of the memory-intensive layers is lower than that of the compute-intensive layers, which necessitates the solution for accelerating the memory-intensive layers.

In this dissertation, we propose two processing-in-memory architectures to accelerate memory-intensive layers and models. By locating the processing elements near the DRAM datapath, processing elements can operate independently. Thus the PIM architectures can utilize expanded internal memory bandwidth the same as the channel bandwidth multiplied by the number of processing elements. With the ample internal bandwidth provided by the PIM architectures, memory-intensive layers can be accelerated. To further accelerate the PIM architectures, we apply optimizations that exploit the characteristics of models' use cases and their workloads.

1.1 Accelerating RNNs on Edge

On edge devices such as mobile or IoT devices, Neural Networks (NNs) are frequently used for applications interacting with users. For example, Natural Language Processing (NLP) is widely used for devices such as Siri, Google Assistant, and Amazon Echo, and expected to become a key application in mobile in the near future. RNN [107] is crucial for NLP [8, 38]. Currently, edge devices typically rely on datacenters in performing a majority of RNN inference [33, 60, 124]. However, due to latency and energy burdens in transferring raw data for RNN inference over the networks, there is significant merit to perform RNN inference within the edge devices.

RNNs have many variants (e.g., LSTM [48] and GRU [20]), all of which spend most of the execution time for matrix-vector multiplication (MV-mul). A characteristic of MV-mul is that matrix elements, which correspond to weights in RNN, are used only once per vector, resulting in a limited degree of data reuse. The on-chip storage of edge devices, sized up to several megabytes (MB) [43], is typically too small to store all the weights whose sizes are reaching tens to hundreds of MB. Therefore, for each MV-mul, the matrix elements of RNN must be read from the main-memory DRAM. This is in sharp contrast to the characteristics of CNN, which has a high degree of data reuse for an input feature map while processing numerous weight channels [16].

When a matrix does not fit in on-chip memory, its MV-mul performance depends on DRAM bandwidth [33, 45, 60], which is much lower than the on-chip storage bandwidth of a processor. Also, in energy efficiency, the data transfer energy between a DRAM device and a processor dominates that of other

operation types (e.g., multiply-accumulate (MAC) operations) [49]. Thus, the performance and energy efficiency of RNN inference is determined by the bandwidth and energy efficiency (J/b) of main-memory DRAM.

To solve the memory bottleneck and reduce J/b of DRAM accesses, there have been many near-data processing studies [11, 30, 34, 50, 63, 68, 78, 110]. They reduced off-chip memory traffic by adding operation units or accelerators near main memory, either within a DRAM die or a logic layer of a 3D-stacked memory. However, they do not fully utilize the internal DRAM bandwidth or do not consider the power constraint of DRAM. Also, they cannot perform acceleration operations and other memory requests from processors simultaneously, so they hardly function as the main memory.

In this paper, we propose the MViD (MV-mul in DRAM) architecture, which can handle both MV-mul operations and memory requests from processors (processor requests) under the maximum power budget of DRAM devices which is determined by normal DRAM operations. MViD places MAC units near the datapath I/O within DRAM banks to improve the performance of MV-mul by utilizing abundant DRAM internal bandwidth. In particular, we analyze the operational limit due to the DRAM power constraint and confirm that MAC units can be added only to half of the banks for the current mobile DRAM standard, LPDDR4 [76].

To further reduce the total amount of computation, MViD performs a sparse MV-mul operation exploiting sparsity and quantization. Processing sparse MV-mul is actively studied in various ways depending on the characteristics of the matrix [45, 46, 104, 122], and its effectiveness depends on finding an appropriate sparse data format by understanding the characteristics of the matrix.

To find the optimum data format for MViD, we explore its design space by adjusting sparsity and quantization bits [45] of RNN weight matrices without accuracy loss. Also, we explore possible mappings of a weight matrix to a DRAM page.

MViD can act as the main memory by allowing other memory requests from a processor to be processed during MV-mul. MViD resolves the power and row-buffer conflicts that occur when MV-mul and processor-side requests are processed simultaneously by controlling the pace of MV-mul operation. For this, we implement MV-bank (bank performing MV-mul) control logic, which can slow down or pause the MV-mul operation. Also, MViD minimizes command/address path utilization overhead by leveraging the existing DRAM interface. MViD further improves the performance of MV-mul by placing the non-MV-mul workload data of the processor to the DRAM banks that do not perform MV-mul.

1.2 Accelerating Recommendation Model

Personalized recommendation systems aim to provide the content preferred by users based on their experience. Companies such as Facebook [42], YouTube [21], and Alibaba [115] are applying deep learning to their recommendation systems to maximize the accuracy of selecting user-preferred content. Recommendation systems based on deep learning (RecSys) have recently gained significant attention within the research community due to their industrial importance. For instance, Facebook states that recommendation systems (e.g., the Deep Learning Recommendation Model (DLRM) [42]) account for 80% of the AI inference

cycles in their datacenters.

A RecSys model utilizes the features of the target user and item to predict the click-through-rate (CTR), which indicates the probability of the user clicking the item (Figure 1.1). The input features consist of both sparse, categorical features and dense, continuous features. A dense feature is a vector whose elements are floating-point numbers. The vector is passed through bottom fully-connected (FC) layers and is collectively translated to one intermediate vector. A sparse feature is represented as a sequence of indices. A number of vectors are fetched from an embedding table using the indices of a sparse feature, which are subsequently reduced to one intermediate vector using element-wise operations (henceforth referred to as tensor gather-and-reduction (GnR)). These intermediate vectors produced by GnR operations are combined through feature interaction, which is conducted by a batched matrix multiplication, and passed through top FC layers to produce a CTR.

Prior work [42] has shown that both FC layers and GnR take up a significant fraction of the end-to-end inference time of RecSys. While there have been

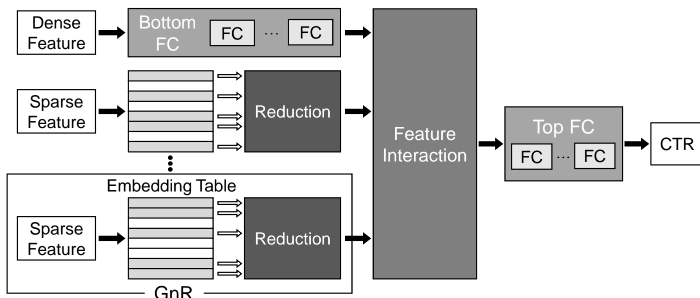


Figure 1.1: Simplified architecture of a RecSys model. We mainly target accelerating GnR (gather-and-reduction), which gathers embedding vectors from embedding tables and reduces them to produce one vector per table. FC and CTR stand for fully connected layer and click-through-rate.

numerous studies focusing on accelerating FC layers, only a handful of prior works have explored the acceleration of GnR, whose characteristics significantly differ from those of FC layers.

Because the embedding table used in RecSys requires a large memory capacity, the embedding table must be stored in main memory. The performance of the GnR operation is constrained by the main-memory bandwidth as the vectors for the GnR operation are fetched from main memory. Increasing the memory bandwidth in a memory channel accompanies an increase in the number of pins or the frequency of the pins, both of which are highly costly. Consequently, TensorDIMM [73] and RecNMP [63] are two recent studies that explored the efficacy of near-data processing (NDP) for accelerating GnR. These architectures accelerate the GnR operation by placing processing elements (PEs) dedicated to each rank (adopting rank-level parallelism) in the buffer chip of DRAM, thereby utilizing the internal bandwidth which is equal to the channel bandwidth multiplied by the number of ranks in the channel. However, we observe that rank-level parallelism does not fully reap the maximum potential of NDP acceleration, leaving significant performance left on the table.

We propose an NDP-based GnR accelerator called the TRiM (Tensor Reduction in Memory) architecture, which is based on DDR4/5 DRAM. Our key approach is to utilize the hierarchical, tree topology of the DRAM datapath structure [23] to seamlessly extract additional internal bandwidth compared to rank-level-parallelism-based NDPs, improving the GnR performance. Compared to prior approaches, a key contribution of our study is the identification of effective embedding table mapping schemes when using multiple ranks/bank-groups/banks. In our work, we compare the internal bandwidth utilization

and energy efficiency in various mapping schemes. As with the existing DRAM interface, if command/address (C/A) signals are transferred only through the C/A path, we cannot fully utilize all PEs placed in/near the main-memory architecture. Therefore, we propose and analyze multiple C/A transfer schemes that can amplify the C/A signal bandwidth without significantly modifying the conventional DRAM interface. Among the embodiments of TRiM architectures, we find the optimal design that can effectively process GnR according to the workload characteristics.

Another important contribution is a detailed analysis of the load-imbalance issue that occurs when multiple ranks/bank-groups/banks process different numbers of lookups per GnR operation, a key limitation of NDP architectures including our design and RecNMP. To address this problem, we propose hot-entry replication by utilizing the workload characteristic according to which the number of accesses is skewed to a small fraction of entries in the embedding table in RecSys. This scheme improves load balancing without any additional modifications to the DRAM interface. We also develop a novel data reliability solution for TRiM architectures that cannot exploit the conventional rank-level error correction code (ECC). Because GnR accesses the embedding tables in a read-only manner, we repurpose the existing on-die ECC [57] to only detect but not correct errors during GnR.

1.3 Research Contributions

In this dissertation, we make the following contributions:

- We propose MViD, which adds MAC units inside DRAM to solve the memory

bottleneck of RNN inference in edge devices and to improve energy efficiency.

- MViD deploys MAC units by carefully considering various constraints (internal bandwidth, power limit, and off-chip bandwidth) that could occur when adding MAC units in the current mobile DRAM standard. For the first time to the best of our knowledge, MViD can perform processor requests simultaneously with MV-mul; therefore, MViD functions as the main memory.
- Through sparse matrix formatting, quantization, and bank partitioning, MViD improves the throughput of inference in Deep Speech 2, up to 7.2× compared to the baseline system without MViD. MViD also guarantees that MV-mul operations do not hoist processor requests.
- We conduct a quantitative analysis of existing NDP solutions aimed at accelerating the GnR operation.
- We propose TRiM, an NDP architecture that accelerates the GnR operation by utilizing the features of the DRAM data/control path structure without causing significant changes in the conventional DRAM interface.
- We propose hot-entry replication utilizing the characteristics of the RecSys workload to alleviate the load imbalance problem that occurs in TRiM.
- The optimized TRiM architecture based on DDR4/5 DRAM improved the performance of GnR by up to 7.7× and 3.9×, respectively, compared to the baseline and state-of-the-art architectures.

1.4 Outline

The organization of this disseration is as follows. Chapter 2 describes the characteristics of memory-intensive machine learning operations and the organization and operation of our target memory device for PIM architecture, DRAM. In Chapter 3 and 4, we propose processing-in-memory architectures, MViD and TRiM, that accelerates mobile inference of RNNs and recommendation models. Chapter 5 describes a discussion about applying PIM microarchitectures to accelerate various memory-intensive applications. Related works are presented in Chapter 6. Conclusions and future works are presented in Chapter 7.

Chapter 2

Background

2.1 Memory-intensive Machine Learning Applications

Machine learning applications are used to handle various tasks such as computer vision, natural language processing, and personalized recommendation. Because these applications require a large amount of computation to process large datasets, they require a lot of computational and memory resources.

Machine learning applications consist of various arithmetic operations such as convolutional (CONV) layers, fully-connected (FC) layers, activation functions (e.g., ReLU), and embedding layers. The computational characteristics of these applications are determined by those of the operations constituting the applications.

The roofline performance model [117] shows the computational characteristics of the operation in computer systems. The X-axis of a roofline graph rep-

This chapter is based on [65, 66, 99].

©2020 IEEE. Reprinted, with permission, from Byeongho Kim, Jongwook Chung, Eojin Lee, Wonkyung Jung, Sunjung Lee, Jaewan Choi, Jaehyun Park, Minbok Wi, Sukhan Lee, and Jung Ho Ahn, "MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks," IEEE Transactions on Computers, April 2020.

"TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory" ©2021 by Byeongho Kim, Jaehyun Park, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1145/3466752.3480080.

"TRiM: Tensor Reduction in Memory" ©2020 by Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1109/LCA.2020.3042805.

resents the arithmetic intensity, the number of arithmetic operations per memory access. The Y-axis represents the number of arithmetic operations per second. The performance of the operation on the region under a diagonal line is limited by memory bandwidth, where the slope of the diagonal line means the memory bandwidth. The performance of the operation on the region under a horizontal line is limited by the peak arithmetic performance of the processor.

Through the roofline analysis as shown in Figure 2.1, it is possible to identify the performance bottleneck of the operation for machine learning. Operations such as a CONV operation and its base operation, MatMul (GEMM), have high arithmetic intensity, and execution time is affected by the computing power of the underlying system. These operations are called compute-intensive operations. Conversely, matrix-vector multiplication (GEMV) and embedding operation (Embedding) have low arithmetic intensity, and the execution time of

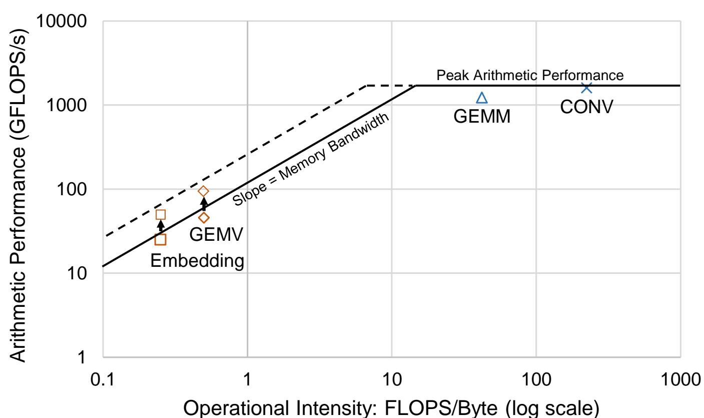


Figure 2.1: Roofline model for four arithmetic operations, where the performance ceiling is based on Intel Xeon Gold 6138 [27] with six memory channels, each with DDR4-2400 modules achieving up to 115.2GB/s. The performance of the memory-intensive operation improves if the memory bandwidth of the system increases.

those operations is affected by memory bandwidth. These operations are called memory-intensive operations.

Various companies emphasized that there is a high demand for accelerating memory-intensive operations when running machine learning applications on edge devices [13] and datacenters [33, 60]. Facebook [9, 24] reported that the server usage for the recommendation model, which is mainly composed of the embedding operations, accounts for more than 70% of the total machine learning inference server usage and has increased about four times in two years. Also, Facebook emphasized the acceleration of the recommendation model through many studies [24, 29, 63, 64, 100, 109, 116]. Google [13] highlights the need for accelerating matrix-vector multiplication on edge devices running RNN models and Transducer models.

As shown in Figure 2.1, increasing the memory bandwidth of the system improves the performance of memory-intensive operations. However, due to the memory wall problem [119], where the memory bandwidth improvement is much lower than the computing power improvement, a solution that simply increases memory bandwidth of the system is costly, and the computer system is bottlenecked by the memory subsystem. To overcome the memory wall problem, we introduce Processing-in-Memory (PIM) microarchitectures that can additionally utilize memory bandwidth.

2.2 DRAM Organization and Operations

It is necessary to understand how a DRAM device is organized and operates to analyze its power constraint and further improve the performance by leveraging

the internal data transfer bandwidth.

The datapath of popular main-memory systems such as DDR4/5 DRAM [54, 57] takes a hierarchical tree structure (see Figure 2.2(a)), i.e., a memory channel as a root node (depth-0) consisting of a primary host memory controller (MC) and multiple, secondary DRAM ranks (depth-1) connected through a depth-1 data bus. Each rank consists of several DRAM chips, all receiving the same command/address information by broadcasting and transferring the corresponding data accordingly. The datapath entering the rank is physically

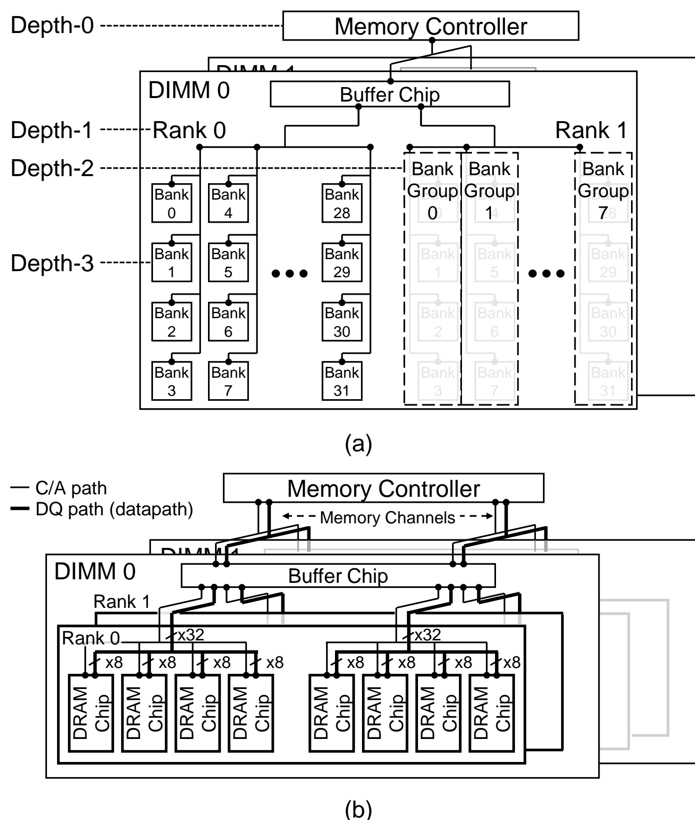


Figure 2.2: (a) Simplified DRAM datapath assuming a single memory channel and (b) module organization of DDR5 DRAM with 2 DIMMs × 2 ranks connecting two memory channels.

divided and connected to each DRAM chip (see Figure 2.2(b)). To maximize memory-level parallelism, several ranks can physically be housed within a DIMM module.

In DDR5 DRAM, a rank consists of eight bank-groups (depth-2), each packed with four banks (depth-3). Similar to the depth-1 data bus, the depth-2 data bus and the depth-3 data bus are shared between one rank and eight bank-groups, and between one bank-group and four banks, respectively. Each bank consists of a 2D array of DRAM cells; each DRAM page (often called row) is controlled by a wordline (WL), and each column is connected to local bitlines (BLs). The sequence of a DRAM read is as follows. 1) Activation (`ACT`): the cells of a selected DRAM page in a bank driven through a WL share charges with the corresponding BLs, and bitline sense amplifiers (BLSAs) detect the small voltage difference of BLs due to charge sharing and amplify it. 2) Read (`RD`): a portion of column bits of data latched in the BLSA are transferred to GIO SAs through the global I/O. The data is amplified again in the GIO SA and transferred to the I/O multiplexer (mux) through inter-bank datalines and leaves the DRAM die. Reading the data in another column of an activated page needs only `RD`. To access data at a page other than the currently activated page in a bank, we should precharge BLs and BLSAs (`PRE`) prior to `ACT` and `RD`. The minimum time interval from `ACT` to `PRE` is tRAS, the time to restore data to the cells.

All of the banks in a DRAM chip operate independently, albeit only one bank can occupy depth-1/2/3 data buses at any given time. Therefore, `RD` can be issued at a minimum tCCD (column to column delay) interval to prevent a conflict in the data buses (often called inter-bank datalines). The time interval between two `ACT`s within a channel is limited to tRRD (row to row activation

delay). `ACT` can be performed simultaneously on multiple banks, but only four `ACT`s within tFAW (four activation window) can be issued due to the power limit.

The notion of a bank-group did not exist until DDR4. DDR4/5 DRAM started adopting such a concept to retain the frequency of DRAM banks at a low level while increasing the data transfer rate and hence the bandwidth. Because the frequency inside a bank-group bus is lower than that outside a bank-group, the consecutive read delay within a bank-group (tCCD_L) is greater than that between the bank-groups (tCCD_S). The bank-group organizes a level of the hierarchy between the ranks and banks, connecting hierarchical multi-drop buses among them.

DDR5 DRAM allows one DIMM to be connected to two channels, giving it a physically different configuration from previous DDR generations where one DIMM is connected to one channel. However, as the two channels operate independently, it can be seen that they have separate data/control paths.

The control path of the main-memory systems is structured similarly. An MC sends command and address (C/A) signals to all connected DRAM ranks through a C/A bus in a broadcast manner. Each rank drops the signals if it is not the right destination. A buffer chip or chipset connects an MC and the ranks in a module to alleviate signal integrity issues [56].

Processing-in-Memory (PIM) architecture is an architecture that processes operations near the DRAM cell with the higher internal bandwidth by utilizing the characteristics of the DRAM datapath. Because DRAM banks can operate independently, if the processing elements (PEs) are located near the DRAM datapath and these PEs do not share the datapath, PEs can operate indepen-

dently. Thus, the operations can be processed utilizing the bandwidth the same as the channel bandwidth multiplied by the number of PEs, and we suggest PIM microarchitectures to accelerate bandwidth-hungry operations.

As the baseline memory for Section 3, we used an LPDDR4 DRAM device [55] which is composed of two memory channels. Multiple LPDDR4 dies can be connected (e.g., through die stacking) to form a multi-rank package with two channels. For the baseline memory used in Section 4, we assume a data/control path dedicated to each rank in the buffer chip as in MCN DIMM [7], which is also assumed in the previous studies [63,73] (Figure 2.2(b)).

Chapter 3

MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks

3.1 Background and Motivation

3.1.1 Energy-efficient RNN Mobile Inference

RNN inference is mostly performed in datacenters yet [33, 60]; however, to reduce the latency and improve the energy efficiency of serving RNNs, there is a strong demand for conducting RNN inference closer to service requests, such as in mobile and IoT devices. These devices typically do not have large on-chip memory due to power and cost issues.

RNNs have several variants, such as LSTM (long short-term memory) and GRU (gated recurrent unit), depending on the existence of certain gate types.

This chapter is based on [65, 66, 99].

©2020 IEEE. Reprinted, with permission, from Byeongho Kim, Jongwook Chung, Eojin Lee, Wonkyung Jung, Sunjung Lee, Jaewan Choi, Jaehyun Park, Minbok Wi, Sukhan Lee, and Jung Ho Ahn, "MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks," IEEE Transactions on Computers, April 2020.

"TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory" ©2021 by Byeongho Kim, Jaehyun Park, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1145/3466752.3480080.

"TRiM: Tensor Reduction in Memory" ©2020 by Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1109/LCA.2020.3042805.

RNN operation consists mainly of matrix-vector multiplication (MV-mul), element-wise sum/multiply, and activation function such as tanh or sigmoid, but MV-mul dominates the execution time. For example, a typical configuration of Deep Speech 2 (DS2) [8], a popular RNN benchmark performing end-to-end speech recognition, is composed of two convolutional layers and five GRU layers where the size of most GRU weight matrices is 1600×1600. When running on an Intel Skylake-based server [27], the FLOP and execution time of MV-mul take 86.8% and 88.6% of the total (see Figure 3.1). The profiling results in the latest GPUs [32] are also similar.

Single GRU layer has six matrices (three for processing an input vector and the other three for processing the vector computed in the previous time step),

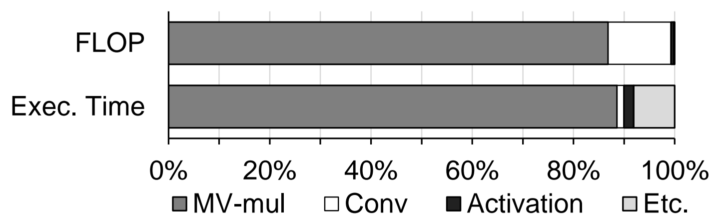


Figure 3.1: The FLOP and execution time breakdown of Deep Speech 2 [8] in an Intel Skylake-based server [27].

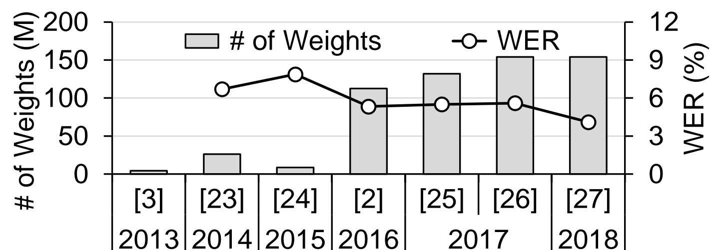


Figure 3.2: The aggregate size of weight matrices and word error rate (WER) of representative ASR models over time [8, 17, 37, 38, 44, 86, 106]. Not all models use the same data set.

and the size of each matrix is mostly 1600×1600 with DS2. Assuming that the size of each element is 16 bits [45], the size of a GRU layer is 30.7 MB. When we perform whole RNN inference, it executes five GRU layers and needs 145 MB (the size of three matrices of the first GRU layer is slightly smaller than the other ones) of storage for weight matrices, which is too large to fit in the on-chip memory (mostly up to several MBs) of mobile/IoT devices. In MV-mul, each vector element is reused by the number of rows within the weight matrix, whereas each weight element is used only once. The on-chip memory size of processors grows over time, but the aggregate size of weight matrices increases as well for better recognition quality (see Figure 3.2). Therefore, the weight elements, which cannot be reused for RNN inference, must be fetched from main-memory DRAM devices.[1]

From an energy efficiency perspective (i.e., J/b), reading data (several to dozens of bits) from a DRAM device requires orders of magnitude more energy than performing a multiply-accumulate (MAC) operation or reading the same amount of data from an SRAM with few KBs of capacity [16, 49]. From a performance perspective, the bandwidth of off-chip DRAM devices is much lower than that of on-chip memory in a processor, making the execution time of MV-mul limited by off-chip DRAM bandwidth. Therefore, it is critical to reduce the energy consumption and increase the bandwidth of main-memory DRAM accesses in improving the energy efficiency and performance of RNN inference.

[1]We can reuse weight elements if we perform RNN inference over a batch of input or a group of time steps. However, mobile/IoT devices typically process one or few time steps of a request at a time to reduce service latency, limiting the degree of this reuse. We interchange the terms of a DRAM device and a DRAM die.

3.1.2 How to Improve the Energy Efficiency and Bandwidth of DRAM Accesses in MV-mul

The best way to improve the energy efficiency of multiplying a large matrix (located at main-memory DRAM) with a vector is to perform all or most of MV-mul within DRAM dies. If a processor performs MV-mul by fetching weight elements from a separate main-memory DRAM device, inter-die communication consumes most of operating cost (data transfer energy) and capital cost (large silicon/packaging area, such as pins, pads, and TSVs) regardless of the ways interconnecting processor and DRAM dies, such as conventional package, silicon interposer (2.5D [58]), and face-to-face die stacking (3D) using TSVs [96].

If we perform MV-mul within a DRAM device but close to its inter-die I/O, we can save inter-die communication energy. However, this does not provide any performance gain unless we utilize multiple DRAM dies within a memory rank [11]. A way to achieve an additional performance benefit in MV-mul is to exploit a structural unit within a DRAM device, bank. A DRAM die has several to few dozens of banks, where each bank can operate independently and concurrently. Therefore, by conducting MV-mul in the edge of bank's datapath I/O, we can improve both performance (multiplied by the number of banks) and energy efficiency (by not dissipating energy for moving weight matrices across banks, which is a significant portion of DRAM read energy according to [95]).

Performing MV-mul on each DRAM bank saves the data transfer energy of weight elements; but, if the vector elements are from a processor, they should traverse farther through inter-die I/O and inter-bank datapath to reach the DRAM bank. Hence, one might regard that performing MV-mul on a bank

provides no energy saving. However, as described in Section 3.1.1, 1) vector elements are reused by the number of matrix rows and 2) a vector is much smaller than a weight matrix (by the number of matrix rows), so we can save most of the energy for fetching a vector by providing a small storage per bank to store the vector elements.

Still, there exists an important feasibility issue in performing MV-mul at all the banks of a DRAM device. DRAM banks can operate in parallel; however, as mentioned in Section 2.2, their operations are limited by various timing constraints. This limitation is because all banks share one or few inter-die I/O paths (called channels), which determine the maximum rate of meaningful activity (related to the data store, retrieval, and retention, which are the primary purposes of DRAM as main memory). Therefore, a DRAM die has a power limitation, which determines the design of its power delivery networks, such as the number of power and ground pins and the density of on-chip power/ground wires.

The cost of DRAM is very sensitive to this power limitation because a DRAM device uses few (typically two to three) metal layers, and hence populating more power/ground wires and pins would significantly increase fabrication cost and die size. If all the DRAM banks perform bursty data read operations for MV-mul, a DRAM device can dissipate power exceeding the aforementioned power limit. For example, on the mobile LPDDR4 DRAM device we use as a baseline, concurrently reading data from all DRAM banks to their global-I/O (GIO) sense amplifiers (SAs) would dissipate 1.7× of its power limitation. As an edge device executes applications other than RNNs as well, it is a huge waste of precious DRAM resources to design a DRAM device with a higher

power budget only for MV-mul. Therefore, it is desired to perform MV-mul under the maximum power budget determined by normal DRAM operations.

As explained in Section 2.2, DRAM reaches its peak power consumption level when each channel of DRAM performs four `ACT`s in tFAW and multiple `RD`s at tCCD interval. If we perform a DRAM read but omit inter-bank and inter-die datapath by supporting the data to the MAC units next to GIO SAs, we can save energy. According to the detailed analysis later explained in Section 3.3.1, we can conduct up to four `RD`s simultaneously when performing MV-mul within DRAM banks without exceeding the peak power consumption level.

3.2 MV-mul in DRAM

3.2.1 Exploiting Quantization and Sparsity in RNN's Matrix Elements

Under this DRAM power constraint, to further reduce the total amount of computation, we exploit a characteristic of RNN that a significant portion of its weight elements is redundant. As mentioned in [45], even if we turn a majority of the weight elements of an RNN into zeros, the impact of this pruning on the RNN inference is negligible. We verify this by conducting an experiment on DS2 [8], whose network model size is specified in Section 3.1.1 with LibriSpeech dataset [98], following the pruning strategy explained in [89]. [89] prunes down the values below certain thresholds into zeros, after every few training epochs. From the experiment, we observed no significant surge in word/character error rates with the sparsity (the portion of zero values within a matrix) of the matrices being 25%, 50%, and 75%.

When we compare a sparse matrix format (specifying non-zero elements

(NZs) with their positions) with a dense matrix format (listing all the matrix elements), if sparsity is higher than (position bits) / (data bits + position bits), the sparse matrix format represents a matrix with fewer bits. Because reading weight matrices dominates the energy and the execution time of MV-mul, it

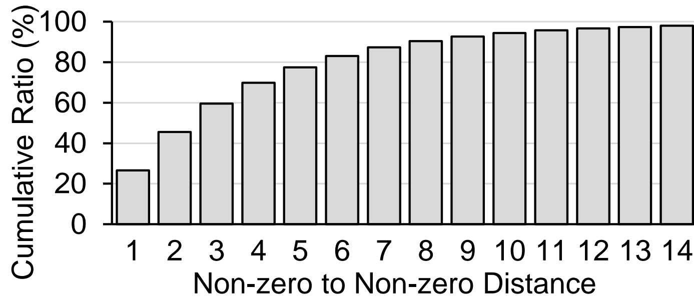
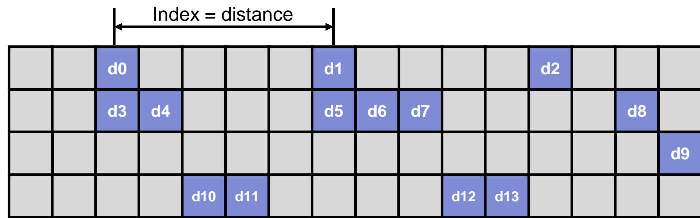


Figure 3.3: Distribution of the distance between two adjacent non-zero values (NZs) in the matrices of GRU layers (75% sparsity) in Deep Speech 2 [8]. More than 98% of NZs have a distance of less than 15.



(a) An example of a sparse matrix

Data	idx	Data	idx	Data	idx	Data	idx	Data	idx	...
d0	0x2	d1	0x4	d2	0x4	0	0xF			
d3	0x2	d4	0x0	d5	0x3	d6	0x0	d7	0x0	...
0	0xE	d9	0x0	2	0xF					
d10	0x4	d11	0x0	d12	0x4	d13	0x0	3	0xF	...

(b) Its sparse matrix representation

Figure 3.4: (a) An exemplary sparse matrix and (b) its representation using the delta encoding format. dN denotes the N-th non-zero value of a matrix in row-major order. Each row in the representation table includes up to one row of the sparse matrix.

is beneficial to exploit the sparse matrix formats for matrices whose sparsity values are above this threshold. There are several formats for representing a sparse matrix, exemplified by Compressed Sparse Row/Column (CSR/CSC), Coordinate (COO), and Diagonal (DIA) [12].

The distribution of NZs in a matrix determines the size efficiency (bytes per NZ) of a specific format. For RNN applications such as DS2, the NZs of the weight matrices are relatively uniform-randomly distributed (see Figure 3.3). In this case, there is no significant difference in the bytes per NZ between sparse formats except for the formats favoring special distributions. In this paper, we propose delta encoding, a variant of CSR.

CSR stores the absolute column indices of NZs and the number of NZs per row, together with the NZ values. By contrast, our delta encoding stores the column distance from the previous NZ as an index to represent the matrix with fewer index bits in storing (data, index) pairs (see Figure 3.4). Also, instead of storing the number of NZs per row, we reserve the maximum value which can be represented by an index to specify the end of a row. When the index indicates the end of a row, the corresponding data field stores the absolute address of the row. For example, if we dedicate four bits per index, we define 0xf (the maximum distance) as the end of a row and use the values from 0x0 to 0xe as the distance between two NZs. If the distance is greater than 0xe, we place one or more dummies (whose data value in the (data, index) pair is zero).

In addition to exploiting sparsity through pruning, we also apply quantization for a further reduction in size representing the matrix. When we applied both pruning and quantization to DS2, we observed that reducing the precision of data till 12 bits incurs no noticeable degradation in inference accuracy, which

is consistent with the results reported in [45].

In our sparse matrix format, the frequency of dummy values added depends on the size of index bits and the sparsity of a matrix. We study an optimal index bit minimizing the total size of a matrix that is formatted at each sparsity value using the GRU layers of DS2. The experiment shows that using (12 bits, 4 bits)

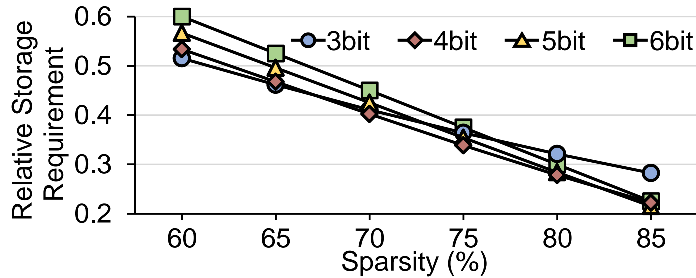


Figure 3.5: The storage requirement for a matrix with different index bits when using our sparse matrix encoding format (delta encoding). The storage requirement is compared to that of a dense matrix without pruning.

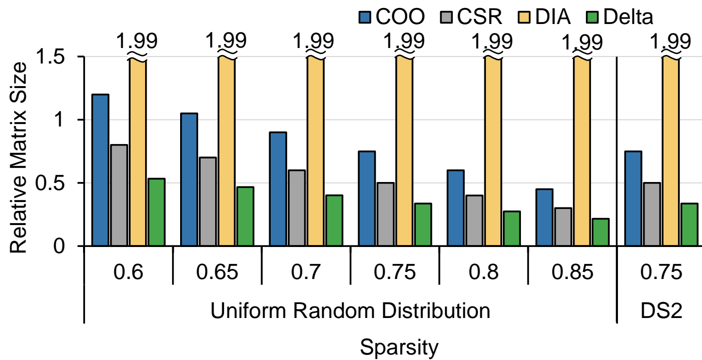


Figure 3.6: The relative size of the tested matrices encoded with various sparse matrix formats normalized to the size of the dense format [12]. We set the size of the dense format as the baseline per matrix. The matrices are categorized into two types: ones synthesized with the given sparsity values under a uniform random distribution and the other collected from the real dataset (a GRU layer of DS2), which has a sparsity of 75%. Our delta encoding format gives the smallest size per NZ among all the tested formats in our target sparsity ranges.

for (data, index) pairs is optimal for the sparsity range of 65% to 80%, where the portion of dummy pairs is less than 5% (see Figure 3.5). As a result, we assume that a (data, index) pair is (12 bits, 4 bits), and matrices are 1600×1600 with a sparsity of 75% by default in this paper.

To verify the effectiveness of our delta encoding format on the sparse matrices, we compared the size of sparse matrices encoded with various sparse formats. From the experiment, the delta encoding format reduces the size of matrices by 32.7% compared to the CSR format for the DS2 dataset with the sparsity of 75% (see Figure 3.6).

By applying quantization and sparse formatting, the total size of weight matrices in the DS2 decreases, but it is still 36 MB, which would not fit on-chip caches of the mobile devices. Therefore, it is effective to perform MV-mul in DRAM; even if the size of matrices and the (data, index) pairs would vary as ASR applications evolve, we believe that the qualitative observation and analysis of our proposed MV-mul accelerator would stay unchanged.

3.2.2 The Operation Sequence of MV-mul in DRAM

To perform MV-mul within DRAM banks, we should locate a weight matrix and an input vector within banks capable of MV-mul, called MV-banks. As we can conduct four `RD`s simultaneously within banks due to the power constraint, each channel in a DRAM device has four MV-banks. So the weight matrix is divided into four sub-matrices and distributed to the MV-banks. As the input vector is reused and much smaller than the matrix, we duplicate the input vector to each MV-bank equipped with a dedicated SRAM (iv-SRAM). An alternative is to use another DRAM bank or another row in the same DRAM

bank for storing the input vector. However, when processing sparse matrix-vector multiplication, accesses to the input vector are not sequential (consecutive). Therefore, it is required to access a larger size of DRAM data than is needed for SRAM because the granularity of a DRAM read (256 bits) is much coarser than that of an SRAM read (12 bits); it is not efficient in terms of throughput and energy.

The power and area overheads of iv-SRAM depend on the number of entries and the size of each entry. The input vector is stored in a dense format, making iv-SRAM sized to hold 1,600 entries, each storing 12-bit value. Its area is equivalent to 2.07MB of DRAM cells (1.61% of a 128MB DRAM bank), and it dissipates just 1.9% of the total power for MV-mul. A more detailed power, area, and timing analysis is described in Section 3.3.1. Each MV-bank also has a separate SRAM to hold an output vector (ov-SRAM), where each element is 24-bit long. In our case of ov-SRAM with 400 entries, the area corresponds to 0.3 MB DRAM cells, and its power consumption is negligible because ov-SRAM has low utilization.

The main operation sequence of MV-mul is to 1) write an input vector to the iv-SRAM in MV-banks, 2) perform MV-mul in each MV-bank by sequentially performing the inner-product computation, and 3) read the generated output vector from the ov-SRAMs. We assume that the matrix elements are already stored in each MV-bank in the form of our sparse matrix format. The detailed operation sequence is as follows. First, writing the input vector is processed through broadcasting, because all iv-SRAMs in a DRAM device need the same input vector. The input vector from the processor is sent to each bank and stored in iv-SRAM simultaneously.

The core inner-product computation sequence in each MV-bank is as follows. **Data reading** reads data from an activated DRAM page. An LPDDR4 device stores 256 bits in GIO SAs in one read. For the size of a weight element being 16 bits, there are 16 elements per DRAM `RD`. **Index decoding** process is required as the weight element is encoded using a sparse format. The index refers to the relative column distance from the previous NZ location minus one. Therefore, we can find the absolute address needed for fetching a proper input vector element by accumulating the index values. The index of the first element represents the index of the first NZ data. Moreover, a 4-bit compare logic is required to identify the row end index (index value 0xf). **Input vector fetching** is performed using the absolute addresses, each pointing to one of the 16 weight elements. iv-SRAM serves 16 weight elements for decoded addresses within tCCD. **MAC executing** process conducts inner product through MAC units populated within an MV-bank. Because 16 weight elements and vector elements are fetched every tCCD, 16 MAC operations must be performed. We designed and compared various MAC units with different numbers of pipeline stages (more details in Section 3.3.1). We identified that populating 16 MAC units having a cycle time of tCCD is better than populating fewer MAC units with a shorter cycle time (through aggressive pipelining) in power and area perspectives. Therefore, we place 16 MAC units next to the GIO SAs of an MV-bank. **Output storing** process stores the result of the MAC operation in the ov-SRAM only when the row end index exists at the decoded index through the index decoder. The output store method depends on how the matrix elements are mapped to DRAM pages, as described below.

When MV-mul is finished in all the participating MV-banks, the results in

ov-SRAM are transferred to the processor. The activation function is done by the processor; because the output vector is small in size, it gives little benefit of implementing the function within a DRAM die.

There are multiple ways to map a weight matrix represented with our delta encoding format to DRAM pages. As mapping changes, organization and operation sequence change accordingly. Considering that, we choose a mapping that we call **Single-Row per Read (SR)**, where elements from a single DRAM read (256 bits) belong to one matrix row (see Figure 3.7).

The elements fetched on GIO SAs belong to one matrix row so that index decoding can be done sequentially using one accumulator or through parallel prefix sum. It is not possible to sequentially decode 16 indices within tCCD using one accumulator, so we use a 6-level parallel prefix sum unit. The column indices from the parallel prefix sum unit are used to fetch the input vector values from the iv-SRAM. Each MAC unit accumulates the product of an input vector element and a weight element until the DRAM read includes the end of a matrix row. If the current DRAM read includes the end of a row, the partial sums stored in all the MAC units of an MV-bank are accumulated through an adder tree, which takes 5 ns (same as tCCD).

SR is inefficient in that on average a half of weight elements in a DRAM

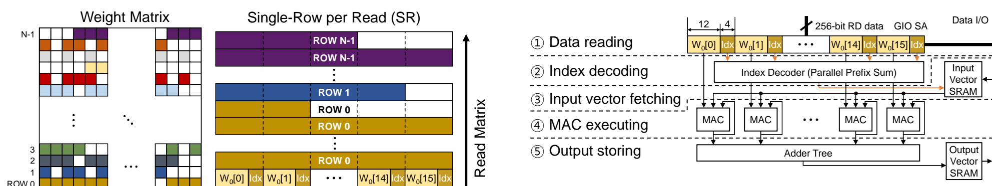


Figure 3.7: Mapping of the weight matrix to DRAM reads and pipeline stages for Single-Row per Read.

read (8 for LPDDR4) are padded with zero values per matrix row (performance degradation about 2% compared to an ideal configuration where MAC units are fully occupied in a 1,600×1,600 matrix). So we may consider other mapping types where the elements from a single DRAM read might belong to different matrix rows. However, from the perspective of implementation, the latter mapping types require higher hardware complexity. Hence, we choose SR as a mapping of our delta encoding format.

In utilizing this MViD architecture, we should consider how a processor controls MViD. For a DS2 matrix we use for evaluation, 1.25 MB of matrix data distributed in each bank must be read and multiplied with an input vector. If all the DRAM `RD` commands for operations are transferred from the memory controller (MC), it will cause a huge CA bus utilization overhead. Therefore, a single MV-mul command, including the size of the initial address and the weight matrix is passed to the MV-bank to perform all MV-mul operations.

MV-banks process MV-mul through taking commands that write/read the vector to/from the input/output SRAM (`WR-iv`, `RD-ov`). To define these two commands, we used the reserved-for-future-use (RFU) commands in the LPDDR4 specification [55]. `WR-iv` command broadcasts the same input vector to all MV-banks. The input vector broadcasted through inter-bank datalines is simultaneously written to the iv-SRAMs in the MV-banks. Once the broadcast is over, MV-mul can be started. Therefore, there is no separate DRAM command for starting MV-mul. `RD-ov` should be processed after the entire MV-mul operation is completed. Because conventional DRAM interfaces do not have a mechanism to notify the end of a specific operation to MC, the primary device in the bus interface, the MC periodically sends `RD-ov` to check if MV-mul is

completed (polling). This requires the MV-bank control unit (MCU) checking the status of the MV-bank; MCU is located close to the inter-die I/O within a DRAM device. If MV-mul is over, MCU returns a predefined value for `RD-ov`. Then, the MC issues `RD-ov` against to retrieve the output vector values.

3.2.3 Concurrently Serving Requests from Processors and Performing MV-mul in DRAM

A computer system that performs RNN inference can simultaneously run other applications. Therefore, the performance of the other applications can be drastically reduced if MV-mul operations within DRAM block the other memory requests. For example, in a system stacking four LPDDR-3200 dies each with eight MV-banks, the minimum time required to read 1.25MB of data and perform MV-mul is 6.125 μs if all MAC units are used for one MV-mul. If DRAM devices do not accept another memory request during the MV-mul operation, the processor would wait for the main memory response at least for 6.125 μs. Therefore, a DRAM device with MV-banks must provide a mechanism to service other memory requests while performing MV-mul.

Careful coordination is required for DRAM to serve both MV-mul and memory requests from a processor (processor requests) concurrently. Because performing MV-mul in MV-banks dissipates the maximum DRAM power, this power limit must be considered to serve processor requests. Moreover, row-buffer conflicts, which occur due to a request to DRAM row different from the active row, must be considered if a processor request heads to an MV-bank that is actively performing MV-mul.

Method for achieving concurrency: We achieve the coordination by sending a

single MV-mul command (composed of `WR-iv` and `RD-ov` explained in Section 3.2.2) to the DRAM and by controlling the progress of MV-mul inside the DRAM. This coordination requires control generator units (CGUs) to convert a single MV-mul command into a sequence of fine-grained sub-commands. A CGU is located at each MV-bank and generates DRAM `ACT`, `PRE`, and `RD` commands to read weight elements, compute vector indices, and perform MAC operations, obeying DRAM's timing constraints.

CGUs must slow down or pause the progress of MV-mul within MV-banks to serve processor requests. When a processor request reaches a non-MV-bank, the DRAM device should secure enough power budget. This can be achieved by slowing down the progress of MV-banks. According to our analysis (whose methodology explained in Section 3.3.1), slowing down all MV-banks into half of their original rate provides a sufficient power budget to process the processor requests with half of the rate a DRAM device can serve the requests. It can be achieved by increasing both tCCD and tRRD twice of their original values. By contrast, when a processor request reaches an MV-bank, we need to slow down all MV-banks and also pause the target MV-bank to deal with the row-buffer conflict.

Implementing commands for slow-down and pause: If a MC sends separate slow-down and pause commands prior to all memory requests, this would cause a huge CA bus utilization overhead. We alleviate this overhead by embedding the slow-down and pause information in existing DRAM commands.

First, the transmission of any normal DRAM command (i.e., `RD`, `WR`, `ACT`, and `PRE`) can be a signal to notify slow-down to MV-banks as all MV-banks must be slowed down to process normal DRAM commands to non-MV-banks.

Therefore, if MV-banks are not in a slow-down state when MCU receives any normal DRAM command, it broadcasts the slow-down command (`SD`) to the CGUs of all MV-banks and then sends the original DRAM command to the target bank. The timing of the original DRAM command should be increased by 3 tCK due to command decoding (1 tCK) and `SD` broadcast (2 tCK). This timing adjustment is only needed when none of the MV-banks are in a slow-down or pause state.

Second, `PRE` can be augmented to notify the target MV-bank to pause. To process a request from a processor on an MV-bank actively processing MV-mul, we must precharge the currently active DRAM row to resolve this row-buffer conflict. Therefore, we define a pause PRE command (`p-PRE`) by modifying `PRE` and use it to pause an MV-bank prior to `PRE`. The timing (latency) of `p-PRE` should be different from that of `PRE` (tRP) because a MC does not exactly know what operation is being performed in the MV-bank when it sends `p-PRE`. For example, if `p-PRE` is sent while the target MV-bank is activating a row for MV-mul, the precharge operation must be delayed by up to tRAS because the MV-bank must restore the row being activated. Therefore, in order not to violate the internal DRAM timing constraints, the timing of `p-PRE` must be at least tRAS+tRP (= tRC).

The worst case for `p-PRE` is when the target MV-bank just started `PRE` for MV-mul. There are two options for CGU in this case: one is to stop MV-mul and to directly enter the pause state when `PRE` is finished, and the other is to perform the planned `ACT` followed by `RD` for MV-mul and then `PRE` for pause. The former favors a processor request as the time of `p-PRE` can be tRC. The latter favors the progress of MV-mul, at the cost of a higher timing value of

p-PRE being tRC+tRP. We prefer the progress of MV-mul and set the timing of p-PRE to tRC+tRP, leaving the quantitative comparison of these two options as future work.

After all processor requests are processed, MV-banks must resume or return to the original processing throughput (speed-up). Because PRE is issued at the end of serving requests from a processor, we leverage PRE once again to notify speed-up (s-PRE) and resume (r-PRE). MC checks the state of each MV-bank and sends s-PRE or r-PRE command to MCU when speed-up or resume is needed. When MCU receives s-PRE, it sends the speed-up signal to the CGUs of all MV-banks and then sends PRE to the target bank. When MCU receives r-PRE, it is passed to the CGU of the target MV-bank, and the CGU resumes MV-mul after precharge. In LPDDR4, PRE has three unused CA bits; we use these to distinguish PRE from p-PRE, s-PRE, and r-PRE.

Policies for slow-down and pause: If we slow down or pause MV-banks every time MC receives a request, the throughput of MV-mul would be deteriorated significantly. To prevent such a throughput drop, MC checks the memory request queue at every time interval (tIV). When the aggregate number of pending requests over multiple tIVs surpasses a certain threshold (nTH), MV-mul is slowed down or paused. In determining whether to slow down or pause, MC checks the following metrics: an aggregated number of memory requests to normal banks (num_req_nonMV), and an aggregated number of memory requests to each MV-bank (num_req_MV[n]). The policy that we propose is:

- num_req_nonMV $\geqq$ nTH $\rightarrow$ slow down all MV-banks.
- num_req_MV[n] $\geqq$ nTH $\rightarrow$ pause the n-th MV-bank and slow down the

other MV-banks.

When a pause happens, we also slow down all MV-banks that are not paused due to the power budget (check a detailed analysis in Section 3.3.1). Also, once an MV-bank enters pause or slow-down state, MV-mul resumes or speeds up after the MV-bank processes all requests that can be processed in that state. Thus, processor requests heading to the MV-banks that are already paused have no additional delay due to the `p-PRE` command.

We use the same `nTH` for both slow-downs and pauses, making pauses less frequent than slow-downs when memory requests are evenly distributed among banks (which are common cases). This design choice is reasonable as a pause is more costly than a slow-down as a sequence of `PRE` and `ACT` should follow a pause to resolve a row-buffer conflict.

We accumulate num_req_nonMV and num_req_MV[n] over multiple `tIV`s, not resetting them at the end of every `tIV`. This accumulation prevents a MC from waiting until MV-mul is finished when few requests (less than `nTH`) from a processor form dependency on future requests (forming a deadlock). The frequency of slow-down and pause can be controlled by changing `nTH` and `tIV`. Large `nTH` and `tIV` help MV-mul finish earlier at the cost of a longer tail latency for requests from a processor, and vice versa for small `nTH` and `tIV`. We can adaptively control both `nTH` and `tIV` values to either favor MV-mul over processor requests or vice versa.

3.2.4 Put It All Together: MViD Architecture

Putting it all together, we propose MViD architecture, as illustrated in Figure 3.8, whose design decisions are made through a series of design space exploration

made in the previous sections. We populate an MCU close to the inter-die I/O within a DRAM device, which orchestrates the whole MV-mul process. 16 MAC units are placed near GIO SAs in four banks, as only up to four concurrent `RD` operations are permitted under the power constraint of the baseline LPDDR4 DRAM. Each of these MV-banks has a CGU, which takes commands and addresses that come to the MV-bank from the MCU and converts them to the fine-grained sub-commands. An iv-SRAM and an ov-SRAM are placed per MV-bank to store the input/output vectors. We used a parallel prefix sum logic for index decoding, and an adder tree to aggregate the output from the 16 MAC units.

The core inner-product computation part of MViD consists of five pipeline stages: data reading, index decoding, input data fetching, MAC executing, and output storing. First, 256-bit data are read from the DRAM bank and latched to GIO SAs, followed by the index decoding stage, which comes with parallel prefix sum and fetching stage, which loads 16 input elements from iv-SRAM. Then, MAC operations are executed, and the results are added up by an adder tree per weight matrix row, forming an output value, which is stored into an ov-SRAM. MCU and CGU deal with requests from a processor through slowing down or pausing MV-mul.

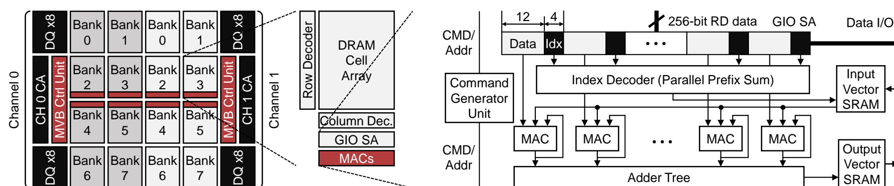


Figure 3.8: The MViD architecture.

We augment LPDDR4 DRAM commands with `p-PRE`, `s-PRE`, `r-PRE`, `WR-iv`, and `RD-ov` to slow down or pause MV-banks, to upload an input vector followed by starting an MV-mul operation, and to check if MV-mul is over and then retrieve the output vector from ov-SRAMs. A CGU generates a repeated sequence of `ACT`, multiple `RD`s, and `PRE` for the aforementioned core computation. By contrast, an MCU manages slow-down and pause of MV-banks.

Because MV-mul operates within DRAM devices, not on a processor, the data transfers of the input/output vectors are controlled in a similar way to DMA between the processor and MViD. Data can be transferred to MViD by giving start address, data length, and operation type to the register allocated for MViD. The MC uses the commands described above to transfer the vector to multiple MV-banks and gather the output vector. When MV-mul is over, the MC sends an interrupt to the processor to indicate the end of MV-mul and transfers data to the processor. Besides, there is a cache coherence issue between the processor and MViD. We can deal with this by either 1) flushing cache lines that hold weight matrices prior to every MV-mul or 2) marking that memory region as uncacheable, which was also proposed by [30].

3.2.5 Additional Optimization Schemes

MViD architecture can be augmented with a bank partitioning scheme to enhance performance further. The progress of MV-mul is delayed by slow-downs and pauses of the MV-banks, and pauses are caused by the necessity of serving processor requests to the specific MV-banks. By exploiting the bank partitioning method [75], it is possible to store data of the non-MV-mul workloads from a processor in non-MV-banks, thereby preventing MV-banks from serv-

Table 3.1: Energy/area/timing of MViD components.

Component	Energy (pJ)	Area (μm^2)	Cycle time (ns)
DRAM RD (to GIO SA)	2.54 pJ/b		5.00
MAC	1.31 pJ/op	6,318	5.00
Input vector SRAM	1.17 pJ/acc	30,048	1.25
Output vector SRAM	0.53 pJ/acc	3,931	2.50
Input decoder (SR)	1.23 pJ/op	5,067	5.00
Adder tree (SR)	1.39 pJ/op	6,071	5.00

ing processor requests. Because no MV-bank is paused, MV-mul and processor requests can be served only experiencing slow-downs, leading to additional performance enhancement. In the perspective of memory capacity, only non-MV-banks can be used as storage by applying bank partitioning.

3.3 Evaluation

3.3.1 Power/Area/Timing Analysis

We analyzed the power, area, and timing for each major component of MViD (LPDDR4-3200 DRAM device, MAC unit, input decoder, adder tree, and iv-/ov-SRAM). We calculated the power and energy of DRAM based on the IDD specification of LPDDR4-3200.[2] We designed a MAC unit, input decoder, and adder tree as Verilog, and synthesized them with 20nm DRAM process.[3] We added 100% area redundancy to the result considering PnR (place and route). We set the cycle time constraint of MAC unit and SR components to 5 ns (200

[2]Because IDD values of LPDDR4 is not publicly available, we refer to [76] and use the projection values from LPDDR3 values considering voltage, transfer rate, and fabrication process scaling.

[3]We synthesized based on FreePDK45 [91] and calculated power, area, and timing by multiplying scale factor considering transistor size and process characteristics as there is no public 20nm DRAM library.

MHz) so that it can operate within tCCD. We used CACTI [113] to model iv-SRAM with 1,600 of 12-bit entries and ov-SRAM with 400 of 24-bit entries.

Power and energy: We define the maximum power of one DRAM channel as the power when bursty data reads occur consecutively with the maximum number (four) of `ACT`s every tFAW. The calculated maximum power is higher than the power of IDD5 (all bank refresh) and IDD7 (`ACT-RD` burst), which are the power-hungry IDD values in the LPDDR4 specification. We calculated the power of DRAM read for MV-mul (read only up to GIO SA) by referring to FGDRAM [95], and scaled it considering DRAM page size, the number of banks, and data path length. Within the maximum power, we can perform MV-mul simultaneously up to four banks per channel (see Figure 3.9). Most of the MV-mul power of MViD is consumed for internal bank `RD`, and the power consumption of MAC, SRAM, and other components account for 2.7%, 1.9%, and 0.4% of the total power for MV-mul operation.

We cannot process DRAM requests from a processor with original DRAM timing values because there is a limited power budget for normal DRAM com-

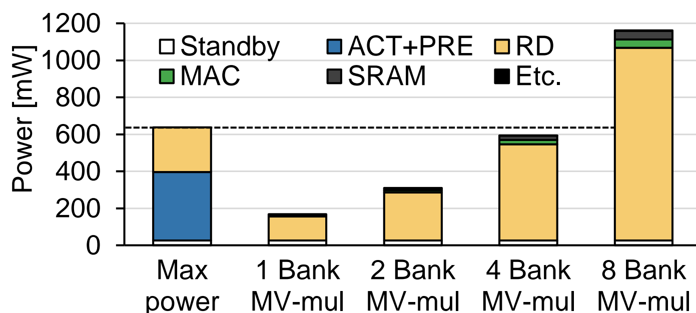


Figure 3.9: The breakdown of LPDDR4 peak power and MViD power consumption; LPDDR4 consumes maximum power with consecutive `RD`s and concurrent `ACT`s obeying the tFAW constraint.

mands even if MV-banks are slowed down or partially paused. When four MV-banks are slowed down to half the frequency, four MV-banks consume 300.8 mW, so that the available power budget is 337.0 mW. By doubling tCCD and tRRD, the maximum power required for normal DRAM commands is 326.2 mW, which is operable within the given power budget. When two MV-banks are paused, and the other two MV-banks are slowed down to half the frequency, these four MV-banks consume 163.4 mW, resulting in a power budget of 474.4 mW. When tRRD is set to double, the maximum power required for normal DRAM commands is 441.7 mW, which is also operable within a given power budget.

The energy for DRAM RD is the largest among the energy required for each component to perform MV-mul in MViD, as summarized in Table 3.1. The energy required for internal bank RD is 2.54 pJ/b, so 650.24 pJ is required for 256-bit read operation. The energy per MAC operation is 1.31 pJ/op, and 20.98 pJ is required for every DRAM RD because 16 MACs operate for processing 256-bit data. Each MAC operation requires access to an iv-SRAM, 1.17 pJ/acc $\times$ 16 acc = 18.72 pJ being required for 256-bit data. The input decoder is activated once for each DRAM RD, so 1.23 pJ/read is used. Moreover, the ov-SRAM and the adder tree require 0.53 pJ/acc and 1.39 pJ/op, respectively, which occupies a small fraction of the total energy because only one write is required per matrix row.

Area and timing: The area of a MAC is 6,318 μm^2, and the area of an iv-SRAM and an ov-SRAM is 30,048 and 3,931 μm^2, respectively. The areas of the input decoder and the adder tree are 5,067 and 6,071 μm^2. In the 8Gb LPDDR4 die, the area overhead of the MViD components required for half of

Table 3.2: Default simulation parameters.

Resource	Value
Number of cores, MCs	4 cores, 2 MCs
Per core:	
Frequency	2.4 GHz
Issue policy	Out-of-Order
L1 I/D $ type/size/associativity	Private/16KB/4
L2 $ type/size/associativity	Private/512KB/16
Hardware (linear) prefetch	On
Per memory controller (MC):	
# of channels, Request queue size	2 Ch, 32 entries
Memory standard	LPDDR4-3200
Scheduling/DRAM page policy	PAR-BS [88]/Open

the entire bank is equivalent to 3.69% of the die size. MAC and SR components can operate once per tCCD (5 ns). Doubling the frequency of a MAC (400 MHz) can reduce the number of MACs by half, but it is not used because the area increases by more than twice in the synthesis result. Due to DRAM internal frequency constraints, we use iv-SRAM[4] with 1.25 ns of cycle time to access 16 times within tCCD. Also, the cycle time of ov-SRAM is 2.5 ns because it requires only one access within tCCD.

3.3.2 Performance/Energy Evaluation

We simulated a chip-multiprocessor system to evaluate the performance of MViD. We modified McSimA+ [5] with the default parameters summarized in Table 3.2 for simulation. We measured the performance of DS2 by applying the effect of MViD on MV-mul based on DS2 data in Figure 3.1 measured on a real machine. MViD can perform MV-mul in four MV-banks per rank.[5]

[4]We configure iv-SRAM as two 2-port SRAM to allow four accesses in one cycle time.

[5]We assume that maximum power and the number of MV-banks increase in proportion to the number of ranks.

In our evaluation, we used the SPEC CPU2017 [81] benchmark suite for multi-programmed workloads running with MV-mul of DS2. Although SPEC CPU2017 is not a representative mobile workload, it is good enough to understand how MViD is affected by the memory intensity of host workloads. Using Simpoint [108], we extracted a representative simulation point of each application, consisting of 100M instructions. Each multi-programmed workload consists of two applications, mix-high (mcf, lbm) composed of memory-intensive applications, mix-low (exchange2, imagick) composed of memory non-intensive applications, and mix-med (xz, xalancbmk) composed of applications in-between.

We quantified the performance and energy efficiency of MViD by comparing it with the baseline configuration equipped with LPDDR4 DRAM in various conditions, through which we made the following key observations.

First, MViD provides a significant benefit in both performance (speed-up) and energy efficiency (energy-delay product (EDP)) across a wide range of matrix sparsity in MV-mul. Figure 3.10 shows the speed-up and relative EDP of a single-rank MViD when only DS2 is performed in the evaluated system. MV-mul of DS2 is performed in either a processor (Base) or MV-banks (MViD) using the dense (-Dense) or sparse (-Sparse) format. We used matrices from a pre-trained model of DS2 [2]. The size of a matrix is independent of sparsity in Dense, and so of the performance. By contrast, as sparsity increases, the size of the matrix of Sparse decreases leading to better speed-up. Sparse outperforms Dense for a sparsity higher than 25%. Executing DS2 in MViD performs about 2.4× on average better than DS2 in Base across a wide range of sparsity because the performance of MV-mul is mostly proportional to memory access band-

width. Considering energy efficiency, even if the power consumption of **MViD** is larger than that of **Base**, EDP on DS2 is improved about 6.5× on average better than **Base** as the execution time of MV-mul is reduced significantly.

Second, MViD still provides a substantial performance gain even when applications from non-MV-mul workloads are executed concurrently. Figure 3.11 (a) shows the relative DS2 throughput and the relative aggregate IPC of non-MV-mul workloads while changing the number (1, 2, and 4) of ranks. The RNN weight matrix of DS2 has a sparsity of 75%. We set `nTH` and `tIV` to 4 and 4 tCK, respectively. For each workload, we set the result of **Base** with one-rank as the baseline.

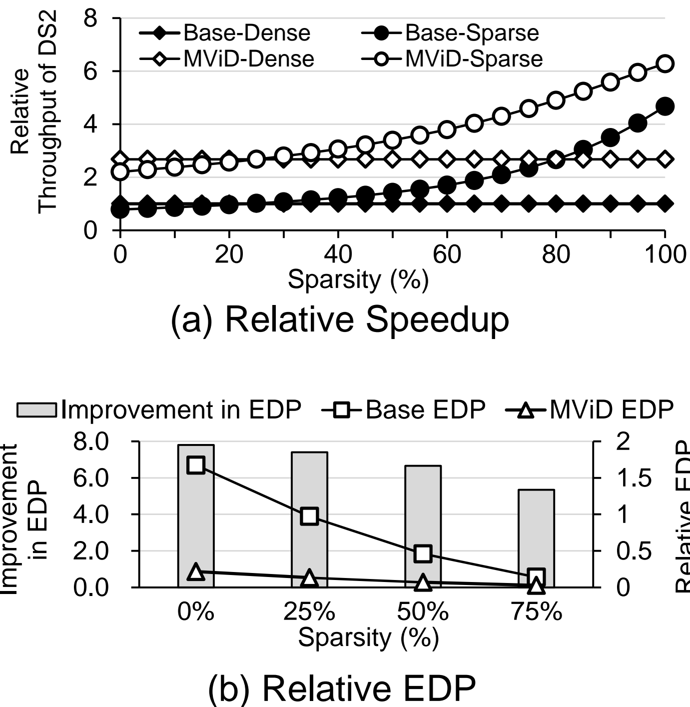


Figure 3.10: The impacts of MViD on DS2 by changing sparsity in (a) performance and (b) energy-delay product (EDP), when DS2 is performed alone. We set the result of **Base-Dense** as the baseline.

When using one DRAM rank, **MViD** performs 2.7×, 2.4×, and 2.2× better than **Base** in DS2 throughput when running with mix-low, mix-med, and mix-high. DS2 performs slower because MV-banks are slowed down or paused due to the requests from a processor. When running with memory non-intensive workloads, MV-banks do not experience frequent disturbance by the requests from a processor, and hence provide the performance gain similar to the case of running DS2 only. As memory intensity of the non-MV-mul workloads increases, the throughput of DS2 in **MViD** drops; however it is still higher than

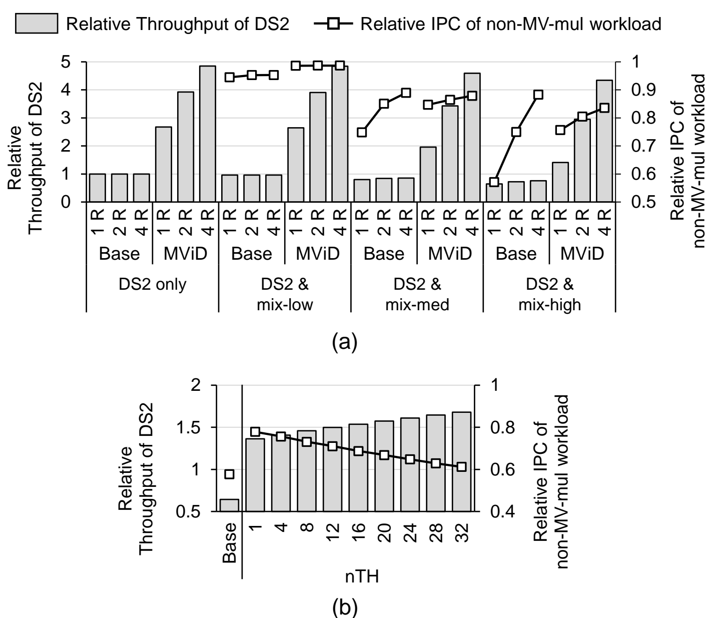


Figure 3.11: The impacts of MViD when DS2 is running solely or with non-MV-mul workloads (mix-low, mix-med, and mix-high). MV-mul of DS2 is performed in either a host processor (**Base**) or MV-banks (**MViD**). For each workload setup, we measure the relative throughput of DS2 and the relative IPC of the non-MV-mul workload, normalized to the throughput of DS2 and the non-MV-mul workload each running alone on the processor with one rank. (a) We change the number of ranks (1R, 2R, 4R) and (b) we change `nTH` while running DS and mix-high on an one-rank MViD.

Base where both MV-mul in DS2 and non-MV-mul workloads compete the same limited off-chip DRAM bandwidth. MViD performs similar or better than Base for the non-MV-mul workloads. On mix-high, mix-med, and mix-low, the aggregate IPC of MViD is 32%, 13%, and 4% higher than that of Base. The increase in the aggregate IPC is different because compared to Base because it takes longer for the aggregate number of pending requests in a MC to exceed `nTH`.

As we populate more DRAM ranks, DS2 throughput of MViD increases significantly, whereas that of Base stays largely unchanged. This is because the aggregate bandwidth on MV-banks of MViD is proportional to the number of ranks, but the off-chip bandwidth of Base is not changed. For example, DS2 throughput of MViD is 5.7× higher than that of Base while running with mix-med. DS2 throughput of Base increases slightly as more ranks are populated because more banks lead to lower row-buffer conflict rates [75]. This reduction in row-buffer conflict rates makes a large impact on the aggregate IPC of the non-MV-mul workloads for Base. For MViD, populating more DRAM banks also increases the aggregate IPC of the non-MV-mul workloads, but not as sensitive as for Base. As the number of banks increases, the memory requests are distributed across the banks. Therefore, it takes longer for a specific MV-bank to have at least `nTH` requests because fewer requests reach a specific MV-bank for the same time interval. It makes a longer tail latency of non-MV-mul workloads compared to that with fewer DRAM banks.

Third, adjusting `nTH` is effective in trading MV-mul throughput with the performance of non-MV-mul workloads. Figure 3.11 (b) shows the relative throughput of DS2 and the relative aggregate IPC of a non-MV-mul workload

(mix-high) as we change nTH on one-rank MViD. We set tIV as 4 tCK because tIV works in a way similar to nTH. When nTH is 1, any memory request heading to MV-banks that are not paused is delayed by tRC+tRP because a p-PRE command should be issued. Still, the aggregate IPC of the non-MV-mul workload of MViD is 35% better than that of Base because non-MV-mul workloads can utilize more aggregate off-chip memory bandwidth when MV-mul is performed on MViD instead of a processor.

As nTH increases, the aggregate IPC of the non-MV-mul workload is decreased. More requests must be queued up in MC to pause an MV-bank; even if an MV-bank is paused less frequently, the increased queuing delay leads to a longer tail latency in serving requests from the host processor. By contrast, the throughput of MV-mul is increased because MV-banks experience interference less often.

Fourth, bank partitioning (MViD-P) improves the performance of both MV-mul and non-MV-mul workloads. Bank partitioning eliminates row-buffer conflicts for MV-banks, so MV-mul operation is only slowed down but

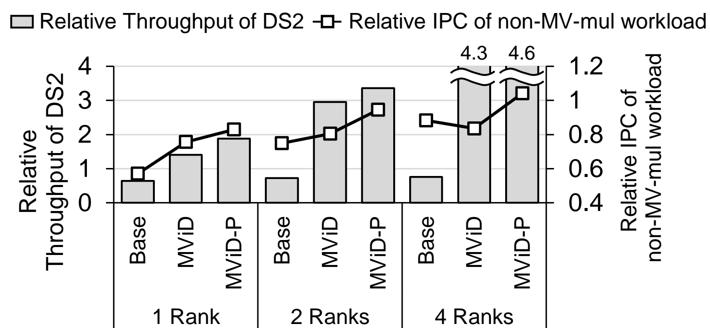


Figure 3.12: The impacts of bank partitioning (MViD-P) when DS2 is running with non-MV-mul workloads (mix-high). The configuration is the same as in Figure 3.11 (a).

not paused. Even if the non-MV-mul workloads experience slightly more frequent row-buffer conflicts as they use fewer DRAM banks (lower bank-level parallelism), the cost of row-buffer conflicts to a non-MV-bank is not as high as that to an MV-bank as the latter often leads to a pause experiencing the delay of `p-PRE` (tRC+tRP). Therefore, DS2 throughput increases by 34% and the aggregate IPC of the non-MV-mul workloads (mix-high) increases by 9.5% on the one-rank **MViD-P**, respectively (see Figure 3.12). With **MViD-P** on the four-rank, the aggregate IPC of mix-high on **MViD** exceeds that of **Base** whereas **MViD** provides 7.2× higher DS2 throughput compared to **Base**.

3.4 Discussion

Workload variation: MViD is robust to changes in the matrix size of the workload. Our proposed MViD is optimized for the 1600×1600 matrices used by a representative end-to-end ASR application. As the model evolves, the size of the matrix used by the RNN may change, and hence, it is desired for MViD to maintain its performance benefit across a wide range of matrix sizes. We experimented with how the performance of MV-mul changes according to the matrix size (see Figure 3.13). Even if the matrix size is reduced to 1/4 (when the number of rows and columns is halved each), the performance of MV-mul is 3.2 times higher than that of Base (only 5% less than the peak speedup of MV-mul). When the size of the matrix is larger than 1600×1600, the matrix is processed after being divided into sub-matrices, and a decrease in performance is no more than 5% as compared with the maximum speedup.

Load balancing: The progress of MV-mul differs between MV-banks because

each MV-bank experiences a different number of pauses due to interference by processor requests, whose distribution is both application and memory-address-mapping specific. This is a critical issue because MV-mul from all MV-banks should finish before the next phase (possibly another MV-mul) starts (forming a strong dependency). Inter-MV-bank load balancing could alleviate this problem. For this, there are two factors to consider, the NZ distribution between MV-banks and the pauses caused by processor requests. However, when we analyzed the weight matrices of various RNN models, the difference in the number of NZ values across MV-banks was within 2%. Also, the memory address mapping makes the processor requests quite evenly distributed over DRAM banks, resulting in the difference between the number of pauses of each MV-bank within 1%. Therefore, we did not devise a particular inter-MV-bank load balancing scheme.

Power budget and the number of processing elements: In a PIM architecture, if multiple processing units are placed in a DRAM module to utilize the internal aggregate bandwidth, there is an issue about exceeding the power budget as more operations are performed within the same duration. In the case of

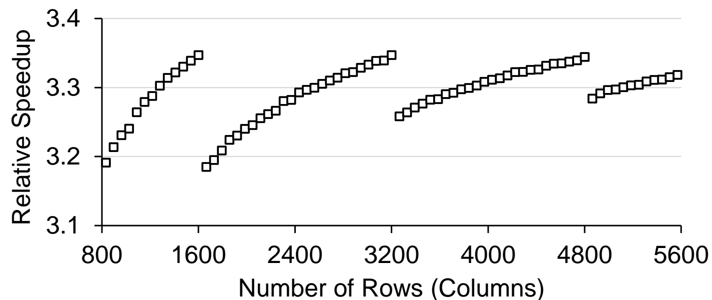


Figure 3.13: The impacts of one-rank MViD on MV-mul by changing the size of the weight matrix. We set the result of Base with one-rank as the baseline.

Samsung's HBM-PIM [77], the number of processing units is half of the total number of banks due to power limitation. [121] emphasizes the importance of the power managing technique by pointing out that high internal aggregate bandwidth causes more power consumption. Different generations of DRAM modules have different power budgets and energy consumption for operation. As UPMEM [35] places processing units for each bank, more processing units can be placed within the DRAM module if the power limitation of the device is not severe.

Chapter 4

TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory

4.1 Prior NDP architectures for accelerating Tensor Gather-and-Reduction

4.1.1 Tensor Gather-and-Reduction in RecSys

Tensor gather-and-reduction (GnR) performs a simple reduction operation (e.g., an element-wise sum for SparseLengthsSum (SLS) in Caffe2 [1]) of embedding vectors collected from multiple embedding table lookups. In DLRM [90], a representative RecSys model, one GnR operation performs generally between 20 and 80 lookups. An embedding table takes the form of a matrix (rank-2 tensor) in which each row holds one embedding vector (rank-1 tensor). The number of elements in a row (hereafter referred to as the vector length, v_{len})

This chapter is based on [65, 66, 99].

©2020 IEEE. Reprinted, with permission, from Byeongho Kim, Jongwook Chung, Eojin Lee, Wonkyung Jung, Sunjung Lee, Jaewan Choi, Jaehyun Park, Minbok Wi, Sukhan Lee, and Jung Ho Ahn, "MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks," IEEE Transactions on Computers, April 2020.

"TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory" ©2021 by Byeongho Kim, Jaehyun Park, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1145/3466752.3480080.

"TRiM: Tensor Reduction in Memory" ©2020 by Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1109/LCA.2020.3042805.

typically ranges from 32 to 256 [103]. The on-chip storage of a processor is too small to store all of the embedding tables of RecSys, the size of which can exceed hundreds of GBs. Therefore, embedding vectors are mostly read from the main-memory DRAM.

As the compute to memory access ratio of GnR is extremely low with little locality (i.e., several KBs to MBs of DRAM reads over several tens to hundreds of GBs of embedding tables), GnR is highly memory intensive. This property renders GnR a prime candidate for acceleration using near-data processing (NDP [4,11,30]) at the processor-memory interface. The NDP architecture places processing elements (PEs) near the DRAM datapath I/O; thus, data read from the DRAM chip are processed within or near the DRAM chip. By fully utilizing the characteristics of the DRAM data/control path, the NDP architecture can achieve high performance.

Because the datapath of DRAM has a hierarchical (tree-like) bus structure, the bus between the MC and the rank can only be utilized by one rank at a time. However, if a processing unit for GnR is placed within the buffer chip, the datapath between the MC and the buffer chip is not utilized during GnR. Therefore, all ranks can simultaneously transfer data to the corresponding processing units in the connected buffer chips. We exploit this characteristic to accelerate GnR with a higher internal bandwidth.

4.1.2 Prior NDP accelerators for GnR

Figure 4.1 explains how the baseline system (`Base`) and prior NDP accelerators handle GnR. In `Base` using a conventional module-based DDR5 DRAM, only one rank can occupy the bus and transfer data in a channel at a time be-

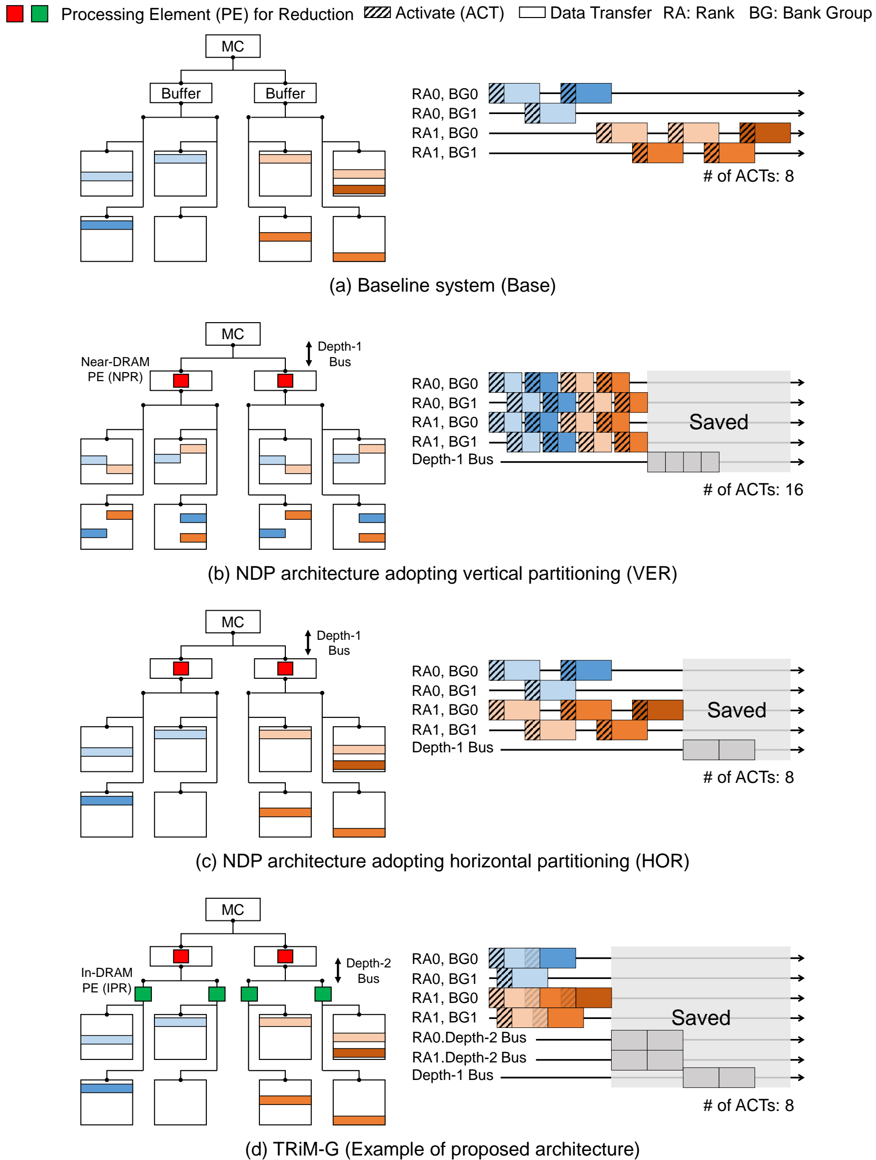


Figure 4.1: Exemplar GnR in the baseline (`Base`) and the evaluated NDP architectures. Each diagram on the left simplifies conceptual DRAM modules, where each has one DRAM rank consisting of two bank-groups, each packed with two banks, showing how embedding vectors are mapped to the DRAM chips. The timing diagram on the right shows the states of bank-groups and buses while transferring data or conducting partial GnR near/in DRAM.

cause all ranks in the channel share the depth-1 bus. TensorDIMM [73] and RecNMP [63], two recently proposed NDP accelerators for GnR, employ one processing element (PE) per rank, locating them inside the buffer chips. Each PE performs GnR for (a portion of) an embedding vector in parallel. As the depth-1 bus is not utilized during GnR in each module, each PE in a rank can receive data independently. Therefore, PEs in different ranks can receive data in parallel, and the aggregated bandwidth for GnR can be as high as the channel bandwidth multiplied by the number of ranks in a channel (rank-level parallelism). Furthermore, as the embedding vectors for GnR are transferred to the buffer chips but not through the depth-1 bus, data transfer energy is saved. Only the reduced vectors are transferred through the depth-1 bus.

A key factor that differentiates `Base` from prior NDP accelerators for GnR is the manner in which the embedding tables are mapped across the memory subsystem. In `Base`, the elements in an embedding vector exist at consecutive

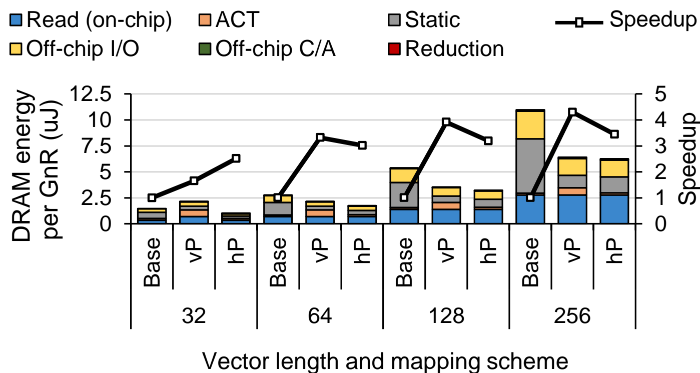


Figure 4.2: Speedup and DRAM energy breakdown of `Base` and the state-of-the-art NDP accelerators when performing GnR. The evaluation assumes a DDR5-4800 DRAM module with four ranks (Section 4.3 details the workloads and the configuration for both the `Base` and NDP architectures).

addresses in a single DRAM row. Therefore, the embedding vector elements are sequentially transferred to the host MC after activating the DRAM row that contains the target vector.

TensorDIMM and RecNMP adopt different embedding table mapping and reduction strategies. TensorDIMM splits the embedding table vertically so that each partition only has a portion of the embedding vector, each of which is mapped to different ranks (vertical partitioning, vP). Because the PE dedicated to each rank produces a portion of a reduced vector from GnR, the reduced vector in each PE is concatenated with the other portions at the host. In contrast, RecNMP evenly distributes the entries of an embedding table to each rank (horizontal partitioning, hP). After performing the reduction of vectors in the PEs, each PE sends a partial sum of the vectors to the host, and the host reduces the partial sums. Because the vector is reduced by an element-wise operation, the elements that arrive at the buffer chip first can be reduced and transferred to the host before the entire vector arrives at the buffer chip. Moreover, data transfer to the host can be overlapped by the subsequent reduction of vectors by another GnR in the PEs.

While the performance benefits of employing NDP for GnR acceleration are clear, we observe several important design overheads that the prior works do not address appropriately. An NDP architecture adopting vP (VER) distributes one vector to multiple ranks evenly such that the number of row activations (ACTs) for a GnR operation becomes proportionally larger as a function of how many ranks are used to (vertically) partition the embedding table (e.g., Figure 4.1(b) incurs 2× more ACTs compared to Figure 4.1(a, c)). Consequently, a key limitation of VER is that it consumes significantly more energy in ACTs for GnR

than `Base` and HOR (NDP architecture adopting hP). Moreover, if the partitioned vector size in VER is smaller than the DRAM access size, the internal data read/write bandwidth is wasted due to redundant data reads.

With regard to the hP strategy, HOR must transfer different C/A signals to each rank because each PE processes embedding lookups of different entries. Therefore, HOR requires a higher C/A bandwidth than VER, which broadcasts C/A signals to all ranks that process the different elements of the same embedding vector. RecNMP mitigates this high C/A bandwidth pressure by compressing a pair of activate/precharge (ACT/PRE) commands and several read (RD) commands to one custom instruction because all elements in an embedding vector exist in one DRAM row. However, the load-imbalance issue also becomes problematic in HOR as each rank may receive a different number of embedding lookups for GnR.

4.1.3 Quantitative Analysis

To quantify the pros and cons of VER and HOR in GnR, we compare the performance and DRAM energy consumption of VER and HOR against `Base` without caching recently accessed embeddings when sweeping the values of v_{len} from 32 to 256 (see Figure 4.2). The performance of VER and HOR is significantly improved because both can utilize the abundant internal bandwidth of the DRAM. VER shows up to 4.3× higher performance (similar to the number of ranks per channel, N_{rank}) when v_{len} is 256, but the achieved speedup from VER becomes only 1.6× when v_{len} is 32. For a v_{len} value of 64, each rank is assigned with 16-length partitioned vector elements, which correspond to 64B, identical to the single DRAM access size. Because there is little spatial local-

ity when accessing the embedding table, vectors are mostly read from different DRAM rows. Therefore, in most cases, a separate instance of row activation is required for every vector read operation within a given rank. However, when v_{len} is 32 where the partitioned embedding vector size of VER is smaller than the minimum DRAM access granularity of 64B, half of the DRAM bandwidth is wasted, resulting in the achievement of only half of the performance benefit of v_{len} of 64. HOR overcomes the aforementioned limitation of VER when the GnR's memory access stream can sufficiently utilize the internal bandwidth of all ranks, even when v_{len} is 32. However, the performance of HOR decreases by about 10% to 20% compared to VER due to the load-balancing issue.

HOR is more energy-efficient than VER because the ACT energy of VER is four times larger than that of `Base` and HOR (equivalent to N_{rank}). In particular, when v_{len} is 32 and 64, VER consumes more DRAM energy than `Base` and HOR because the ACT energy accounts for a large portion of the total DRAM energy consumption. As v_{len} increases, the ACT energy is amortized over the off-chip I/O energy, the DRAM read energy, and the DRAM static energy. When v_{len} is 256, off-chip data transfers from a buffer chip to a host MC are significantly reduced in VER and HOR compared to those in `Base`. Also, the DRAM static energy in NDP architectures is reduced due to the speedup in GnR. Thus, the energy consumption of VER and HOR decreases by 42% and 43% over `Base`, respectively. Owing to the wasted internal bandwidth, as mentioned above, for VER when v_{len} is 32, there is no significant difference in the energy consumption between the two v_{len} values of 32 and 64. The C/A signaling and reduction operation slightly affects the total energy consumption.

4.1.4 Additional Schemes for Accelerating GnR

RecNMP applied additional optimizations to improve GnR performance. First, it implements a cache (RankCache) inside the buffer chip to store entries with high access rate. RecNMP improves the GnR performance by taking advantage of the temporal locality of access to the embedding table in RecSys workloads. Second, to alleviate the load imbalance problem that occurs in HOR, RecNMP puts GnR operations together in a single GnR batch to process several GnR operations at a time. When processing a single GnR operation (when the number of GnR operations per batch, N_{GnR}, is 1), the number of lookups (N_{lookup}) handled by each rank is highly uneven, resulting in a significant performance drop due to the load-imbalance issue. However, if multiple GnR operations (when $N_{GnR} > 1$) are processed at once, the total number of lookups in the batch increases, which relieves the load-imbalance problem.

4.2 Tensor Reduction in Memory

Although TensorDIMM and RecNMP achieve decent speedups, we observe that there are substantial untapped, further performance improvement opportunities to be gained by leveraging the internal data transfer bandwidth at the DRAM datapath. Because the datapath of DRAM is organized as a tree structure, if there are PEs dedicated to memory nodes (e.g., ranks, bank-groups, and banks) above a certain depth in an NDP architecture, embedding vectors can be hierarchically reduced from the depth. As N_{rank} (up to several) is much smaller than N_{lookup} (dozens) for GnR, the GnR speedup achievable when utilizing rank-level parallelism is fundamentally limited; there is room for further performance

improvements by exploiting finer-grained memory-level parallelism.

4.2.1 Basic Concept for TRiM

We present TRiM (Tensor Reduction in Memory), a DRAM-based NDP architecture tailored to the GnR operation. Our proposal is based on the key observation that the datapath constituting any given ranks/bank-groups/banks exhibits a tree-like interconnect topology. TRiM opens up opportunities to conduct the GnR operation hierarchically by employing the NDP-based PE unit per memory node (e.g., a bank, bank-group, or rank). In TRiM, each memory node can perform the GnR operation independently without using the bus closer to the root node (MC). Therefore, TRiM can utilize internal bandwidth equal to the channel bandwidth multiplied by the number of memory nodes (N_{node}). By adopting finer-grained memory-level parallelism, N_{node} increases, and the internal bandwidth increases accordingly.

Embodiments of the DDR4/5-based TRiM architecture include TRiM-R/G/B, corresponding to the depth **R**ank/bank-**G**roup/**B**ank to which a PE is allocated. Figure 4.3 shows the simplified architecture of TRiM-G/B. In TRiM-G, there is a PE per bank-group. The PEs are located inside the DRAM

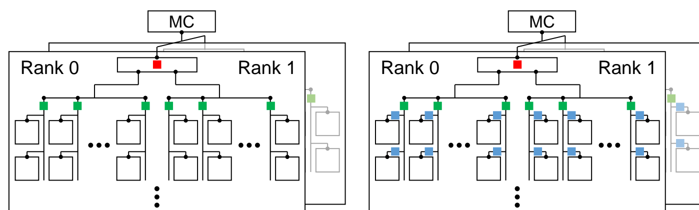


Figure 4.3: High-level overview of TRiM-G/B (left/right). Red/Green/Blue squares correspondingly denote PE located near the DRAM datapath dedicated per Rank/bank-Group/Bank.

chip so that the bus between the bank-group and the rank is not used for reduction within bank-groups. Similar to TRiM-G, TRiM-B has a PE for each bank, and the PEs are also located inside the DRAM chip.

To utilize the internal bandwidth provided by TRiM fully, an effective embedding table mapping scheme is needed. As discussed in Section 4.1.2, different mapping schemes (vP or hP) present different tradeoffs. When TRiM adopts vP, N_{node} rows must be activated to read a vector. Accordingly, the ACT energy increases. Moreover, the internal bandwidth is wasted when the partitioned vector size is smaller than the minimum DRAM read granularity (64B for DDR4/5). In contrast, when TRiM adopts hP, the C/A bandwidth requirement increases proportionally to N_{node}, and load-imbalance issue arises. One can also imagine a hybrid of vP and hP (vP-hP) when N_{node} is large enough; for example, vP is applied between memory nodes in different ranks, whereas hP is applied between memory nodes in different bank-groups. Unfortunately, such a design point inherits the shortcomings of both hP and vP as the ACT energy increases proportionally to N_{rank} and the C/A bandwidth usage increases proportionally to the number of bank-groups. Moreover, the load imbalance and internal bandwidth waste issues still exist.

Because adopting vP or vP-hP deteriorates the performance and energy efficiency of TRiM, we employ hP as the mapping scheme for the TRiM architecture. Given that RecNMP also employs an hP mapping scheme, we henceforth refer to RecNMP without RankCacahe as TRiM-R. Then, the increase in N_{node} entails a proportional increase in the C/A bandwidth demand. We address the issues of the C/A bandwidth shortage and the load-imbalance under TRiM-R/G/B using our novel two-stage C-instr transfer and hot-entry replication

scheme, the details of which are discussed later in this section.

Figure 4.1(d) highlights the key benefits of TRiM against prior NDP proposals for RecSys (see Section 4.2.4 for more details on the TRiM architecture). First, in-memory-node PE for Reduction (IPR) dedicated to each memory node gathers a series of embedding vectors from the corresponding DRAM banks within its local memory node, generating the final (partially) reduced vector in an accelerated manner. Multiples of IPR-reduced vectors are collected by the near-memory-node PE for Reduction (NPR) in the aggregate for the next level of reduction at the parent memory node. The final output reduced from NPR is eventually transferred back to the host MC, and then reduced at the host. Transferring the reduced vectors of a GnR batch and performing the element-wise GnR operations of the subsequent GnR batch can be done in parallel without sharing the datapath of DRAM and hence becoming overlapped.

By conducting the entire in-memory-node reduction operation within the shared tree datapath, our proposed NDP architecture reaps the abundant memory bandwidth (i.e., channel bandwidth $\times$ N_{node}) unlocked with in-memory processing, achieving superior performance. Also, TRiM significantly improves the energy efficiency as the data that reaches the NPR of the parent memory node is already partially reduced using our proposed IPR units, reducing the power requirement. Maximally unlocking the potential of in-memory processing requires TRiM to have the ability to supply GnR commands sufficiently to the NDP units. In the following section, we discuss our proposal for amplifying the C/A bandwidth to help realize the maximum degree of memory-level parallelism.

4.2.2 How to Provision C/A Bandwidth

C/A signals for GnR must be sufficiently supplied to all memory nodes in TRiM. We leverage a scheme proposed in RecNMP [63], which compresses ACT, sequential RDs, and PRE into a C-instr. One C-instr consists of 85 bits and takes charge of one embedding vector lookup. A C-instr is decoded to conventional DRAM commands in the command decoder located in each memory node, which is then transferred to the corresponding bank.

To avoid performance bottlenecks from an insufficient C/A bandwidth, MC must supply N_{node} C-instrs within the minimum time for a memory node to start processing consecutive C-instrs (referred to as $t_{C-instr}$). Assuming that TRiM fully utilizes the internal bandwidth, $t_{C-instr}$ is identical to the time to read the vector from a memory node, which is proportional to v_{len} and the read cycle. When transferring C-instrs through C/A pins (see Figure 4.4(a)), the following condition must be met to provide C-instr sufficiently to all memory nodes:

$$t_{C-instr} \geq N_{node} \cdot \frac{C\text{-}instr\ bits}{C/A\ bandwidth} \tag{4.1}$$

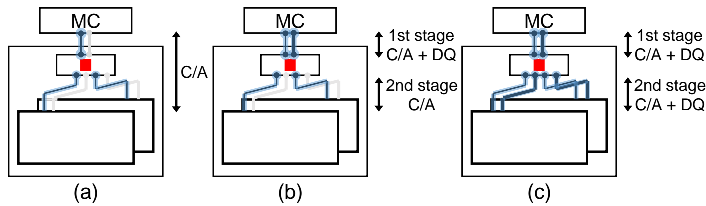


Figure 4.4: Various methods for transferring C-instr, the C/A signal to process the GnR operation, to the memory node in the TRiM architecture. (a) Using only C/A pins, adopting a two-stage C-instr transfer scheme and (b) only using C/A pins, or (c) C/A and DQ pins together at the second stage.

The right side of Eqn. (4.1) is the time taken to deliver C-instr to all memory nodes, which is (time to transfer one C-instr)$\times$N$_{node}$. In DDR5, the C/A bandwidth is 14 bits/cycle, meaning that C-instr can be sufficiently supplied up to five memory nodes when v$_{len}$ is 64. Consequently, only utilizing C/A pins cannot provide sufficient bandwidth to TRiM-G/B with multiple ranks. C-instrs can be transferred with a much higher bandwidth by using DQ pins (data pins) in addition to C/A pins. If a C-instr passes through the buffer chip and is transferred into a DRAM chip directly, inefficiency arises because only a portion of the DQ pins from the buffer chip are connected to each DRAM chip. Our key approach is to transfer C-instr only to the buffer chip instead (not to the DRAM chip) using the DQ pins because all DQ pins from the MC are connected to the buffer chip, which helps to amplify the effective C/A bandwidth as follows:

$$\mathrm{t}_{C\text{-}instr} \geq N_{node} \cdot \frac{C\text{-}instr\ bits}{(DQ_{MC} + C/A)\, bandwidth} \tag{4.2}$$

When the C-instr is transferred from the MC to the buffer chip using DQ pins and C/A pins together in DDR5, 5.6$\times$ more bandwidth (624-bit / 8 cycle) can be utilized than when transferring a C-instr using C/A pins only.

We propose a two-stage C-instr transfer scheme that delivers a C-instr through a path from the MC to the buffer chip, using another path from the buffer chip to the DRAM chip. In the first stage, a C-instr is transferred using C/A pins and DQ pins together. In the second stage, we consider two design points (see Figures 4.4 (b) and (c)), to use only C/A pins (Eqn. (4.3)), or to use C/A pins and DQ pins together (Eqn. (4.4)). These two stages can be pipelined as they do not share a datapath. That is, all buffer chips, which have dedicated

data/control paths connecting the ranks, can independently transfer C–instrs to the DRAM chip. Thus the aggregate C/A bandwidth of the second stage grows proportionally to N_{rank}.

$$t_{C\text{-}instr} \geq \frac{N_{node}}{N_{rank}} \cdot \frac{C\text{-}instr\ bits}{C/A\ bandwidth} \tag{4.3}$$

$$t_{C\text{-}instr} \geq \frac{N_{node}}{N_{rank}} \cdot \frac{C\text{-}instr\ bits}{(DQ_{chip} + C/A)\ bandwidth} \tag{4.4}$$

When sending a C–instr, using both C/A and DQ pins can utilize more band–width than using only C/A pins. However, if C/A and DQ pins are used to–gether, there may be a conflict between the transfer of a partially reduced vector from the IPR to the NPR and the transfer of C–instr. This can also cause an additional delay due to the bus turnaround of the datapath. Thus, if Eqn. (4.3) is satisfied, it is better to use only C/A pins in the second stage.

When two DDR5 ranks are populated per memory channel, we measure the C/A bandwidth requirement in TRiM–R/G/B by changing v_{len} from 32 to 256 and the bandwidth provision according to the C–instr supply method

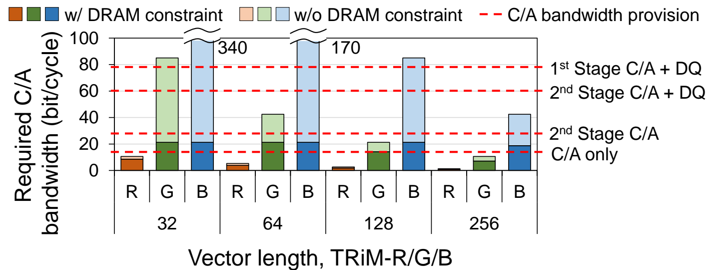


Figure 4.5: C/A bandwidth requirement to utilize all memory nodes for each TRiM architecture and bandwidth provision for each method of transferring C–instr. In TRiM–G/B, the required bandwidth for C–instr is significantly reduced due to DRAM timing constraints compared to that without these constraints.

used (see Figure 4.5). The light bar graph in Figure 4.5 is the C/A bandwidth requirement without any constraint when the DRAM access size and read cycle are set to (64B, 8), and the dark bar graph is the C/A bandwidth requirement when considering the various timing constraints (e.g., tRRD and tFAW). The red dotted lines represent the bandwidth provision according to the C-instr supply method: using 1) only C/A pins as in a conventional DRAM (C/A only), 2) C/A and DQ pins to a buffer chip (1st stage C/A+DQ) but only C/A pins from the buffer chip (2nd stage C/A), and 3) C/A and DQ pins from the buffer chip (2nd stage C/A+DQ).

The larger the v_{len}, the longer it takes to process one C-instr; thus, the C/A bandwidth requirement decreases. When the DRAM timing constraints are not considered, the bandwidth requirement inversely decreases proportionally to v_{len}. In TRiM-G/B, the required C/A bandwidth for a C-instr is significantly reduced by the DRAM constraints especially those limiting frequent activations within the rank, which consequently limit the multiple memory nodes in the same rank from operating in parallel.

The two-stage C-instr transfer scheme is compelling for increasing the effective bandwidth for C-instr (more than 2× compared to when using C/A pins only). We choose to use only C/A pins in the second stage because the C/A bandwidth provision of this approach is sufficient to fully supply C-instrs for TRiM-R/G/B with v_{len} from 32 to 256.

4.2.3 Exploring NDP Unit Placement

The optimal TRiM design point varies depending on the characteristics of the GnR workload. Allocating the PE for a certain memory depth allows GnR to

exploit the internal bandwidth up to the channel bandwidth multiplied by N_{node} in a memory channel. However, if the load is not sufficiently distributed to the PE or the time to read an embedding vector is relatively short, TRiM could suffer from low utilization of the internal DRAM data transfer bandwidth.

In Figure 4.6, we measure the throughput of GnR compared to `Base` according to N_{node} while varying N_{lookup} and v_{len}, assuming 2/4-rank DDR5-4800. In the TRiM architecture, the performance of GnR improves as N_{node} increases because the available aggregate bandwidth increases proportionally to N_{node} as each memory node can utilize a dedicated datapath. Therefore, the rightmost end of the heatmaps with the largest number of memory nodes performs best.

However, the speedup saturates when adopting excessive parallelism. When multiple banks share a datapath, one bank can occupy the datapath by reading data while the other banks are preparing data, meaning that the data preparation time can be hidden and the internal bandwidth can be highly utilized. Using

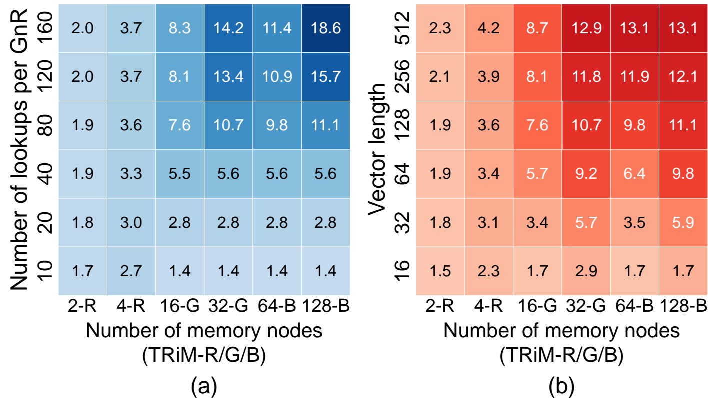


Figure 4.6: Heatmaps showing the speedup of TRiM-R/G/B over `Base` (a) according to N_{lookup} when v_{len} is 128 and (b) according to v_{len} when N_{lookup} is 80. The number of memory nodes of TRiM-R/G/B is 2/16/64 in 1 DIMM × 2 ranks and 4/32/128 in 2 DIMM × 2 ranks.

finer-grained parallelism reduces the number of banks in one memory node and thus reduces the number of banks sharing the datapath. In this case, the data preparation time might not be completely hidden and the utilization rate deteriorates. This demerit is pronounced when v_{len} is low, where the ratio of the data preparation time is relatively high, as shown in the bottom right of Figure 4.6(b). Also, limiting the frequency of activation in DRAM chips saturates the performance improvement as N_{node} increases.

To utilize the aggregate bandwidth obtained by increasing N_{node} fully, N_{lookup} should be sufficiently large. When this is the case, the time to perform a reduction in the memory node overlaps with the time for transferring the partially reduced vector to the PE at the parent memory node. Then, the speedup is eventually bounded by the ratio of the internal aggregate bandwidth in TRiM over the channel bandwidth. In contrast, if N_{lookup} is too small, the time to undertake reduction in the memory node is completely overlapped by the time for the transfer of the partially reduced vector to the PE at the parent node, limiting the speedup (see the lower right part in Figure 4.6(a)).

Considering the results in Figure 4.6, TRiM-G is most efficient, in common RecSys models with v_{len} of 20-80 and N_{node} of 32-256. Although TRiM-B shows better performance at some points, it incurs over 4× more area overhead than TRiM-G as the NDP unit is employed per bank, not per bank-group. As this area overhead can also lead to greater energy overhead to keep the DRAM read/write latency unchanged [94,112], we consider TRiM-G as a better option compared to TRiM-B.

We set TRiM-G with 16 memory nodes as the default configuration. When N_{lookup} is 80, TRiM-G with 32 memory nodes (2 DIMMs × 2 ranks) performs

better than TRiM-G with 16 memory nodes (1 DIMM × 2 ranks). However, if the lookups are evenly distributed across the memory nodes, we observe that there is no significant difference in performance between these two TRiM-G configurations. The load-imbalance issue is mitigated by applying hot-entry replication, which will be described later. Then, in DDR5, an embedding table is stored only in 1 DIMM × 2 ranks × 8 bank-groups, allowing multiple embedding tables to be looked up concurrently where performance improvements can be multiplied by the number of DIMMs.

Designing TRiM "in-memory" (rather than "near-memory" at the buffer chip) allows the utilization of the abundant internal DRAM bandwidth while consuming much less energy thanks to the shorter datapath. The trade-off lies in the higher area overhead of the "in-memory" approaches; DRAM processes with fewer metal layers require more area for designing logic elements, and the power and timing constraints of a DRAM chip hinder the maximal use of its internal bandwidth.

4.2.4 TRiM-G Organization and Operations

Based on our design space exploration, this section presents our TRiM-G architecture (see Figure 4.7), where an NDP unit tailored for GnR is employed per bank-group as well as per rank. IPR for TRiM-G is located between the bank-group I/O multiplexer (MUX) and the global I/O MUX. NPR for TRiM-G is placed on the buffer chip. IPR consists of 32-bit floating-point multiply-add (add for NPR) units (MACs) for vector reduction and includes registers to store the partial reduction of the vectors. Each buffer chip has a queue for the temporary storage of the C-instrs transferred from MC. The C-instr decoder is

located in the IPR to decode a C-instr to DRAM commands (ACT, RD, and PRE) and to send internal DRAM commands to each bank considering bank interleaving.

One 85-bit C-instr consists of target-address (34-bit), weight (32-bit), nRD (5-bit), batch-tag (4-bit), opcode (3-bit), skewed-cycle (6-bit), and vector-transfer (1-bit). The target-address is the starting address of a vector. The weight contains 32-bit floating-point data for supporting the weighted-sum operation. The nRD is the number of DRAM read commands per vector, which also indicates the size of the vector to be processed per C-instr. The batch-tag identifies the C-instr belonging to the same GnR operation in a GnR batch. The opcode determines the type of element-wise reduction operation (e.g., sum or weighted sum). The skewed-cycle indicates when the current C-instr starts operation at the memory node after arrival, and the vector-transfer instructs each memory node to transfer a partially reduced vector to the parent

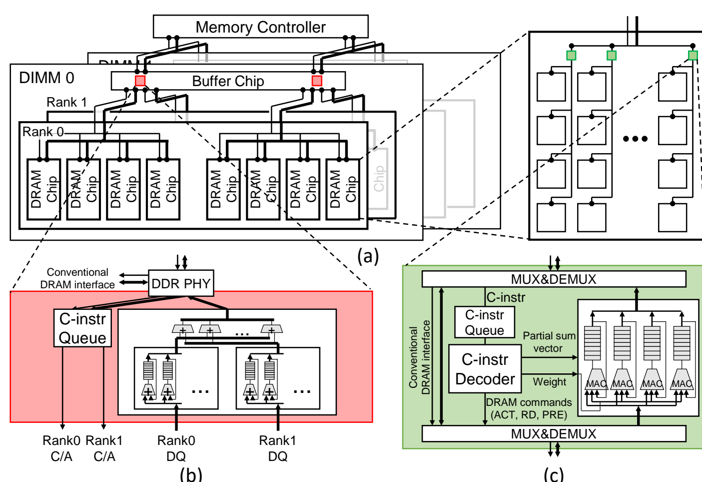


Figure 4.7: (a) Overall architecture of TRiM-G, (b) NPR, and (c) IPR assuming an ×8 data I/O bit-width. Four 32-bit MACs are placed in the IPR.

memory node. The vector-transfer bit is set to 1 for the last C-instr in the batch.

TRiM-G operates as follows. At the first stage of the two-stage C-instr transfer, up to 7 C-instrs enter the C-instr queue of the NPR in the buffer chip for every eight cycles exploiting C/A and DQ paths. In the C-instr queue of the NPR, a C-instr is transferred to the C-instr queue of the IPR in the target bank-group through the C/A path in order (the second stage). For each C-instr in the queue, the decoder in the IPR starts sending DRAM commands (i.e., ACT, PRE, and RDs) into the internal DRAM after the skewed-cycle. Whenever a RD is sent, the MAC units in the IPR accumulate data after a delay of tCL (access time). If the C-instr with the vector-transfer bit set enters the NPR queue, the NPR alternately sends commands to each IPR to transfer a partially reduced vector to the NPR by occupying the depth-2 data bus. After the first of these commands is sent, the adders in the NPR accumulate the partial sum of each IPR for every tCCD. The command for transferring a partially reduced vector is defined using reserved-for-use (RFU) commands. When accumulation for each rank is completed, other adders in the NPR combine the partial sums of the ranks. Finally, the MC reads the partial sums of the DIMMs.

4.2.5 Host-side Architecture for TRiM

Hot-entry Replication for Balancing Loads: Figure 4.8 shows the distribution of the maximum loads (the number of lookups) across the memory nodes per GnR, normalized to a perfectly balanced load. The performance of TRiM is bound to the memory node that has the largest number of lookups. As N_{node} increases, the number of lookups for a single memory node decreases, but the load

imbalance becomes more severe. RecNMP alleviates this problem by batching GnR operations, processing multiple GnR operations at one time. Batching incurs an area overhead that is proportional to N_{GnR} because dedicated registers are required to hold a partial reduction of vectors. Applying a batch of 8 GnR operations to TRiM-G causes an additional 2.5% of DRAM chip overhead.

We tackle the load-imbalance challenges with our novel hot-entry replication scheme, which copies frequently accessed (hot) entries to each memory node; this scheme is motivated by the replication method often used in datacenters [92]. We observe that a few entries occupy a large portion of the lookup

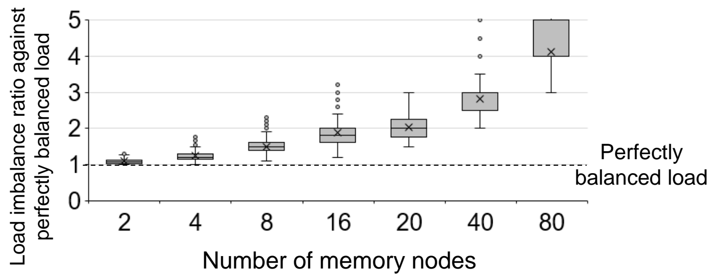


Figure 4.8: Distribution of the load imbalance ratio, the largest number of lookups among memory nodes in each GnR, normalized to the number of lookups of a memory node assuming that loads are evenly distributed to all memory nodes. N_{lookup} is 80.

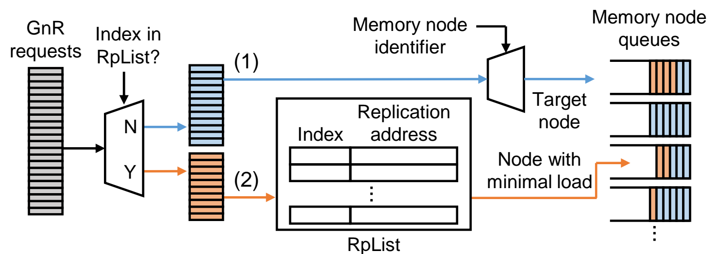


Figure 4.9: Execution flow of lookup request distribution through hot-entry replication.

requests (more details about the dataset are given in Section 4.3). Load balancing is done by distributing the lookup request, which corresponds to the C-instr. Only hot requests, ones that head toward hot entries, are redirected to the replicated entries at the nodes with lower loads. Hot entries are statically determined by profiling embedding table access traces, after which they are replicated and stored at the same address (bank, row, column for TRiM-G) in each memory node.

Figure 4.9 illustrates the execution flow for the distribution of lookup requests. There is a list of replicated entries (RpList), where the replicated entries exist at the same relative locations across all memory nodes. All lookup requests in a GnR batch are classified based on whether the target index of the request is on the RpList. Lookup requests other than hot requests are put in the request queue dedicated to each memory node. Then, hot requests are distributed to the queue of the memory node with the minimal load.

There is a trade-off between the number of hot entries (N_{hot}) and the speedup because the load balance improves as N_{hot} increases, but the capacity overhead rises (proportional to N_{node}). Because both hot-entry replication and batching mitigate the load-imbalance issue, N_{hot} and N_{GnR} should be configured considering the area and memory capacity overheads.

Storing hot-entries in the host cache can improve the performance of GnR operations, but hot-entry replication is preferred for the following reasons. First, the degree of performance improvement by the host cache is limited because only a small portion of entries can be stored in the cache. The embedding table corresponding to a size of hundreds of GBs can be stored in the main memory, whereas the size of the host cache is few tens of MBs; only about a hun-

dredth of a percent of the embedding table can be stored in the cache. Second, even if the performance is improved by the method described above, it could degrade the performance of other operations. As found in earlier work [63], if hot entries are stored in the cache, data such as the weights of the FC layer required for the RecSys model can be evicted. Consequently, this not only reduces the performance of other operations, but also increases the latency. With TRiM, this effect is even greater because FC takes up most of the time.

RecNMP uses the cache in the buffer chips to exploit the temporal locality of hot entries. However, using the cache on the DRAM side breaks the conventional interface by which the DRAM access latency is deterministic, making it unsuitable for TRiM. Also, transferring signals for cache hits from DRAM to MC is expensive as polling is required and an additional scheduler should exist per memory node.

Programming, Memory Model, and Data Placement: TRiM leverages the programming and memory models in previous NDP architectures [11, 30, 63]. The host runs a RecSys application and offloads a portion of the GnR operation to TRiM, similar to CUDA [84]. By ensuring that the contiguous virtual ad-

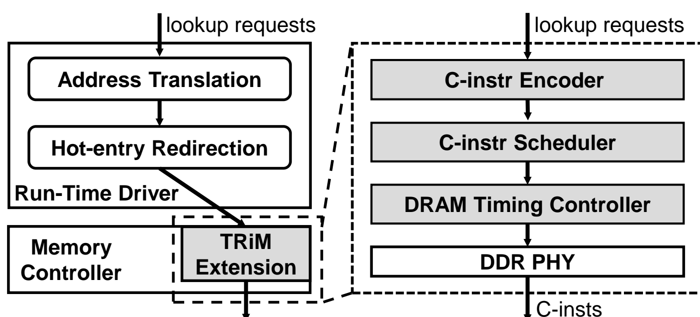


Figure 4.10: An overall execution flow of the lookup requests on the host-side.

dress range for an embedding table is mapped to a contiguous physical address range, the physical address of each embedding vector index can be identified from the starting address of the embedding table stored in a TRiM-specific driver. When placing the embedding table in DRAM, the TRiM-specific driver evenly distributes the embedding table to the memory nodes exploiting DRAM address mapping. The host and TRiM have their own independent memory region within one physical memory space to avoid memory consistency issues. Following [11, 30], we address cache coherence issues by marking the memory region for TRiM as uncacheable and shuffle data in an embedding table entry to place a consecutive 32-bit floating-point value into a DRAM chip for the IPR.

Execution Flow on the Host side: Figure 4.10 shows how lookup requests are processed on the host side prior to being transferred to TRiM. As N_{hot} is configurable, the RpList can be enlarged according to the embedding table size and the replication rate of hot-entry replication (p_{hot}), representing a burden to put the RpList on the MC. Accordingly, we assign a run-time driver tailored for TRiM to distribute hot requests. It receives lookup requests and undertakes the distribution of the hot requests. The lookup requests are transferred to the MC with the TRiM extension, and the C-instr encoder encodes the requests to C-instrs. Then, the C-instr scheduler reorders the C-instrs for each GnR batch considering that multiple memory nodes operate simultaneously. After the completion of a scheduling task for a GnR batch, the DRAM timing controller sets the skewed cycle in the C-instrs. The C-instrs are then transferred to the DRAM, obeying the DRAM timing constraints.

4.2.6 Schemes for Improving Reliability

In TRiM-G/B, the GnR operation is performed inside the DRAM chip and hence the conventional rank-level error correction code (ECC) cannot be applied for error detection and correction, necessitating a different way to ensure reliability when reading data for GnR inside a DRAM chip. Modern DRAM chips have started to adopt on-die ECC, which operates inside the DRAM chip (die) to improve data integrity further [14,93,97,101,111]. In particular, DDR5 uses single-bit error correction (SEC) codes for the on-die ECC [57]. However, the reliability of DRAM only with conventional on-die ECC (SEC) is lower than that with rank-level ECC (typically supporting SECDED, SEC with double-bit error detection (DED)) and on-die ECC together.

To improve the reliability of data being read during GnR to a level equivalent to DED, we propose to repurpose the on-die SEC code to detect double-bit errors. This is possible because the embedding tables are read-only while performing GnR and the hamming code used for SEC has a minimal hamming distance of 3, which can be used to detect double-bit errors if correction is not needed [102]. During GnR, the parity bits are calculated for the embedding table entry being read, similar to a normal DRAM write, and those calculated are compared to the stored parity bits. If a mismatch is identified, an error is reported, and the table entry should be reloaded from storage. The overhead of supporting DED for TRiM is minimal as most of the ECC logic is reused; only a simple comparator is added to detect a mismatch between the parity bits.

Table 4.1: Timing/energy parameters of 16Gb DDR5-4800 ×8 DRAM chips and NDP units.

Parameters	Values
Clock frequency (1/tCK)	2,400 MHz
Cycle time (tRC)	48.64 ns
ACT to RD, Access, PRE time (tRCD, tCL, tRP)	16.64 ns
Read to read between different bank-groups (tCCD_S)	8 tCK
Read to read to the same bank-group (tCCD_L)	12 tCK
Four activate window (tFAW)	13.31 ns
ACT energy	2.02 nJ
On-chip read/write energy	4.25 pJ/b
Read energy to bank-group (BG) I/O MUX	2.45 pJ/b
Off-chip I/O energy	4.06 pJ/b
MAC unit energy in IPR	3.23 pJ/Op
Adder energy in NPR	0.90 pJ/Op

4.3 Experimental Setup

Simulation framework: We modified Ramulator [69] to evaluate the performance and energy consumption of TRiM compared to those of the baseline system (`Base`) and two state-of-the-art NDP architectures, RecNMP and TensorDIMM. The C-instr generator, decoder, and hot-entry replication module are implemented inside our Ramulator-based simulation framework. We set `Base` and the NDP architectures to use commodity DRAM modules, DDR5-4800, 1 DIMM with 2 ranks per memory channel (see Table 4.1). `Base` was simulated in the CPU trace-driven mode with 32MB of last-level cache, which is large enough to saturate the performance improvement due to the temporal locality in our synthetic traces.

Benchmarks: We utilize representative RecSys models published in prior work [42]. The vector reduction operation of GnR utilizes an element-wise sum operator (SLS) and N_{lookup} is set to 80 with 32-bit floating-point elements. Following [118], we vary v_{len} from 32 to 256 in our experiments. Because the real

trace used in prior works [42, 63] is not publicly available, we generate a synthetic embedding table access trace with the algorithm discussed in [90] using the publicly available Criteo dataset [22, 103]. Our synthetic trace shows temporal locality similar to the traces presented in [29, 63]. We set N_{GnR} to 4 and the default p_{hot} rate to 0.05%.

Power and area: To analyze the power and area for the NDP architectures including TRiM, we estimate the power consumption of DDR5 DRAM with the DDR4 datasheets from industry [87,105] and the off-chip I/O with CACTI [59]. We calculated the power of the DRAM read for the IPR (read-only up to bank-group I/O) by referring to FGDRAM [95] and scaled it properly considering the DRAM page size, the number of banks, and the datapath length. We first synthesized both the IPR and NPR units using the Synopsys Design Compiler with 40nm CMOS technology at frequencies of 200 MHz (for IPR) and 300 MHz (for NPR) respectively, which is demonstrated to operate in the other recent NDP architecture implemented in the real DRAM devices [77]. Then, we scaled the result of the IPR to a 20nm DRAM process assuming that a DRAM process is 10$\times$ less dense compared to an ASIC process of an equivalent feature size considering fewer metal layers and slower transistors, referring to [25, 110].

4.4 Evaluation

This section initially evaluates the effect of TRiM's various design optimizations by incrementally applying them on top of `Base`. We then quantify the performance and energy-efficiency benefits of TRiM-G over TensorDIMM and RecNMP. We also analyze the speedup of TRiM over various p_{hot} values and

evaluate the effects of replication and batching together.

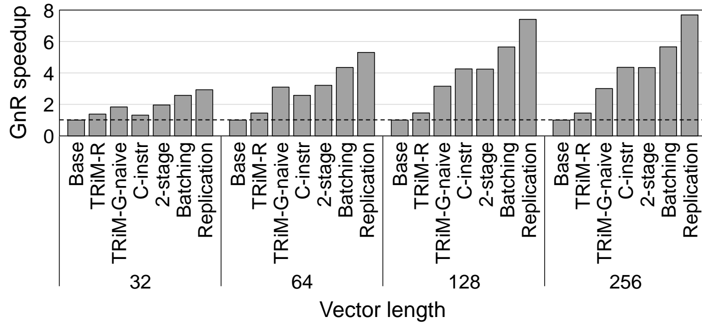


Figure 4.11: The GnR speedup of the TRiM architectures when applying the TRiM-R, TRiM-G-naive (rank-/bank-group-level parallelism), C-instr (instruction compression), 2-stage (2-stage C-instr transfer), Batching (GnR operation batching), and (hot-entry) Replication schemes.

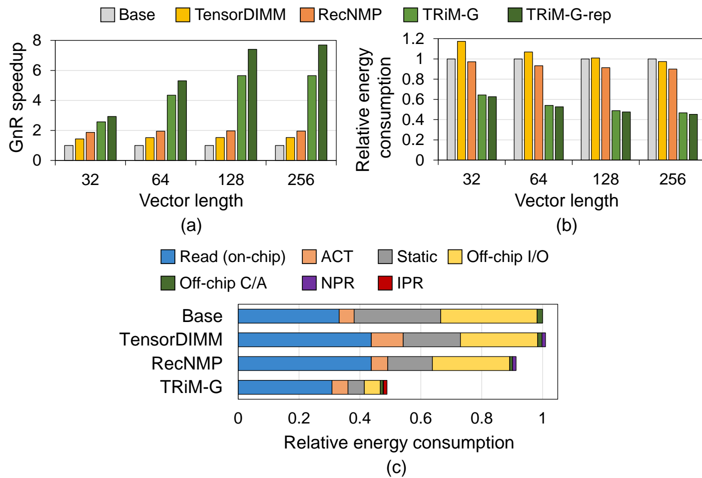


Figure 4.12: (a) GnR speedup and (b) relative DRAM energy consumption of TensorDIMM, RecNMP, TRiM-G, and TRiM-G with hot-entry replication (TRiM-G-rep), and (c) energy consumption breakdown when v_{len} is 128.

4.4.1 Performance and Energy Efficiency

Figure 4.11 shows the speedup in six scenarios of TRiM when gradually applying the modified design of the C/A signals and optimizations at various values of v_{len}. The first two scenarios are `TRiM-R` and `TRiM-G-naive`, each corresponding to the TRiM architecture with rank or bank-group level parallelism without any change in the C/A interface. The next two scenarios are `C-instr` and `2-stage`, each applying the instruction compression scheme (proposed in [63]) and the two-stage C-instr transfer scheme to increase the effective C/A bandwidth. The last two scenarios are `Batching` and `Replication`, batching GnR operations (proposed in [63]) and a hot-entry replication scheme to alleviate the load-imbalance issue. In later experiments, `2-stage` and `Replication` correspond to TRiM-G and TRiM-G-rep, respectively.

In general, the effectiveness of our proposal is clearly demonstrated by the gradually increasing performance as additional optimizations are incrementally applied, but with varying degrees of effectiveness under different v_{len} values. `TRiM-R` improves the performance by up to 1.46× over `Base`. `TRiM-R` can utilize up to twice (N_{rank}) as much internal bandwidth relative to that by `Base`. However, `TRiM-R` cannot benefit from the cache in the host; thus, its speedup is less than the increase in the internal bandwidth. `TRiM-G-naive` also uses 8× more memory nodes and 8× more internal bandwidth than `TRiM-R`, but its performance is only slightly higher than that by `TRiM-R`. This occurs because `TRiM-G-naive` is more strictly affected by the C/A bandwidth limitation and DRAM constraints than `TRiM-R`.

`C-instr` achieves up to 45% higher performance than `TRiM-G- naive` with various values of v_{len} because more DRAM commands can actually be trans-

ferred by exploiting the instruction compression scheme. However, if v_{len} is 32 or 64, the performance of `C-instr` is degraded because the number of cycles sending one lookup request by ACT-RDs-PRE is smaller than that by C-instr. `2-stage` improves the performance by 50% and 24% when v_{len} is 32 and 64, respectively, as it can amplify the bandwidth for the C-instr transfer, which is limited when v_{len} is low. Compared to `TRiM-G-naive`, the schemes for increasing the effective C/A bandwidth achieve a speedup of 1.2× on average. There is an additional speedup of 67% on average via batching GnR operations (`Batching`) and hot-entry replication (`Replication`), mitigating the load-imbalance issue. In conclusion, TRiM-G applying all of the optimizations achieves speedups of up to 7.7× over `Base` and up to 5.3× over `TRiM-R`.

TRiM-G improves the performance over TensorDIMM and RecNMP owing to the increased internal bandwidth realized by exploiting finer bank-group-level parallelism. Figure 4.12 shows the speedup and relative DRAM energy consumption for `Base`, TensorDIMM, RecNMP, and TRiM-G when varying v_{len}. We linearly scale the speedup and static energy from RankCache in a previous study [63] to the results here. TRiM-G achieves a speedup of up to 5.7× over `Base`, and up to 3.7× and 2.9× over TensorDIMM and RecNMP, respectively (see Figure 4.12(a)). Despite the fact that N_{node} of TRiM-G is 8× larger than that of TensorDIMM and RecNMP, the speedup of TRiM-G is lower than 8× because the frequency inside a bank-group bus is lower than that outside a bank-group, which reduces the peak bandwidth by 33%, and the load imbalance becomes worse due to the increase in N_{node}. When v_{len} is 32 and 64, the speedup is relatively low due to the limitation of the ACT frequency, while TRiM-G demands more frequent ACTs for higher bandwidth utilization. If

v_{len} exceeds 128, the speedup is nearly saturated by the internal bandwidth.

As shown in Figure 4.12(b), TRiM-G consumes up to 55%, 54%, and 50% lower DRAM energy than `Base`, TensorDIMM, and RecNMP, respectively. Figure 4.12(c) shows the energy consumption breakdown when v_{len} is 128. TRiM-G dissipates 7% less on-chip read energy than `Base` with the host-side cache. Compared to RecNMP, TRiM-G consumes 30% less on-chip read energy and 79% less off-chip I/O energy due to the decreased data transfers from the IPR to the NPR, and 63% less static energy due to the reduced execution time. The energy consumption by the NPR and IPR for TRiM-G is negligible, accounting for 0.24% and 2.47%, respectively.

Hot-entry replication can improve the performance by up to 36% by alleviating the load imbalance. TRiM-G with hot-entry replication achieves a speedup of up to 7.7× over `Base`, and up to 5.0× and 3.9× correspondingly over TensorDIMM and RecNMP (see Figure 4.12(a)). The impact of hot-entry replication on the energy efficiency is negligible because this scheme does not

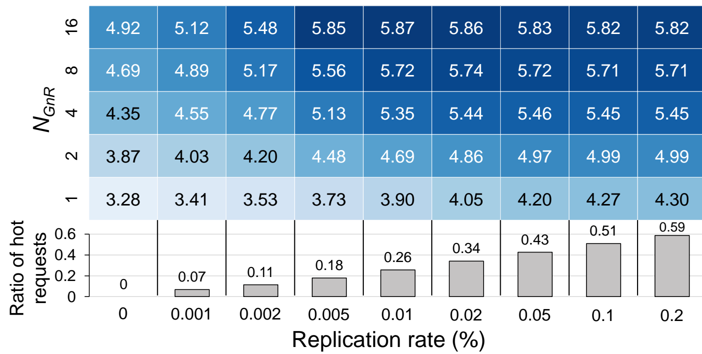


Figure 4.13: Speedup of TRiM-G over `Base` with various p_{hot} and N_{GnR} values (heatmap), the ratio of hot requests over all requests (bar graph), averaged over speedups while varying v_{len} from 32 to 256.

change the total number of lookups in GnR (see Figure 4.12(b)).

4.4.2 Sensitivity Study of Hot-entry Replication

Increasing p_{hot} can improve the performance, as the load balancing improves. In Figure 4.13, the heatmap shows the speedup of TRiM-G over `Base` according to N_{GnR} and p_{hot}, averaged over the speedups while v_{len} ranges from 32 to 256. The bar graph shows the ratio of hot requests to all requests according to p_{hot}. As more requests are distributed to memory nodes with lower loads as p_{hot} increases, the load imbalance is alleviated. With p_{hot} set to 0.05% and with a hot request rate of 42%, the speedup is nearly saturated, being 0.2% lower than that for a perfectly balanced workload and 25% higher than TRiM-G without hot-entry replication when N_{GnR} is set to 4.

Because increasing N_{GnR} (batching) and p_{hot} (hot-entry replication) causes area and memory capacity overhead, both should be carefully selected in consideration of the diminishing return of speedup. If N_{GnR} exceeds 8 without replication, the speedup is saturated to around 5.0. With a small p_{hot} of hot-entry replication, the speedup surpasses 5.0 when N_{GnR} is 4. Thus, we set N_{GnR} to 4 considering that larger values return a low-performance gain with a large register file requirement, which degrades the DRAM access latency further. Also, we set p_{hot} to 0.05% such that it saturates the speedup and incurs only 0.8% of the memory capacity overhead.

4.4.3 Design Overhead

The total area overhead of IPR is 2.03mm^2 per 16Gb DDR5 DRAM die [67], which corresponds to 2.66%, assuming that each chip has a $\times 8$ data I/O bit-

width and (v_{len}, N_{GnR}) is (256, 4). Each IPR, one per bank-group, includes four MACs and two 1KB register files considering double buffering. Because adding an IPR incurs a small amount of overhead, increasing the driver strength of the inter-bank datapath by a small degree keeps the DRAM access latency unchanged with a minimal increase in the read/write energy [112]. The area of the NPR is 0.361mm^2, similar to the area of RecNMP without RankCache.

4.5 Discussion

Applying TRiM to DLRM training: During DLRM training, the gradient of the reduced vector is required to update the embedding table for the backward pass of the embedding layer. There is a gradient vector with the same shape as the embedding vector per batch. For the TRiM architecture with horizontal partitioning, all memory nodes require gradient vectors to update the embedding table. By using the datapath inside the TRiM architecture, gradient vectors can be transferred to all memory nodes. Therefore, the TRiM architecture can utilize the ample internal aggregate bandwidth during DLRM training. Because the update of the embedding table is conducted through the element-wise operation between a target embedding vector and a gradient vector, DLRM training can be handled by adding an array of ALU units and registers to the existing processing unit of the TRiM architecture.

Processing multiple embedding tables: Because the size of a single embedding table does not match the size of the DIMM module with the TRiM architecture (called a TRiM module), a processing method should be devised when locating several embedding tables into the TRiM module. Embedding tables are mapped

in the address space for TRiM. During a GnR operation, the identification of the embedding table and the embedding vector index are translated into the physical address by the TRiM driver of the host system. Therefore, the processing method of the GnR operation is independent of the number of embedding tables in the TRiM module.

Sensitivity of hot-entry replication scheme to highly skewed dataset: The dataset used in the industry also has a characteristic that the number of accesses is skewed to a small number of entries [3]. However, the skewness of the industry dataset may be different from that of the publicly available dataset. As the access frequency distribution becomes more skewed, load imbalance is sufficiently mitigated by copying a small number of hot-entries to all memory nodes. Thus, the capacity overhead due to replication is further reduced. However, if hot-entries are not evenly distributed among the banks in the memory node (e.g., bank-group), the number of memory accesses may be concentrated to a small number of banks. This can lead to memory bandwidth underutilization.

Sensitivity of TRiM design selection according to the DRAM configuration: The DRAM configuration varies according to the DRAM generation, which affects the selection of the optimal TRiM design. The number of DRAM banks is the most important factor in the design selection. DDR3/DDR4/HBM2 have 8/16/16 DRAM banks per rank, respectively. If the embedding vector length of the model used is sufficiently long and the number of embedding vectors required per GnR operation is large enough, that the performance of TRiM can be improved linearly with the number of memory nodes. However, if a PE is placed in every bank, data preparation time cannot be hidden at all. This causes low internal bandwidth utilization so that the speedup is much worse

than the expected performance improvement the degree of which is the same as the number of banks. The optimal performance can be obtained when each memory node has at least 4 banks thus fully utilizing the bandwidth, and it is expected that TRiM achieves a speedup by a factor of about the number of the quarter of the total banks.

Chapter 5

Discussion

Applying PIM architectures to various memory-intensive applications: We found a use case in demand for memory-intensive machine learning applications, and we accelerated these applications through the proposed NDP architectures (MViD and TRiM). However, machine learning applications that provide optimal performance and accuracy when processing specific tasks evolve rapidly, so there is a possibility that applications that are mainly used now will not be used within a few years. Therefore, in order for PIM architectures to be exploited generally, it is necessary to accelerate not only specific applications but also primitive operations that are widely used in various applications.

Commonly used memory-intensive operations include GEMV, sparse matrix-vector multiplication (SpMV), and sparse matrix-matrix multiplication (SpMM). GEMV operation is a primitive operation widely used in machine learning, including FC layers of the neural network models [77]. SpMV and SpMM

This chapter is based on [65, 66, 99].

©2020 IEEE. Reprinted, with permission, from Byeongho Kim, Jongwook Chung, Eojin Lee, Wonkyung Jung, Sunjung Lee, Jaewan Choi, Jaehyun Park, Minbok Wi, Sukhan Lee, and Jung Ho Ahn, "MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks," IEEE Transactions on Computers, April 2020.

"TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory" ©2021 by Byeongho Kim, Jaehyun Park, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1145/3466752.3480080.

"TRiM: Tensor Reduction in Memory" ©2020 by Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1109/LCA.2020.3042805.

are mainly used for matrix multiplication or matrix-vector multiplication that receives a matrix or vector with a sparse format as input. Moreover, those operations are often used in graph processing (e.g., graph convolutional network [70]), as the method of representing graph relationships in graph analysis tasks is generally expressed in a matrix with the sparse format.

TRiM can be applied to accelerate GEMV operations. TRiM can store the weight matrices in DRAM cells and receive the input vectors from the host, temporarily storing them in the buffer. The register files for storing the partial sums of vectors in IPR can be used as the buffer for temporarily storing the input/output vectors for the GEMV operation in each memory node. With proper support from the software stack, TRiM can accelerate the memory-bound GEMV by fully exploiting the internal aggregate bandwidth of DRAM devices.

SpMM operations (especially, sparse-matrix dense-matrix multiplication) can also be accelerated by TRiM architecture. By processing an SpMM operation in the same way as a GnR operation, SpMM can be performed in a way that utilizes output stationary dataflow [15]. The dense matrix is stored in the DRAM cell similar to the way the embedding table is stored. Position and value information of non-zero elements in a single row of the sparse matrix is transferred to the TRiM architecture the same way as the indices of a GnR operation are sent to the TRiM architecture. This allows SpMM operations to be handled the same way as GnR operations.

MViD is an architecture to accelerate SpMV, but if matrix elements are processed without decoding for the sparse format, GEMV operations can also be accelerated. The degree of GEMV performance improvement is expected to

be about four times, which is half of the total number of banks on a single-rank configuration.

Chapter 6

Related work

Near-Data Processing: A large body of prior work has sought to place the processing logic closer to the memory. Similar to MViD and TRiM, recent works proposed DRAM or DIMM module-based near-data processing architectures [6, 7, 30, 50, 79, 80, 83]. Chameleon [11] proposed an NDP architecture based on Load-Reduced DIMM. Adding logic to stacked memory solutions was also explored in prior works [28, 31, 39, 68, 78, 82]. TETRIS [34] adds a simple data accumulator near the DRAM banks to accelerate the CNNs, but the proposed architecture cannot fully utilize the internal DRAM bandwidth. McDRAM [110] is closest to MViD in that it placed 2,048 MAC units within a DRAM die for accelerating MLPs and RNNs. However, McDRAM did not consider the sparsity of the matrices, the power constraints of the DRAM devices, and concurrency in accessing DRAM from processors while performing MV-mul. CHoNDA [18] considered concurrent host access during near-data

---

This chapter is based on [65, 66, 99].

©2020 IEEE. Reprinted, with permission, from Byeongho Kim, Jongwook Chung, Eojin Lee, Wonkyung Jung, Sunjung Lee, Jaewan Choi, Jaehyun Park, Minbok Wi, Sukhan Lee, and Jung Ho Ahn, "MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks," IEEE Transactions on Computers, April 2020.

"TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory" ©2021 by Byeongho Kim, Jaehyun Park, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1145/3466752.3480080.

"TRiM: Tensor Reduction in Memory" ©2020 by Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1109/LCA.2020.3042805.

processing. However, CHoNDA lacked a detailed analysis of the power generated by processing data within the DRAM. [72] presents an HBM processing-in-memory chip that exploits bank-level parallelism. However, this architecture is inefficient when used to perform reduction operations because it neither organizes PEs hierarchically nor allows PEs to access non-local memory. [64] presents a real near-data processing architecture that locates FPGA chip in the buffer chip of DRAM and exploits rank-level parallelism. But as the DRAM chip is not modified, this architecture cannot exploit finer-grained parallelism.

Computational optimization of RNN: Persistent RNN [26] alleviated the memory bandwidth bottleneck of LSTM by storing a portion of weight elements of an LSTM layer in GPU's on-chip memory. Sparse Persistent RNN [125] extended [26], covering sparse matrices as well. It compresses a sparse matrix into a densely packed matrix. However, both are difficult to apply to mobile environments with limited on-chip memory capacity. Zhang et al. [124] reduced the amount of off-chip memory transfers by exploiting inter- and intra-cell parallelism of LSTM in a mobile GPU with limited on-chip memory, which is orthogonal to MViD.

NN accelerators: Fowers et al. [33] exploited pipeline parallelism to utilize processing units effectively and to reduce the service latency of few batches by distributing RNN weights across on-chip memory of FPGAs. However, they assumed all data can fit in on-chip and did not consider the sparsity of the weight matrices. EIE [46] proposed a hardware model capable of sparse matrix-vector multiplication exploiting data pruning for MLP and fully-connected layers. ESE [45] proposed an optimized architecture for FPGA by applying pruning and quantization for LSTM models. Cambricon-X [122] suggested an accel-

erator architecture for a wide range of DNN models by supporting both pruned sparse matrices and dense matrices. Although EIE, ESE, and Cambricon-X are similar to MViD in that they accelerate sparse matrices, they suffer from the off-chip memory bandwidth bottleneck when MLP/RNN models do not fit in on-chip memory with limited capacity.

Accelerating RecSys: The computer systems community has recently seen growing interest in accelerating RecSys [40 – 42, 51, 52, 62 – 64, 73, 116]. FAFNIR [10] exploits a tree structure to perform all reductions in rank-level parallel NDP units, reducing the off-chip data movement and the number of connections between the cores and the NDP units. However, FAFNIR requires a separate chip outside of the DRAM module and is not compatible with the memory controller on the conventional processor die. Tensor Casting [74] proposed a rank-level parallelism NDP architecture for training RecSys models. In addition to the method of accelerating embedding layers, [109] exploited complementary partitions that reduced the number of embedding vectors while preserving the uniqueness of the embedding, and [120] proposed a low-precision embedding table with a high-precision cache.

Chapter 7

Conclusion

The demand for memory-intensive operations is increasing as interest in various deep neural network models is growing. Because it is costly to increase the main-memory bandwidth of the system, the Processing-in-Memory architectures that implement the processing unit in/near the main-memory to supply higher aggregate internal bandwidth have been proposed. In this dissertation, we propose two PIM architectures that accelerate memory-intensive operations used in various neural network models.

We have proposed MViD, a near-data processing architecture that accelerates matrix-vector multiplication (MV-mul) in RNNs by performing MV-mul with multiply-accumulate (MAC) units inside main-memory DRAM. MViD maximizes computational and energy efficiency by using a sparse matrix format and reducing weight precision through quantization. MViD populates the MAC units only on a portion of the DRAM banks considering the limited power budget of DRAM. We proposed an optimized sparse matrix format for MViD

This chapter is based on [65, 66, 99].

©2020 IEEE. Reprinted, with permission, from Byeongho Kim, Jongwook Chung, Eojin Lee, Wonkyung Jung, Sunjung Lee, Jaewan Choi, Jaehyun Park, Minbok Wi, Sukhan Lee, and Jung Ho Ahn, "MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks," IEEE Transactions on Computers, April 2020.

"TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory" ©2021 by Byeongho Kim, Jaehyun Park, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1145/3466752.3480080.

"TRiM: Tensor Reduction in Memory" ©2020 by Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn is licensed under CC BY 4.0. https://doi.org/10.1109/LCA.2020.3042805.

called delta encoding. To process requests from a processor concurrently with MV-mul, we allow MViD to slow-down or pause the progress of MV-mul. Furthermore, we solve the performance overhead caused by a pause in MV-mul through bank partitioning. Our evaluation shows that MViD improves the throughput of Deep Speech 2, an MV-mul dominant application, by up to 4.9× and 7.2× compared to the baseline system with four DRAM ranks when running Deep Speech 2 alone and with memory-intensive applications, respectively.

Also We have proposed TRiM, a near-data processing (NDP) architecture for accelerating tensor gather-and-reduction (GnR) operations in recommendation systems. First, we identified the challenges of state-of-the-art NDP architectures for accelerating GnR and the potential for further energy-efficiency improvements by unlocking the inherent bandwidth amplification opportunities within the DRAM chip's tree-topology-based datapath by populating processing elements for a reduction at the datapath. We proposed a two-stage instruction transfer scheme to amplify the control bandwidth by splitting the data path into two stages and pipelining them. The hot-entry replication scheme alleviates the load imbalance problem. We also improve the data reliability of TRiM by repurposing the existing on-die ECC to only detect and not correct errors during GnR as it accesses the embedding tables in a read-only manner. TRiM improves the performance of GnR by up to 7.7× and 3.9× compared to the DDR5-based baseline system and the state-of-the-art NDP architecture, respectively.

REFERENCES

[1] "Caffe2," https://caffe2.ai, 2017.

[2] "Deep Speech 2 Pytorch," https://github.com/SeanNaren/deepspeech.pytorch, 2017.

[3] "FBGEMM," https://github.com/pytorch/FBGEMM/tree/a5dd4824b9c5d80a698b1910859e72ecaef9c498, 2021.

[4] J. Ahn, M. Erez, and W. J. Dally, "Scatter-Add in Data Parallel Architectures," in Proceedings of the IEEE 11th International Symposium on High-Performance Computer Architecture, 2005.

[5] J. Ahn, S. Li, S. O, and N. P. Jouppi, "McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling," in Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2013.

[6] M. Alian and N. S. Kim, "NetDIMM: Low-Latency Near-Memory Network Interface Architecture," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019.

[7] M. Alian, S. W. Min, H. Asgharimoghaddam, A. Dhar, D. K. Wang, T. Roewer, A. McPadden, O. O'Halloran, D. Chen, J. Xiong, D. Kim, W.-m. Hwu, and N. S. Kim, "Application-Transparent Near-Memory Processing Architecture with Memory Channel Network," in Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture, 2018.

[8] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. Hannun, T. Han, L. Johannes, B. Jiang, C. Ju, B. Jun, P. LeGresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, D. Yogatama, B. Yuan, J. Zhan, and Z. Zhu, "Deep Speech 2: End-to-End Speech Recognition in English and Mandarin," in Proceedings of the 33rd International Conference on Machine Learning, 2016.

[9] M. J. Anderson, B. Chen, S. Chen, S. Deng, J. Fix, M. Gschwind, A. Kalaiah, C. Kim, J. Lee, J. Liang, H. Liu, Y. Lu, J. Montgomery, A. Moorthy, N. Satish, S. Naghshineh, A. Nayak, J. Park, C. Petersen, M. Schatz, N. Sundaram, B. Tang, P. Tang, A. Yang, J. Yu, H. Yuen, Y. Zhang, A. Anbudurai, V. Balan, H. Bojja, J. Boyd, M. Breitbach, C. Caldato, A. Calvo, G. Catron, S. Chandwani, P. Christeas, B. Cottel, B. Coutinho,

A. Dalli, A. Dhanotia, O. Duncan, R. Dzhabarov, S. Elmir, C. Fu, W. Fu, M. Fulthorp, A. Gangidi, N. Gibson, S. Gordon, B. P. Hernandez, D. Ho, Y. Huang, O. Johansson, S. Juluri, and et al., “First-Generation Inference Accelerator Deployment at Facebook,” arXiv:2107.04140, 2021.

[10] B. Asgari, R. Hadidi, J. Cao, D. E. Shim, S.-K. Lim, and H. Kim, “FAFNIR: Accelerating Sparse Gathering by Using Efficient Near-Memory Intelligent Reduction,” in Proceedings of the IEEE 27th International Symposium on High-Performance Computer Architecture, 2021.

[11] H. Asghari-Moghaddam, Y. H. Son, J. Ahn, and N. S. Kim, “Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems,” in Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, 2016.

[12] N. Bell and M. Garland, “Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors,” in Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, 2009.

[13] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, “Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks,” in 2021 30th International Conference on Parallel Architectures and Compilation Techniques, 2021.

[14] S. Cha, S. O, H. Shin, S. Hwang, K. Park, S. J. Jang, J. S. Choi, G. Y. Jin, Y. H. Son, H. Cho, J. Ahn, and N. S. Kim, “Defect Analysis and

Cost-Effective Resilience Architecture for Future DRAM Devices," in Proceedings of the IEEE 23rd International Symposium on High Performance Computer Architecture, 2017.

[15] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in Proceedings of the 43rd Annual ACM/IEEE International Symposium on Computer Architecture, 2016.

[16] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers, 2016.

[17] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, "State-of-the-Art Speech Recognition with Sequence-to-Sequence Models," in Proceedings of the 43rd IEEE International Conference on Acoustics, Speech and Signal Processing, 2018.

[18] B. Y. Cho, Y. Kwon, S. Lym, and M. Erez, "CHoNDA: Near Data Acceleration with Concurrent Host Access," arXiv:1908.06362, 2019.

[19] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in Pro-

ceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 2014.

[20] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 2014.

[21] P. Covington, J. Adams, and E. Sargin, "Deep Neural Networks for YouTube Recommendations," in Proceedings of the 10th ACM Conference on Recommender Systems, 2016.

[22] CriteoLabs, "Kaggle Display Advertising Challenge Dataset," http://labs.criteo.com/2014/02/download-kaggle-display-advertising-challenge-dataset, 2014.

[23] W. J. Dally and B. P. Towles, Principles and Practices of Interconnection Networks. Elsevier, 2004.

[24] Z. Deng, J. Park, P. T. P. Tang, H. Liu, J. Yang, H. Yuen, J. Huang, D. Khudia, X. Wei, E. Wen, D. Choudhary, R. Krishnamoorthi, C.-J. Wu, S. Nadathur, C. Kim, M. Naumov, S. Naghshineh, and M. Smelyanskiy, "Low-Precision Hardware Architectures Meet Recommendation Model Inference at Scale," IEEE Micro, vol. 41, no. 5, 2021.

[25] F. Devaux, "The true Processing In Memory accelerator," in 2019 IEEE Hot Chips 31 Symposium, 2019.

[26] G. Diamos, S. Sengupta, B. Catanzaro, M. Chrzanowski, A. Coates, E. Elsen, J. Engel, A. Hannun, and S. Satheesh, “Persistent RNNs: Stashing Recurrent Weights On-Chip,” in International Conference on Machine Learning, 2016.

[27] J. Doweck, W. F. Kao, A. K. y. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, “Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake,” Micro, IEEE, vol. 37, no. 2, 2017.

[28] M. Drumond, A. Daglis, N. Mirzadeh, D. Ustiugov, J. Picorel, B. Falsafi, B. Grot, and D. Pnevmatikatos, “The Mondrian Data Engine,” in Proceedings of the ACM/IEEE 44th Annual International Symposium on Computer Architecture, 2017.

[29] A. Eisenman, M. Naumov, D. Gardner, M. Smelyanskiy, S. Pupyrev, K. Hazelwood, A. Cidon, and S. Katti, “Bandana: Using Non-volatile Memory for Storing Deep Learning Models,” in Proceedings of Machine Learning and Systems, 2019.

[30] A. Farmahini-Farahani, J. Ahn, K. Morrow, and N. S. Kim, “NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules,” in Proceedings of the 21st International Symposium on High-Performance Computer Architecture, 2015.

[31] I. Fernandez, R. Quislant, C. Giannoula, M. Alser, J. Gomez-Luna, E. Gutierrez, O. Plata, and O. Mutlu, “NATSA: A Near-Data Processing

Accelerator for Time Series Analysis," in Proceedings of the International Conference on Computer Design, 2020.

[32] D. Foley and J. Danskin, "Ultra-Performance Pascal GPU and NVLink Interconnect," Micro, IEEE, vol. 37, no. 2, Mar/Apr 2017.

[33] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in Proceedings of the 45th ACM/IEEE International Symposium on Computer Architecture, 2018.

[34] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems, 2017.

[35] J. Gomez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture," arXiv:2105.03814, 2021.

[36] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT press, 2016.

[37] A. Graves and N. Jaitly, "Towards End-to-End Speech Recognition with Recurrent Neural Networks," in Proceedings of the 31st International Conference on Machine Learning, 2014.

[38] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in Proceedings of the 38th IEEE International Conference on Acoustics, Speech and Signal Processing, 2013.

[39] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, and Y. Xie, "iPIM: Programmable In-Memory Image Processing Accelerator Using Near-Bank Architecture," in Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture, 2020.

[40] 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," in Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture, 2020.

[41] U. Gupta, S. Hsia, J. Zhang, M. Wilkening, J. Pombra, H.-H. S. Lee, G.-Y. Wei, C.-J. Wu, and D. Brooks, "RecPipe: Co-Designing Models and Hardware to Jointly Optimize Recommendation Quality and Performance," in Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture, 2021.

[42] U. Gupta, C.-J. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, M. Hempstead, B. Jia, H.-H. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. Zhang, "The Architectural Implications of Facebook's DNN-Based Personalized Recommendation," in Proceedings of the IEEE 26th International Symposium on High Performance Computer Architecture, 2020.

[43] M. Halpern, Y. Zhu, and V. J. Reddi, "Mobile cpu's rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction," in Proceedings of the 22nd International Symposium on High-Performance Computer Architecture, 2016.

[44] K. J. Han, A. Chandrashekaran, J. Kim, and I. Lane, "The CAPIO 2017 Conversational Speech Recognition System," CoRR, vol. abs/1801.00059, 2017.

[45] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. J. Dally, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," in Proceedings of the 25th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017.

[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 Proceedings of the 43rd ACM/IEEE International Symposium on Computer Architecture, 2016.

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[48] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, 1997.

[49] M. Horowitz, "Computing's Energy Problem (and what we can do about it)," in International Solid-State Circuits Conference, 2014.

[50] W. Huangfu, X. Li, S. Li, X. Hu, P. Gu, and Y. Xie, "MEDAL: Scalable DIMM Based Near Data Processing Accelerator for DNA Seeding Algorithm," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019.

[51] R. Hwang, T. Kim, Y. Kwon, and M. Rhu, "Centaur: A Chiplet-Based, Hybrid Sparse-Dense Accelerator for Personalized Recommendations," in Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture, 2020.

[52] B. Hyun, Y. Kwon, Y. Choi, J. Kim, and M. Rhu, "NeuMMU: Architectural Support for Efficient Address Translations in Neural Processing Units," in Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems, 2020.

[53] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler, "Data Movement Is All You Need: A Case Study on Optimizing Transformers," in Proceedings of Machine Learning and Systems, 2021.

[54] JEDEC, "DDR4 SDRAM Standard," 2017.

[55] JEDEC, "Low Power Double Data Rate 4 (LPDDR4)," 2017.

[56] JEDEC, "DDR4 Registering Clock Driver," 2019.

[57] JEDEC, "DDR5 SDRAM Standard," 2020.

[58] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh, "NoC Architectures for Silicon Interposer Systems: Why Pay for More Wires when You Can Get Them (from Your Interposer) for Free?" in Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014.

[59] N. P. Jouppi, A. B. Kahng, N. Muralimanohar, and V. Srinivas, "CACTI-IO: CACTI with Off-chip Power-Area-Timing Models," in IEEE Transactions on Very Large Scale Integration, vol. 23, no. 7, 2015.

[60] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in Proceedings of the 44th ACM/IEEE International Symposium on Computer Architecture, 2017.

[61] W. Jung, D. Jung, B. Kim, S. Lee, W. Rhee, and J. Ahn, "Restructuring Batch Normalization to Accelerate CNN Training," in Proceedings of

Machine Learning and Systems, 2019.

[62] D. Kalamkar, E. Georganas, S. Srinivasan, J. Chen, M. Shiryaev, and A. Heinecke, "Optimizing Deep Learning Recommender Systems Training on CPU Cluster Architectures," in International Conference for High Performance Computing, Networking, Storage and Analysis, 2020.

[63] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H.-H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C.-J. Wu, M. Hempstead, and X. Zhang, "RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing," in Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture, 2020.

[64] L. Ke, X. Zhang, J. So, J.-G. Lee, S.-H. Kang, S. Lee, S. Han, Y. Cho, J. H. Kim, Y. Kwon, K. Kim, J. Jung, I. Yun, S. J. Park, H. Park, J. Song, J. Cho, K. Sohn, N. S. Kim, and H.-H. S. Lee, "Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM," IEEE Micro, no. 01, 2021.

[65] B. Kim, J. Chung, E. Lee, W. Jung, S. Lee, J. Choi, J. Park, M. Wi, S. Lee, and J. Ahn, "MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks," IEEE Transactions on Computers, vol. 69, no. 7, 2020.

[66] B. Kim, J. Park, E. Lee, M. Rhu, and J. Ahn, "TRiM: Tensor Reduction in Memory," IEEE Computer Architecture Letters, vol. 20, no. 1, 2021.

[67] D. Kim, M. Park, S. Jang, J.-Y. Song, H. Chi, G. Choi, S. Choi, J. Kim, C. Kim, K. Kim, K. Koo, S. Song, Y. Kim, D. U. Lee, J. Lee, D. Kim, K. Kwon, M. Han, B. Choi, H. Kim, S. Ku, Y. Kim, J. Kim, S. Kim, Y. Seo, S. Oh, D. Im, H. Kim, J. Choi, J. Chung, C. Lee, Y. Lee, J.-H. Cho, J. Chun, and J. Oh, "23.2 A 1.1V 1ynm 6.4Gb/s/pin 16Gb DDR5 SDRAM with a Phase-Rotator-Based DLL, High-Speed SerDes and RX/TX Equalization Scheme," in 2019 IEEE International Solid-State Circuits Conference, 2019.

[68] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in Proceedings of the 43rd ACM/IEEE International Symposium on Computer Architecture, 2016.

[69] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," IEEE Computer Architecture Letters, vol. 15, no. 1, 2016.

[70] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," arXiv:1609.02907, 2016.

[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in Advances in Neural Information Processing Systems, 2012.

[72] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, S. O, H.-S. Yu, H. Lee, S. Y. Kim, Y. Cho, J. G. Kim, J. Choi, H.-S. Shin, J. Kim, B. Phuah, H. Kim, M. J. Song, A. Choi, D. Kim, S. Kim, E.-B.

Kim, D. Wang, S. Kang, Y. Ro, S. Seo, J. Song, J. Youn, K. Sohn, and N. S. Kim, “25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications,” in 2021 IEEE International Solid- State Circuits Conference, 2021.

[73] Y. Kwon, Y. Lee, and M. Rhu, “TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning,” in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019.

[74] Y. Kwon, Y. Lee, and M. Rhu, “Tensor Casting: Co-Designing Algorithm-Architecture for Personalized Recommendation Training,” in Proceedings of the IEEE 27th International Symposium on High-Performance Computer Architecture, 2021.

[75] E. Lee, J. Chung, D. Jung, S. Lee, S. Li, and J. Ahn, “Work as a Team or Individual: Characterizing System-level Impacts of Main Memory Partitioning,” in IEEE International Symposium on Workload Characterization, 2017.

[76] S. Lee, H. Cho, Y. Son, Y. Ro, N. S. Kim, and J. Ahn, “Leveraging Power-Performance Relationship of Energy-Efficient Modern DRAM Devices,” IEEE Access, vol. 6, 2018.

[77] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, S. O, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, “Hardware Architecture and Software Stack for PIM Based on Commer-

cial DRAM Technology : Industrial Product," in 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture, 2021.

[78] J. Li, X. Wang, A. Tumeo, B. Williams, J. D. Leidel, and Y. Chen, "PIMS: A Lightweight Processing-in-Memory Accelerator for Stencil Computations," in Proceedings of the International Symposium on Memory Systems, 2019.

[79] S. Li, A. O. Glova, X. Hu, P. Gu, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "SCOPE: A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator," in Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture, 2018.

[80] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-based Reconfigurable In-Situ Accelerator," in Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, 2017.

[81] A. Limaye and T. Adegbija, "A Workload Characterization of the SPEC CPU2017 Benchmark Suite," in IEEE International Symposium on Performance Analysis of Systems and Software, 2018.

[82] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-Memory for Energy-Efficient Neural Network Training: A Heterogeneous Approach," in Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture, 2018.

[83] S.-L. Lu, Y.-C. Lin, and C.-L. Yang, "Improving DRAM Latency with Dynamic Asymmetric Subarray," in Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture, 2015.

[84] D. Luebke, "CUDA: Scalable Parallel Programming for High-Performance Scientific Computing," in 2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, 2008.

[85] S. A. McKee, "Reflections on the Memory Wall," in Proceedings of the 1st Conference on Computing Frontiers, 2004.

[86] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-End Speech Recognition using Deep RNN Models and WFST-based Decoding," in IEEE Workshop on Automatic Speech Recognition and Understanding, 2015.

[87] Micron, "Calculating Memory Power for DDR4 SDRAM," 2017.

[88] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," in Proceedings of the 35th ACM/IEEE International Symposium on Computer Architecture, 2008.

[89] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring Sparsity in Recurrent Neural Networks," in Proceedings of the 5th International Conference on Learning Representations, 2017.

[90] M. Naumov, D. Mudigere, H.-J. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko,

S. Pereira, X. Chen, and M. Smelyanskiy, "Deep Learning Recommendation Model for Personalization and Recommendation Systems," arXiv:1906.00091, 2019.

[91] NCSU, "FreePDK45," https://www.eda.ncsu.edu/wiki/FreePDK45:Contents, 2011.

[92] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling Memcache at Facebook," in 10th USENIX Symposium on Networked Systems Design and Implementation, 2013.

[93] S. O, S. Kwon, Y. H. Son, Y. Park, and J. Ahn, "CIDR: A Cache Inspired Area-Efficient DRAM Resilience Architecture against Permanent Faults," IEEE Computer Architecture Letters, vol. 14, no. 1, 2015.

[94] S. O, Y. H. Son, N. S. Kim, and J. Ahn, "Row-Buffer Decoupling: A Case for Low-Latency DRAM Microarchitecture," in Proceedings of the ACM/IEEE 41st Annual International Symposium on Computer Architecture, 2014.

[95] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems," in Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, 2017.

[96] R. Oh, B. Lee, S.-W. Shin, W. Bae, H. Choi, I. Song, Y.-S. Lee, J.-H. Choi, C.-W. Kim, S.-J. Jang, and J. S. Choi, "Design Technologies for

a 1.2V 2.4Gb/s/pin High Capacity DDR4 SDRAM with TSVs," in IEEE Symposium on VLSI Circuits, Digest of Technical Papers, 2014.

[97] T.-Y. Oh, H. Chung, J.-Y. Park, K.-W. Lee, S. Oh, S.-Y. Doo, H.-J. Kim, C. Lee, H.-R. Kim, J.-H. Lee, J.-I. Lee, K.-S. Ha, Y. Choi, Y.-C. Cho, Y.-C. Bae, T. Jang, C. Park, K. Park, S. Jang, and J. S. Choi, "A 3.2 Gbps/pin 8 Gbit 1.0 V LPDDR4 SDRAM With Integrated ECC Engine for Sub-1 V DRAM Core Operation," IEEEjournal of Solid-State Circuits, vol. 50, no. 1, 2015.

[98] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR Corpus Based on Public Domain Audio Books," in Proceedings of the 40th IEEE International Conference on Acoustics, Speech and Signal Processing, 2015.

[99] J. Park, B. Kim, S. Yun, E. Lee, M. Rhu, and J. Ahn, "TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory," in Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture, 2021.

[100] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. S. Khudia, J. Law, P. Malani, A. Malevich, N. Satish, J. M. Pino, M. Schatz, A. Sidorov, V. Sivakumar, A. Tulloch, X. Wang, Y. Wu, H. Yuen, U. Diril, D. Dzhulgakov, K. M. Hazelwood, B. Jia, Y. Jia, L. Qiao, V. Rao, N. Rotem, S. Yoo, and M. Smelyanskiy, "Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications," arXiv:1811.09886, 2018.

[101] M. Patel, J. Kim, T.-M. Shahroodi, H. Hassan, and O. Mutlu, "Bit-Exact ECC Recovery (BEER): Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics," in Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture, 2020.

[102] W. W. Peterson and E. J. Weldon, Error-correcting codes. MIT press, 1972.

[103] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "MLPerf Inference Benchmark," in Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture, 2020.

[104] F. Sadi, J. Sweeney, T. M. Low, J. C. Hoe, L. Pileggi, and F. Franchetti, "Efficient SpMV Operation for Large and Highly Sparse Matrices using Scalable Multi way Merge Parallelization," in Proceedings of the 52th Annual IEEE/ACM International Symposium on Microarchitecture, 2019.

[105] Samsung Electronics, "8Gb B-die DDR4 SDRAM," https://www.samsung.com/semiconductor/global.semi/file/resource/2017/

11/8G_B_DDR4_Samsung_Spec_Rev2_1_Feb_17-0.pdf, 2017.

[106] G. Saon, G. Kurata, T. Sercu, K. Audhkhasi, S. Thomas, D. Dimitriadis, X. Cui, B. Ramabhadran, M. Picheny, L.-L. Lim, B. Roomi, and P. Hall, "English Conversational Telephone Speech Recognition by Humans and Machines," Interspeech, 2017.

[107] M. Schuster and K. K. Paliwal, "Bidirectional Recurrent Neural Networks," IEEE Transactions on Signal Processing, vol. 45, no. 11, 1997.

[108] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, 2002.

[109] H.-J. M. Shi, D. Mudigere, M. Naumov, and J. Yang, "Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems," in Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020.

[110] H. Shin, D. Kim, E. Park, S. Park, Y. Park, and S. Yoo, "McDRAM: Low Latency and Energy-Efficient Matrix Computations in DRAM," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, 2018.

[111] Y. H. Son, S. Lee, S. O, S. Kwon, N. S. Kim, and J. Ahn, "CiDRA: A Cache-inspired DRAM Resilience Architecture," in Proceedings of the

IEEE 21st International Symposium on High Performance Computer Architecture, 2015.

[112] Y. H. Son, S. O, Y. Ro, J. W. Lee, and J. Ahn, "Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations," in Proceedings of the ACM/IEEE 40th Annual International Symposium on Computer Architecture, 2013.

[113] S. Thoziyoor, J. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies," in Proceedings of the 35th ACM/IEEE International Symposium on Computer Architecture, 2008.

[114] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is All You Need," in Advances in Neural Information Processing Systems, 2017.

[115] M. Wang, C. Meng, G. Long, C. Wu, J. Yang, W. Lin, and Y. Jia, "Characterizing Deep Learning Training Workloads on Alibaba-PAI," in 2019 IEEE International Symposium on Workload Characterization, 2019.

[116] M. Wilkening, U. Gupta, S. Hsia, C. Trippel, C.-J. Wu, D. Brooks, and G.-Y. Wei, "RecSSD: Near Data Processing for Solid State Drive Based Recommendation Inference," in Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems, 2021.

[117] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," Communications of the ACM, vol. 52, no. 4, 2009.

[118] C.-J. Wu, R. Burke, E. Chi, J. Konstan, J. McAuley, Y. Raimond, and H. Zhang, "Developing a Recommendation Benchmark for MLPerf Training and Inference," arXiv:2003.07336, 2020.

[119] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," ACM SIGARCH Computer Architecture News, vol. 23, no. 1, 1995.

[120] J. A. Yang, J. Huang, J. Park, P. T. P. Tang, and A. Tulloch, "Mixed-Precision Embedding Using a Cache," arXiv:2010.11305, 2020.

[121] C. Zhang, T. Meng, and G. Sun, "PM3: Power Modeling and Power Management for Processing-in-Memory," in 2018 IEEE International Symposium on High Performance Computer Architecture, 2018.

[122] 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 Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, 2016.

[123] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep Learning Based Recommender System: A Survey and New Perspectives," ACM Computing Surveys, vol. 52, no. 1, 2019.

[124] Z. Zhang, C. Xie, J. Wang, W. Zhang, and X. Fu, "Towards Memory Friendly Long-Short Term Memory Networks (LSTMs) on Mobile

GPUs," in Proceedings of the 51st Annual IEEE/ACM International Sym-posium on Microarchitecture, 2018.

[125] F. Zhu, J. Pool, M. Andersch, J. Appleyard, and F. Xie, "Sparse Persistent RNNs: Squeezing Large Recurrent Networks On-Chip," in Proceedings of the 6th International Conference on Learning Representations, 2018.

국문초록

최근 많은 신경망 연구들이 관심을 받으면서, RNN 모델 혹은 추천 시스템 모델과 같은 메모리 집약적 신경망 모델들이 다양한 작업을 처리하기 위해서 등장하고있다. RNN 모델과 추천 시스템 모델은 대부분의 실행 시간 동안 각각 행렬-벡터 곱을 연산하고 임베딩 레이어를 처리한다. 임베딩 레이어의 기본 연산인 GnR 연산은 여러개의 임베딩 벡터를 모은 다음 이들을 합치는 동작을 한다. RNN 처리시 필요한 행렬과 추천 시스템 모델 처리시 필요한 임베딩 테이블은 재사용성이 낮고 이들의 크기는 계속 증가하여 온칩 스토리지에 저장될 수 없기 때문에 행렬-벡터 곱 및 GnR 연산의 성능 및 에너지 효율성은 주 메모리 DRAM의 성능 및 에너지 효율성에 의해 결정된다. 따라서 DRAM 내에서 이러한 연산을 처리하는 방식이 관심을 끌고있다.

본 논문에서는 먼저 DRAM 뱅크 내부에 MAC 유닛을 배치하여 행렬-벡터 곱을 수행하는 MViD라는 주 메모리 구조를 제안한다. 그리고 더 높은 계산 효율성을 위해 희소 행렬 형식을 사용하고 양자화를 활용한다. DRAM 장치가 사용할 수 있는 제한된 전력 때문에 DRAM 뱅크의 일부에만 MAC 장치를 구현한다. 전력 제한 조건을 충족하면서 프로세서의 메모리 요청을 동시에 처리하기 위해 행렬-벡터곱을 늦추거나 일시 중지하도록 MViD를 설계한다. 그 결과로 MViD가 메모리 집약적 워크로드로 Deep Speech 2의 추론을 실행하면서 4개의 DRAM

랭크를 사용하는 프로세서에서 행렬-벡터곱을 처리하는 기준 시스템에 비해 7.2 배 더 높은 처리량을 제공한다는 것을 보여준다.

그리고 우리는 추천 시스템을 가속하기 위한 메모리 근처 처리 구조인 TRiM을 제안한다. DRAM 데이터 경로가 계층적 트리 구조를 갖는다는 사실을 기반으로 TRiM은 DDR4/5 랭크/뱅크그룹/뱅크 수준에서 DRAM 내부 벡터 감소 장치로 DRAM 데이터 경로를 강화한다. 병렬로 실행되는 여러 벡터 감소 장치에 명령을 효과적으로 제공하기 위해 DRAM의 인터페이스를 수정한다. 또한 벡터 감소 장치에서 발생하는 부하 불균형을 완화하기 위해 호스트 측 구조에 핫 임베딩 벡터 복제를 제안한다. DDR5를 기반으로 하는 최적의 TRiM 설계는 DRAM 칩의 2.66%에 해당하는 크기 오버헤드만으로 최대 7.7배 및 3.9배의 속도 향상을 달성하고 임베딩 벡터 수집의 에너지 소비를 55% 및 50% 줄인다.

주요어: 프로세싱-인-메모리, 메모리 근처 처리, DRAM 내부 처리, 메모리 집약적, 메모리 세부구조

학번: 2017-22676