Accelerating Personalized Recommendation with Cross-level Near-Memory Processing

Haifeng Liu*
Long Zheng*†
Huazhong University of Science and Technology
Zhejiang Lab
{hfliu,longzh}@hust.edu.cn

Yu Huang*
Chaoqiang Liu*
Huazhong University of Science and Technology
Zhejiang Lab
{yuh,chqliu}@hust.edu.cn

Xiangyu Ye*
Jingrui Yuan*
Huazhong University of Science and Technology
Zhejiang Lab
{xyye,jryuan}@hust.edu.cn

Xiaofei Liao*
Huazhong University of Science and Technology
xfliao@hust.edu.cn

Hai Jin*
Huazhong University of Science and Technology
hjin@hust.edu.cn

Jingling Xue
University of New South Wales
jingling@cse.unsw.edu.au

ABSTRACT

The memory-intensive embedding layers of the personalized recommendation systems are the performance bottleneck as they demand large memory bandwidth and exhibit irregular and sparse memory access patterns. Recent studies propose *near memory processing* (NMP) to accelerate memory-bound embedding operations. However, due to the load imbalance caused by the skewed access frequency of the embedding data, existing NMP solutions that exploit fine-grained memory parallelism fail to translate the increasingly massive internal bandwidth to performance improvements, leading to resource underutilization and hardware overhead.

We propose an efficient yet practical fine-grained NMP accelerator for embedding operations. We architect ReCross, a cross-level NMP architecture that exploits rank, bank-group, and subarray-level memory parallelism in a unified DIMM-based memory system by supporting rank, bank-group, and bank-level NMP to accommodate various bandwidth requirements of embedding data. In addition, we present a novel embedding partitioning technique to quantify the bandwidth requirements of embedding tables and allocate them to appropriate NMP levels. ReCross innovatively collaborates the data and architecture characteristics for the NMP embedding layer acceleration, achieving high resource utilization and performance. Our evaluation shows that ReCross outperforms a state-of-the-art bank-group level NMP solution, TRiM-G, by 2.5× with nearly the same area overhead and bank-level NMP solution, TRiM-B, by 1.8× with an area overhead reduction of 4×.

*Authors are with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology in HUST.
†Long Zheng is the corresponding author.

ISCA '23, June 17–21, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0095-8/23/06…$15.00
https://doi.org/10.1145/3579371.3589101

CCS CONCEPTS

• **Computer systems organization** → **Parallel architectures**.

KEYWORDS

Near-memory-processing, memory system, DIMM, DRAM

ACM Reference Format:
Haifeng Liu, Long Zheng, Yu Huang, Chaoqiang Liu, Xiangyu Ye, Jingrui Yuan, Xiaofei Liao, Hai Jin, and Jingling Xue. 2023. Accelerating Personalized Recommendation with Cross-level Near-Memory Processing. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3579371.3589101

1 INTRODUCTION

Recommendation systems have become an internet infrastructure due to their industrial importance. They have been widely deployed in a wide variety of applications, including e-commerce [55, 64], search engine [28], media service [10, 14], and social network [50]. To provide fast, accurate, and personalized recommendation services, deep learning has recently been applied in the recommendation systems and attracted increasing attention [15, 19, 50, 71]. These deep learning-based personalized recommendation systems play an increasingly important role and consume substantial data center resources spent on AI [18]. For instance, the *deep learning-based recommendation model* (DLRM) occupies over 80% of inference cycles in Facebook's AI data center [18].

To process the personalized interaction between users and items, deep learning-based recommendation systems employ an embedding layer, which performs embedding table lookup (gathering) and pooling (reduction) operations to transform sparse categorical features into a dense representation. However, unlike compute-intensive CNNs and DNNs, the embedding layer that exhibits large storage requirements, low compute intensity, and irregular memory access patterns requires vast memory bandwidth and bottlenecks the system performance in the conventional architecture [44, 65].

Recently, *processing in memory* (PIM) technology has been demonstrated as a promising solution to accelerate memory-bounded workloads [9, 21, 22, 49, 58]. One of the PIM approaches is *near memory processing* (NMP) which integrates *processing elements*

(PEs) in the memory side to leverage massive memory level parallelism and reduce data movements [12, 47]. TensorDIMM [38] and RecNMP [42] are two NMP architectures that exploit the rank-level memory parallelism for accelerating embedding operations by adding a PE for each rank inside the DIMM buffer. The reduction operations are offloaded to the NMP PEs, and only the reduced results are transferred to the host, significantly reducing data movements.

By placing the PE inside the DRAM chip, TRiM [52] exploits fine-grained bank-group and bank-level memory parallelism, referred to as TRiM-G and TRiM-B, respectively. Since the gathered embedding vectors are reduced in the PEs, the internal bandwidth is scaled with the number of *memory nodes*, e.g., ranks, bank-groups, and banks (corresponding to rank, bank-group, and bank-level NMP, respectively). In the conventional DRAM architectures, the number of banks is usually several times more than the number of bank-groups, which is several times more than that of ranks. Therefore, exploiting fine-grained NMP shows great potential. However, existing fine-grained NMP solutions fail to translate the increasingly massive internal bandwidth into performance benefits, leading to massive resource underutilization and remarkable hardware overhead. For example, the TRiM-B achieves only a speedup of up to 1.31× over the TRiM-G, with 4× more PEs.

In this paper, we observe that the embedding data exhibit a skewed access frequency that a small fraction of data take up the vast majority of memory accesses, bottlenecking the performance of the embedding layer (discussed in §3.1). However, existing NMP solutions consider improving memory parallelism exclusively from the perspective of the underlying memory architecture, regardless of the embedding data characteristics. Thus, a symmetrical NMP architecture is often applied to treat all data equally, resulting in severe load imbalance and resource underutilization. Unlike existing NMP techniques that purely pursue higher internal memory bandwidth of a memory architecture [38, 42, 52], we argue that it is of great importance to consider both the embedding data and memory architecture characteristics simultaneously.

To achieve this goal, we present an efficient yet practical hardware and software co-design for embedding layer acceleration. In hardware design, we architect ReCross, a cross-level NMP architecture that exploits rank, bank-group, and bank-level NMP to accommodate different memory bandwidth requirements of different embedding data. Targeting the small fraction of frequently accessed data, we innovatively exploit the subarray level parallelism in the bank-level NMP, making a great leap forward in performance. In software design, we conduct a statistical analysis to distinguish the bandwidth requirements of different embedding data. The variability of the embedding tables leads to a varying spectrum of memory access distributions, exhibiting various requirements for the underlying NMP implementations. To address this, we propose a bandwidth-aware partitioning algorithm to allocate embedding tables appropriately in the corresponding NMP levels with optimal performance. Compared with prior studies [38, 42, 52], ReCross offers cross-level memory parallelism to accommodate different data requirements, naturally addressing load imbalance and resource under-utilization. This paper makes the following contributions:

- We conduct an in-depth analysis of embedding data to characterize the performance bottleneck and comprehensively explore the potential of fine-grained memory parallelism.

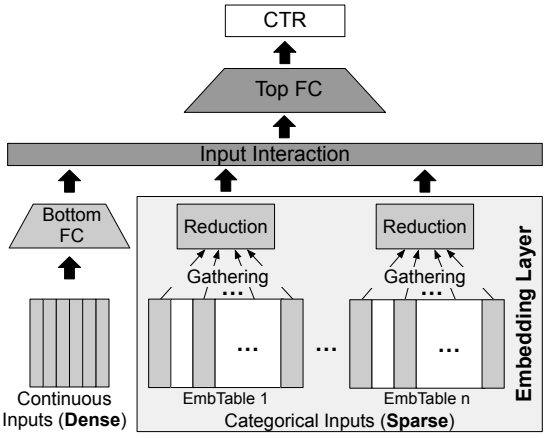

Figure 1: DLRM overview

- We architect a cross-level NMP architecture, ReCross, with rank, bank-group, and bank-level NMP, to accommodate different memory bandwidth requirements by considering the skewed embedding access frequency, and exploit innovatively the subarray level parallelism in the bank-level NMP for the most frequently accessed data.
- We conduct a statistical analysis to quantify the bandwidth requirements of embedding data and present a bandwidth-aware partitioning algorithm for each embedding table to be allocated appropriately in corresponding NMP level.
- We have implemented ReCross and evaluated it on the embedding layer of DLRM [50]. Results show that ReCross outperforms the state-of-the-art back-group level NMP TRiM-G (bank-level NMP TRiM-B) by 2.5× with nearly the same area overhead (1.8× with an area overhead reduction of 4×).

2 BACKGROUND

In this section, we introduce the *deep learning-based recommendation system* (DLRM), the modern DRAM-based main memory system, and existing NMP solutions for DLRM.

2.1 Personalized Recommendation Models

In this work, we adopt the Facebook's production-scale deep learning recommendation model [50]. Figure 1 provides a DLRM overview. As we can see, DLRM comprises two *fully connected* (FC) layers and an embedding layer with various embedding tables. The bottom FC layer deals with dense inputs containing personal information, including age, gender, and other preferences. The embedding layer handles the categorical inputs containing the related item indices collected from users' past interactions. In addition, the outputs from the bottom FC and the embedding layers are combined together as input to the top FC layer to produce the *click-through rate* (CTR), indicating the items' priority to users.

The embedding layer comprises embedding tables of various numbers and sizes, depending on the use case. Each embedding table is a two-dimensional matrix containing millions of rows, each of which is an unique embedding vector usually containing 32 to 256 dimensions [66]. For each embedding operation, a gathering operation is applied to collect the related embedding vectors by looking up embedding tables. The number of vectors collected for

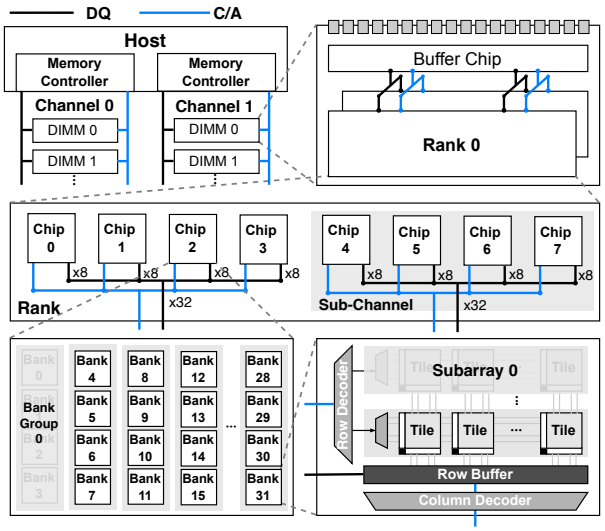

Figure 2: The main memory architecture overview

each operation is often 20 to 80 [42]. Then, the collected vectors are aggregated by a reduction operation (e.g., element-wise summation, average, concatenation) [50]. Typically, embedding operations are processed in batches to improve the throughput [18].

Compared to the FC layers that occupy relatively small storage, an embedding layer can take up hundreds of GBs to several TBs for production-scale recommendation models. Currently, an embedding table of such a large footprint is often stored in a capacity-optimized main memory [18]. Besides, unlike the compute-intensive FC layers that perform regular *multiplier and accumulation* (MAC) operations and are widely studied for acceleration, the embedding layer exhibits irregular memory access patterns and low compute intensity. Thus, its performance is limited on conventional architectures. As such, it is imperative to accelerate embedding operations due to the rapid growth of the recommendation model scales.

2.2 Modern Main Memory Architecture

As the embedding data size exceeds the capacity of the on-chip memory, the production-scale environment usually stores them in the main memory. Figure 2 shows the hierarchical architecture of a modern DDR-based main memory system. Its memory controllers sit at the topmost level. Each memory controller controls a memory *channel* that can be accessed independently. Each memory channel is connected to several *ranks* that share a *command/address* (C/A) and *data* (DQ) bus. The ranks are physically housed in the *Dual In-line Memory Modules* (DIMMs), each of which can be packed with multiple ranks to maximize memory parallelism and meet large memory capacity requirements. Due to the limited DRAM chip I/O data width (usually 4, 8, or 16), multiple chips are often populated into one rank and operated in a lock-step manner to provide the 64-bit width. The DRAM chip is organized by multiple *bank-groups*, each of which comprises several *banks*.

A DRAM bank can be logically viewed as a monolithic array of rows with a row decoder, a column decoder, and a row buffer. Implementing a large array often causes long bit-line and word-line, imposing a great challenge to the reliability and latency of the signal. Thus, a modern DRAM bank is usually composed of multiple DRAM cell tiles [7, 24, 33, 62], each of which is typically sized of 512×512 [63]. As shown in Figure 2, all tiles in the horizontal direction share the same set of global word lines with a local row decoder and are operated in a lock-step manner. Such a *row of tiles* is referred to as a *subarray* [24]. Even though only one subarray is activated for each operation, multiple accesses to the same bank (even to different subarrays) are still serial in modern DRAM, leaving ample opportunities to exploit a higher level of memory parallelism.

The concept of bank-group is first introduced from DDR4, in which a bank-group is packed with several banks that share a local I/O gating [26]. By interleaving the accesses to bank-groups, DDR4 increases the data transfer rate and bandwidth while keeping the DRAM internal operating frequency at a low level. DDR5 [27] further separates a 64-bit channel into two sub-channels, as shown in Figure 2. Each sub-channel can operate independently and provide 64 bytes of data with a new default burst length of 16, reaching a higher data transfer rate while keeping the same internal frequency as DDR4. Besides, DDR5 doubles the number of bank-groups per rank and keeps the same number of banks per bank-group as DDR4. Therefore, DDR5 is able to provide a larger monolithic device capacity by up to 64Gb per DRAM chip.

2.3 NMP for DLRM Acceleration

Since gathering operations demand massive memory bandwidth and reduction operations are lightweight at a low compute intensity, the embedding layer is amenable to acceleration using the *near memory processing* (NMP) technique. Recent studies [38, 42, 52] offload the embedding gathering and reduction operations into the main memory, allowing transferring only the result vector (instead of all gathered embedding vectors) to the host, reducing the data movement between the main memory and host significantly.

The hierarchical architecture of the memory system allows exploiting memory parallelism at different levels. TensorDIMM [38] and RecNMP [42] utilize the rank-level NMP by placing a *processing element* (PE) inside the rank buffer. This way, embedding vectors from one rank can be reduced locally, and only the reduced partial result is transferred to the host. TRiM [52] further proposes to exploit bank-group and bank-level NMP by integrating PEs inside the DRAM chip near the bank-group and bank, increasing the internal bandwidth for boosting performance.

With such fine-grained NMP being exploited, the number of PEs increases significantly, yielding significant extra hardware overhead. For example, if each rank consists of 8 bank-groups, then 8 PEs will be placed in each DRAM chip for the bank-group level NMP. Given that each bank-group comprises 4 banks, bank-level NMP further requires 4× more PEs. Integrating so many PEs in a highly-integrated DRAM chip not only poses a great challenge to the chip fabrication but also induces prohibitive area and energy overhead.

Moreover, previous works increase the internal bandwidth exclusively, without considering data characteristics of DLRM itself, resulting in limited performance improvement at expensive hardware overhead. For example, the bank-level NMP TRiM-B [52] achieves only 1.31× speedup compared to bank-group level NMP TRiM-G but induces 4× more area overhead. This paper explores the potential of fine-grained memory parallelism and the practical NMP performance roofline to accelerate embedding operations.

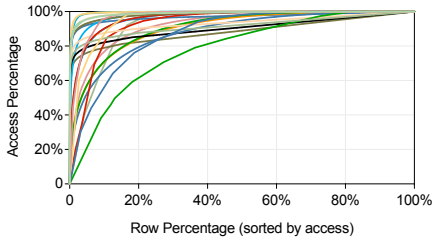

Figure 3: Accumulated access frequency of embedding tables for Kaggle [2] (each line denotes an embedding table)

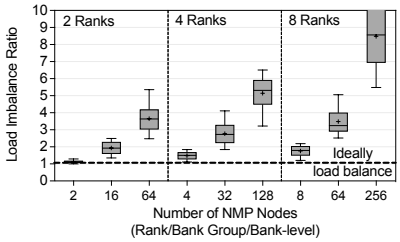

Figure 4: Load imbalance ratios of different NMP levels for 2-, 4-, and 8-rank configuration

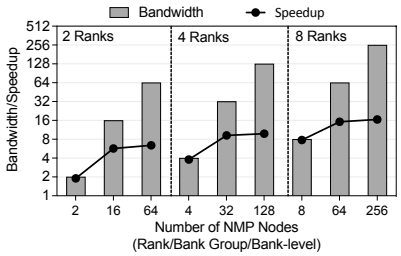

Figure 5: Normalized speedups and internal bandwidth of different NMP levels for 2-, 4-, and 8-rank configuration

3 MOTIVATION ANALYSIS

3.1 Skewed Access *vs.* Symmetrical Architecture

Observation 1: *The vast majority of accesses to an embedding table are centered on a few rows, while the symmetrical NMP architecture in existing solutions treats all data equally, resulting in load imbalance and resource underutilization.*

Previous works assume random access to an embedding table [18, 38]. In practice, however, the access distribution is highly skewed [30, 42, 57]. For example, the minority popular videos are likely to be visited by most of the users, while the majority of the others are rarely accessed. That is, most of the accesses to an embedding table are on a small fraction of its embedding rows. This so-called *long-tail phenomenon* is widespread throughout the internet industry [6, 20, 51]. Figure 3 shows the percentage of cumulative accesses to each embedding table on a real-world dataset [2]. As we can see, a small percentage of data (< 20%) can take up most of the accesses while most of the others have a few, showing a *high temporal locality*. Unfortunately, these few frequently accessed rows are randomly distributed in the arbitrarily large embedding tables, exhibiting a relatively *low spatial locality*.

However, existing NMP solutions adopt a symmetrical architecture, in which all the data are handled at the same NMP level. As the embedding tables in the baselines are allocated contiguously in the memory and a row index also serves as the memory offset. Given that the highly accessed embedding rows are randomly distributed across the memory nodes (e.g., banks), their different access frequencies will directly lead to different access times to the memory addresses, causing imbalanced accesses. In this context, the performance in the NMP architecture is bounded by the slowest memory node with the most lookups. Figure 4 illustrates the distribution of load imbalance ratios, which are computed by dividing the largest number of lookups among the memory nodes in one operation by the number of lookups in an ideally balanced load that is evenly distributed to all memory nodes. As the number of memory nodes increases with increasingly finer NMP granularity, the average number of lookups per memory node decreases, but the load imbalance worsens, limiting performance improvements.

To address the load imbalance issue, RecNMP [42] uses a specialized cache in the PE to avoid memory access for the hot entries cached. However, the cache size is limited, and even just 0.1% of hot entries for a large-scale model can be too large to cache. TRiM [52] proposes hot-entry replication to distribute accesses to hot entries across multiple nodes. However, its effectiveness is closely related to the number of replications and the percentage of such replications over all the data, which vary for different datasets. Besides, scheduling to access different replications also brings extra control overhead. However, all existing solutions assume the underlying memory nodes with the same parallelism, but it is inconsistent with the skewed nature of embedding data access. This is the root cause of the load imbalance arising in recommendation systems and motivates the development of ReCross, which uses different NMP nodes to offer cross-level memory parallelism to accommodate different data requirements, and consequently, addresses the load imbalance issue adequately.

3.2 Successive Access *vs.* Serial Operation

Observation 2: *The high data temporal locality leads to successive accesses to the same bank. However, bank operations are inherently serial in modern memory architectures, limiting the performance of exploiting fine-grained NMP.*

In a hierarchical memory architecture, the internal bandwidth is more intensive in the lower NMP level [52]. As shown in Figure 5, the finer the granularity of NMP is, the more PEs are integrated. Thus, the memory parallelism, and consequently, the internal bandwidth are increased. However, we observe that the performance gains of DLRM models do not increase significantly. Specifically, the rank-level NMP speedup is nearly linear scaling. As the finer-grained NMP is applied, the practical performance growth (normalized to the theoretical performance) decreases gradually. In particular, the bank-level NMP, with a 4× higher internal bandwidth, achieves almost no benefits over the bank-group level NMP. The load imbalance may become worse for the bank-level NMP since it has more memory nodes, i.e., banks. Also, since the bank-level NMP performs serial operations for the accesses to the same bank, the most imbalanced memory node, i.e., the bank with the most accesses, slows down the performance significantly.

Figure 6 shows the execution timeline of four successive accesses to two banks at different memory parallelism levels. The bank-group level NMP in Figure 6(a) allows different bank-groups to run in parallel, but the two banks in the same bank-group have to be operated alternately due to the occupation of the DQ bus from the banks to the bank-group PE. As shown in Figure 6(b), the bank-level NMP allows the banks in the same bank-group to operate in parallel since the embedding vectors are accumulated in the PE dedicated to each bank. However, due to the serial semantics upon the same bank,

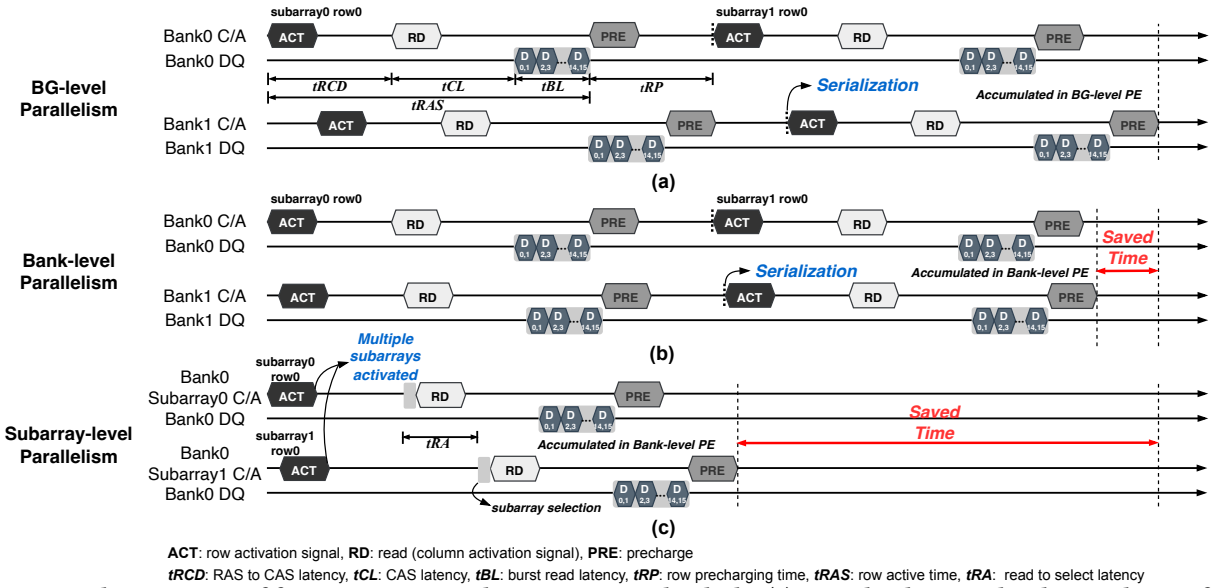

Figure 6: The execution of four successive read accesses to two banks by (a) using bank-group level NMP that performs serial bank operations and reduces embedding vectors using the bank-group level PEs, (b) proceeding exactly as in (a) except that bank-level NMP and PEs are used, and (c) using subarray-parallel bank-level NMP that allows simultaneous accesses to different subarrays of the same bank and reduces embedding vectors using the bank-level PEs

the performance benefits offered by bank-level memory parallelism are not reaped fully, yielding a slight performance improvement over the bank-group level. Thereby, the successive bank access caused by the skewed access frequency distribution and the load imbalance architecture become the major bottleneck of the bank-level NMP, hindering performance improvement.

In addition, existing NMP solutions that solely explore higher internal bandwidth from massive memory parallelism will incur significant hardware overhead. For example, the bank-level NMP will induce 4× more PEs for each DRAM chip against the bank-group level NMP, resulting in significant area overhead. Since the internal area of DRAM is rather limited for high-density storage integration, it is worth considering the area efficiency of the NMP architecture, which has been inadequately considered.

3.3 Cross-level Asymmetrical NMP Architecture

Previous efforts to explore performance gains solely from finer-grained NMP levels have significant limitations, we are inspired to develop an efficient yet practical NMP solution to accelerate embedding operations by considering both embedding data and NMP architecture characteristics. Different from the symmetrical NMP architectures, the philosophy of our architecture is to provide memory parallelism for embedding data with corresponding bandwidth requirements, e.g., access frequency. For example, the most frequently accessed data that require massive bandwidth will be handled by a low NMP level (such as bank-level), while the majority of the tail data that are hardly accessed are stored at a high NMP level (such as rank-level). However, as discussed previously, existing bank-level NMP solutions fail to satisfy the bandwidth requirements of the frequently accessed data, yielding little performance gains.

Is the bank-level parallelism the roofline we can achieve? The answer is negative. As described in §2.2, a modern DRAM bank consists of multiple subarrays, and only one subarray is activated simultaneously (i.e., the serial bank access), leaving much space to exploit a higher memory parallelism [35]. Figure 6(c) illustrates exploiting subarray parallelism in bank-level NMP. Compared to the bank-serial operation that requires *tRC* (i.e., row cycle time, the sum of *tRAS* and *tRP*) for each access, the subarray level parallelism overlaps the latency of *tRCD* and *tRP* when different subarrays are accessed in the same bank, reducing the access latency significantly. Besides, the number of subarrays in one bank can often exceed 128 in the modern DRAM [35], showing great potential for massive internal bandwidth.

In ReCross, we exploit the subarray level parallelism in bank-level NMP to improve the performance of vast successive bank accesses to the frequently-accessed data. Considering the skewed embedding accesses, we further present an asymmetrical NMP architecture, which offers different NMP levels of rank, bank-group, bank with respect to rank, bank-group, and subarray level parallelism to accommodate various bandwidth requirements. The rationale behind exploiting all three NMP levels is twofold. First, the variability of the embedding tables leads to varying access distributions. Applying only two NMP levels and partitioning the data into only two parts may not adapt well to all data and is not flexible enough to meet different requirements, while applying three NMP levels makes it more likely to accommodate a wide variety of situations. Besides, in the tree-like structure of the DRAM datapath, the accessed data must span these three memory levels (i.e., bank, bank-group, and rank) to reach the memory controller. In this case, exploiting three NMP levels also minimizes the amount of data transferred as they are reduced promptly. In this way, ReCross provides sufficient resources at different NMP levels to different data with a minimal number of PEs in an area-efficient manner, improving load balance and resource utilization.

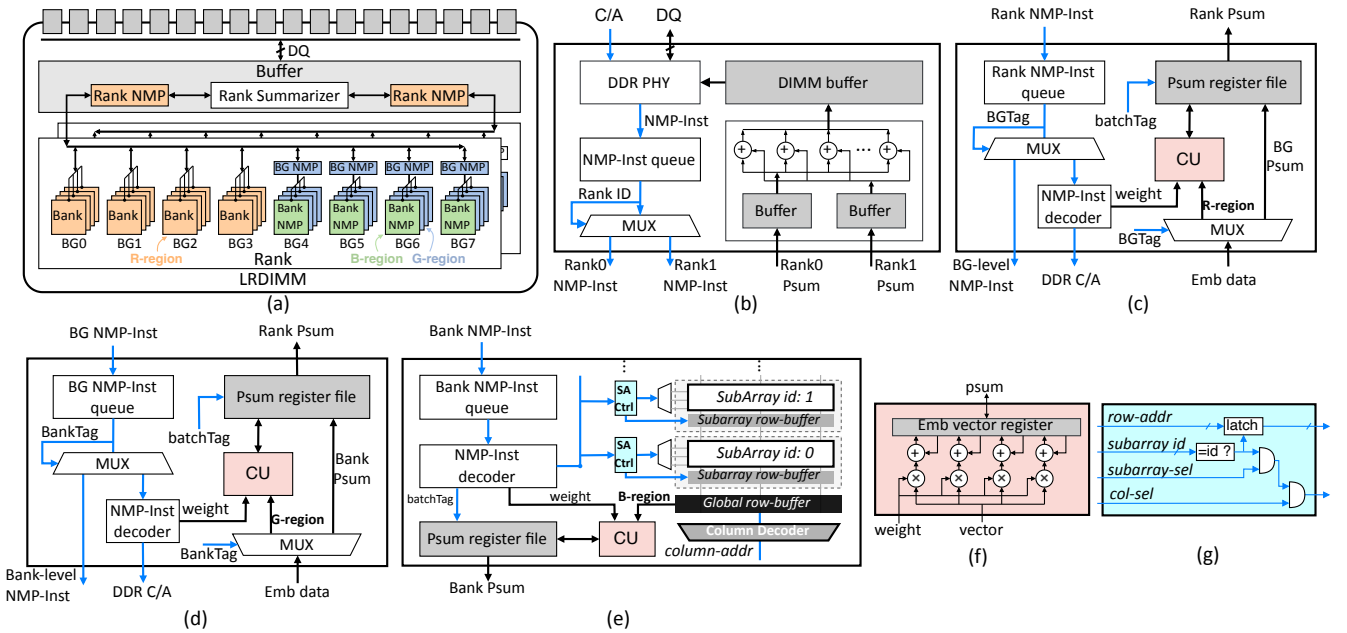

Figure 7: (a) Architecture overview of ReCross, (b) Rank summarizer, (c) Rank-level NMP, (d) Bank-group level NMP, (e) Subarray-parallel bank-level NMP, (f) Computation unit, and (g) Subarray access controller

4 RECROSS

In this section, we describe ReCross, which leverages multiple memory level parallelism in a unified DIMM-based NMP architecture with hardware-software co-designs.

4.1 Architecture

The overall architecture of ReCross is shown in Figure 7(a). Compared with the commercial Load-Reduced DIMMs, we modified it to support cross-level NMP and integrated three types of NMP, i.e., rank, bank group, and bank-level PE. Each rank is equipped with one rank-level PE, ensuring minimum rank-level memory parallelism. Partial bank groups in each rank (e.g., 4 in each as Figure 7(a) illustrates) are combined with one PE to support bank group level NMP. Partial banks inside the NMP-featured bank group (e.g., 1 in each as Figure 7(a) illustrates) are modified to support subarray-parallel bank-level NMP. The memory space is split into three regions corresponding to three levels of NMP as well as memory parallelism, i.e., R-region refers to the rank-level NMP, G-region refers to the bank group-level NMP, and B-region refers to the bank-level NMP. The ratio of these three regions is configured as 4:3:1 by default as Figure 7(a) illustrated, and §5.4 explores more configurations.

For a DIMM containing multiple ranks, a rank summarizer is employed to summarize their partial results (`Psum`), reducing data transfers further. Figure 7(b) shows the detailed logic of the rank summarizer, which dispatches NMP instructions to ranks and accumulates the reduced `Psums` from different ranks. Of course, the modified DIMM can also serve as a regular main memory device since all the conventional interfaces are reserved.

Rank-Level NMP. Figure 7(c) shows the design of the rank-level NMP PE, which contains two key functions: NMP instruction decoding and embedding vector reduction. Since ReCross contains three levels of NMPs, it is necessary to identify an NMP instruction so that it can be processed at the correct NMP level. We use two 1-bit tags, i.e., `BGTag` and `bankTag`, to accomplish it. `BGTag` indicates whether this embedding vector is located in the R-region. If so, it will be decoded and processed inside the rank-level NMP. Otherwise, this instruction is expected to be processed at a lower-level NMP. Consequently, the rank-level NMP can receive data from either a lower-level NMP or accessing the R-region. The former can be directly accumulated since they have already been reduced, while the latter needs to be reduced first.

Note that ReCross can be modified to support various reduction operations, e.g., summation, weighted summation, and quantized operation. In this work, we use the weighted summation as in the previous work [42, 52]. Figure 7(f) shows the detailed logic of the computation unit, which performs the general cumulative multiplication operations and can be widely applied to large applications, e.g., matrix vector multiplication.

Bank-Group Level NMP. Figure 7(d) shows the architecture of bank-group level NMP. Similar to the rank-level PE, the bank-group level PE also contains two key functions: NMP instruction decoding and embedding vector reduction. For each NMP instruction, the bank-group level PE needs to distinguish if it belongs to itself or a bank-level NMP, which is indicated by the `bankTag`. If the instruction belongs to itself, it will be decoded and accessed directly, or it will be sent to the bank-level PE. The reduction processing is similar to the rank-level. The data from bank-level NMP can be directly accumulated, while the other needs to be reduced first.

Bank-Level NMP with Subarray Parallelism. The conventional DRAM sends the data to the global row buffer immediately once a word-line is activated, which monopolizes the global bit-lines and limits activation to one subarray at a time. To overcome this, inspired by [35], we decouple the subarray activation and data transmission to allow multiple subarrays to be activated simultaneously by temporarily storing the data in the local row buffer.

As Figure 7(e) shows, the *subarray access controller* (shown in Figure 7(f)) is designed to control the data transmission from the local to the global row buffer. A specific row is connected to the local row buffer when it is activated by the row address (*row-addr*) and *subarray id* and is connected to the global row buffer only when it is designated by the subarray selection *subarray-sel* signal. In this way, multiple subarrays in a bank can be activated simultaneously, and only the designated one is connected to the global bit-lines at a time, ensuring safe and correct operation.

After being read out from the global row buffer, the vectors will be reduced directly. Note that all the added signals are generated by the NMP-inst decoder of a bank-level PE without impacting the other banks. Thus, the additional hardware overhead incurred is lightweight, as discussed in §5.5. The simultaneous activation of multiple subarrays also introduces a new challenge requiring the subarray selection signal to stay unchanged once one subarray is activated. Otherwise, another activated one may receive the signal to transfer incorrect data. To address this, a new timing constraint is introduced to control the waiting time before a new subarray selection signal is sent, i.e., *tRA* in Figure 6. Moreover, the constraints such as *tFAW* and *tRRD* that limit one activated row per bank are still obeying [35].

By interleaving access to the subarrays, memory parallelism is scaled with the number of subarrays. However, successively accessing a subarray happens in two cases: (1) accessing the same row but on a different column address, and (2) accessing different rows. For case (1), since the target row is already in the local row buffer, the data can be accessed directly in a short latency with a row buffer hit. For case (2), serial operations can incur as in the case of conventional DRAM. To optimize the row buffer hits and avoid the serial operations, we present a locality-aware scheduling policy based on the widely used scheduling FR-FCFS [56]. We have modified it to be subarray-aware by ensuring that the requests to the same local row buffer are prioritized over those in different subarrays and then over those in the same subarray but with different rows.

4.2 NMP Instruction and C/A Bandwidth

Like RecNMP [42], ReCross leverages the instruction compression technique by encoding all the signals, tags, and other information into an 82-bit instruction. We now explain each of its parts. The 3-bit `opcode` field indicates a reduction operation, e.g., summation, weighted sum, or any other quantized operation. The 3-bit `DDR cmd` field represents the DDR commands of ACT, RD, and PRE. The 34-bit `addr` field represents the physical address of the target embedding vector. The 3-bit `vsize` field indicates the vector size and also denotes the number of DRAM reads per embedding vector. The 32-bit `weight` field denotes the floating-point weight used for the weighted summation. The tags are set at runtime by the host-side memory controller. The `batchTag` identifies if the vectors belong to the same embedding operation, and the `batchTags` of the instructions in the same embedding operation are set identically. The `lastTag` is used to indicate the end of a batch, and only the last instruction in each batch is set to 1, meaning that the reduced results can be transferred to the host. The `BGTag` and `bankTag` co-determine the NMP level and are generated according to the `addr`. `BGTag` is set to 1(0) when the bank group field of the physical address (mis)matches with an NMP-featured bank group. `bankTag` is set to 1 iff `BGTag` is valid and the bank field of the physical address matches an NMP-featured bank.

The PEs may stall with a low utilization if the C/A signals are not supplied promptly. Ideally, an NMP instruction to be processed must be transferred before its preceding instruction is completed. Consequently, the instruction transfer time is equal to the access time of the embedding vector, which is proportional to the vector length. However, the C/A bandwidth of DDR5 is only 14 bits per cycle [37], which is sufficient for only five memory nodes for a vector length of 64 (256 bytes). To cope with this, as in TRiM [52], we have applied a *two-stage instruction transfer technique* to use both DQ and C/A pins to transfer NMP instructions from the memory controller to the DRAM buffer chip. Since embedding vectors are stored in PEs without occupying the DQ bus until the end of this batch, during the embedding operation, the DQ bus is idle and can be used to transfer NMP instructions. In this way, the total 94 pins (14 C/A pins, 80 D/Q pins) can be used to transfer the instructions. Besides, considering the DRAM time constraints (e.g., t_{RA} and $t_{CCD_{S/L}}$), the provided C/A bandwidth is sufficient for the bank-level NMP with all the 32 banks equipped with PEs [52]. Therefore, we set up the C/A bandwidth to be sufficient for the default ReCross configuration in the evaluation section.

Table 1: Descriptions of the Used Parameters

Parameter	Description
I	EmbTable Set
$vsize_i$	vector size of EmbTable i
num_i	Number of Rows of EmbTable i
f_i	Access Distribution Function of EmbTable i
$pool_i$	Average Pooling of EmbTable i
$prob_i$	Probability of Accessing EmbTable i
$batch$	Average Batch Size
J	Memory Region Set
cap_j	Capacity of Memory Region j
bw_j	Internal Bandwidth of Memory Region j

4.3 Embedding Data Partition and Placement

ReCross has divided the memory space into three regions with different NMP levels and internal bandwidth capabilities. We explain how the embedding tables are allocated to these memory regions with appropriate memory parallelism for better performance. We first profile the embedding tables using statistical analysis and quantify the bandwidth requirements of the embedding data. Next, we formulate data partitioning as an optimization problem that can be solved quickly via a linear programming solver. After a partitioning decision is produced, the embedding tables are divided and mapped to the ReCross architecture for processing.

Data Characterization. For each embedding table, two types of data are considered. The first is its specification information that can be accessed directly, including its embedding row count, vector length, and data type size. The other is the access information captured during the training phase, including its access probability, the access distribution, and the average number of accessed embedding rows each time. The hardware configuration, e.g., the memory capacity and bandwidth at each memory region, is also considered. Table 1 lists the parameters used.

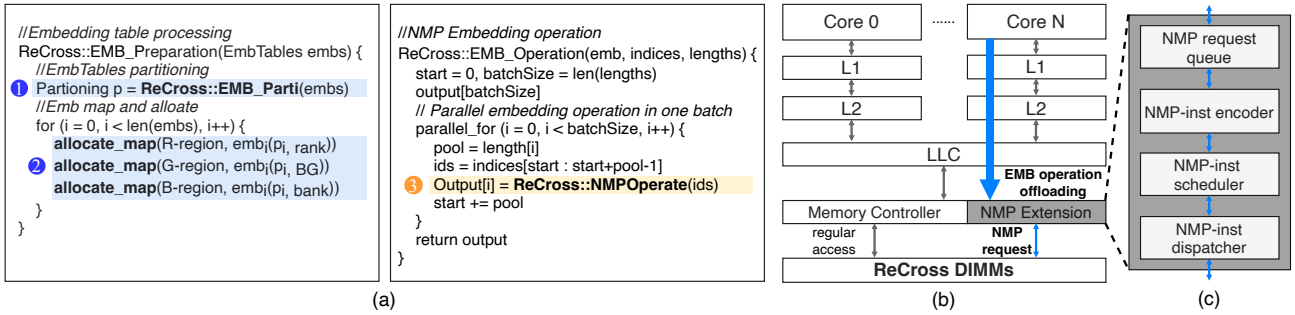

Figure 8: (a) Programming framework of ReCross, (b) Host-ReCross interaction, and (c) The NMP extension

Based on such statistical information, we estimate the memory bandwidth requirement of the embedding data. Let p_{ij} be the percentage of embedding vectors in the i-th embedding table in the j-th memory region. For the embedding table i at one operation, we can estimate the volume of data accessed from region j by $d_{ij} := f_i(p_{ij}) \times pool_i \times vsize_i$, and for a batch of operations, the total volume of data accessed from region j is $D_j := \sum_j d_{ij} \times prob_i \times batch$. Thereby, the access latency for each memory region can be estimated by $t_j = \frac{D_j}{bw_j}$. Since the three NMP levels are processed in parallel, the three memory regions can be accessed simultaneously. Thus, the total latency for one batch of embedding operations t is determined by the slowest NMP level, that is, $t := max(t_j)$. We use t to estimate the performance of the partitioning scheme p_{ij}.

Data Partitioning. Our goal is to make a partitioning decision for each embedding table with minimized overhead, i.e., the embedding operation latency under the resource constraints of memory capacity at each memory region, which can be considered as a constrained optimization problem:

$$0 \le p_{ij} \le 1 \ \forall i \in I, \ \forall j \in J \tag{1}$$

$$\sum_j p_{ij} = 1 \ \forall i \in I \tag{2}$$

Equ. (1) defines the domain of a partitioning decision and Equ. (2) ensures all embedding tables are allocated to memory.

When dividing the embedding tables among different memory regions, we must ensure that the memory capacity of each level is considered. Thus, we have the following constraints:

$$\sum_i p_{ij} \times num_i \times vsize_i \le cap_j \ \forall j \in J \tag{3}$$

where $num_i \times len_i \times dbytes_i$ is the total memory footprint occupied by the i-th embedding table and p_{ij} indicates the amount of data in the i-th embedding table assigned to the j-th memory region. Equ. (3) ensures that the accumulated number of the partitions from the embedding tables assigned to each memory region does not exceed its memory capacity. Thus, the partitioning problem can be formalized as:

$$\textbf{Minimize } t \quad \textbf{Subject to } \text{Equ. (1), Equ. (2), and Equ. (3)} \tag{4}$$

which is a *linear programming (LP)* [60] that can be solved efficiently using the LP solvers such as Gurobi [3]. The preprocessing overhead is discussed in detail in § 5.6.

Data Placement. After a partitioning decision is generated, the placement of each embedding table is determined. The embedding tables are usually stored continuously in the memory, and an embedding row can be accessed by the base address and its index, which is served as the memory offset. However, the embedding rows of the partitions are selected based on their access frequency, and they are randomly distributed in the embedding table. To end this, we create a mapping table for each embedding table that maps the index of each embedding row to its physical address, which allows the host to access an embedding table via the indexes in the standard manner, regardless of the underlying storage locations.

4.4 Execution Flow

Programming Model. Like the previous works [8, 12, 47], ReCross divides the application into host calls running on the CPU (i.e., the recommendation system) and NMP kernels (i.e., the embedding operation) being offloaded to NMP PEs. Figure 8(a) describes the programming framework in ReCross. Before performing an embedding operation, embedding tables will be first partitioned and allocated to different levels of NMP memory. For each embedding table, a partitioning decision (❶) is generated. Then, an allocation and remapping phase is performed to allocate the embedding tables into the corresponding memory regions and map the index of the embedding vector with a physical address (❷), allowing the host to access an embedding table by indexing as usual. When an embedding operation starts, the indices of each embedding operation are sent to the NMP kernel (❸), which offloads the embedding operation task to the NMP system. Specifically, the NMP kernel is responsible for address mapping and NMP request generation. Once the results are returned, the embedding operation is finished.

Host-ReCross Interaction. Figure 8(b) illustrates the execution flow from the perspectives of the host and the memory controller. The NMP embedding operation launched by the host will be first offloaded to the memory, in which an NMP extension is tailored to handle them. Figure 8(c) depicts the NMP extension, in which a queue is used to store the requests, each of which is encoded into one NMP instruction by the encoder. Then, the scheduler reorders the instructions according to the modified locality-aware scheduling policy. Afterward, the dispatcher transfers the instructions to the corresponding NMP PEs inside the ReCross DIMMs. Finally, the reduced results are returned to the host via the DRAM interface once the offloaded embedding operation is complete.

4.5 Discussion

Embedding Table Updates. The embedding partitioning decisions are based on the prior accessing distribution of the training data. While in some cases, the embedding tables are updated in real-time, i.e., in the online training system [48], the embedding tables are inserted (new data come in) and modified (old data change)

continuously. For these updated embedding data, we treat them as cold data and store them in the capacity-optimized memory region (i.e., rank-level) as they are never accessed before.

Access Frequency Changes. The embedding data access frequency could vary over time (e.g., a rarely accessed row could be popular, and vice versa), resulting in the existing embedding placement being non-optimal. To handle this, we can apply *dynamic embedding scheduling* in the three NMP levels according to the real-time data access frequency. Specifically, we can count the accessing frequency of embedding rows over a fixed time interval (e.g., 5 minutes), and for these highly-accessed items (e.g., top 1000) in the lower parallelism NMP level, allocate them to the high-parallel bank-level NMP, and vice versa. By applying this, ReCross also enables dynamic scheduling of the embedding data according to the real-time accessing frequency. We leave the potential exploration of alternative designs that address this for future work.

5 EXPERIMENTAL EVALUATION

In this section, we evaluate efficiency and effectiveness of ReCross against the state-of-the-art DLRM NMP solutions.

5.1 Experimental Setup

Methodology. We implement ReCross on a cycle-accurate simulator Ramulator [36] to evaluate its performance and effectiveness. Ramulator is modified to emulate the designs of ReCross, including the instruction encoder and decoder, and the arithmetic logic of PEs. Table 2 summarizes the configuration and parameters of ReCross. The subarray parallelism [35] is modeled by adjusting parts of the DRAM timing constraints to estimate the performance of multiple accesses to the same bank simultaneously. We synthesize the PEs using 40nm CMOS technology at 300MHz clock frequency (the same as DDR5-4800 internal frequency), and the area and power of the PEs are derived from Synopsys Design Compiler. For the other compared works, we synthesized their PE designs faithfully according to their configurations. The DRAM energy of DDR5 is obtained from the MICRON DDR4 power calculator [46]. The off-chip I/O data are obtained from CACTI-IO [29].

Datasets and Benchmarks. We evaluate ReCross on the representative *Deep Learning Recommendation Model* (DLRM) from Facebook [50]. Since the production-scale datasets used in RecNMP [42] are not available, we use the open-source real-world Criteo Ad datasets [1, 2], which have similar data characteristics as the production-scale datasets. The default number of embedding vectors per embedding operation is 80. The production-scale embedding operations are shown to have the batch sizes between 1 and 100 [53]. We use 32 by default and scale it from 1 to 128 in a sensitive study. We evaluated the performance of ReCross on varying embedding vector lengths, ranging from 16 to 256, to analyze its effectiveness across different scenarios.

Baselines. We evaluate ReCross against a CPU baseline and four state-of-the-art NMP solutions, TensorDIMM [38], RecNMP [42], TRiM-G [52], and TRiM-B [52]. The CPU baseline performs all embedding operations on a 16-core processor (as shown in Table 2). TensorDIMM and RecNMP leverage the rank-level NMP. TensorDIMM adopts a vertical partitioning to embedding tables, while RecNMP splits embedding tables horizontally. The cache size of RecNMP is 1 MB for each rank-level PE, and the hot-entry replication proportion of TRiM is 0.05%, as declared in their papers. To make apple-to-apple comparisons, all the baselines have the same hardware configuration in Table 2.

Table 2: System Configuration

CPU	16 cores, x86-64 ISA, 2.2GHz, Broadwell, 32MB LLC
Memory Controller	64-entry RD/WR request queue, FR-FCFS scheduler
DRAM Organization	DDR5-4800 ×8 device, 1 DIMM per channel, 2 Ranks per DIMM, 8 bank-groups per Rank, 4 Banks per bank-group, 256 subarrays per Bank
Timing Parameters	$tRCD$=40, tCL=40, tRP=40, $tRAS$=76, tRC=116, tBL=8, $tCCD_S$=8, $tCCD_L$=12, $tFAW$=32
Energy and Latency Parameters	DRAM ACT energy=2 nJ, DRAM RD/WR=4.2 pJ/bit, off-chip I/O=4 pJ/bit, FP32 adder=0.9pJ/Op, FP32 mult=2.4pJ/Op

5.2 Overall Performance

We evaluate the performance in the execution time of the embedding operation of ReCross against the baselines with various vector lengths, batch sizes, and the number of ranks.

Figure 9 shows the speedups of ReCross against the state-of-the-art solutions with the vector lengths ranging from 32 to 256 (under a batch size of 32). Compared with the CPU baseline, TensorDIMM, RecNMP, TRiM-G, and TRiM-B, ReCross achieves the speedups of 15.5×, 9.3×, 7.9×, 2.5×, and 1.8×, respectively. The superiority of ReCross lies in the fact that prior NMP solutions rely primarily on the number of memory nodes (NMP PEs) to improve memory parallelism and internal bandwidth, while ReCross provides different levels of memory parallelism for different data types, exploiting the bandwidth more fully. Especially for the frequently accessed data, which heavily bottlenecks the performance, we expose the subarray level parallelism that reduces the latency significantly. Note that the speedups are low when the vector length is 16 or 32 against the long vector cases. This is because the short vector incurs a high C/A bandwidth requirement. Thus, the C/A signals are unsaturated, limiting performance gains. Despite this, ReCross still yields considerable benefits over the existing NMP solutions.

Figure 10 depicts the speedups of ReCross against the state-of-the-art solutions with different batch sizes ranging from 1 to 128 (with a vector length of 64). In production cases, the batch size is selected according to different optimization goals, i.e., a small batch size induces lower latency while a large batch size helps improve throughput [18]. We have evaluated ReCross with different batch sizes to demonstrate its superiority for all batch-size cases. We can also notice that as the batch size increases, more embedding operation requests are sent to the memory, better utilizing the memory parallelism and improving the performance slightly.

Figure 11 shows the speedups of ReCross against the state-of-the-art solutions with different rank counts. ReCross is superior to the existing solutions in all scenarios. The main reason lies in that ReCross sufficiently leverages the finer-grained subarray parallelism in the bank-level NMP, providing higher internal bandwidth within each rank. ReCross is designed inside the rank and ensures well scalability for the number of ranks, which is more pronounced for the increasing embedding data size in the forthcoming scenarios.

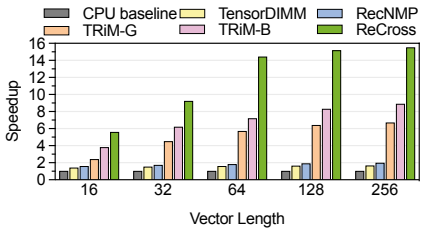

Figure 9: Normalized speedups with different vector lengths

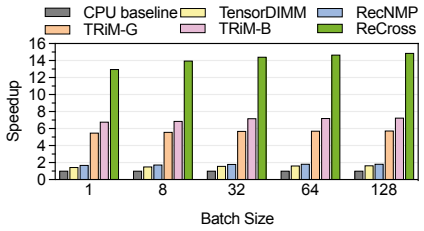

Figure 10: Normalized speedups with different batch sizes

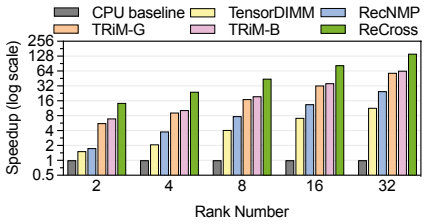

Figure 11: Normalized speedups with different rank counts

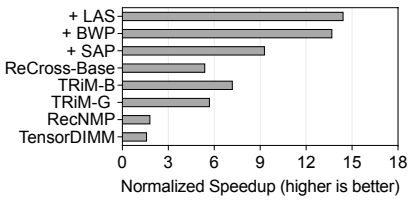

Figure 12: Performance breakdown of ReCross's designs

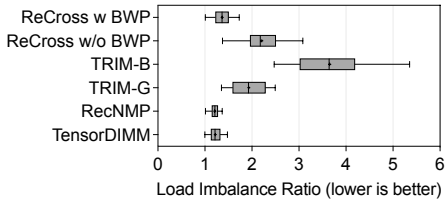

Figure 13: The load imbalance ratio comparison of ReCross against the baselines

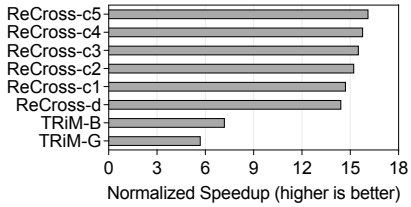

Figure 14: Normalized speedups of ReCross under different configurations

5.3 Effectiveness

We evaluate the effectiveness of ReCross in terms of three optimizations: *subarray-level parallelism* (SAP) in the bank-level NMP, *bandwidth-aware embedding partitioning* (BWP), and *locality-aware scheduling* (LAS). Figure 12 shows the results of gradually applying these optimizations to a ReCross-Base design, which excludes SAP, BWP, and LAS. ReCross-Base outperforms the CPU baseline by 5.4×, roughly the same as the TRiM-G. However, subarray-level parallelism improves performance significantly to 9.3× when using a simple embedding table partitioning. Despite this improvement, the crude partitioning method fails to consider the fine-grained data characteristics, leading to a high load-imbalance ratio. As shown in Figure 13, ReCross without BWP has an even higher load-imbalance ratio than TRiM-G. However, BWP generates a globally optimal partitioning strategy that balances the bandwidth requirements, reducing the load-imbalance ratio and boosting performance to 13.7×. Finally, applying LAS further improves the performance to 14.4× due to the increased interleaved accesses.

5.4 Configuration Exploration

The default configuration of ReCross (denoted as ReCross-d) comprises 1/4/4 rank-/bank-group/bank-level PEs in one rank. The ratio of the capacity of the R-/G-/B-region (R:G:B) is 16:12:4. To explore the performance roofline of ReCross, we evaluate its performance with five other configurations: (1) ReCross-c1: 1/4/8 rank-/bank-group/bank-level PEs with R:G:B of 16:8:8, (2) ReCross-c2: 1/4/16 rank-/bank-group/bank-level PEs with R:G:B of 16:0:16, (3) ReCross-c3: 1/8/8 rank-/bank-group/bank-level PEs with R:G:B of 0:24:8, (4) ReCross-c4: 1/8/16 rank-/bank-group/bank-level PEs with R:G:B of 0:16:16, (5) ReCross-c5: 1/8/32 rank-/bank-group/bank-level PEs with R:G:B of 0:0:32.

Figure 14 shows the results. As we can see, there are no significant performance differences after adding more bank-group- and bank-level PEs. In particular, even if all banks are applied with the bank-level NMP, the performance of ReCross-c5 does not improve much. The main reason is the skewed distribution of the embedding data access frequency that the majority of data are in a long tail that is seldom accessed. Thus continuing to increase bank-level NMP affects only the tail data, impacting performance negligibly. However, with the addition of more PEs, the additional area rapidly increases and outweighs the performance gains, significantly reducing area efficiency. Overall, ReCross-d captures a nice sweet point to achieve the highest area efficiency by considering both embedding characteristics and NMP capability.

5.5 Energy/Area Consumption

Energy. Figure 15 illustrates the normalized energy savings achieved by ReCross compared to state-of-the-art solutions. ReCross reduces energy consumption by 58.5%, 57.2%, 51.9%, 28.5%, and 23.7% compared to the CPU baseline, TensorDIMM, RecNMP, TRiM-G, and TRiM-B, respectively. The reduction in energy consumption can be attributed to ReCross's reduced on-chip read access latency resulting from its row-interleaved subarray-level parallelism, as well as the reduction of off-chip I/O energy consumption due to the movement of most reduction operations inside the bank. Shorter execution time also contributes to the reduction of static energy. Furthermore, the energy overhead caused by additional PEs and support for subarray parallelism only accounts for 3.5% and 0.28%, respectively. Note that due to more rows being activated simultaneously, the DRAM power of ReCross will increases. However, since only a small portion of banks (i.e., one-eighth) are applied the subarray parallelism, the power increase is slightly and well within the thermal density limits.

Area. Table 3 shows the area overhead introduced by extra NMP PEs. TensorDIMM and RecNMP only apply the rank-level NMP PEs in the DIMM buffer. TRiM and ReCross exploit the bank-group- and bank-level NMP that integrate PEs in the DRAM chip. The computing logic of bank-group- and bank-level PEs involves only multipliers and adders, which can be implemented using the DRAM process [41]. According to the prior study [67, 69], the area

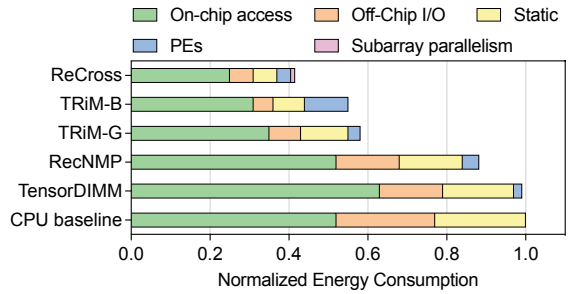

Figure 15: Energy consumption breakdown

of compute-logic fabricated in the DRAM process can be up to 2× larger than that in the CMOS process, thus we have selected an older generation of the CMOS process at 40nm to make conservative estimates of the PE area overhead. In general, the extra area of ReCross is 0.34 mm^2 per DIMM and 2.35 mm^2 per DRAM chip, which consumes a small fraction considering the buffer chip and DRAM chip can be more than 100 mm^2 [45] and 75 mm^2 [34], respectively. ReCross yields a similar area overhead as TRiM-G as they have the same number of NMP PEs, but this is smaller than what is required by TRiM-B that is equipped with more PEs.

5.6 Partitioning Overheads

Partitioning Overhead. The embedding partitioning problem is *linear programming* (LP), as described in § 4.3, and it can be solved by polynomial time complexity that scales with the number of embedding tables [32]. For the evaluated dataset with 26 sparse features, the partitioning decision can be generated within 5s using the LP solver Gurobi [3], and for the largest production-scale models with hundreds of embedding tables [43], the LP problem can also be solved in tens of seconds. Note that the partitioning only needs to be processed only once after the model is offline trained, the partitioning overhead is therefore negligible relative to expensive model training time (hours to days).

Mapping Table Overhead. Besides, the mapping table requires 34 bits per embedding row. Note that the embedding dimension in production cases is usually between 32 to 256 with at least a 128-byte vector [66]. The mapping table size often takes only a small fraction (less than 4%) of the model size. As for the latency of accessing the mapping table, it is just like when the TLB cache is missing in the physical address translation process, which is around 10ns and only takes up a small fraction of the whole process (which can be over 100ns).

6 RELATED WORK

Recommendation Acceleration. With the wide adoption of personalized recommendation systems in the industry, a lot of research has been dedicated to optimizing and accelerating the recommendation models at production scales [4, 11, 16–18, 23, 31, 40, 43, 48, 57, 61, 65, 70]. Gupta *et al.* [18] analyze the memory bounded, low computation intensity, and irregular memory access of DLRM [50], and propose a set of optimizations, such as batching techniques and co-location of inference, to improve the performance of at-scale recommendation. ReShard [57] analyzes data characteristics of the recommendation model and proposes a sharding technique for efficient recommendation training. In addition, there are also

Table 3: Extra Area Overhead Breakdown

	Rank PE (per Buffer Chip)	BG/Bank PE (per DRAM chip)
TensorDIMM	0.28 mm^2	None
RecNMP	0.54 mm^2	None
TRiM-G	0.36 mm^2	2.03 mm^2
TRiM-B	0.36 mm^2	11.5 mm^2
ReCross	0.34 mm^2	2.35 mm^2

research efforts on accelerating the embedding layer [13, 59, 68]. Shi *et al.* [59] propose to reduce the size of embedding data by applying complementary partitions. Ginart *et al.* [13] explore the mixed dimension embeddings to reduce the vector dimension.

Near Data Processing for Embedding Operation. Considering the memory-bounded characteristics of the embedding layer, many research attempts [5, 30, 38, 39, 42, 44] leverage DRAM-based *near memory processing* (NMP) to accelerate embedding operations by integrating logic in or near the memory. FAFNIR [5] uses a rank reduction tree in which all partial results from the rank-level NMP units are reduced, minimizing the data movement from the memory to the host. However, FAFNIR still utilizes rank-level parallelism as in TensorDIMM [38] and RecNMP [42], improving little the internal bandwidth. SPACE [30] relies on a heterogeneous NMP architecture that places the most frequently accessed data inside the bandwidth-optimized 3D-stacked DRAM [25, 54] and the others inside the capacity-optimized DIMMs. However, the cost of the 3D-stacked DRAM is an order of magnitude higher than that of the commercial DIMM. Its capacity is limited to support future-looking scenarios with extremely-large data scales. To the best of our knowledge, ReCross is the first NMP solution that enables providing different NMP levels to accelerate the skewed embedding data on a unified DIMM-based commodity memory.

7 CONCLUSION

This paper proposes an efficient yet practical fine-grained NMP solution with hardware-software co-design for accelerating embedding operations in deep-learning-based personalized recommendation systems. We architect ReCross, a cross-level NMP architecture that provides the rank, bank-group, and bank-level NMP for exploiting rank, bank-group, and subarray-level memory parallelism to accommodate different bandwidth requirements of different embedding data. We also propose a novel bandwidth-aware embedding partitioning technique to quantify the bandwidth requirements of embedding tables and allocate them to appropriate NMP levels for optimal performance. Our evaluation on the embedding layer of DLRM shows that ReCross can significantly outperform the state-of-the-art DIMM-based NMP solutions in terms of both performance and resource efficiency.

8 ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers of HPCA (2023) and ISCA (2023) for their insightful comments. This work is supported by the National Key Research and Development Program of China under Grant No. 2022YFB4501403 and the National Natural Science Foundation of China under Grant No. 62072195, 61825202, and 61832006. This work is also supported by Zhejiang Lab (Grant No. 2022P10AC02).

REFERENCES

[1] 2013. Criteo AI Labs Ad Terabyte. https://labs.criteo.com/2013/12/download-terabyte-click-logs/.

[2] 2014. Criteo AI Labs Ad Kaggle. https://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset.

[3] 2023. Gurobi Optimizer Reference Manual. https://www.gurobi.com.

[4] Bilge Acun, Matthew Murphy, Xiaodong Wang, Jade Nie, Carole-Jean Wu, and Kim M. Hazelwood. 2021. Understanding Training Efficiency of Deep Learning Recommendation Models at Scale. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021*. IEEE, 802–814.

[5] Bahar Asgari, Ramyad Hadidi, Jiashen Cao, Da Eun Shim, Sung Kyu Lim, and Hyesoon Kim. 2021. FAFNIR: Accelerating Sparse Gathering by Using Efficient Near-Memory Intelligent Reduction. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021*. IEEE, 908–920.

[6] Erik Brynjolfsson, Yu (Jeffrey) Hu, and Duncan Simester. 2011. Goodbye Pareto Principle, Hello Long Tail: The Effect of Search Costs on the Concentration of Product Sales. *Management Science* 57, 8 (2011), 1373–1386.

[7] Kevin Kai-Wei Chang, Donghyuk Lee, Zeshan Chishti, Alaa R. Alameldeen, Chris Wilkerson, Yoongu Kim, and Onur Mutlu. 2014. Improving DRAM performance by parallelizing refreshes with accesses. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture, HPCA 2014*. IEEE, 356–367.

[8] Dan Chen, Hai Jin, Long Zheng, Yu Huang, Pengcheng Yao, Chuangyi Gui, Qinggang Wang, Haifeng Liu, Haiheng He, Xiaofei Liao, and Ran Zheng. 2022. A General Offloading Approach for Near-DRAM Processing-In-Memory Architectures. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022*. IEEE, 246–257.

[9] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In *Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016*. IEEE, 27–39.

[10] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems, RecSys 2016*. ACM.

[11] Assaf Eisenman, Kiran Kumar Matam, Steven Ingram, Dheevatsa Mudigere, Raghuraman Krishnamoorthi, Krishnakumar Nair, Misha Smelyanskiy, and Murali Annavaram. 2022. Check-N-Run: a Checkpointing System for Training Deep Learning Recommendation Models. In *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022*. USENIX Association, 929–943.

[12] Amin Farmahini Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. 2015. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture, HPCA 2015*. IEEE, 283–295.

[13] Antonio A. Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2021. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. In *Proceedings of the IEEE International Symposium on Information Theory, ISIT 2021*. IEEE, 2786–2791.

[14] Carlos Alberto Gomez-Uribe and Neil Hunt. 2016. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems* 6, 4 (2016), 13:1–13:19.

[15] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, Xiuqiang He, and Zhenhua Dong. 2018. DeepFM: An End-to-End Wide & Deep Learning Framework for CTR Prediction. *CoRR* abs/1804.04950 (2018).

[16] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2020. DeepRecSys: A System for Optimizing End-To-End At-Scale Neural Recommendation Inference. In *Proceedings of the 47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020*. IEEE, 982–995.

[17] Udit Gupta, Samuel Hsia, Jeff Zhang, Mark Wilkening, Javin Pombra, Hsien-Hsin Sean Lee, Gu-Yeon Wei, Carole-Jean Wu, and David Brooks. 2021. 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, MICRO 2021*. ACM, 870–884.

[18] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim M. Hazelwood, Mark Hempstead, Bill Jia, Hsien-Hsin S. Lee, Andrey Malevich, Dheevatsa Mudigere, Mikhail Smelyanskiy, Liang Xiong, and Xuan Zhang. 2020. The Architectural Implications of Facebook's DNN-Based Personalized Recommendation. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture, HPCA 2020*. IEEE, 488–501.

[19] Kim M. Hazelwood, Sarah Bird, David M. Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture, HPCA 2018*. IEEE, 620–629.

[20] Oliver Hinz, Jochen Eckert, and Bernd Skiera. 2011. Drivers of the Long Tail Phenomenon: An Empirical Analysis. *Journal of Management Information Systems* 27, 4 (2011), 43–70.

[21] Yu Huang, Long Zheng, Pengcheng Yao, Qinggang Wang, Xiaofei Liao, Hai Jin, and Jingling Xue. 2022. Accelerating Graph Convolutional Networks Using Crossbar-based Processing-In-Memory Architectures. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture, HPCA 2022*. IEEE, 1029–1042.

[22] Yu Huang, Long Zheng, Pengcheng Yao, Qinggang Wang, Haifeng Liu, Xiaofei Liao, Hai Jin, and Jingling Xue. 2022. ReaDy: A ReRAM-Based Processing-in-Memory Accelerator for Dynamic Graph Convolutional Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 3567–3578.

[23] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. 2020. Centaur: A Chiplet-based, Hybrid Sparse-Dense Accelerator for Personalized Recommendations. In *Proceedings of the 47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020*. IEEE, 968–981.

[24] Kiyoo Itoh. 2013. *VLSI Memory Chip Design*. Vol. 5. Springer Science & Business Media.

[25] JEDEC. 2013. High Bandwidth Memory (HBM) DRAM.

[26] JEDEC. 2017. DDR4 SDRAM Standard.

[27] JEDEC. 2021. DDR5 SDRAM Standard.

[28] Qianghuai Jia, Ningyu Zhang, and Nengwei Hua. 2019. Context-aware Deep Model for Entity Recommendation in Search Engine at Alibaba. *CoRR* abs/1909.04493 (2019).

[29] Norman P. Jouppi, Andrew B. Kahng, Naveen Muralimanohar, and Vaishnav Srinivas. 2015. CACTI-IO: CACTI With OFF-Chip Power-Area-Timing Models. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23, 7 (2015), 1254–1267.

[30] Hongju Kal, Seokmin Lee, Gun Ko, and Won Woo Ro. 2021. SPACE: Locality-Aware Processing in Heterogeneous Memory for Personalized Recommendations. In *Proceedings of the 48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021*. IEEE, 679–691.

[31] Dhiraj D. Kalamkar, Evangelos Georganas, Sudarshan Srinivasan, Jianping Chen, Mikhail Shiryaev, and Alexander Heinecke. 2020. Optimizing deep learning recommender systems training on CPU cluster architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020*. IEEE/ACM, 43.

[32] Narendra Karmarkar. 1984. A New Polynomial-Time Algorithm for Linear Programming. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, 1984*. ACM, 302–311.

[33] Brent Keeth, R. Jacob Baker, Brian Johnson, and Feng Lin. 2007. *DRAM Circuit Design: Fundamental and High-speed Topics*. Vol. 13. John Wiley & Sons.

[34] Dongkyun Kim, Minsu Park, Sungchun Jang, Jun-Yong Song, Hankyu Chi, Geunho Choi, Sunmyung Choi, Jaeil Kim, Changhyun Kim, Kyung Whan Kim, Kibong Koo, Seonghwi Song, Yongmi Kim, Dong-Uk Lee, Jaejin Lee, Dae Suk Kim, Ki Hun Kwon, Minsik Han, Byeongchan Choi, Hongjung Kim, Sanghyun Ku, Yeonuk Kim, Jong-Sam Kim, Sanghui Kim, Youngsuk Seo, Seungwook Oh, Dain Im, Haksong Kim, Jonghyuck Choi, Jinil Chung, Changhyun Lee, Yongsung Lee, Joo-Hwan Cho, Junhyun Chun, and Jonghoon Oh. 2019. 23.2A 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 *Proceedings of the IEEE International Solid-State Circuits Conference, ISSCC 2019*. IEEE, 380–382.

[35] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. 2012. A case for exploiting subarray-level parallelism (SALP) in DRAM. In *Proceedings of the 39th International Symposium on Computer Architecture, (ISCA 2012)*. IEEE, 368–379.

[36] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (2016), 45–49.

[37] Neal Koyle and Sujeet Ayyapureddi. 2020. Micron ® DDR5: Key Module Features.

[38] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2019. 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, MICRO 2019*. ACM, 740–753.

[39] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2021. Tensor Casting: Co-Designing Algorithm-Architecture for Personalized Recommendation Training. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021*. IEEE, 235–248.

[40] Youngeun Kwon and Minsoo Rhu. 2022. Training personalized recommendation systems from (GPU) scratch: look forward not backwards. In *Proceedings of the 49th Annual International Symposium on Computer Architecture, ISCA 2022*. ACM, 860–873.

[41] Suk Han Lee, Shinhaeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, Jinhyun Kim, Seongil O, Anand Iyer, David Wang, Kyomin Sohn, and Nam Sung Kim. 2021.

Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology : Industrial Product. In *Proceedings of the 48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021*. IEEE, 43–56.

[42] Ke Liu, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim M. Hazelwood, Bill Jia, Hsien-Hsin S. Lee, Meng Li, Bert Maher, Dheevatsa Mudigere, Maxim Naumov, Martin Schatz, Mikhail Smelyanskiy, Xiaodong Wang, Brandon Reagen, Carole-Jean Wu, Mark Hempstead, and Xuan Zhang. 2020. RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing. In *Proceedings of the 47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020*. IEEE, 790–803.

[43] Ke Liu, Udit Gupta, Mark Hempstead, Carole-Jean Wu, Hsien-Hsin S. Lee, and Xuan Zhang. 2022. Hercules: Heterogeneity-Aware Inference Serving for At-Scale Personalized Recommendation. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture, HPCA 2022*. IEEE, 141–144.

[44] Ke Liu, Xuan Zhang, Jinin So, Jong-Geon Lee, Shinhaeng Kang, Sukhan Lee, Songyi Han, YeonGon Cho, Jin Hyun Kim, Yongsuk Kwon, KyungSoo Kim, Jin Jung, IlKwon Yun, Sung Joo Park, Hyunsun Park, Joon-Ho Song, Jeonghyeon Cho, Kyomin Sohn, Nam Sung Kim, and Hsien-Hsin S. Lee. 2022. Near-Memory Processing in Action: Accelerating Personalized Recommendation With AxDIMM. *IEEE Micro* 42, 1 (2022), 116–127.

[45] P. J. Meaney, L. D. Curley, G. D. Gilda, M. R. Hodges, D. J. Buerkle, R. D. Siegl, and R. K. Dong. 2015. The IBM z13 memory subsystem for big data. *IBM Journal of Research and Development* 59, 4/5 (2015).

[46] Micron. 2017. Micron: System Power Calculator (DDR4).

[47] Hadi Asghari Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. 2016. Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016*. IEEE, 50:1–50:13.

[48] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, Jie Amy Yang, Leon Gao, Dmytro Ivchenko, Aarti Basant, Yuxi Hu, Jiyan Yang, Ehsan K. Ardestani, Xiaodong Wang, Rakesh Komuravelli, Ching-Hsiang Chu, Serhat Yilmaz, Huayu Li, Jiyuan Qian, Zhuobo Feng, Yinbin Ma, Junjie Yang, Ellie Wen, Hong Li, Lin Yang, Chonglin Sun, Whitney Zhao, Dimitry Melts, Krishna Dhulipala, K. R. Kishore, Tyler Graf, Assaf Eisenman, Kiran Kumar Matam, Adi Gangidi, Guoqiang Jerry Chen, Manoj Krishnan, Avinash Nayak, Krishnakumar Nair, Bharath Muthiah, Mahmoud khorashadi, Pallab Bhattacharya, Petr Lapukhov, Maxim Naumov, Ajit Mathews, Lin Qiao, Mikhail Smelyanskiy, Bill Jia, and Vijay Rao. 2022. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture, ISCA 2022*. ACM, 993–1011.

[49] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2020. A Modern Primer on Processing in Memory. *CoRR* abs/2012.03112 (2020).

[50] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. *CoRR* abs/1906.00091 (2019).

[51] María Olmedilla, M. Rocío Martínez-Torres, and Sergio L. Toral Marín. 2019. The superhit effect and long tail phenomenon in the context of electronic word of mouth. *Decision Support Systems* 125 (2019), 113120.

[52] Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn. 2021. TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2021*. ACM, 268–281.

[53] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Shanker Khudia, James Law, Parth Malani, Andrey Malevich, Nadathur Satish, Juan Miguel Pino, Martin Schatz, Alexander Sidorov, Viswanath Sivakumar, Andrew Tulloch, Xiaodong Wang, Yiming Wu, Hector Yuen, Utku Diril, Dmytro Dzhulgakov, Kim M. Hazelwood, Bill Jia, Yangqing Jia, Lin Qiao, Vijay Rao, Nadav Rotem, Sungjoo Yoo, and Mikhail Smelyanskiy. 2018. Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications. *CoRR* abs/1811.09886 (2018).

[54] J. Thomas Pawlowski. 2011. Hybrid memory cube (HMC). In *Proceedings of the IEEE Hot Chips 23 Symposium, HCS 2011*. IEEE, 1–24.

[55] Mohammad R. Rezaei. 2021. Amazon Product Recommender System. *CoRR* abs/2102.04238 (2021).

[56] Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter R. Mattson, and John D. Owens. 2000. Memory access scheduling. In *Proceedings of the 27th International Symposium on Computer Architecture, ISCA 2000*. IEEE, 128–138.

[57] Geet Sethi, Bilge Acun, Niket Agarwal, Christos Kozyrakis, Caroline Trippel, and Carole-Jean Wu. 2022. RecShard: statistical feature-based memory optimization for industry-scale neural recommendation. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2022*. ACM, 344–358.

[58] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016*. IEEE, 14–26.

[59] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, SIGKDD, 2020*. ACM, 165–175.

[60] Gerard Sierksma and Yori Zwols. 2015. *Linear and Integer Optimization: Theory and Practice*. CRC Press.

[61] Xuan Sun, Hu Wan, Qiao Li, Chia-Lin Yang, Tei-Wei Kuo, and Chun Jason Xue. 2022. RM-SSD: In-Storage Computing for Large-Scale Recommendation Inference. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture, HPCA 2022*. IEEE, 1056–1070.

[62] Aniruddha N. Udipi, Naveen Muralimanohar, Niladrish Chatterjee, Rajeev Balasubramonian, Al Davis, and Norman P. Jouppi. 2010. Rethinking DRAM design and organization for energy-constrained multi-cores. In *Proceedings of the 37th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2010*. ACM, 175–186.

[63] Thomas Vogelsang. 2010. Understanding the Energy Consumption of Dynamic Random Access Memories. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2010*. IEEE, 363–374.

[64] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, SIGKDD 2018*. ACM, 839–848.

[65] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. 2021. RecSSD: near data processing for solid state drive based recommendation inference. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2021*. ACM, 717–729.

[66] Carole-Jean Wu, Robin Burke, Ed H. Chi, Joseph A. Konstan, Julian J. McAuley, Yves Raimond, and Hao Zhang. 2020. Developing a Recommendation Benchmark for MLPerf Training and Inference. *CoRR* abs/2003.07336 (2020).

[67] Xinfeng Xie, Zheng Liang, Peng Gu, Abanti Basak, Lei Deng, Ling Liang, Xing Hu, and Yuan Xie. 2021. SpaceA: Sparse Matrix Vector Multiplication on Processing-in-Memory Accelerator. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021*. IEEE, 570–583.

[68] Jie Amy Yang, Jianyu Huang, Jongsoo Park, Ping Tak Peter Tang, and Andrew Tulloch. 2020. Mixed-Precision Embedding Using a Cache. *CoRR* abs/2010.11305 (2020).

[69] Amir Yazdanbakhsh, Choungki Song, Jacob Sacks, Pejman Lotfi-Kamran, Hadi Esmaeilzadeh, and Nam Sung Kim. 2018. In-DRAM near-data approximate acceleration for GPUs. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques, PACT 2018*. ACM, 34:1–34:14.

[70] Mark Zhao, Niket Agarwal, Aarti Basant, Bugra Gedik, Satadru Pan, Mustafa Ozdal, Rakesh Komuravelli, Jerry Pan, Tianshu Bao, Haowei Lu, Sundaram Narayanan, Jack Langman, Kevin Wilfong, Harsha Rastogi, Carole-Jean Wu, Christos Kozyrakis, and Parik Pol. 2022. Understanding data storage and ingestion for large-scale deep recommendation model training: industrial product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture, ISCA 2022*. ACM, 1042–1057.

[71] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep Interest Evolution Network for Click-Through Rate Prediction. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*. AAAI Press, 5941–5948.