EMS-ı: An Efficient Memory System Design with Specialized Caching Mechanism for Recommendation Inference

YITU WANG, SHIYU LI, and QILIN ZHENG, Duke University, USA
ANDREW CHANG, Samsung Semiconductor, Inc., USA
HAI LI and YIRAN CHEN, Duke University, USA

Recommendation systems have been widely embedded into many Internet services. For example, Meta's deep learning recommendation model (DLRM) shows high prefictive accuracy of click-through rate in processing large-scale embedding tables. The SparseLengthSum (SLS) kernel of the DLRM dominates the inference time of the DLRM due to intensive irregular memory accesses to the embedding vectors. Some prior works directly adopt near data processing (NDP) solutions to obtain higher memory bandwidth to accelerate SLS. However, their inferior memory hierarchy induces low performance-cost ratio and fails to fully exploit the data locality. Although some software-managed cache policies were proposed to improve the cache hit rate, the incurred cache miss penalty is unacceptable considering the high overheads of executing the corresponding programs and the communication between the host and the accelerator. To address the issues aforementioned, we propose EMS-ı, an efficient memory system design that integrates Solide State Drive (SSD) into the memory hierarchy using Compute Express Link (CXL) for recommendation system inference. We specialize the caching mechanism according to the characteristics of various DLRM workloads and propose a novel prefetching mechanism to further improve the performance. In addition, we delicately design the inference kernel and develop a customized mapping scheme for SLS operation, considering the multi-level parallelism in SLS and the data locality within a batch of queries. Compared to the state-of-the-art NDP solutions, EMS-ı achieves up to 10.9× speedup over RecSSD and the performance comparable to RecNMP with 72% energy savings. EMS-ı also saves up to 8.7× and 6.6 × memory cost w.r.t. RecSSD and RecNMP, respectively.

CCS Concepts: • **Hardware** → **Memory and dense storage**; • **Computer systems organization** → *Data flow architectures;*

Additional Key Words and Phrases: Recommendation system, compute express link

ACM Reference format:
Yitu Wang, Shiyu Li, Qilin Zheng, Andrew Chang, Hai Li, and Yiran Chen. 2023. EMS-ı: An Efficient Memory System Design with Specialized Caching Mechanism for Recommendation Inference. *ACM Trans. Embedd. Comput. Syst.* 22, 5s, Article 100 (September 2023), 22 pages.
https://doi.org/10.1145/3609384

This work was supported by National Science Foundation under grant CNS-1822085 and the National Science Foundation IUCRC memberships from Samsung and other companies.
Authors' addresses: Y. Wang, S. Li, Q. Zheng, H. Li, and Y. Chen, Duke University, Durham, North Carolina, USA, 27707; emails: {yitu.wang, shiyu.li, qilin.zheng, hai.li, yiran.chen}@duke.edu; A. Chang, Samsung Semiconductor, Inc., San Jose, USA, 95134; email: andrew.c1@samsung.com.

1539-9087/2023/09-ART100 $15.00
https://doi.org/10.1145/3609384

1 INTRODUCTION

Recommendation systems have been widely used in many daily life applications such as advertising [1], social networks [35], and video clip recommendation [40]. Various recommendation models have been also proposed, among which deep learning recommendation model (DLRM) [26] demonstrates high efficiency in processing large-scale user and item data. Hundreds of embedding tables may be included in the DLRM and each embedding table could consist of millions of embedding vectors. Hence, the DLRM could consume hundreds to thousands of GBs memory. The operation kernels in the DLRM include multi-layer perceptron (MLP), SparseLengthSum (SLS) and self-defined feature interaction. SLS dominates the inference time of the DLRM because it requires irregular and sparse accesses to the embedding vectors among all the large embedding tables [15]. The unique data access pattern induces low utilization of memory bandwidth and poor data locality in the cache.

Some prior works [15, 33, 37] accelerate the inference of DLRM by directly using near data processing (NDP) solutions to increase the memory bandwidth. Other works [2] try to improve the data locality by using customized software-managed cache techniques, e.g., retaining the frequently accessed embedding vectors in the cache. However, all these works have some prominent weaknesses: In NDP solutions, for example, significant changes in the internal architecture of DRAM or SSD leads to high research and development (R&D) cost and may potentially increase the manufacturing cost. Besides, using only DRAM to store hundreds/thousands of GBs of embedding vectors induces high memory cost as DRAM chips are much more expensive than SSDs. Although SSD-based NDP platforms have low memory cost, the limited I/O bandwidth and low computing power lead to high inference latency. Moreover, the memory hierarchy of current NDP accelerators only has a small one-level cache residing above the DRAM or NAND flash chips. The inferior memory hierarchy fails to fully exploit the data locality of the DLRM workloads. In the solutions of software-managed caches, a host CPU is usually needed to run the corresponding algorithms to maintain a self-defined cache policy when a cache miss happens. Hence, besides the latency of DRAM or SSD access, the cache miss penalty also includes the execution time of the algorithm (~10 ms) and the communication latency between the host CPU and the accelerator (~100ns). The total latency of the cache miss may be overlapped during the training the DLRM through increasing the batch size to a large value, e.g., 2048 or more to only improve the throughput [5, 20, 25]. However, the approach adopted in the training is unacceptable for the inference of DLRM where the batch size could not be very large [15, 37] (usually no greater than 64) because both the inference latency and the inference throughput should be taken into account.

To address the issues in the prior works, we propose EMS-I, an efficient memory system design with a specialized caching mechanism and a inference kernel for recommendation system inference. The memory system of EMS-I is based on the combination of the memory components on FPGA (URAM/BRAM/HBM) and an SSD, which achieves a sweet point between the memory cost and the performance of DLRM inference. The variety of memory resources provides the opportunity of building a hierarchical memory system with multi-level caches to be customized for the DLRM workloads. However, in the conventional memory system, storage devices like SSD are not included. To integrate the SSD into our memory system, we adopt Compute Express Link (CXL) [7] to expand the memory space and build a unified memmory space consisting of the host memory, SSD and the HBM on the FPGA. In EMS-I, we develop L1 cache on the URAM, regard the HBM as L2 cache (which is mapped to the unified memory space as aforementioned) and use the internal DRAM cache in the SSD as L3 cache. Through implementing the back-invalidation coherence policy in CXL, the FPGA can directly communicates with the SSD via the CXL switch without the coordination of the host. We customize the caching mechanism to incorporate the software-managed cache technique into the hardware design of the cache controller to eliminate

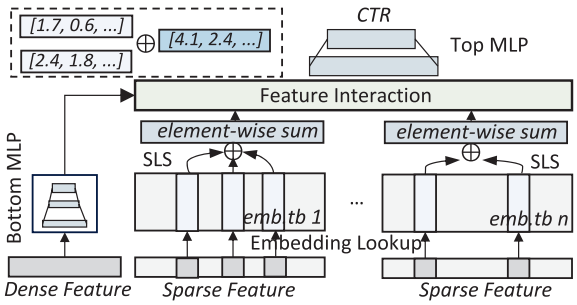


Fig. 1. General architecture of deep learning recommendation model and element-wise summation in *SparseLengthSum*.

the extra cache miss penalty. In addition, through a characteristic study, we found that the cache hit rate of the DLRM workload is sensitive to the cache replacement granularity in a naïve case of two-level caches. Inspired by the observation, we support the flexible adjustment of the replacement granularity in EMS-I. We also propose a novel prefetching mechanism to fetch the highly correlated embedding vectors together to improve the overall cache hit rate in the three-level caches of EMS-I. We develop an inference kernel to process the SLS and Multi-layer perceptrons (MLPs) in the DLRM on the HBM-enabled FPGA to avoid the internal hardware changes in the memory devices. Specially, we develop a hierarchical SLS array and propose a customized mapping scheme for the L1 cache in the inference kernel to support the multi-level parallelism existing in the SLS operation and explore the data locality within a batch of input queries. *As far as we know, we are the first to accelerate the inference of recommendation system from the view of memory system design.*

We summarize our contributions as follows.

- We conduct a comprehensive characteristic study for the DLRM workloads in the perspective of memory system and get meaningful insights to guide the design of EMS-I.
- We propose EMS-I, an efficient memory system design for recommendation system based on two CXL devices, an HBM-enabled FPGA and an SSD.
- We specialize the caching mechanism, including the delicate hardware cache controller to reduce the evictions of popular embedding vectors and the flexible adjustment of the cache replacement granularity.
- We develop a novel prefetching mechanism according to the correlation of the embedding vectors.
- We develop an efficient inference kernel and a customized mapping scheme to process the DLRM inference considering the multi-level parallelism in the SLS operation.

Experimental results show that EMS-I achieves up to 10.9× speedup over the state-of-the-art SSD-based NDP solution – RecSSD. EMS-I also achieves a comparable performance to the DRAM-only accelerator – RecNMP with 72% energy savings. In terms of memory cost, EMS-I is up to 8.7× and 6.6× cost-effective over RecSSD and RecNMP, respectively.

2 PRELIMINARY

2.1 Recommendation Model – DLRM

The general architecture of DLRM is illustrated in Figure 1. The input of one query consists of dense features and sparse features, showing the characteristics of the user and the item. The dense feature is fed into a bottom MLP. Meanwhile, each sparse feature, which is actually a one-hot or

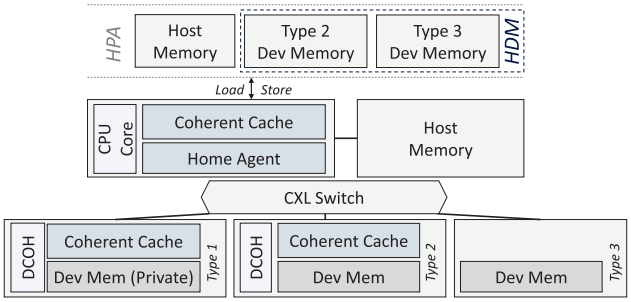


Fig. 2. CXL system architecture including type-1, type-2, and type-3 devices. The host physical address (HPA) is shown on the top where type-2 and type-3 device memory is host-managed device memory (HDM).

multi-hot vector, is translated into the embedding indices of each embedding table. SLS consists of two steps, embedding lookup and element-wise summation. According to the embedding indices of each embedding table, SLS first looks up the corresponding embedding vectors and then performs the element-wise summation on the embedding vectors as shown in the top left of Figure 1. The number of the embedding vectors that are looked up is referred to as pooling factor. For example, in Figure 1, the pooling factors of embedding Table 1 and embedding table n are 3 and 2, respectively. The results of the bottom MLP and SLS are processed by the feature interaction module, where the self-defined functions can be implemented such as concatenation and inner-product. Finally, the results of the feature interaction are inferred by a top MLP to get the click-through rate (CTR), which shows the possibility that the item is clicked by the user.

Meta develops a series of DLRMs such as RMC1, RMC2 and RMC3 [26]. The differences among the three specific models are mainly the size of MLP and embedding tables, and the number of embedding tables. According to [26], in general, RMC1 has small MLP, and small and few embedding tables. RMC2 has more embedding tables with small size than RMC1. The size of MLP and embedding tables are the largest in RMC3. In fact, the number and size of the embedding tables depends on the specific datasets, which also determines the specific architecture of the recommendation model. Hence, to some extent, it is unnecessary to differentiate RMC1, RMC2 and RMC3 when applying DLRMs to the real world as long as the model follows the workflow in Figure 1. In this paper, the models that we use are based on RMC series and the number of embedding tables is adjusted when running the model on different datasets.

2.2 Coherence Interconnect – CXL

CXL is a dynamic multi-protocol technology designed to support the low-latency and high-bandwidth communication among different CXL devices [7], which enables multiple heterogeneous devices to share the memory space with cache coherency. A typical CXL system consisting of a host, a CXL switch and multiple CXL devices, is illustrated in Figure 2. Based on the physical layer of PCIe, CXL provides a set of protocols including I/O semantics similar to PCIe (CXL.io), caching protocol semantics (CXL.cache), and memory access semantics (CXL.mem). According to the combination of the protocols, three types of devices are defined in CXL. With CXL.io and CXL.cache, type-1 devices are usually used as NICs. With all three protocols, type-2 devices are usually developed as accelerators that have both computing and memory resources. Using CXL.io and CXL.mem, type-3 devices can be recognized as the memory expansion of the system. Thanks to CXL.io protocol that enables the CXL device to expose device registers to host physical address (HPA), all the CXL hardware modules share a unified memory space, which includes the host memory, type-2 and type-3 device memory. Type-2 and type-3 device memory is referred to

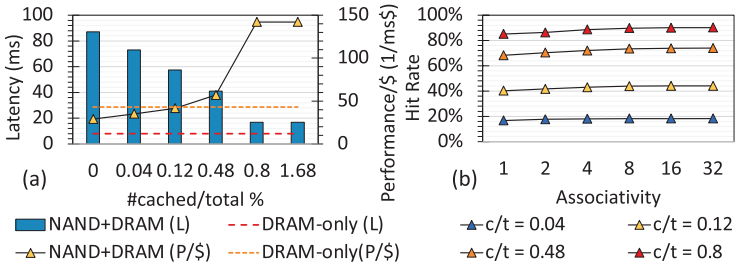


Fig. 3. (a) Comparison of latency (batch size = 64) and performance/$ among NAND-only, NAND+DRAM and DRAM-only platforms; (b) The influences of cache capacity and associativity on hit rate on the NAND+DRAM platform.

as host-managed device memory (HDM) when it is exposed to the host and can be accessed by the host via CXL.mem. Type-2 devices can buffer data residing in the HPA into its internal cache via CXL.cache. The internal coherent cache in type-2 is maintained by device coherency engine (DCOH) to guarantee the cache coherency across the host and other CXL devices.

HDM comes in three types according to the approach to maintaining the coherence: host-managed coherence (HDM-H), device managed coherence (HDM-D) and device managed coherence with back-invalidation (HDM-DB). EMS-I is mainly based on the coherency model of HDM-D with device bias mode and HDM-DB to reduce the overhead of the interruption from the host. The coherency model of HDM-D with device bias mode enables the device to access its device memory, which is mapped as HDM, directly without communicating with the host. The coherency model of HDM-DB allows direct snooping on each device. This coherency model enables P2P communication among CXL devices if using a CXL 3.0 switch where a P2P unordered I/O (UIO) is implemented. The corresponding logic of coherence protocol is implemented in the device's DCOH. Detailed specification is described in [7] and our work benefits from the application of the functionality of CXL.

3 CHARACTERISTIC STUDY

We conduct a characteristic study to get meaningful insights to guide the design of an efficient memory system for DLRM workloads. In this study, we assume that only CPU is available for computing. The simulation environment is based on gem5 [22] with the integration of SimpleSSD [13] and ramulator [18].

3.1 NAND-only v.s. DRAM-only v.s. NAND+DRAM

Firstly, we study the impact of the choices of memory devices on the latency of a batch (batch size is set to 64) of queries and the corresponding performance-memory cost ratio, i.e., performance/$, which is defined as $1/(latency \times memory\ cost)$. We assume that DRAM is 4.6$/GB and NAND flash is 0.42$/GB referring to [8, 32]. We use the synthetic workload generated by DeepRecSys [10] framework. For the NAND+DRAM platform, we adopt DRAM as the cache of NAND with LRU replacement policy. The x-axis in Figure 3(a) is the ratio of the number of the cached embedding vectors to the number of all the embedding vectors, which suggests DRAM with different sizes. As shown in Figure 3(a), the NAND-only (#cached/total % = 0) platform gets the highest latency due to the limited I/O bandwidth and high-latency data accesses to the NAND flash chips. In addition, the performance/$ of the NAND-only platform is also the lowest though NAND flash is relatively cheap. In contrast, the DRAM-only platform achieves the lowest latency, but its performance/$ is still not optimal. From the figure, we can observe that increasing DRAM capacity helps

Table 1. The Overall Cache Hit Rate with Vector and Block Replacement Granularity for the Lower Level Cache

	Syn 1	Syn 2	Criteo day3	fb_dlrm
Vector	83.6%	89.2%	96.1%	77.1%
Block	81.2%	90.6%	97.2%	75.4%

the NAND+DRAM get lower latency and higher performance/$. When the ratio, #cached/total (c/t), increases to 0.008, the NAND+DRAM platform gets the highest performance/$ and achieves comparable performance as the DRAM-only platform. *Hence, we believe that the platform with the combination of NAND and DRAM is the optimal choice to achieve the sweet point between the performance and the memory cost.*

3.2 Impact of Cache Capacity and Associativity

Figure 3(b) illustrates that on the NAND+DRAM platform, how the cache size (c/t) and associativity influences the DRAM cache hit rate. We can observe that the cache hit rate increases significantly as the size of the DRAM cache increases. In comparison, the associativity marginally improves the DRAM hit rate if the LRU cache replacement policy is implemented like most common caches. Specifically, after the associativity increases to 8, the hit rate saturates. *We can conclude that high associativity fails to improve the cache hit rate on DLRM workloads.*

3.3 Considerations in the Hierarchical Cache System

Motivated by the observations in Sections 3.1 and 3.2, we further explore the potential of building the hierachical caches in a more realistic system with a device-attached memory (DRAM) and an SSD. We regard the device-attached memory as the upper level cache and the internal DRAM cache in the SSD as the lower level cache. Developing the appropriate cache replacement policies is crucial to the cache hit rate for the two-level caches. Prior works mainly focus on improving the temporal data locality in the device-attached memory through LRU or software-managed cache replacement policies. However, those works neglect the opportunity of exploiting the existing spatial locality in the dataset, i.e., embedding vectors that are stored together may be accessed together. Intuitively, we think that the replacement granularity of the cache that is closest to where the original embedding vectors are stored may have an influence on the cache hit rate. Hence, we develop the internal DRAM cache in the SSD with two choices for the replacement granularity, the size of the embedding vector (Vector) or the size of the page buffer (Block) in the NAND flash. We conduct a toy experiment on four datasets as shown in Table 1. By configuring the parameters of generating synthetic workloads in DeepRecSys, we can get the workloads with lower spatial locality (Syn 1) and higher spatial locality (Syn 2). We assume that the internal DRAM in the SSD caches 2 million embedding vectors while the accelerator-attached memory caches 10 million embedding vectors (although the lower level cache is smaller than the upper level cache here, the assumption makes sense because in a realistic system, the device-attached memory is usually larger than the size of the internal DRAM cache (only 4 GBs) in the SSD). From the results in Table 1, we observe that the size of replacement granularity of the SSD internal DRAM cache indeed affects the overall cache hit rate. *Thus, we get the insight that the choice of the replacement granularity of the internal DRAM cache in the SSD should be "workload-aware".*

4 DESIGN OF EMS-I

According to our insights above, we build EMS-I, consisting of an SSD, an accelerator based on FPGA and a CPU host. Using CXL protocols, the SSD and the accelerator are developed as CXL

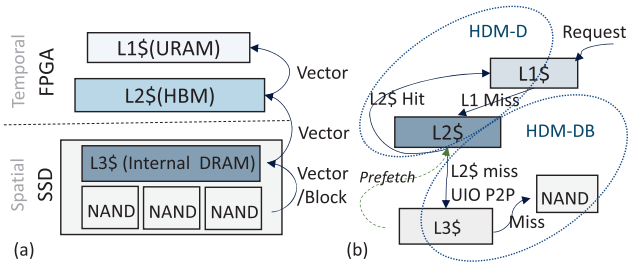


Fig. 4. (a) Memory system hierarchy and cache mapping of EMS-I; (b) Data path and coherency models of EMS-I.

type-3 and type-2 devices, respectively. The SSD and the accelerator are connected via a CXL switch. We integrate the two CXL devices into the CXL system considering the requirements of both storage and computation. The CXL-SSD is mainly used to store the huge amount of embedding vectors while the CXL-FPGA is developed to customize the inference kernels for the parallel SLS operations with its high-bandwidth memory and on-chip URAM. Note that EMS-I is designed for the inference rather than the training of the recommendation models. In the scenario of training, GPU, which has the efficient recommendation model training framework such as torchrec [34], can be developed as type-2 device and integrated into the CXL system following the similar memory system design of EMS-I with larger batch size. We leave it to the future works.

4.1 Memory System Design

We choose the HBM-enabled FPGA Xilinx VU57P as the platform for the accelerator due to the high bandwidth requirement of SLS. Besides the HBM, the accelerator-attached memory resources also include the on-chip URAM and BRAM. Inspired by our characteristic study, we develop all the accelerator-attached memory resources as the upper level caches and the internal DRAM cache in the SSD as the lower level cache. However, we further divide the levels of the memory on the FPGA. We develop the URAM as the L1 cache (also the internal cache of the accelerator) to support the multi-level parallelism of the SLS operation in the DLRM inference and the HBM as the L2 cache to improve the temporal data locality on the FPGA to secure a high cache hit rate, as illustrated in Figure 4(a). In addition, the internal DRAM inside the SSD is developed as the L3 cache to explore the potential spatial data locality in the dataset to further improve the overall cache hit rate and support the prefetching mechanism. The replacement granularities of the L1 and L2 cache are both "Vector" while the replacement granularity of the L3 cache can be adjusted to "Vector" or "Block" according to different datasets.

We develop the SSD as the type-3 device and the accelerator as the type-2 device in the CXL system. The memory space of the SSD and the HBM on the accelerator is mapped as the HDM. Figure 4(b) shows the data path and the corresponding coherence models of our design. When the accelerator receives the request from the host, the coherency model of HDM-D with device bias mode is firstly triggered to make the accelerator directly access the L2 cache (the HBM, mapped to HPA) and buffer the data into the L1 cache via CXL.cache without the interruption from the host CPU. If the cache miss happens in the L2 cache on the accelerator, then the accelerator communicates with the SSD using the coherency model of HDM-DB. The missed data in the L2 cache and the corresponding highly correlated data is fetched or prefetched from the L3 cache through the UIO, which enables P2P communication between the SSD and the accelerator without the data passing the host CPU. Note that the data transfer from the L3 cache to the L2 cache uses the CXL.io protocol and the L3 cache only buffers data from the NAND chips within the SSD rather

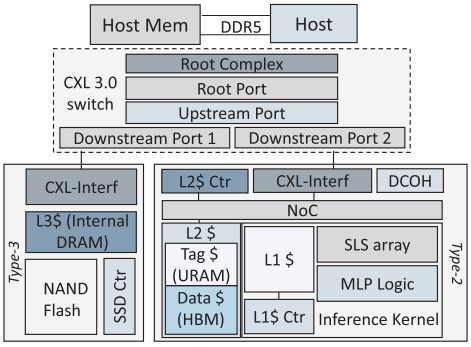


Fig. 5. Overall architecture of EMS-1.

than from other CXL devices because for the workloads of DLRM inference, there are no writes to the SSD.

4.2 Hardware Architecture

Figure 5 illustrates the overall hardware architecture of EMS-1. Both the SSD and the accelerator have their own CXL interfaces (CXL-interf) and the accelerator has the DCOH to maintain the coherence protocols. All the hardware modules within the accelerator are connected via a specialized NoC. The inference kernel for DLRM workloads consists of an L1 cache controller, an L1 cache that buffers the embedding vectors to be inferred, an SLS array and an MLP logic. The specialized inference kernel considers multi-level parallelism during the inference of DLRM and is illustrated in detail in Section 4.6. The L2 cache mechanism is based on the LRU policy but also modified for reducing the evictions of popular embedding vectors from the L2 cache, which will be described in Section 4.3 in detail. The architecture of CXL 3.0 switch is based on the UIO. The SSD and the accelerator are directly connected to the two separate downstream ports. Because we only have two CXL devices, the data path between the two devices is fixed. Hence, the host CPU just needs to configure the switch only once. The SSD controller is modified to support the flexible replacement granularity, i.e., Vector or Block, for the L3 cache.

4.3 Cache Specialization

To avoid the high latency of running software-managed cache strategies as described in Section 1, we customize the cache mechanism of the accelerator from the hardware side.

4.3.1 Resource Allocation and Tag Compression. The memory resources of Xilinx VU57P include 8.8625 MB BRAM, 33.75 MB URAM, and 16 GB HBM. When allocating the memory resources for the L1 and L2 cache, besides the cache size, the peripheral information such as tag, LRU bits, and other self-defined bits should also be taken into account, especially the memory consumption of the cache tags. As we know, one cache address consists of a tag, a set index, and an offset. Assuming that the memory space is M, the cache associativity is A, the size of a cache block is B, and the cache size is C, then we can get the memory consumption of the tags of the cache as shown in the following equation,

$$\begin{aligned} MemConsp(tags) &= \frac{C}{B}\left(log_2 M - log_2 \frac{C}{AB} - log_2 B\right) \\ &= \frac{C}{B}\left(log_2 \frac{MA}{C}\right). \end{aligned} \tag{1}$$

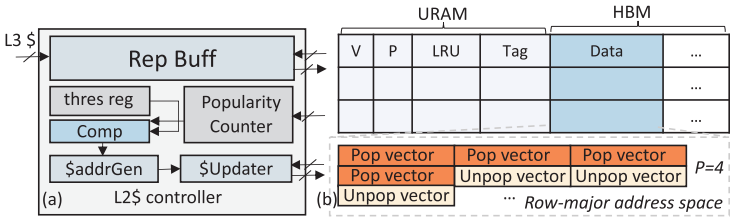


Fig. 6. (a) The L2 cache controller; (b) The L2 cache structure.

We aim to store the tags in the URAM instead of HBM or the host's DRAM to avoid the extra cache miss penalty induced by the high latency of looking up tags. Hence, the L2 cache consists of a tag cache based on the URAM and a data cache based on the HBM as shown in Figure 5. The tag cache also buffers other peripheral information for the L2 cache as described in Section 4.3.2. Note that the each entry in the tag cache is aligned with each entry in the data cache as illustrated in Figure 6(b). We assume that the size of L1 cache is 16 MB and all the 16 GB HBM is used as the L2 cache. Meanwhile, the memory space is set as 1TB, which is enough for most of the datasets. Then, for the L2 cache, we get the constraint,

$$\frac{16GByte}{B}\left(log_2\frac{1TByte\times A}{16GByte}\right) < (33.75-16)MByte. \tag{2}$$

We can observe that by reducing the associativity and enlarging the block size, the memory consumption of the tags can be reduced. Remember that in Section 3.2, we get the conclusion that high associativity just marginally improves the cache hit rate. Hence, setting associativity to 8 is a perfect choice here. Finally, we set the block size to 4096 bytes to satisfy the constraint. Note that the cache access granularity is the size of the cache block. In such a memory allocation, the memory consumption of the tags is 8 MB for the L2 cache, with the remaining memory resources allocated to other peripheral information in the cache structure, which will be shown in the next. We specialize the L1 cache organization with the inference kernel design in Section 4.6.

4.3.2 Customized Cache Replacement Policy. Our L2 cache replacement policy is based on the LRU policy. Hence, the cache structure includes the corresponding LRU bits. Meanwhile, we modify the policy to reduce the evictions of popular embedding vectors. Firstly, we attach a popularity bit "1" to each popular embedding vector in the preprocessing of the dataset. We sample 30% of the input queries and do the statistics on the access frequency of each embedding vector. If the access frequency of an embedding vector is higher than some pre-defined threshold, then the embedding vector is recognized as a popular embedding vector. Correspondingly, popularity bits (P) are also added in the cache structure to show the number of popular embedding vectors in the block, as shown in Figure 6(b). When a cache block is selected to be replaced by the LRU policy, the popularity counter will check the popularity bits of the block and compare the popularity value with a pre-defined threshold as illustrated in Figure 6(a). If the popularity value is larger than the threshold, then the block will not be replaced, and a new address will be generated to select a new cache block. Otherwise, the block will be selected to be replaced, and the corresponding valid bits (V), LRU bits and tag will be updated.

So far, it is not difficult to find that the access granularity (the size of the cache block) is larger than the replacement granularity ("Vector") of the L2 cache, which could induce the underutilization of the L2 cache. Note that EMS-ı processes a batch of queries simultaneously so the missed embedding tables are also fetched in a batch. When a cache miss happens in the L2 cache, a batch of the missed embedding vectors, which are indicated by the current batch of quries,

are firstly loaded into the replacement buffer from the L3 cache as shown in Figure 6(a). Then, the missed embedding vectors are grouped according to their tags. The data in a replaced cache block is updated by a group of the embedding vectors with the same tag. Hence, a large group size helps load more embedding vectors into a replaced cache block, which finally improves the L2 cache utilization. Through increasing the batch size of the input queries, the group size of the missed embedding vectors can be implicitly increased. In addition, our prefetching mechanism (see Section 4.5) can further increase the group size by fetching the correlated embedding vectors along with the missed embedding vectors.

4.4 Support for Flexible Replacement Granularity

The address of NAND flash consists of row address and column address. The row address is used to address the logic units, blocks and pages. The column address is used to access the data in the page buffer. Hence, although the access granularity of transferring data from the NAND flash array to the page buffer is usually the page size, e.g., 4KB, the SSD controller can access more fine-grained data, e.g., the size of an embedding vector, from the page buffer using specific column address. We modify the flash transaction layer (FTL) on the SSD controller to adjust the address of the data to be accessed according to the choice of the replacement granularity. RM-SSD [33] supports the fine-grained data access in the storage with a similar way.

4.5 Correlation-based Prefetching Mechanism

There exist mainly two challenges in the design of the prefetching mechanism for the DLRM inference. Firstly, the access pattern of embedding vectors can vary significantly from one dataset to anther one. Even within the same dataset, we observe different access patterns across different embedding tables. Secondly, the memory and consumption requirement of the prefetching mechanism should be reasonable and within the budget of hardware resources to improve the overall performance. To address the first challenge, we need to find a general solution that can be adapted to different access patterns. For the second challenge, we need to tune the hyperparameters of the prefetching mechanism to achieve the sweet point between the hardware resource consumption and the performance.

The prefetching of EMS-I happens when fetching the embedding vectors missed in the L2 cache, the correlated embedding vectors of the missed embedding vectors are prefetched from the SSD. Our prefetching mechanism is based on the correlation between the embedding vectors because our insight is that *no matter what the specific access pattern is, highly related embedding vectors are much more likely to be accessed together*. Hence, the prefetching mechanism requires two steps. Firstly, we generate the correlation information of the embedding vectors. Secondly, the correlation-based prefetching is performed during the runtime of DLRM inference using the generated correlation information. The two steps mentioned above are referred to as $Pref_{train}$ and $Pref_{infer}$, respectively.

4.5.1 $Pref_{train}$. The goal of $Pref_{train}$ is to get the correlation information, the score matrix and the correlation dictionary, as shown in Figure 7. Each element $S_(i, j)$ in the score matrix showing how many times $vector_i$ and $vector_j$ are accessed together by one query. We sample 30% of the input queries of the whole targeted dataset to do the statistics. Meanwhile, the indices of the top K popular embedding vectors are also obtained along with the score matrix. Then, a correlation dictionary is abstracted from the score matrix for the top K popular embedding vectors. The key of the dictionary is the index of the K embedding vectors, and each value list consists of each popular embedding vector's correlated embedding vectors with top E scores as illustrated in Figure 7. Note that we constrain the distance between the index of the popular embedding vector and the index

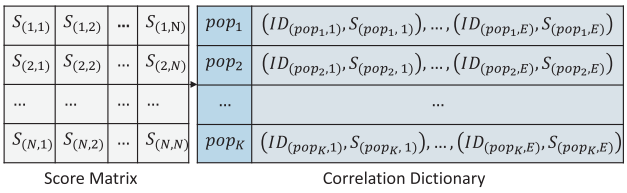


Fig. 7. The score matrix and correlation dictionary in the prefetch mechanism of EMS-I.

Table 2. Parameters of the Prefetching Mechanism

Parameter	Description
W	window size of correlated embedding vectors
$S_{(i,j)}$	freq. of $vector_i$ and $vector_j$ are fetched together
K	length of the abstracted correlation dictionary
E	length of the value list of each key
L	number of the prefetched vectors from each list

ALGORITHM 1: The Training of the Correlation-based Prefetching Mechanism

Input: Dataset D; Sample rate R; W; K; E; L

1: Sample R of D to get the input traces for training, $Trace_{train}$
2: $N = max(index\ in\ Trace_{train})$; create a score matrix M_s with size $N \times N$ and each element in M_s is $S_{(i,j)}$
3: **for** each input trace t in $Trace_{train}$ **do**
4: **for** each index i in t **do**
5: $S_{(i,i)} += 1$
6: **for** each index j in $t \setminus \{i\}$ **do**
7: $S_{(i,j)+=1}$
8: $keys$ = indices of top $K(\ S_{(i,i)}\ for\ i\ in\ range(0, N)\ )$; create a correlation dictionary $Dict_{cor}$ with $keys$
9: **for** each key pop_k in $keys$ **do**
10: Fetch top E elements $\{S_{pop_k,e}\}$from $S_{pop_k,1}$ to $S_{pop_k,N}$ and the corresponding indices $\{ID_{(pop_k,e)}\}$
11: Create a list for pop_k with each element as $(ID_{(pop_k,e)}, S_{(pop_k,e)})$
12: **return** $Dict_{cor}$

of the correlated embedding vector within W, which can ensure that the missed embedding vector and its correlated embedding vectors have the same cache tag so that they can be loaded into the replacement buffer together. Some key parameters of $Pref_{train}$ are summarized in Table 2 and Algorithm 1 illustrates the corresponding pseudo code. Note that the final output of $Pref_{train}$ is the correlation dictionary. The score matrix is intermediate.

4.5.2 $Pref_{infer}$. During the runtime of the inference of DLRM workloads, when the index of a missed embedding vector in the L2 cache hits the key of the correlation dictionary, top L correlated embedding vectors associated with the key will be fetched together with it from the SSD. The prefetching mechanism is maintained by the modified SSD controller. The pseudo code is illustrated in Algorithm 2.

Because we only store the correlation dictionary, our prefetching mechanism does not consume much memory resources. For example, a typical setting of the hyperparameters is that

ALGORITHM 2: The Inference of the Correlation-based Prefetching Mechanism

Input: Correlation dictionary $Dict_{cor}$; Index of the missed embedding vector in the L2 cache id_{miss}

```
for each pop_k in keys do
    if id_miss == pop_k then
        Pref_cand = Dict{"pop_k"}
        break
Sort Pref_cand with the value of S_(pop_k, e) in the descending order
Pref_indices = the first L ID_(pop_k,:) in the sorted Pref_cand
return Pref_indices
```

$W = 100$, $L = 10$, $K = 30000$ and $E = 100$ for the different embedding tables in the Criteo dataset. Assuming that a tuple of $(ID_{(pop_k,e)}, S_{(pop_k,e)})$ is 8 byte in total, the corresponding memory consumption of the correlation matrix is only 23 MBs. In EMS-1, we still have two constraints for tth hyperparameters. Firstly, considering the architecture of the L2 cache, we constrain $W < 2^{bit\_len(set\ index)+bit\_len(offset)}$ because we want to ensure that the prefetched embedding vectors share the same tag with the missed embedding vector so that they can be loaded into the same cache block to improve the L2 cache utilization. Secondly, L should not be large to avoid the situation that one cache block cannot buffer all the prefetched embedding vectors.

4.6 Inference Kernel Design

From Figure 1, we observe three-level parallelisms of SLS in the inference DLRM workloads, which are (i) inter-table parallelism: SLS operations on different embedding tables can be performed in parallel, (ii) intra-table parallelism: element-wise summation of multiple embedding vectors within an embedding table can be performed in parallel, (iii) intra-vector parallelism: multiple elements of an embedding vector can be fetched and computed in parallel. Correspondingly, from the the perspective of hardware, the inference kernel is hierarchically designed for the three-level parallelisms of SLS and the L1 cache is disaggregated for the SLS units. From the perspective of software, we specialize a mapping scheme to allocate the disaggregated cache resources to buffer the embedding vectors.

4.6.1 Hardware Specialization. Figure 8(a) shows the overall architecture of the inference kernel, which consists of an L1 cache controller, an SLS array and an MLP logic. In the L1 cache controller, the embedding vector address queue buffers the addresses of the embedding vectors to be performed SLS from the host. Note that the addresses are translated by the host according to the logical indices of the embedding vectors. Using the addresses, the L1 cache controller fetches the missed embedding vectors from the L2 cache and load the data into its read buffer. Considering the refreshing of the SSD FTL, the addresses of the embedding vectors may dynamically change. Hence, the memory space of one embedding table can be broken into several discrete pieces. The embedding table address registers record the address ranges (start and end addresses of the discrete pieces) of the memory space of each embedding table so that the L1 mapper can recognize which embedding table that each fetched embedding vector belongs to and then sends the embedding vectors to the corresponding SLS Unit's URAM block via the AXI bus, following the mapping scheme described in Section 4.6.2.

On the right of Figure 8(a), the SLS array and the MLP logic are illustrated. The SLS array includes 14 SLS Units and each SLS unit has its own URAM blocks. The L1 cache in the inference kernel consists of 488 URAM blocks, which is within the budget of 16 MBs as aforementioned in

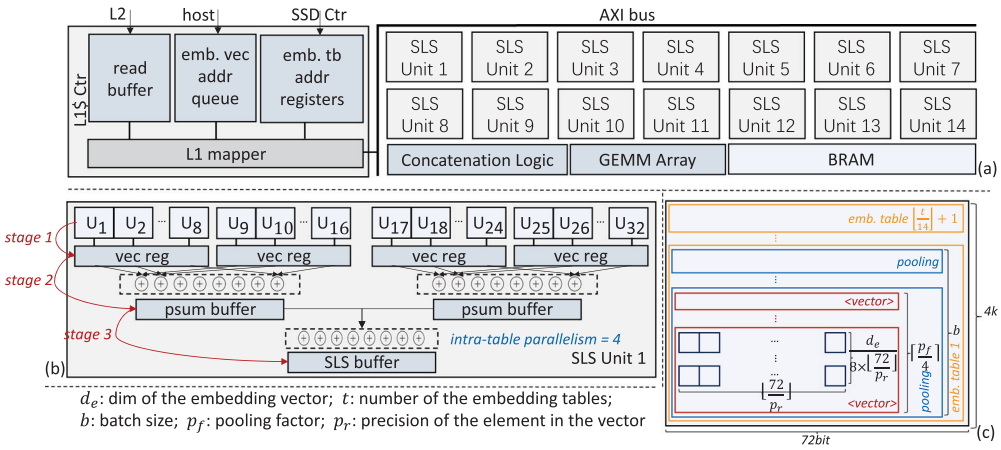


Fig. 8. (a) The overall architecture of the inference kernel of EMS-I; (b) The architecture of the SLS Unit 1; (c) The allocation of the URAM block within the SLS Unit 1 in (b).

Section 4.3.1. To maximize the bandwidth of the URAM blocks and the three-level parallelisms of SLS operation, the cascade size of the URAM is set to 1 and the 488 URAM blocks are evenly assigned to the 14 SLS Units. As for the MLP logic which is not the focus of our work, we adopt a similar design to SmartRec [30] using a GEMM array to perform the matrix-vector multiplication in the MLP layers. The concatenation logic is used to concatenate the results of each SLS operations as the input of the top MLP. The BRAM is used to store the parameters of the MLP layers.

The internal architecture of the SLS Unite is shown in Figure 8(b). There are 32 URAM blocks in each SLS Unit and every 8 URAM blocks are allocated to a vector register to buffer one embedding vector. Two embedding vectors are fed into one group of adders to perform the element-wise summation, the result of which is stored in the psum buffer. The bitwidth of the vector register equals the total bitwidth of the URAM blocks that are allocated to it. The results from two psum buffers can be further performed the element-wise summation on if the pooling vector is larger than 2 and the final result of the SLS is stored in the SLS buffer. Obviously, there exists a three-stage pipeline in the SLS Unit as shown in Figure 8(b). The SLS operation of one embedding table is only processed in one SLS Unit, the reason of which will be introduced in Section 4.6.2. Hence, we get that in our design, the inter-table parallelism is 14, the intra-table parallelism is 4. In general, *we summarize that the inter-table parallelism depends on the number of SLS Units and the intra-table parallelism depends on the number of vector registers or groups of adders in each SLS Unit.* We further discuss the design space in Section 5.3.8.

4.6.2 Mapping Scheme. Figure 8(c) shows the mapping of the embedding vectors in a URAM block in SLS Unit 1, suggesting the support of the three-level parallelisms of SLS operation and exploiting the data locality within a batch of embedding vectors in one embedding table. To support the inter-table parallelism, the embedding vectors from one embedding table are mapped to the 32 URAM blocks in one SLS Unit so that the SLS array can process SLS operations of 14 embedding tables simultaneously. When the number embedding table of the DLRM is larger than 14 (which is the common case), the embedding vectors from different embedding tables are allocated to one SLS Unit, as shown in the yellow rectangles in Figure 8(c). To avoid redundant accesses to the embedding vectors from the L2 cache within an inference batch, the L1 cache controller fetches a batch of embedding vectors together and maps a batch of embedding vectors from the same embedding table in one SLS Unit. In Figure 8(c), the b rectangles represent the embedding vectors indicated by

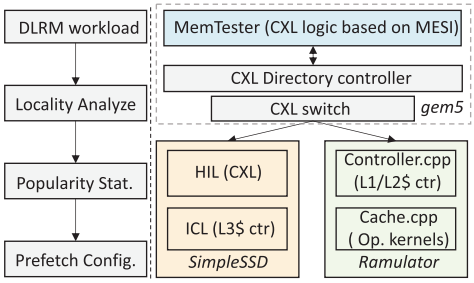


Fig. 9. Simulation framework.

b input queries in a batch. To support the intra-table parallelism, different embedding vectors that are performed by one SLS operation are mapped to different URAM blocks in one SLS Unit. We map one embedding vector to 8 URAM blocks, with each URAM block storing 1/8 of the embedding vector. So there are at most 4 embedding vectors can be fetched and performed element-wise summation in parallel. If the pooling factor is larger than 4, more than one embedding vectors that are indicated by one query are allocated to one URAM block as shown in the red rectangles in Figure 8(c). Meanwhile, the intra-vector parallelism is also supported because the elements from different dimensions can be loaded from different URAM blocks. Using the notations in Figure 8(c), the elements from $\lfloor \frac{72}{p_r} \rfloor$ dimensions of one embedding vector can be loaded from one URAM block once because the bitwidth of the URAM block is 72 bit. Hence, the elements from $\lfloor \frac{72}{p_r} \rfloor \times 8$ dimensions of one embedding vector can be loaded from 8 URAM blocks once and are performed the element-wise summation with another embedding vector in parallel. *We can conclude that the intra-vector parallelism is determined by* $\lfloor \frac{72}{p_r} \rfloor \times$ *the number of URAM blocks that are allocated to one vector register.* If the dimension of one embedding vector is larger than $\lfloor \frac{72}{p_r} \rfloor \times 8$, then more than $\lfloor \frac{72}{p_r} \rfloor$ elements of one embedding vector are allocated to one URAM block as shown in the squares in Figure 8. Considering the resource budget of URAM, we can get the constraint of the mapping scheme using the notations in Figure 8,

$$\frac{d_e}{8} \times \left\lceil \frac{p_f}{4} \right\rceil \times b \times \leq \left\lceil \frac{t}{14} \right\rceil \leq 4000. \tag{3}$$

Normally, we decrease the batch size b if the constraint is not satisfied.

5 EVALUATION

5.1 Experimental Methods

5.1.1 Simulation Framework. Figure 9 shows the simulation framework of EMS-I. We first perform the preprocessing on the DLRM workload to analyze the data locality in the dataset to determine the L3 cache replacement granularity, determine the threshold of access frequency to select the popular embedding vectors, and set the parameters of the prefetching mechanism. Due to the lack of an available open-source CXL system simulator, we modify the MESI protocol provided by the Ruby memory system in gem5 to implement the behavioral logic of CXL protocols. We use Ramulator in cpu-driven mode to simulate the memory system on FPGA and integrate the hardware behavior models of the inference kernel, which are abstracted from the synthesized Xilinx HLS with the frequency set to 300 MHz. In addition, we also integrate the SimpleSSD into gem5 and modify the host interface layer (HIL) and internal cache layer to support the CXL coherence models, L3 cache replacement policy, and our prefetching mechanism. We refer to [21] to get the reasonable CXL latency parameters. The allocation of memory resources follows the

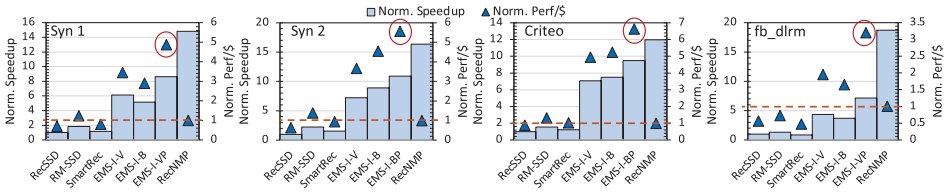


Fig. 10. Speed up normalized to RecSSD and Performance/$ normalized to RecNMP. The batch size is set to 64.

description in Section 4. The settings of the emulated SSD and HBM are referred to [28] and [39], respectively. The simulator is memory trace based which means that we do not need to get the actual embedding vectors (train the DLRM models). The memory traces, which are translated from the indices of the embedding vectors in the datasets, drive the simulator to emulate the behavior of the storage and memory devices in EMS-I.

5.2 Baselines and Benchmarks

We compare EMS-I with RecSSD [37] (SSD-only), RecNMP [15] (DRAM-only), RM-SSD [33], and SmartRec [30] for DLRM workloads. RM-SSD implements an internal FPGA within the SSD while SmartRec directly adopts SmartSSD which connects an SSD to an FPGA via the PCIe switch. We build the baselines according to their design philosophy in our simulation framework. As for the datasets, we use the synthetic workloads generated by DeepRecSys (Syn 1 and Syn 2; 40 embedding tables for each; different spatial data locality through adjusting the specific parameters), Criteo [6] (26 embedding tables) and fb_dlrm [9] (first 128 embedding tables). As for the recommendation model, we use the basic settings of MLP and dimension of embedding vectors of RMC2 [26] but align the number of embedding tables of the model with the number of the embedding tables with the specific dataset that the model runs on. Hence, in our experiments, the models that run on the 4 datasets share the same bottom MLP structure but have different top MLP structures, which are 256-128-64 and (number of the embedding tables × dimension of the embedding vector) - 128-64-1, respectively. The MLP is not deep or complex in our recommendation models because this work mainly focuses on addressing the issue of data access to and SLS operations on the embedding tables rather than accelerating the MLP. We do not consider the quantization of the embedding vector in this work so the default precision of each element of the embedding vector is 32-bit. The default dimension of the embedding vector is set to 64 unless otherwise specified.

5.3 Evaluation Results

5.3.1 Speedup. Figure 10 illustrates the normalized speedups of different designs over RecSSD. To evaluate the influence of the replacement granularity of the L3 cache on the performance, we implement EMS-I with "Vector" replacement granularity (EMS-I-V) and "Block" replacement granularity (EMS-I-B). In addition, based on the better replacement granularity, we further implement EMS-I with the prefetching mechanism (EMS-I-VP/BP). The performance of RecSSD and SmartRec is not as good as the others'. RecSSD only has a 4GB internal DRAM as the cache, which is shared by the operations of the FTL layer of SSD and the workloads of DLRM. Moreover, all the computing tasks in RecSSD are completed by the embedded cores, which cannot satisfy the requirement of the computation of SLS and MLP. Although SmartRec adopts the FPGA to compute, the attached memory is just a single channel 4GB DRAM. The memory bandwidth is only 19 GB/s. By implementing a low-end FPGA inside the SSD, RM-SSD outperforms RecSSD and SmartRec benefiting from the stronger computing power and higher SSD internal bandwidth. However, none of the

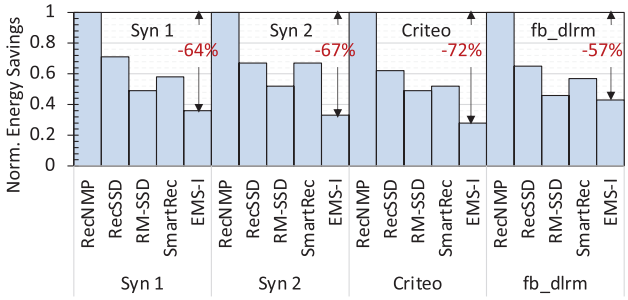


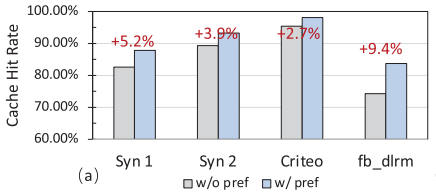
Fig. 11. Energy savings normalized to RecNMP.

aforementioned designs develops a multi-level cache, let alone a customized caching mechanism. Hence, most embedding vectors should be accessed from the NAND flash chips. Benefiting from the customized three-level cache mechanism, over 83% accessed embedding vectors can be found in our caches. Overall, when the batch size is set to 64, EMS-I can achieve up to 10.9×, 6.1×, and 8.2× over RecSSD, RM-SSD, and SmartRec, respectively. EMS-I does not outperform RecNMP but achieves comparable performance. This is mainly because the access latency of DRAM could be 10× lower than that of SSD. We can also observe that EMS-I-B outperforms EMS-I-V on Syn 2 and Criteo. This is because Syn 2 and Criteo illustrate good spatial data locality, i.e., the embedding vectors in a replaced block are highly possible to be accessed in the future. On the datasets of Syn1 and fb_dlrm, which do not show much spatial data locality, "Vector" replacement granularity is more suitable. With our correlation-based prefetching mechanism, the performance of EMS-I is further improved.

5.3.2 Performance/$. Figure 10 also shows the performance/$ in terms of memory cost. Although our design adopts HBM-enabled FPGA, HBM is naturally based on 3D-stacked DRAM, and its price is a little higher than DRAM, but it still applies to our insight in Section 3. We estimate the price of HBM referring to [8, 11]. We can observe that EMS-I achieves the highest performance/$. Because through the dedicated design of the memory system, EMS-I achieves the sweet point between performance and memory cost. The performance/$ of EMS-I is up to 6.6× over that of RecNMP because all the embedding vectors should be stored in the DRAM in RecNMP. The price of DRAM is over 11× higher than that of SSD. Although RecSSD only uses SSD, it still achieves the lowest performance/$ on three datasets due to its poor performance.

5.3.3 Energy. We also evaluate the energy savings, which are normalized to RecNMP, of different designs as shown in Figure 11. The energy savings of EMS-I compared to RecNMP mainly comes from the power consumption of DRAM. The power consumption of 1TB DRAM is ~380W. Overall, EMS-I can achieve up to 72% energy savings compared to RecNMP. Due to the low power consumption of the SSD, the power consumption of RecSSD and RM-SSD is lower than EMS-I. However, EMS-I executes much faster than RecSSD and RM-SSD as shown above. Hence, EMS-I can still outperform RecSSD and and RM-SSD in terms of energy savings.

5.3.4 Cache Hit Rate. Figure 12(a) shows the total cache hit rate of the three-level cache in EMS-I and Figure 12(b) shows the specific setting of the prefetching mechanism for each DLRM workload. EMS-I finally achieves over 83% cache hit rate over all the datasets thanks to the large L2 cache and the customized cache replacement policies. With our prefetching mechanism, the cache hit rate increases because the utilization of the L2 cache is improved. Remember that due to the requirement of compression of tags, we enlarge the block size of the L2 cache and add a replacement buffer. Although when processing a batch of queries' cache misses we can load



(b)

	Syn 1	Syn2	Criteo	fb_dlrm
W	100	100	100	100
K	40000	40000	30000	100000
E	100	100	100	100
L	8	5	10	6

Fig. 12. (a) Overall cache hit rate of EMS-I w/ and w/o the prefetching mechanism; (b) The settings of the prefetching mechanism for different DLRM workloads.

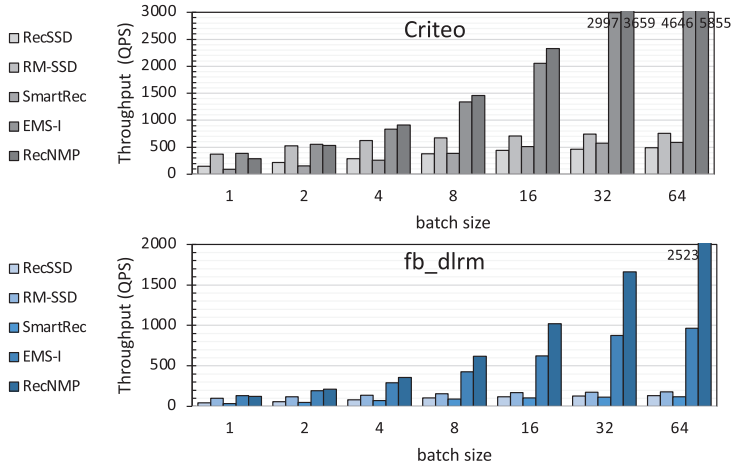


Fig. 13. The throughput of various designs on Criteo and fb_dlrm, with batch size from 1 to 64.

several embedding vectors with the same tag at a time into the replacement buffer, only 57% of the replacement buffer is written on average. However, with the prefetching, more correlated embedding vectors are written into the replacement buffer. The average utilization of replacement buffer is improved to 86.4%. Hence, the cache hit rate is improved.

5.3.5 Analysis of Batch Size v.s. Throughput. We explore the impact of batch size on the throughput of various designs as shown in Figure 13. We select Criteo [6] and fb_dlrm [9] in this experiment because they are officially released by Kaggle [14] and Meta [24], whose data could be more realistic. In general, we can observe that the throughput improvement of the design that has more DRAM or HBM is more sensitive to the increase of the batch size. This is mainly because the bandwidth of SSD I/O of SmartRec and the internal SSD bandwidth of RecSSD and RM-SSD (~tens of GB/s) are lower and saturate faster than the memory bandwidth of DRAM/HBM on RecNMP/EMS-I (~hundreds of GB/s) when the batch size of the input queries increases. In addition, the SSD access latency (~30μs) is much higher than DRAM and HBM access latency (~70μs and ~110μs), which naturally makes the throughput gap between the in/near-SSD designs and the in/near-DRAM/HBM designs larger when processing a larger batch of embedding vectors. Although there are some optimizations for the MLP inference in RM-SSD, the throughput of RM-SSD does not obviously improve as the batch size increases because for the embedding-dominated recommendation models (see the settings in Section 5.2), the throughput is constrained by the SLS operations. The execution time of MLP inference only accounts for ~6% of the total execution time on our RM-SSD baseline. It is also not difficult to find that the throughputs of all the designs on fb_dlrm are not as high as on Criteo because fb_dlrm has many more embedding tables than

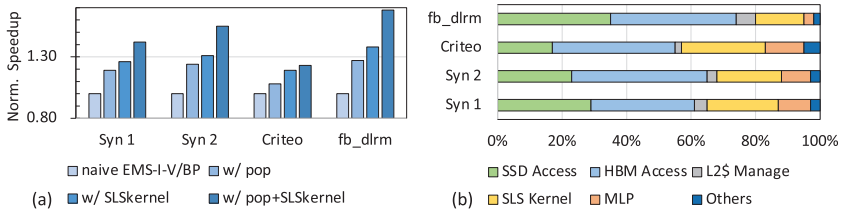


Fig. 14. (a) Ablation study of EMS-I; (b) Execution time breakdown of EMS-I.

Criteo, i.e., the model running on fb_dlrm is more embedding-dominated. Benefiting from the specialized SLS kernel and multi-level cache mechanism, the throughput of EMS-I can compete with RecNMP though there exists a gap due to much fewer DRAM resources. It is not difficult to observe that the batch size just increases to 64, which is a typical value of the batch size of the inference of recommendation models [15, 37]. The batch size of the inference cannot be very large because there is a trade-off between the latency of the inference of one query and the throughput of a batch of queries. Although increasing the batch size can improve the throughput, it also induces higher latency, which may violate the requirement of the maximal response time (tens or a few hundreds of milliseconds) of some online recommendation services like e-commerce website. In fact, when the batch size exceeds 128, the overall latency will increase exponentially [15]. In the scenario of training recommendation models, where latency is not a significant issue, the batch size can be set to large values like 1024, 2048 and even larger [5] to just improve the overall throughput. In addition, the number of the total accessed embedding vectors in a batch is equal to $batch\ size \times average\ pooling\ factor\ per\ table \times \#embedding\ tables$, which is large enough to exploit our memory system design when the batch size is set to 64, especially when the number of embedding tables is also large.

5.3.6 Ablation Study. We perform an ablation study to show the benefits of the multiple techniques that are proposed. In fact, the effects of flexible replacement granularity and prefetching mechanism have been shown in Figure 10. We further evaluate the influences of the popularity bit mechanism in the L2 cache (w/ pop) and the specialized SLS kernel (w/ SLSkernel) on the overall speedup. Note that the flexible replacement granularity, prefetching mechanism, popularity bit and specialized SLS kernel hardly influence each other. The baselines for the four workloads are the naïve EMS-I only with suitable replacement granularity and the correlation-based prefetching mechanism (naïve EMS-I-V/BP). As illustrated in Figure 14(a), with popularity bit, up to 1.27× speedup can be achieved because the popular embedding vectors, which are frequently accessed, tend to stay in the L2 cache instead of being easily evicted. Hence, the accesses to the SSD are correspondingly reduced. The specialized SLS kernel design can help improve the performance by up to 1.31× because we set the URAM cascade size to 1 to support the three-level parallelism so that we can maximally exploit the bandwidth of the on-chip URAMs. Combining the popularity bit mechanism and the specialized SLS kernel, EMS-I achieves the best performance by up to 1.68× over the baseline.

5.3.7 Overhead Analysis. Figure 14(b) shows the execution time breakdown of EMS-I on the four workloads. Overall, the SSD access including the prefetching execution, and HBM access account for around 60% of the total execution time. Running fb_dlrm takes the largest portion (74%) of the total execution time to access SSD because fb_dlrm shows the most irregular and sparse memory access pattern. The SLS kernel and MLP logic averagely takes around 21% and 8.5% of the total execution time, respectively. The execution time of L2 cache management and other parts like the coordination CXL account for less than 11%.

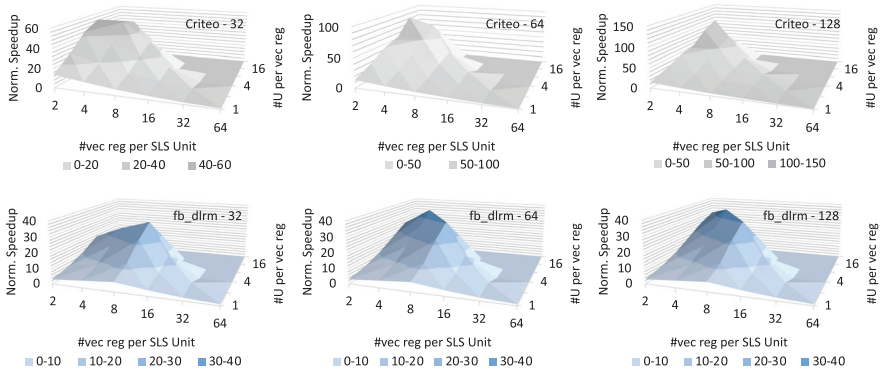


Fig. 15. The speedup of SLS array normalized to the performance of array that is configured to 1 URAM block per vector register and 64 vector registers per SLS Unit with different dimensions of the embedding vector on Criteo and fb_dlrm.

5.3.8 Design Space Discussion. From Section 4.6, we know that the three-level parallelism of SLS operation is affected by the mapping scheme for L1 cache according to the specific hardware setting of the SLS array and the characteristics of the embedding vector. In Figure 15, we show the performance of the SLS array with different hardware configurations and different dimensions of the embedding vector. The total hardware budget of the design space is 448 URAM blocks, 448 adders that can be grouped flexibly, 3.5 KB vector registers and the corresponding required psum/SLS buffer. To be specific, the design space of the SLS array consists of the number of vector registers in each SLS Unit (#vec reg per SLS Unit), the number of URAM blocks that are allocated to one vector register (#U per vec reg) and the dimension of the embedding vector. We define the point that makes the SLS array achieve the optimal performance on the (#vec reg per SLS Unit, #U per vec reg) space as the "optimal point". From the Figure 15, we can firstly observe that the "optimal point" is on the left of the (#vec reg per SLS Unit, #U per vec reg) space with different batch sizes on both of the workloads. This is because the most of the pooling factors in the two workloads are smaller than 16 so that the workloads do not need a high intra-table parallelism. Hence, when #vec reg per SLS Unit is larger than 16, around half of the URAM blocks and groups of the adders in one SLS Unit are idle during the runtime. Specifically, the value of #vec reg per SLS Unit of the "optimal point" is 2 on Criteo but is 4 or 8 on fb_dlrm beacuse the average pooling factor in fb_dlrm is larger than that in Criteo. We also observe that the "optimal point" shifts to the top left of the (#vec reg per SLS Unit, #U per vec reg) space as the dimension of the embedding vector increases. This is because as the dimension increases, the workload requires higher intra-vector parallelism, i.e., higher #U per vec reg. To be specific, if #U per vec reg is set to 1, then it takes 64 rows in a URAM block to store a 128-dimensional embedding vector so that moving the whole embedding vector from the URAM block to the vector register requires 64 reads. In this scenario, allocating more URAM blocks to a vector register with larger bitwidth to support the parallel data loading is more beneficial to the performance of the SLS array. This is because increasing #U per vec reg secures a high resource utilization with a fixed hardware budget considering the dynamic pooling factors and workload imbalance among the embedding tables. Hence, *we can give the insight that when the dimension of the embedding vector is large, the priority of designing the SLS array should be given to increase the intra-vector parallelism.* Taking all the six "optimal points" in Figure 15 into account, the specific configuration of the SLS array is 8 URAM blocks per vector register and 4 vector registers per SLS Unit as described in Section 4.6. Thanks to the reconfigurability of FPGA, the configuration can be flexibly adjusted in the design space if the SLS array is applied to other different DLRM workloads.

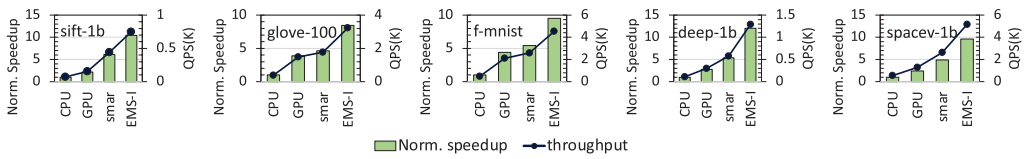


Fig. 16. Normalized speedup and throughput of EMS-I on 5 ANNS workloads.

5.3.9 Evaluation on Another Application - Approximate Nearest Neighbor Search. Besides recommendation system, EMS-I can also be applied to other applications which also have dynamic and irregular memory access pattern. For example, keeping the same memory system and basic hardware platform, we modify the EMS-I to adapt to the application of approximate nearest neighbor search (ANNS) [3], which aims to dynamically find the approximate nearest neighbors for a given query. ANNS has been a key retrieval technique for vector database and many data center applications, such as person re-identification. We select a popular graph-traversal-based ANNS algorithm, HNSW [23], and evaluate the performance of EMS-I by running the algorithm on 5 typical ANNS datasets - sift-1b [29], glove-100 [27], f-mnist [38], deep-1b [4] and spacev-1b [31]. We compare EMS-I with CPU, GPU and SmartSSD [17]. As shown in Figure 16, EMS-I can achieve up to 12.01×, 4.33× and 2.24× speedup over CPU, GPU and SmartSSD, respectively.

6 RELATED WORKS

To reduce the query latency and improve the throughput of DLRM, there have been many customized DLRM accelerators [12, 15, 16, 19, 36, 37]. Most of the prior works adopt the NDP approach. TensorDIMM [19] firstly employs a DIMM-based NDP unit in a disaggregated GPU memory system to overcome the memory bandwidth bottlenecks of the embedding operation. Unlike TensorDIMM which assumes a GPU-centric system for inference, Centaur [12] focuses on CPU-centric systems and proposes a chiplet based CPU+FPGA solution. RecNMP [15] proposes a lightweight DDR4-compatible near-memory architecture with customized NDP instructions to accelerate the memory-bound *SLS* operations. And then it is implemented in a versatile FPGA-enabled NDP platform called AxDIMM [16]. To reduce the high infrastructure costs of the large and fast DRAM based memories, RecSSD [37] utilize Flash based SSDs with larger capacity for industry-scale recommendation. RecSSD is implemented in FTL firmware and compatible with existing NVMe protocols, requiring no hardware changes. These accelerators focus on the inference task and utilize the knowledge like the distribution of the embedding table entries to address the bottleneck.

7 CONCLUSION

In this work, we obtain the significant insights into designing the efficient memory system and caching mechanism for various recommendation system workloads by performing the comprehensive characteristic studies. We found that the platform with the combination of SSD and DRAM/HBM achieves the sweet point between the memory cost and the performance when processing the DLRM workloads. In this scenario, we also observed that the cache hit rate is sensitive to the cache capacity and the replacement granularity but insensitive to the associativity. Guided by these insights, we propose EMS-I, an efficient memory system including the various memory resources and multi-level caches. EMS-I is customized for the DLRM inference with the delicately designed inference kernel and specialized caching and prefetching mechanisms. Overall, EMS-I outperforms the SOTA NDP solutions in execution latency, energy savings, and memory cost.

REFERENCES

[1] Amazon Personalize 2023. https://aws.amazon.com/personalize/

[2] Ehsan K. Ardestani et al. 2022. Supporting massive DLRM inference through software defined memory. In *ICDCS*. IEEE.

[3] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)* 45, 6 (1998), 891–923.

[4] Artem Babenko and Victor Lempitsky. 2016. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2055–2063.

[5] Keshav Balasubramanian, Abdulla Alshabanah, Joshua D. Choe, and Murali Annavaram. 2021. cDLRM: Look ahead caching for scalable training of recommendation models. In *Proceedings of the 15th ACM Conference on Recommender Systems*. 263–272.

[6] Criteo Kaggle Dataset 2020. https://www.kaggle.com/datasets/mrkmakr/criteo-dataset

[7] CXL 3.0 Specification 2022. https://www.computeexpresslink.org/download-the-specification/

[8] DRAM Market Price 2023. https://electronics-sourcing.com/2022/05/12/dram-price-increases-will-ease/

[9] Facebook DLRM Dataset 2021. https://github.com/facebookresearch/dlrm_datasets

[10] Udit Gupta et al. 2020. DeepRecSys: A system for optimizing end-to-end at-scale neural recommendation inference. In *ISCA*.

[11] HBM Market Price 2023. https://www.networkworld.com/article/3664088/high-bandwidth-memory-hdm-delivers-impressive-performance-gains.html

[12] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. 2020. Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*. IEEE, 968–981.

[13] Myoungsoo Jung et al. 2017. SimpleSSD: Modeling solid state drives for holistic system simulation. *IEEE Computer Architecture Letters* (2017).

[14] Kaggle 2023. https://www.kaggle.com

[15] Liu Ke et al. 2020. Recnmp: Accelerating personalized recommendation with near-memory processing. In *ISCA*.

[16] Liu Ke, Xuan Zhang, Jinin So, Jong-Geon Lee, Shin-Haeng Kang, Sukhan Lee, Songyi Han, YeonGon Cho, Jin Hyun Kim, Yongsuk Kwon, et al. 2021. Near-memory processing in action: Accelerating personalized recommendation with axdimm. *IEEE Micro* 42, 1 (2021), 116–127.

[17] Ji-Hoon Kim, Yeo-Reum Park, Jaeyoung Do, Soo-Young Ji, and Joo-Young Kim. 2022. Accelerating large-scale graph-based nearest neighbor search on a computational storage platform. *IEEE Trans. Comput.* (2022), 1–1. https://doi.org/10.1109/TC.2022.3155956

[18] Yoongu Kim et al. 2015. Ramulator: A fast and extensible DRAM simulator. *IEEE Computer Architecture Letters* (2015).

[19] 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*. 740–753.

[20] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2021. Tensor casting: Co-designing algorithm-architecture for personalized recommendation training. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*. IEEE, 235–248.

[21] Huaicheng Li et al. 2022. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. https://doi.org/10.48550/ARXIV.2203.00241

[22] Jason Lowe-Power et al. 2020. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152* (2020).

[23] Yu A. Malkov and Dmitry A. Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2018), 824–836.

[24] Meta 2023. https://about.meta.com

[25] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, et al. 2021. High-performance, distributed training of large-scale deep learning recommendation models. *arXiv preprint arXiv:2104.05158* (2021).

[26] Maxim Naumov et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv* (2019).

[27] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*. 1532–1543.

[28] PM983 Product Brief 2018. https://www.samsung.com/semiconductor/global.semi.static/

[29] sift-1b 2022. http://corpus-texmex.irisa.fr/

[30] Mohammadreza Soltaniyeh et al. 2022. Near-storage processing for solid state drive based recommendation inference with SmartSSDs®. In *ICPE*.

[31] spacev-1b 2021. https://github.com/microsoft/SPTAG/tree/main/datasets/SPACEV1B
[32] SSD Market Price 2023. https://www.disctech.com/Samsung-PM1725B-3.2TB-MZ-PLL3T2C-MZPLK1T6HCHP-00005-Dell-73KJ7-PCIe-NVMe-SSD?partner=1011&gclid=CjwKCAiAzp6eBhByEiwA_gGq5BswRyE1M-T6X7Gjbw9dlC_GAWnrc0kRwddyzN9IQ6mbkMA3mfSvpxoCmvEQAvD_BwE
[33] 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 *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA'22)*. IEEE, 1056–1070.
[34] torchrec 2022. https://pytorch.org/torchrec/
[35] Frank Edward Walter et al. 2008. A model of a trust-based recommendation system on a social network. *AAMAS* (2008).
[36] Yitu Wang, Zhenhua Zhu, Fan Chen, Mingyuan Ma, Guohao Dai, Yu Wang, Hai Li, and Yiran Chen. 2021. REREC: In-ReRAM acceleration with access-aware mapping for personalized recommendation. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD'21)*. IEEE, 1–9.
[37] Mark Wilkening, Gupta, et al. 2021. RecSSD: Near data processing for solid state drive based recommendation inference. In *ASPLOS*.
[38] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).
[39] Xilinx VU57P HBM 2023. https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus-vu57p.html
[40] Xiangmin Zhou et al. 2015. Online video recommendation in sharing community. In *ICMD*.

Received 23 March 2023; revised 2 June 2023; accepted 13 July 2023