HERO: Local HBM Enhancements to Support Remote Memory Optimizations

ABSTRACT

Contemporary AI compute nodes integrate CPUs, GPUs, and distinct memories. While CPU DDR offers high capacity at low bandwidth, GPU HBM provides high bandwidth but limited capacity. GPUs must use a low-bandwidth, high-latency interface like PCIe to access CPU memory, while access to peer GPU HBM utilizes higher bandwidth interconnects such as NVLink. Consequently, to scale memory, applications distribute data across multiple GPUs, sharing via mechanisms such as (1) CPU-initiated, programmer-managed message passing through memory copies, (2) on-demand, shared memory style cache-line level peer-to-peer (P2P) memory accesses, and (3) on-demand, unified virtual memory page faults. Among these, shared memory P2P provides the best trade-off between programmability and data movement overhead, but naïvely adapting modern workloads to P2P accesses yields limited memory-level parallelism and underutilized NVLink bandwidth.

HERO introduces an *Optimizing Local Buffer (OLB)* in HBM, enabling GPU cores to issue many remote loads without consuming on-chip resources that limit occupancy. However, concurrent accesses to the same remote address can overwhelm on-chip MSHRs, causing stalls and redundant NVLink traffic. To address this, HERO proposes *Extended Inter-GPU Merging*, which leverages spare HBM capacity and bandwidth to eliminate redundant requests. Together, these mechanisms deliver 1.2×–3.4× speedup and up to 43% remote traffic reduction on GPU kernels used in deep learning recommendation models and up to 1.56× speedup on neural networks and HPC graph algorithms over a peer-to-peer baseline. Overall, HERO results in an end-to-end speedup of 1.14× to 2.43× on deep learning recommendation model configurations representative of production workloads.

KEYWORDS

Multi-GPU systems, Unified Virtual Addressing, Accel-Sim

1 INTRODUCTION

Multi-GPU systems are extensively deployed in modern data centers to accelerate computationally intensive workloads that are central to Artificial Intelligence (AI), High-Performance Computing (HPC), and large-scale data analytics across industry sectors [1, 2, 12, 51]. Such infrastructure primarily supports demanding AI and HPC workloads, ranging from large-scale foundation model training and inference for natural language processing, computer vision, and generative AI applications to high-performance scientific simulations, data-intensive analytics, and large-scale numerical computations. Modern online advertising systems demand massive-scale machine learning inference under strict subsecond latency, and serving hundreds of trillions of ad-ranking inferences daily places significant pressure on AI infrastructure [47]. These workloads require massively parallel computing power and high-throughput interconnects (such as NVLink [68] and InfiniBand [71]) to process enormous datasets efficiently.

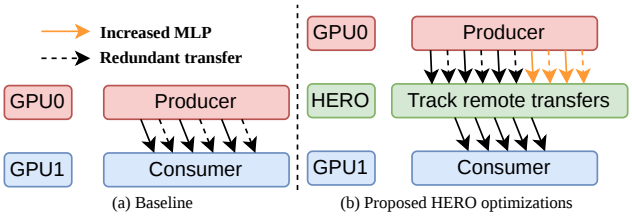


Figure 1: HERO improves upon a baseline multi-GPU system by increasing MLP and eliminating redundant inter-GPU transfers.

Data movement in GPUs can be orchestrated in several ways. Like message-passing based approaches, CPU-initiated memory copies need to be coarse-grained for maximal bandwidth efficiency and orchestrating compute-communication overlap needs expert pipelining strategies by the programmer. Unified Virtual Memory (UVM) [61] reduces this effort by enabling pointer-based remote access through page migration, but its high latency and inefficiencies make it unsuitable for production. Peer-to-peer (P2P) via Direct Cacheline Access (DCA) or Unified Virtual Addressing (UVA) [67] offers a middle ground, pinning data to GPUs and serving accesses at cacheline granularity without page migration. P2P memory accesses preserve the usual shared-memory abstraction and adhere to the existing GPU memory consistency rules, making it straightforward to scale applications from a single GPU to multi-GPU setups. P2P accesses also offer a natural strategy to overlap communication of data with compute. However, naively adapting code to adopt P2P accesses in a kernel comes with significant inefficiencies. This work targets improving P2P efficiency for contemporary workloads.

Two major factors contribute to the slowdown introduced by remote accesses inside the kernel. First, kernels often do not generate enough concurrent memory requests to cover the latency of remote accesses. In the message passing based solution, all in-kernel accesses are to the local HBM. Local accesses have ~ 10× lower latency than remote accesses, meaning the P2P solution requires ~ 10× more in-flight requests to maintain the same performance. However, the occupancy of these kernels is limited by register file usage, rendering traditional loop unrolling ineffective at increasing Memory Level Parallelism (MLP) since it will limit warp occupancy. Kernels optimized for inter-GPU accesses, such as those in communication libraries like NVIDIA Collective Communications Library (NCCL [70]) and similar frameworks [8, 55], perform reductions, scatter, and gather operations with low computational and resource demands on the core. Second, even if enough accesses can be generated concurrently, such GPU kernels generate excessive requests to the same remote cache lines. Although past work [78, 86] has proposed caching data stored on remote GPUs in the local memory hierarchy, such mechanisms are not implemented in contemporary NVIDIA hardware. Part of the reason for the lack of remote caching may be the need to maintain hardware coherence on remote data,

and the complexity and hardware overhead of inter-GPU cache coherence cannot be justified. A key goal of this work is to maximize the MLP achieved by the GPU kernel (and hence, improve the inter-GPU interconnect utilization) while at the same time minimizing the amount of inter-chip traffic in GPUs without relying on cache coherence.

Prefetching is one technique widely used to improve the latency of memory requests subject to the availability of memory bandwidth. Prefetching is only effective when the prefetch is issued far enough ahead of the demand to hide the worst-case latency (timeliness property). Data in GPUs can be prefetched into registers, shared memory, or global memory caches [29]. However, achieving large prefetch distances on GPUs is challenging as the usage of prefetch destinations, such as registers or shared memory, begins to limit occupancy. Prefetching into caches too has limitations due to cache pollution. This motivates a design to achieve large prefetch distances without hurting kernel occupancy.

Modern GPUs employ Miss Status Handling Registers (MSHRs) within their Miss Handling Architecture (MHA) to merge in-flight requests targeting the same cache line, reducing inter-GPU traffic [25, 81]. When a cache miss occurs, the MSHR tracks metadata (e.g., miss type) until data is retrieved from lower memory. A typical MHA configuration includes 32–64 entries, each with 4–8 slots. Primary misses create new entries, while secondary misses merge into slots of existing entries. This merging enables memory parallelism but limits new requests when entries/slots are exhausted, making MHA critical for efficient memory management.

Conventional memory handling structures such as the L1 and L2 MSHRs rely on data caching to reduce the need to use an MSHR entry for future access to the same cache line. However, given that responses to remote (inter-GPU) data requests are cached in L1 but not in the local L2 [18], current systems repeatedly need to bring in redundant copies of the data and waste precious inter-GPU bandwidth and energy in data movement. Prior work has shown that caching remote data in L2 results in significant performance degradation due to L2 flushes at kernel boundaries [86] and L2 interference from local traffic [37]. Contemporary workloads often also have working sets that are larger than the size of GPU L2 caches, thus rendering mere caching of remote data in L2 ineffective. Prior approaches that cache in GPU DRAM or HBM [78, 86] come with significant overheads of maintaining coherence.

Our analysis of several P2P-enabled multi-GPU workloads revealed that the number of in-flight misses for remote accesses to the same cache line can be as high as 120, which requires far more on-chip MSHR resources than contemporary systems can reasonably offer. A naive approach to this issue is adding more entries and slots to the on-chip MSHRs. This method has substantial area overhead and offers poor scaling behavior for future workloads.

To both increase MLP and reduce redundant memory requests over the inter-GPU network, we propose HERO (depicted in Figure 1). HERO introduces the concept of an *Optimizing Local Buffer (OLB)* to the local HBM, allowing GPU cores to initiate many remote loads, without consuming on-chip resources that limit occupancy. Even with an OLB, loads initiated from different cores tend to access the same address on a remote GPU very close in time. This access pattern quickly overwhelms the merge distance of on-chip MSHRs, resulting in unnecessary stalling or excessive redundant traffic over NVLink. HERO introduces *Extended Inter-GPU Merging*, which uses spare HBM capacity and bandwidth to remove redundant traffic from the NVLink connections to mitigate this issue.

The main contributions of this paper are:

- We identify the need to improve the efficiency of P2P-based multi-GPU programs and show that existing optimizations from single-GPU programs do not translate optimally to multi-GPU settings.
- We propose an Optimizing Local Buffer (OLB) that improves inter-GPU interconnect bandwidth utilization and reduces the access latency of remote memory operations.
- We propose minor architectural modifications to the GPU memory system to support infinite requests to remote GPU data without the overhead of caching or maintaining cache coherence.
- Through simulation on a 4-GPU Hopper [66] system interconnected by NVLink [68], HERO yields a speedup of 1.2×-3.4× and remote traffic reduction of up to 43% on the GPU kernels used in deep learning recommendation models using datasets reflective of real-world production memory traces and up to 1.56× speedup on neural networks and HPC graph algorithms, relative to a peer-to-peer baseline. HERO translates to an end-to-end speedup of 1.14×-2.43× on deep learning recommendation model configurations representative of production workloads.

2 BACKGROUND

We summarize the key aspects of multi-GPU architecture, communication mechanisms, and provide background on caching in contemporary multi-GPU systems.

2.1 Multi-GPU architecture

With the slowdown of Moore's law, single GPUs can no longer meet the growing compute and memory demands of modern workloads. Multi-GPU systems address this by coordinating multiple GPUs connected via high-speed fabrics such as PCIe or NVLink, with package-level integration also explored [9]. However, inter-GPU interconnects offer over 3× lower bandwidth and ∼ 10× higher latency than local HBM [53, 86], making it vital to generate sufficient MLP to cover the latency of the interconnect. Furthermore, modern multi-GPU platforms increasingly rely on Remote Direct Memory Access (RDMA) to enable direct accesses to peer GPU memory without CPU involvement, but RDMA operations still inherit the fundamental bandwidth and latency limitations of the underlying interconnect, reinforcing the need for efficient remote-access optimization.

2.2 Multi-GPU communication

Data can be migrated across GPUs via (i) programmer-directed memory copies, (ii) page migration through Unified Virtual Memory (UVM), or (iii) Direct Cacheline Access (DCA).

2.2.1 Host (CPU) initiated DMA using memory copies. APIs such as cudaMemcpy() leverage the GPU DMA engine for GPU–GPU or GPU–CPU transfers. Compute and memory transfers can be pipelined in CUDA, but pipeline parallelism strategies [27] demand

significant programmer effort and workload-specific tuning. Moreover, the communication and computation overlap is more coarse-grained. Computation cannot start on any part of the data until the full copy transfer has been completed.

2.2.2 Page Migration through UVM. Pages are transferred on demand via page faults, typically at a coarse granularity (e.g., 2MB [87]). While this improves bandwidth utilization, high page sharing between GPUs can cause ping-ponging, making UVM's programming convenience a major performance concern [6].

2.2.3 Direct Cacheline Access. Pages can be pinned to a GPU, with remote data accessed at cache-line granularity [67] via Peer-to-Peer (P2P) transfers. Unlike UVM migration, P2P avoids costly SM pipeline, cache, and TLB flushes, and caches RDMA responses in L1. However, inter-GPU latency over PCIe or NVLink remains significant. P2P has been used in prior works [10, 53, 86] and libraries such as NVSHMEM [69], DeepEP [17] and NCCL [70].

2.3 Caching in P2P-based multi-GPU systems

NVIDIA GPUs automatically flush or invalidate all per-SM L1 caches at every kernel launch, as these structures are private, non-coherent, and tied to kernel-specific execution contexts. However, the shared L2 cache is not flushed across kernel boundaries, and its contents persist unless displaced by replacement. Zhang et al. [88] show that context switches flush all L1 caches but leave L2 untouched, implying that flushing L2 at every kernel boundary would incur prohibitive writeback and refill overheads. When explicit L2-flush operations are exposed, prior work [48] finds that they are expensive enough to warrant sophisticated compiler/runtime techniques to reduce their frequency. Similarly, coherence mechanisms for CPU-GPU systems and multi-GPU caches seek to avoid bulk L2 flush and invalidation actions due to their performance impact [7].

Caching remote GPU data has emerged as an attractive optimization for multi-GPU systems, but it raises correctness challenges because current discrete GPUs provide no hardware coherence across device memories. CARVE [86] demonstrates that when a GPU caches remote memory lines in its L2 or in a Remote Data Cache (RDC), these lines must be invalidated, and any dirty remote-data must be written back to the home GPU's memory at kernel boundaries. CARVE also imposes a complication of using HBM capacity for remote data which is critical for memory capacity constrained applications. Their results show that such remote-data flushes can incur multi-millisecond delays, particularly for large RDCs, which can severely limit the practical benefit of remote-data caching. CARVE proposes write-through RDCs and epoch-based tracking to reduce the frequency and cost of these bulk flushes. In contrast, hardware-coherent designs such as HMG [78] eliminate the need for explicit remote-data flushes by providing fine-grain cross-GPU coherence, albeit with higher hardware complexity.

Owing to the complexity of maintaining coherence at the L2, remote GPU memory accesses (over the NVLink) are cached at the target GPU's L2, but not at the L2 of the GPU initiating the request, as reverse-engineered by prior work [18]. Through microbenchmarking, we find that NVIDIA GPUs resort to sending redundant requests over the inter-GPU interconnect upon saturating MSHR resources. In this work, we propose a new mechanism that exploits inter-GPU bandwidth more efficiently while avoiding the overheads of conventional cache-based coherence.

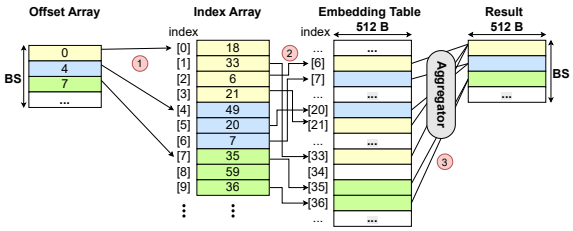


Figure 2: Illustration of the working of the embedding bag operator. BS denotes the batch size. The Offset and Index array are used to read the respective rows from the Embedding Table for each element in a batch.

3 CASE STUDY: CHARACTERIZING A P2P-BASED EMBEDDING BAG KERNEL

In this section, we go over the finer details of GPU kernels amenable to the optimizations proposed in this work. Multi-GPU systems are extensively leveraged to power recommendation systems such as DLRMs [14, 15, 24, 32, 47, 84, 89, 90], which are crucial for delivering personalized and highly relevant content to billions of users worldwide across leading platforms like Netflix [21], Meta [59], and Amazon [80]. Embedding tables dominate contemporary workloads such as Deep Learning Recommendation Models (DLRMs), accounting for over 99% of the weights [22] with industry-scale footprints in the terabyte range [49, 77]. As an illustrative example, we use the embedding bag kernel implementation in PyTorch [52].

3.1 Embedding based deep learning workloads

Figure 2 illustrates the embedding bag operator: offsets index into an array of embedding indices ①, which in turn select embedding rows from the table ②. These rows are then accumulated ③ using functions such as sum, max, or avg [52].

Several prior works have indicated the existence of frequently accessed embedding table entries [3, 46, 73]. Table 1 shows that the percentage of unique entries in a batch is as low as 3.4% as in the case of Meta high-reuse dataset [50]. This is a consequence of the power-law access patterns found in such workloads where a few unique memory addresses contribute to the majority of the accesses [4, 79].

Table 1: Unique entries (%) in each dataset for batch size 2048

Reuse level	High reuse	Medium reuse	Low reuse
Unique entries (%)	3.47	29.87	71.82

Hotness of data in modern day workloads such as DLRMs also needs immense miss handling capability. Figure 3 shows an example of a high watermark of the number of MSHR slots needed during the execution of the embedding kernel of DLRM workloads. MSHRs that are able to handle as many as 120 misses in the same cache block are necessary to prevent stalls due to reservation failures of

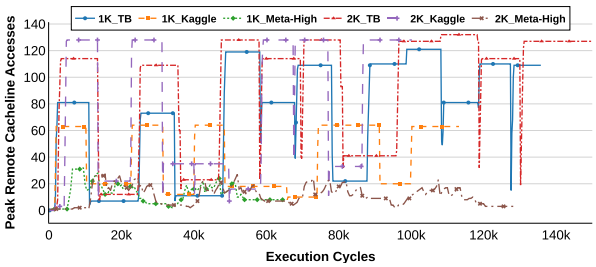


Figure 3: Peak concurrent accesses to a remote cache line during kernel execution for <Batchsize>_<Dataset>, obtained using the simulation framework in Section 5.

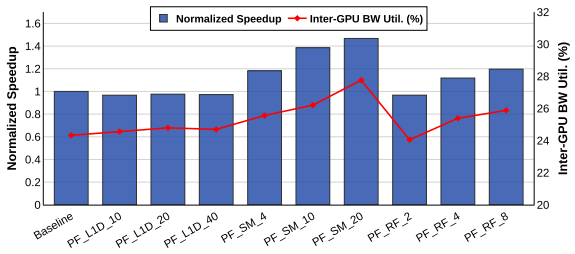


Figure 4: Baseline and Prefetch-optimized EmbeddingBag kernels for the Meta-High reuse dataset (details in Section 5). Silicon measurements were obtained from a 4-GPU Hopper system for a batch size of 2048 and embedding width of 160. The prefetch (PF) configurations are labeled as PF_<PF destination>_<PF distance>. (RF: Register File, SM: Shared Memory)

the MSHR entries. Typically, the number of slots in GPUs is far fewer than required by such workloads. Through careful design of the memory subsystem, we seek to eliminate redundant memory access requests through intelligent merging.

Algorithm 1 Simplified irregular memory access loop in the PyTorch CUDA Embedding Bag operator

```
begin = offsets[bag];
end = offsets[bag+1];
for idx in begin ... end do
    weightRow = indices[idx];
    weightValue = weightFeat[weightRow];
    weightSum += weightValue;
end for
output[bag][featureDim] = weightSum;
```

3.2 GPU Implementation of the Embedding Bag operation

The embedding lookup phase of DLRM involves several parameters and it is vital to understand how high level algorithmic parameters map to the parallelization strategy. Multiple tables on the same GPU are processed sequentially. The embedding bag operator is used to process a given table using the PyTorch CUDA backend kernel [52]. It generates an output matrix of size (BS) × (ED) where BS and ED denote the batch size and embedding dimension respectively. The default kernel uses a static launch configuration with a grid size of (1024,1,1) and a block size of (32,8,1). This results in a large number of CUDA threads and can keep all the SMs of the latest GPUs active (e.g., all 132 SMs in Hopper [66]). Compiling on a Hopper machine results in 74 registers per thread, which limits the number of threadblocks that can be assigned to an SM.

Algorithm 1 highlights the work done by a thread, which is a "number of lookups" worth of gather and reduce operations. The gather is similar to a pointer-chasing operation as we traverse a sequence of arrays to find the indices to load (e.g., the offset array, then the indices array and finally the Embedding table in figure 2). The irregular loads (highlighted in red) caused by this sequence of indirect memory accesses have a significant impact on the performance of the embedding stage in DLRMs. As the indices corresponding to these irregular loads are known well ahead of access, prefetching has been shown to be an effective optimization [29].

3.3 Inefficiencies of a P2P embedding kernel

Prior work [29] identified the *EmbeddingBag* kernel [52] as memory-latency bound on a single GPU and proposed prefetching as an optimization. We first investigate if accessing embedding table entries over the narrow and slower inter-GPU interconnect needs a different solution than previously explored.

Figure 4 shows the performance of prefetch strategies [29] for a P2P-enabled *EmbeddingBag* kernel with remote GPU accesses over NVLink [68]. Prefetching reduces latency by issuing loads early, with destinations in L1D, shared memory (SM) scratch pad, or registers (RF). Shared memory and register prefetching improve performance over a baseline off-the-shelf PyTorch [74] implementation, while L1D prefetching slightly degrades performance due to cache pollution. Despite these gains, interconnect utilization remains low (≈ 27%). Larger prefetch distances could in theory saturate interconnect bandwidth, but in practice require increased register/shared memory usage, reduce kernel occupancy, and limit achievable MLP on latency-bound kernels.

4 HERO ARCHITECTURAL PRINCIPLES

This section describes the guiding architectural principles for our proposed mechanisms. These principles serve to decouple the concepts from the specific implementation details.

There are two key inefficiencies with current systems that motivate our design. To enable long prefetch distances and efficient inter-GPU interconnect utilization, we need a mechanism to unroll a large number of remote loads without affecting occupancy. Next, the power-law memory access characteristics of workloads result in a strong demand for MSHR slots, leading to redundant accesses and wasted interconnect bandwidth. Hence, we need a scalable mechanism to extend the miss handling capability of GPUs to service remote requests. The following section explores the design space of an Optimizing Local Buffer (OLB) that can enable large unroll factors for long latency remote memory accesses without sacrificing kernel occupancy. We further describe the design of an extended inter-GPU merging structure to extend the miss handling capability of GPUs for remote read requests.

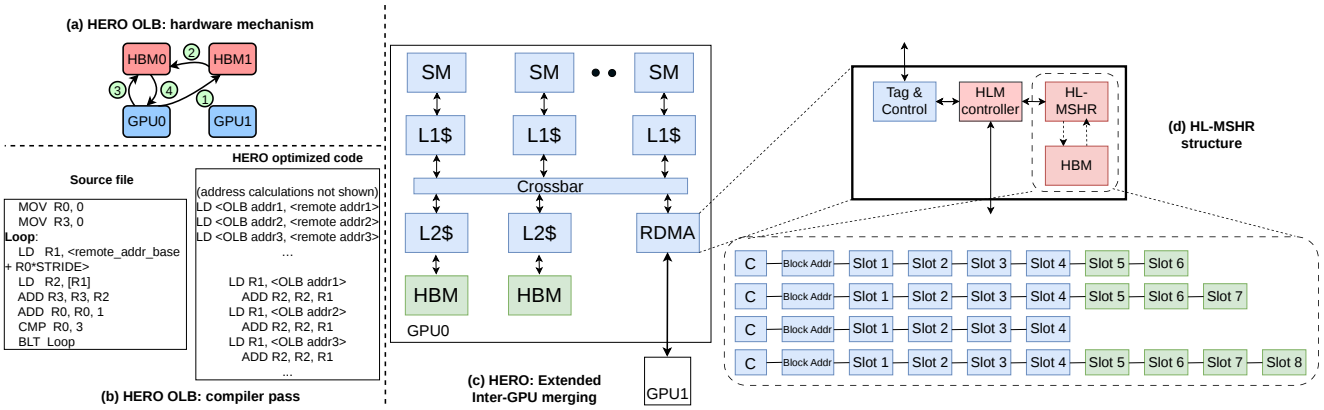


Figure 5: An illustration of the HERO-OLB hardware mechanism (a) and compiler optimizations (b). High-level view of the proposed Hierarchically Linked MSHR (HL-MSHR) in a 2-GPU system (c). HERO extends the RDMA unit (changes shown in red) (d). Also shown in (d) is a logical view of a plausible HL-MSHR state. Green denotes slots in HBM, while blue represents slots stored on-chip. The HBM inside the RDMA denotes the HBM extended MSHR slots

4.1 Optimizing Local Buffer (OLB)

NVIDIA's Ampere architecture [63] introduced the LDGSTS instruction, which is a specialized SASS (low-level assembly) instruction for asynchronous data transfer from global memory to shared memory and enables non-blocking data movement, allowing concurrent execution of compute and memory operations. NVIDIA Hopper [66] and newer architectures extended LDGSTS-based transfer using Tensor Memory Accelerator (TMA) instructions [66]. The TMA allows the asynchronous data transfer of multi-dimensional tiles of data from global memory to shared memory bypassing previous on-chip resources such as registers. This prioritizes shared memory reuse and reduces dependency on registers. Since prefetching over large prefetch distances into shared memory or registers dramatically limits occupancy and parallelism, we need a new staging area. A natural candidate to stage remote loads is the local HBM. An Optimizing Local Buffer (OLB) carved out of the local HBM capacity serves as a mechanism to temporarily stage long-latency remote reads. Upon use of the remote load value, the value is retrieved from the local HBM memory into a register. This circumvents the use of a register or shared memory to hold long latency remote accesses and does not limit occupancy when enabling large prefetch distances. The OLB in the HBM improves the MLP without the overhead of maintaining coherence in the case of an HBM cache [86].

Figure 5 (a) illustrates an example of how the OLB can be used to loop-unroll long-latency loads from a remote GPU. When GPU0 requests data from remote GPU (HBM1) ① (Figure 5(a)), the loads are temporarily staged in HBM0 ②. Upon encountering the load-use instruction, the buffered values are loaded from local memory (HBM0) ③ and the value is consumed in a future instruction ④. As shown in Figure 5(b), a HERO-compiler pass transforms the unoptimized code by unrolling the loop and temporarily staging the remote long-latency loads in the local HBM. The length of unrolling depends on the available MLP in the code and the available size of the OLB.

4.1.1 ***Discussion.*** In the following subsections, we elaborate on the implementation design choices and other tradeoffs.

Implementation. Implementing the OLB can be done in several ways. One possible mechanism could be that the compiler intelligently uses the local stack as the OLB. This optimization assumes that each thread manages its buffer independently of the others. For instance, considering the kernel launch parameters from Section 3.2 and an example EmbeddingBag pooling factor of 200, this configuration yields a buffer size of 200*4B=0.8kB per thread of local stack space dedicated to the OLB. Assuming all threadblocks are active simultaneously for the PyTorch CUDA backend kernel [52] which has 1024 threadblocks each of 32×8=256 threads, the aggregate OLB size is $1024 \times 32 \times 8 \times 0.8\text{kB} \approx 200\text{MB}$, which is a small fraction of the tens of gigabytes of HBM available on contemporary GPUs. In a general case, implementing the OLB via a compiler will be limited by the capacity of the local space and is reserved ahead of time by the compiler.

Alternatively, the OLB can also be implemented as a programmer-managed scratchpad to leverage inter-thread locality and coordinated loading of data. This introduces new programming semantics that enable the programmer to gain access to an HBM-scratchpad. However, a programmer-managed scratchpad is less user-friendly than a compiler-enabled optimization but gives more flexibility to a skilled programmer to use a larger scratchpad than is available on-chip.

When delegated to the compiler, the compiler can make intelligent decisions regarding the length of unrolling to saturate inter-GPU interconnect bandwidth. Using the OLB as a programmer-controlled scratchpad gives the programmer direct control over inter-GPU link utilization. We present a generic architecture whose underlying implementation choices can vary depending on the desired tradeoff between performance through reuse optimizations and programmability. For the purpose of modeling the OLB, we choose a compiler-driven approach and do not seek to further optimize the inter-thread reuse as with an OLB-scratchpad.

Coherence. The OLB is a temporary staging area that enables short-term staging of remote-GPU data before it is consumed by a requesting thread. The OLB does not enforce any coherence between a locally cached copy and the remote data. Any writes to a remote address are directly propagated to the target GPU and the previous locally staged copy (if any) is deemed invalid. This choice of strategy is motivated by the fact that often read-only data is only shared in P2P enabled multi-GPU applications. Hence, we do not need to maintain a full-fledged coherence protocol as done by prior work [78, 86]. We maintain the same coherence model for the OLB as current GPUs maintain for the L1s. Like the SM L1s, the OLB can be flushed at every kernel launch while flushing the L2 is expensive.

Comparison to a remote cache. While bearing some similarities to a remote cache, the OLB differs in its key goal to improve MLP of remote accesses and thus improve the interconnect utilization between GPUs. A remote cache serves no such purpose and comes with additional complexities introduced by maintaining coherence either through hardware or software solutions [86]. Our compiler-driven implementation of the OLB does not capture any temporal reuse between threads as each thread maintains a separate staging area as part of the OLB. A programmer-driven OLB could potentially capture reuse across threads thus reducing the HBM bandwidth demands of the OLB. We only evaluate a compiler-transformation based OLB to illustrate the large potential of staging remote loads in the local HBM.

4.2 Extended Inter-GPU Merging

We propose an extended inter-GPU merging mechanism, a novel approach to scale up the number of miss-handling resources. The key idea is to leverage HBM capacity to store metadata of remote misses. As the metadata required per slot is small relative to the GPU HBM capacity, remote miss-handling capability can be greatly enhanced. Figure 5(c) shows a high-level overview of the proposed architecture with respect to other components in the memory hierarchy (changes are highlighted in red). At its core is the Hierarchically Linked-MSHR (HL-MSHR), managed by a dedicated controller (the HLM controller) placed between the Tag and Control Unit (TCU) and the HL-MSHR itself. Upon receiving a memory request from a core, the TCU inserts the request into an MSHR if the request misses in the cache. With the HL-MSHR, the HLM controller intercepts the signal from the Tag & Control Unit and inserts the request into an HL-MSHR slot, which may involve further interactions with the HBM. The following subsections describe the challenges and operations of the proposed mechanism.

4.2.1 ***Hierarchically Linked MSHRs.*** While the extended inter-GPU merging mechanism can support a large number of remote misses in flight, several challenges must be overcome to realize its potential. First, one must aim to minimize the MSHR hit latency. This is to keep the latency of a remote access as low as possible. Second, access to HBM is slow compared to L2 structures; hence, care must be taken to overlap the accesses to HBM. For example, when the miss returns and the HBM holds additional slots for an entry, the delay to fetch the requester ID from HBM lies on the miss-return critical path and must be minimized. Finally, the proposed mechanism must have minimal hardware overheads.

To minimize the MSHR hit latency, we propose to have no tag comparisons in the HBM. The HBM is purely used to increase the number of slots for an MSHR entry. This is acceptable for workloads exhibiting power law access patterns where there are frequent misses to the same cacheline. Figure 5(d) depicts a possible state of the HL-MSHRs. Every slot in a row shares the same block address and a comparator (depicted as "C") and each slot holds other metadata information such as the bitmask of the data requested and a data buffer (to track write misses) along with other miss-handling information. For instance, conventional MSHRs in such a scenario would be limited to 4 entries each having 4 slots. As a result, if there are more than 3 secondary misses after a primary miss in a conventional MSHR, it would result in a reservation failure as there are no more slots to accommodate additional misses.

Our HLM controller dynamically orchestrates the allocation and deallocation of slots in the HBM. Several additional bits are needed for each MSHR entry compared to the conventional structure.

- **Overflow bit (O)**: This bit is set to 1 if the number of slots demanded spills over to the HBM memory.
- **Base address bits (B):** These bits are valid only if the overflow bit is set. This field tracks the base address of the HBM memory allocated for additional slots.
- **Number of slots bits (N):** These bits are incremented each time a slot is added. During the miss-return path, the number of slots may be used in conjunction with the base address bits to retrieve the requester ID of the miss requests.

4.2.2 ***Operations.*** In the following subsections, we describe how various operations are handled.

Handling primary misses. When a miss is detected by the TCU in the HL-MSHRs, the block address is searched with the tags of all the MSHR entries in parallel using the comparator. If no match is found (i.e., a primary miss), the HLM controller allocates a new entry for it. The overflow bit is initially set to 0 as no slots have been spilled over to the HBM. The cache block address, offset address and requester ID are recorded in the first slot. If the primary miss is a write request, the data is written into the data buffer field.

Handling secondary misses. When the TCU detects a match in the HL-MSHRs, it sends a merge signal to the corresponding MSHR entry. The HLM controller tries to merge the new miss request into one of the slots. There are two possibilities here:

(1) The Overflow bit is not set and there are available slots. In this case, the information of the miss request is recorded in the first available slot. The HLM controller sets the Overflow bit based on whether all the slots are occupied.
(2) If the Overflow bit is set, the HLM controller walks through to the next available slot by making use of the HBM base address pointer.

Deallocation of an HL-entry. An HL-entry is deallocated when the miss returns. However, it needs to first walk through all the slots. It first checks the slots that are local to the HL-MSHR and releases them. If there are other slots allocated in the backup HBM,

it sends out a read request to the addresses of the slots. Once it receives the output of the read request, it zeros out the Overflow bit. Note that each HBM read request is at 32B. Assuming each slot in the HBM takes 2B of metadata, up to 16 secondary-miss returns can be serviced with 1 HBM access.

Handling write misses. Our analysis of several P2P-enabled multi-GPU workloads reveals remote read-heavy accesses that show power law locality. Hence, we optimize our system to only spill over to the HBM in case of remote reads but not remote writes. For remote writes, the miss handling behavior falls back to a statically sized MSHR structure with a fixed number of slots.

4.2.3 ***Overhead.*** Each MSHR entry tracks a base HBM address (for spill slots), an overflow bit, and a counter for total slots. Assuming a 48-bit address, 1-bit overflow, and 10-bit counter (supporting up to 1024 outstanding misses), this adds ≈8B overhead per entry. The overhead can further be reduced by storing offsets to a global base address. Each slot tracks the requester SM ID and cache-block offset (≈2B). Write misses are not spilled to HBM; supporting them would require data buffers per slot but would still add minimal overhead relative to HBM capacity. All slots for an MSHR entry are mapped to the same HBM row, and simulations show no more than 1024 concurrent misses per cache line. Thus, we ignore the effect of modeling the activation of multiple HBM rows to store slot metadata for an MSHR entry.

4.2.4 ***Handling memory fragmentation.*** We allocate at most one HBM page per MSHR entry and try to reuse pages among MSHR entries (it is possible that many MSHR entries can spill onto the same page). With page sizes of 4kB–2MB, each page can hold thousands of remote misses, sufficient for our workloads. If an MSHR entry uses up more than a page to store miss metadata, conventional page-linking (through pointers) can be used. As the total number of pages needed is negligible relative to HBM capacity (tens of GBs), we neglect the effects of storing miss metadata in HBM memory. If allocation of a new page is not possible due to capacity limits or memory fragmentation issues, the HERO mechanism reverts to the baseline policy of limiting the number of accesses in flight.

4.3 Combined working of the mechanisms

As explained in sections 4.1 and 4.2, the OLB boosts the MLP by increasing the number of remote GPU accesses in flight while the Extended Inter-GPU Merging mechanism eliminates redundant accesses over the inter-GPU interconnect thus ensuring efficient use of the interconnect bandwidth. Thus, the mechanisms work synergistically and seek to maximize performance and interconnect bandwidth utilization.

5 METHODOLOGY

Simulator:

We modify the trace-driven, cycle-level Accel-Sim v1.1 [38] simulator to model a Multi-GPU system and use BookSim [31] for interconnect simulation to evaluate our proposed optimizations. We use the NVBit [82] tool v1.7.7 to dump traces for each GPU context and then assign kernels to different GPUs depending on the context ID. The tracer also extracts the address allocation range of each GPU, allowing us to map memory accesses in the kernel to different GPUs. Our baseline configuration for a multi-GPU system is as shown in Table 2. NVIDIA H100 [66] GPUs have also introduced an L2 Request Coalescer (LRC) that processes incoming requests for L2 and tries to coalesce read requests before forwarding them to the L2 cache [60]. Through microbenchmarking, we found that the LRC only merges local GPU requests and not remote GPU requests. Our microbenchmark also revealed that the LRC merging works at a per Graphics Processing Cluster (GPC) [66] granularity and the merge width for local requests is 4. We conservatively choose 8 slots per entry to filter out requests to a remote GPU. We characterize the inter-GPU bandwidth and latency of the H100 interconnected via NVSwitch [68] based on open-source microbenchmarks [65].

Table 2: Multi-GPU configuration

#GPUs	4
#SMs	132 SMs per GPU
SM configuration	Hopper-like SM, 64 warps, 4 subcores, 65536 registers, 256 kB (4 banks) L1 cache + Shared Memory, 1.83 GHz
L2 cache size	50 MB (80 banks)
Inter-GPU Interconnect	All-to-all topology, 368 GB/s (uni-directional), 2.5 μs effective round-trip latency
Memory BW	3 TB/s (bidirectional) per GPU
L1 MSHRs	8 slots per entry
L2 MSHRs	4 slots per entry
Remote MSHRs	8 slots per entry

Software: To build P2P-based DLRMs, we modified the PyTorch backend to replace memcpy with P2P and rewrote the code to remove explicit inter-GPU transfers during embedding lookups. Experiments use PyTorch v2.1.0 [74], source-compiled with CUDA 12.4 and driver 550.90.07.

Table 3: Summary of DLRM model configurations

Name	Tables	Bottom-MLP	Top-MLP
rm1	32	2048-2048-256-64	768-384-1
rm2_1	60	256-128-128	128-64-1
rm2_2	120	1024-512-128-128	384-192-1
rm2_3	170	2048-1024-256-128	512-256-1

Models: Following prior work [22, 24, 30, 56], we use embedding tables with 1M rows, 128 dimensions, and 4-byte precision (512 B per vector). For end-to-end evaluation of Meta models, we use the parameters listed in Table 3 (MLP: Multi-Layer Perceptron).

Datasets: We use homogenized production traces open-sourced by Meta [50], covering memory access patterns with high, medium, and low reuse (Section 3.1). As in prior work [30], we evaluate large batch sizes (1024, 2048) with a pooling factor of 150. We also test our mechanisms on the Criteo Kaggle [44] and Terabyte [45] datasets using model configurations from MLPerf [16]. Finally, we also evaluate our optimizations on Graph Neural Networks (GNNs) [5, 39] and graph algorithms such as BFS [13] across various batch sizes and graph sizes.

5.1 Evaluation

We model the bandwidth, timing, and memory access implications of our mechanisms using simulation across various cache and remote data handling configurations. As described in section 2.3,

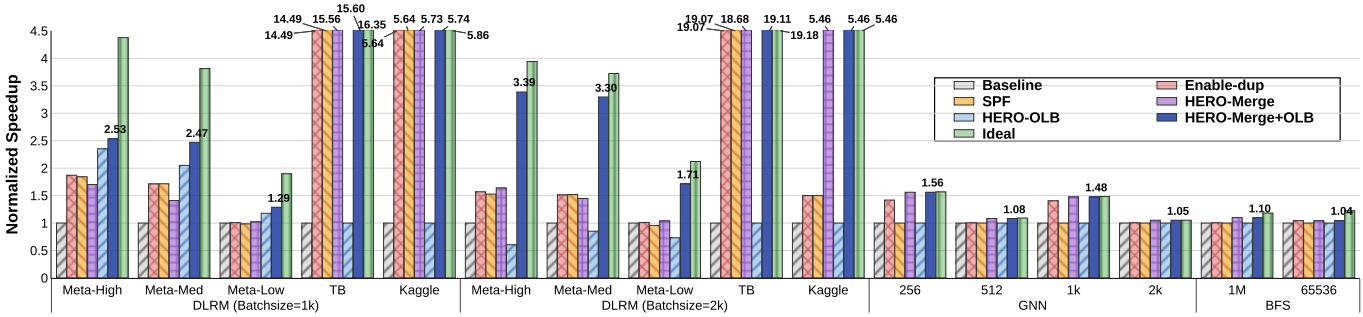


Figure 6: Normalized performance for various workload configurations and optimizations

our caching configuration implements caching of remote GPU accesses at the L1 but not at the L2. The baseline system has finite MSHRs and locks up if the number of memory requests at any point exceeds the MSHR capacity. **Enable-dup** extends the baseline by issuing redundant requests when interconnect bandwidth is available. Finally, we compare against an ideal system with infinite resources to loop unroll as many requests as permitted by the bandwidth (without affecting occupancy) and an infinitely-sized MSHR to handle remote misses denoted as **Ideal**. Our mechanisms are denoted as **HERO-Merge** (HBM extended MSHRs), **HERO-OLB** (HBM carved out as OLB), and **HERO-Merge+OLB**, which represents the cumulative benefit of the proposed mechanisms. We also compare against the best-performing prefetch baseline from prior work [29], denoted **SPF**, which prefetches long-latency loads into shared memory. Datasets follow Section 5, with reuse levels and batch/graph sizes appended where applicable. The GNN workloads are appended with batch size. Graph algorithms such as BFS are appended with the number of nodes in the graph.

5.2 Performance

Figure 6 shows the performance of our mechanisms relative to the baseline. TB datasets achieve the largest gains, underscoring the need for MSHRs that handle bursty requests. Enabling duplicates improves speedup but increases off-chip traffic. Larger batch sizes increase requests per thread in the PyTorch *EmbeddingBag* kernel [52], boosting MLP and speedup. SPF offers little benefit, as prefetch distances beyond 20 are limited by shared memory capacity and occupancy constraints. Speedup strongly correlates with reuse: high-reuse Meta datasets show the greatest improvement. Individually, HERO-Merge and HERO-OLB are bottlenecked by complementary limits, but together they synergistically raise MLP and reduce redundant requests. For example, enabling HERO-OLB increases the number of loads in flight, but these loads cannot be serviced due to limited MSHR capacity and thus face unnecessary stalls (e.g., Meta dataset workloads for a 2k batch size). At batch sizes 1024 and 2048, HERO achieves up to 2.53× and 3.39× speedup on high-reuse Meta datasets, outperforming the next best baseline (SPF) by 37% and 2.5×, respectively. Regarding workloads such as GNNs and graph algorithms such as BFS, the speedup numbers are more muted. Such workloads do not have the ability to increase MLP through unrolling (similar to the case for DLRM models with the Kaggle/TB datasets). Hence, the HERO compiler fails to make use of the OLB mechanism. We observe a speedup of up to 1.56× due to the HERO-Merge optimization for GNNs.

5.3 Remote memory traffic reduction

HERO increases in-flight requests while reducing redundant accesses via extended merging. Figure 7 shows normalized remote traffic of our optimizations relative to the baseline.

While enabling duplicates improves the speedup over the baseline, it comes with the overhead of redundant traffic (as high as 50%). Reduction in remote memory traffic greatly depends on the redundancy in memory access patterns in the dataset. For example, the Meta high reuse dataset can have at most 42% reduction in remote memory traffic in the ideal case. The Meta low-reuse dataset has little or negligible reduction in remote traffic even in the ideal case. The SPF optimization improves performance by reducing memory access latency but issues redundant requests, yielding behavior similar to the duplicate-enabled baseline. The TB and Kaggle kernels see no benefit from prefetching since each sample accesses only one embedding vector, leaving nothing to prefetch. All proposed mechanisms in HERO achieve near-optimal reductions in off-chip traffic when compared to an infinitely sized baseline. GNNs and graph workloads like BFS show limited off-chip traffic reduction mostly because of a lack of sufficient MLP. However, this still results in a non-negligible speedup as the HERO mechanism results in fewer stalls on long-latency memory loads compared to the baseline.

5.4 Memory bandwidth utilization

Our characterization in Section 3 identified low inter-GPU utilization as a key bottleneck for P2P embedding kernels. Figure 8 shows the improvement in remote memory bandwidth for the baseline and the proposed mechanism for a batchsize of 2048. Although the Hopper datasheet specifies 450 GB/s of unidirectional inter-GPU bandwidth, our microbenchmarks measure a sustained peak of 368 GB/s in silicon, which we adopt as the reference peak. The baseline kernel only achieves 100-144 GB/s of remote bandwidth, while HERO achieves close to the peak bandwidth and near an ideal baseline. Optimizations such as Enable-dup and SPF improve remote bandwidth utilization but remain far below peak. HERO also increases local bandwidth by 3.5×, though still well below the peak local bandwidth (3 TB/s).

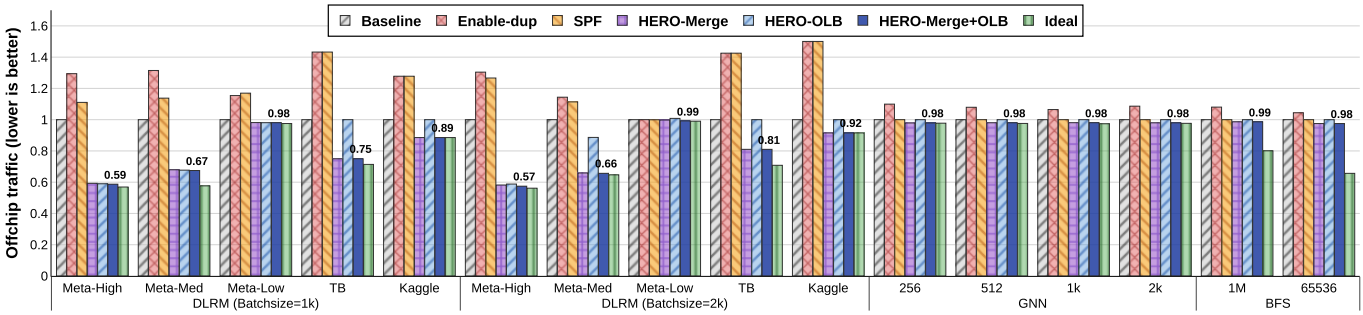


Figure 7: Normalized remote memory traffic for workload configurations and optimizations

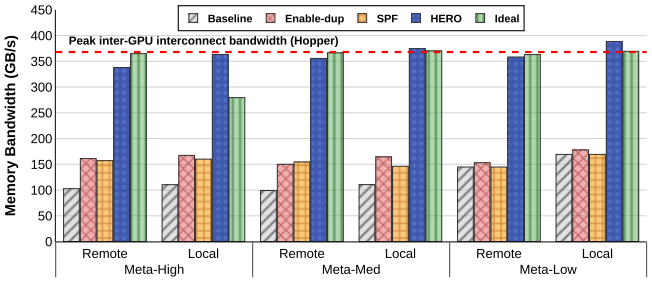


Figure 8: Remote and local memory bandwidths achieved for various mechanisms (Batchsize=2048)

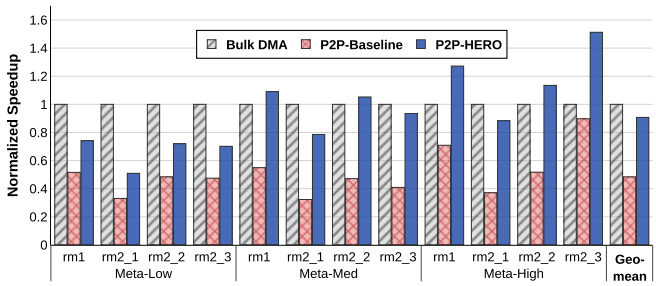


Figure 9: Projected end-to-end speedup (Batchsize=2048)

5.5 End-to-end speedup

Since cycle-accurate end-to-end simulation of ML models is often impractical, we project speedups using a mix of simulation and silicon results, as in prior work [54, 75, 76]. Figure 9 shows projected inference speedups on DLRM models from Table 3, normalized to a bulk DMA baseline. We observe a slowdown of up to 3× for a naive P2P baseline relative to bulk DMA. This slowdown in P2P stems from the inability to saturate inter-GPU interconnect bandwidth and the pressure on MSHRs to serve bursty redundant requests. As the embedding bag speedups increase with data reuse (Figure 6), HERO achieves up to 1.51× speedup on the Meta-High dataset over a memcpy baseline. At lower data reuse levels, we see a slowdown in performance for HERO when compared to the memcpy solution due to insufficient speedup from the mechanisms. Overall, HERO improves the end-to-end speedup by 1.14× to 2.43× over a baseline P2P design.

5.6 Comparison with prior works

We further compare our proposed mechanism against a baseline that implements idealized caching for remote requests at the L2. We call this baseline CARVE* following related work [86]. We duplicate L2 slices for remote requests and thus do not suffer from contention between local and remote requests. Real systems with a unified cache for local and remote requests will face some contention between local and remote traffic. Figure 10 depicts the performance of CARVE* along with other baselines and the proposed mechanism. HERO is able to increase the MLP and reduce redundant accesses through intelligent merging, while remote caching on its own does not increase MLP. Thus, HERO consistently beats CARVE*. HERO-Merge only has the extended merge optimization and shows slightly inferior performance compared to CARVE* for the Meta dataset, where the high watermark (Figure 3) is relatively small. On datasets with a higher watermark such as Kaggle and TB, HERO-Merge is able to outperform CARVE*.

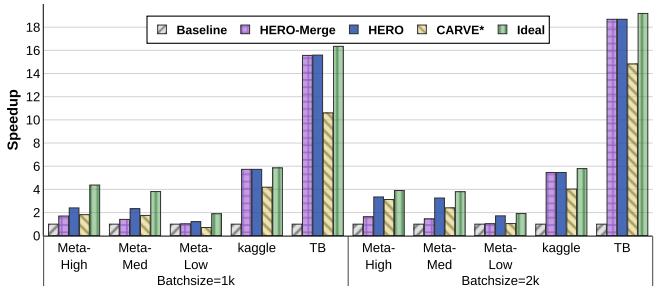


Figure 10: Comparison against prior work CARVE* [86]

5.7 Remote MSHR sizing ablation study

We sweep the number of MSHR slots supported before spilling to HBM in the extended-MSHR mechanism (Figure 11). For example, Merge-4 allows up to four requests per cacheline before allocating additional slots in HBM. Increasing MSHR slots reduces runtime, especially for high-reuse workloads (e.g., Meta High), demonstrating

how HBM provides effectively *infinite* MSHR capacity and sustains performance without large on-chip resources.

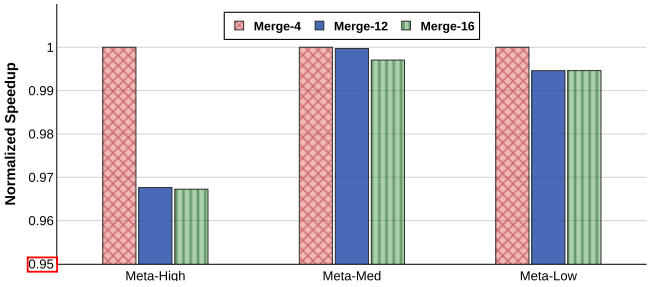


Figure 11: Ablation study on the number of MSHR slots for the HBM extended MSHR (Batchsize=2048)

6 DISCUSSION

Applicability to future GPU generations. The number of Streaming Multiprocessors (SMs) has scaled steadily, with this work assuming 132 SMs per GPU (Hopper-like). Newer architectures such as Blackwell [64] feature 180 SMs, further increasing remote request pressure and benefiting from the HERO-Merge mechanism. Interconnect bandwidth has also scaled steadily; for instance, Blackwell GPUs provide 900 GB/s unidirectional bandwidth, more than double that in Table 2, allowing HERO to unroll about twice as many remote loads in Section 5. More generally, HBM-OLB adapts to different interconnect bandwidths, provided sufficient MLP exists.

Opportunities for GPU kernel fusion. Off-the-shelf PyTorch embedding kernels [52] assume single-table operation with all tables resident in local HBM, scattering results to destination GPUs via cudaMemcpy. These transfers are serialized within a stream, underutilizing interconnect bandwidth. A P2P-based lookup can fuse multiple table accesses into one kernel, improving interconnect utilization. Further, techniques such as OLB can increase memory-level parallelism and better saturate the inter-GPU bandwidth. A P2P-enabled multi-GPU paradigm thus broadens the design space for optimal performance, though its full evaluation is left for future work.

7 RELATED WORK

In this section we highlight the key optimizations explored in related work.

GPU optimizations. Starting with Ampere, NVIDIA GPUs support L2 residency control [62], allowing up to 75% of L2 to be reserved for persistent data. Our micro-benchmarks indicate this applies only to local GPU data and not remote GPU data, likely to avoid inter-GPU coherence overheads. Thus, while effective for caching local embedding entries [29], it cannot be directly applied to remote GPU accesses.

Multi-GPU performance. PROACT [58] combines compile-time and runtime mechanisms to proactively orchestrate shared data in multi-GPU systems, while Finepack [57] transparently improves small-access efficiency and scaling. Other works [9–11, 41, 53, 83, 86] propose software and hardware techniques to enhance multi-GPU performance. For example, Griffin [10] optimizes page migration, and additional studies [40, 77] exploit common data patterns and compression to reduce memory bandwidth and improve performance. Oasis [83] is an object-aware page management scheme for multi-GPU systems that uses logical-object information rather than individual page-level decisions to drive placement and migration, reducing NUMA penalties on inter-GPU memory accesses. CARVE [86] caches remote lines in an HBM-resident Remote Data Cache, but software-managed invalidation and writeback at kernel boundaries impose substantial overhead. Hardware-coherent proposals such as HMG [78] extend coherence across multi-GPU L2 slices to enable remote-data caching without coarse-grain flushes, while also addressing increased contention and directory pressure. NetCrafter [19] improves multi-GPU system performance by reducing and prioritizing network traffic in hierarchical GPU interconnects to alleviate low-bandwidth bottlenecks.

Optimizations for DLRMs. Prior work has explored efficient scheduling and heterogeneous platforms for DLRM inference [22, 26, 35], with Lui et al. [49] addressing distributed system design. Other studies [3, 4, 43, 85] target skewed memory access patterns and propose optimizations for GPU memory use, but still rely on manual memory orchestration without examining multi-GPU microarchitectural implications. Custom accelerators for embedding and MLP stages [20, 23, 28, 33, 34, 36, 42, 46, 72] offer high speedups and efficiency but are hard to adopt commercially due to rapidly evolving models. In contrast, GPUs are already widely deployed [51], and our optimizations can directly benefit next-generation multi-GPU systems.

8 CONCLUSION

Contemporary workloads continue to scale in compute and memory demands, making GPUs the de facto deployment platform. Achieving performance at scale, however, requires manual data orchestration. Peer-to-Peer (P2P) offers a programmer-friendly abstraction albeit at significant performance costs. Prefetching remote accesses offers limited speedup and interconnect bandwidth utilization in a multi-GPU system and is bound by the reduced kernel occupancy when prefetching at large distances. To address this, we propose the Optimizing Local Buffer (OLB), which enables GPU cores to issue a large number of remote loads and increase MLP without sacrificing occupancy. Since skewed memory access patterns overload GPU miss-handling resources and waste interconnect bandwidth, we extend the miss-handling capacity for remote requests. Together, these mechanisms deliver 1.2×–3.4× speedup and up to 43% remote traffic reduction on GPU kernels used in deep learning recommendation models using production-like traces and up to 1.56× speedup on graph neural networks and HPC graph algorithms. Overall, HERO results in an end-to-end speedup ranging from 1.14× to 2.43× on deep learning recommendation model configurations.

REFERENCES

[1] 2025. Meta and Microsoft Data Center Investment Race for 2025. *Tom's Hardware* (2025). https://www.tomshardware.com/news/meta-build-2gw-datacenter Accessed: 2025-11-07.

[2] 2025. OCP 2025 Meta keynote: Scaling the AI Infrastructure to Data Center Scale. In *OCP Global Summit*. https://techblog.comsoc.org/ocp-2025-meta-keynote

Accessed: 2025-11-07.

[3] Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant J Nair. 2021. Accelerating recommendation system training by leveraging popular choices. *arXiv preprint arXiv::2103.00686* (2021).

[4] Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant J. Nair. 2021. Accelerating recommendation system training by leveraging popular choices. *Proc. VLDB Endow.* 15, 1 (Sept. 2021), 127–140. https://doi.org/10.14778/3485450.3485462

[5] Alibaba. [n. d.]. GraphLearn-for-PyTorch(GLT). https://github.com/alibaba/graphlearn-for-pytorch

[6] Tyler Allen and Rong Ge. 2021. In-Depth Analyses of Unified Virtual Memory System for GPU Accelerated Computing. In *Proceedings of the 2021 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)* (St. Louis, MO, USA). ACM, 13. https://doi.org/10.1145/3458817.3480855 Focuses on root-causes of overhead in NVIDIA UVM when used in HPC workloads..

[7] Johnathan Alsop, Xianwei Zhang, Tsung Tai Yeh, Bradford M. Beckmann, Matthew D. Sinclair, Srikant Bharadwaj, Alexandru Dutu, Anthony Gutierrez, Onur Kayiran, Michael LeBeane, Brandon Potter, and Sooraj Puthoor. 2019. Optimizing GPU Cache Policies for MI Workloads. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 243–248. https://doi.org/10.1109/IISWC47752.2019.9041977

[8] AMD. 2025. ROCm Communication Collectives Library (RCCL). https://github.com/ROCm/rccl

[9] Akhil Arunkumar, Evgeny Bolotin, Benjamin Cho, Ugljesa Milic, Eiman Ebrahimi, Oreste Villa, Aamer Jaleel, Carole-Jean Wu, and David Nellans. 2017. MCM-GPU: Multi-chip-module GPUs for continued performance scalability. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 320–332. https://doi.org/10.1145/3079856.3080231

[10] Trinayan Baruah, Yifan Sun, Ali Tolga Dinçer, Saiful A. Mojumder, José L. Abellán, Yash Ukidave, Ajay Joshi, Norman Rubin, John Kim, and David Kaeli. 2020. Griffin: Hardware-Software Support for Efficient Page Migration in Multi-GPU Systems. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 596–609. https://doi.org/10.1109/HPCA47549.2020.00055

[11] Tal Ben-Nun, Michael Sutton, Sreepathi Pai, and Keshav Pingali. 2020. Groute: Asynchronous Multi-GPU Programming Model with Applications to Large-scale Graph Processing. *ACM Trans. Parallel Comput.* 7, 3, Article 18 (June 2020), 27 pages. https://doi.org/10.1145/3399730

[12] Meta Engineering Blog. 2025. Meta's Infrastructure Evolution and the Advent of AI. https://engineering.fb.com/ Accessed: 2025-11-07.

[13] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*. 44–54. https://doi.org/10.1109/IISWC.2009.5306797

[14] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems* (Boston, MA, USA) *(DLRS 2016)*. Association for Computing Machinery, New York, NY, USA, 7–10. https://doi.org/10.1145/2988450.2988454

[15] Joel Coburn, Chunqiang Tang, Sameer Abu Asal, Neeraj Agrawal, Raviteja Chinta, Harish Dixit, Brian Dodds, Saritha Dwarakapuram, Amin Firoozshahian, Cao Gao, Kaustubh Gondkar, Tyler Graf, Junhan Hu, Jian Huang, Sterling Hughes, Adam Hutchin, Bhasker Jakka, Guoqiang Jerry Chen, Indu Kalyanaraman, Ashwin Kamath, Pankaj Kansal, Erum Kazi, Roman Levenstein, Mahesh Maddury, Alex Mastro, Siji Medaiyese, Pritesh Modi, Jack Montgomery, Satish Nadathur, Amit Nagpal, Ashwin Narasimha, Maxim Naumov, Eleanor Ozer, Jongsoo Park, Poorvaja Ramani, Harikrishna Reddy, David Reiss, Deboleena Roy, Sathish Sekar, Arushi Sharma, Pavan Shetty, Aravind Sukumaran-Rajam, Eran Tal, Mike Tsai, Shreya Varshini, Richard Wareing, Olivia Wu, Xiaolong Xie, Jinghan Yang, Hangchen Yu, Tanmay Zargar, Zitong Zeng, Feixiong Zhang, Ajit Matthews, Xun Jiao, Jiyuan Zhang, Emmanuel Menage, Truls Edvard Stokke, and Mohammed Sourouri. 2025. Meta's Second Generation AI Chip: Model-Chip Co-Design and Productionization Experiences. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 1689–1702. https://doi.org/10.1145/3695053.3731409

[16] ML Commons. 2025. MLPerf Inference. ML Commons Blog. https://github.com/mlcommons/inference

[17] Deepseek. 2024. DeepEP. https://github.com/deepseek-ai/DeepEP

[18] Sankha Baran Dutta, Hoda Naghibijouybari, Arjun Gupta, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. 2022. Spy in the GPU-box: Covert and Side Channel Attacks on Multi-GPU Systems. arXiv:2203.15981 [cs.CR]

[19] Amel Fatima, Yang Yang, Yifan Sun, Rachata Ausavarungnirun, and Adwait Jog. 2025. NetCrafter: Tailoring Network Traffic for Non-Uniform Bandwidth Multi-GPU Systems. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 1064–1078. https://doi.org/10.1145/3695053.3731040

[20] Amin Firoozshahian, Joel Coburn, Roman Levenstein, Rakesh Nattoji, Ashwin Kamath, Olivia Wu, Gurdeepak Grewal, Harish Aepala, Bhasker Jakka, Bob Dreyer, Adam Hutchin, Utku Diril, Krishnakumar Nair, Ehsan K. Aredestani, Martin Schatz, Yuchen Hao, Rakesh Komuravelli, Kunming Ho, Sameer Abu Asal, Joe Shajrawi, Kevin Quinn, Nagesh Sreedhara, Pankaj Kansal, Willie Wei, Dheepak Jayaraman, Linda Cheng, Pritam Chopda, Eric Wang, Ajay Bikumandla, Arun Karthik Sengottuvel, Krishna Thottempudi, Ashwin Narasimha, Brian Dodds, Cao Gao, Jiyuan Zhang, Mohammed Al-Sanabani, Ana Zehtabioskuie, Jordan Fix, Hangchen Yu, Richard Li, Kaustubh Gondkar, Jack Montgomery, Mike Tsai, Saritha Dwarakapuram, Sanjay Desai, Nili Avidan, Poorvaja Ramani, Karthik Narayanan, Ajit Mathews, Sethu Gopal, Maxim Naumov, Vijay Rao, Krishna Noru, Harikrishna Reddy, Prahlad Venkatapuram, and Alexis Bjorlin. 2023. MTIA: First Generation Silicon Targeting Meta's Recommendation Systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 80, 13 pages. https://doi.org/10.1145/3579371.3589348

[21] Carlos A. Gomez-Uribe and Neil Hunt. 2016. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 6, 4, Article 13 (dec 2016), 19 pages. https://doi.org/10.1145/2843948

[22] 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 ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 982–995. https://doi.org/10.1109/ISCA45697.2020.00084

[23] 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 *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 870–884. https://doi.org/10.1145/3466752.3480127

[24] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, 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. arXiv:1906.03109 [cs.DC]

[25] Sunpyo Hong and Hyesoon Kim. 2009. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. *ACM SIGARCH Computer Architecture News* 37, 152–163. https://doi.org/10.1145/1555754.1555775

[26] Samuel Hsia, Udit Gupta, Mark Wilkening, Carole-Jean Wu, Gu-Yeon Wei, and David Brooks. 2020. Cross-Stack Workload Characterization of Deep Recommendation Systems . In *2020 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE Computer Society, Los Alamitos, CA, USA, 157–168. https://doi.org/10.1109/IISWC50251.2020.00024

[27] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. *GPipe: efficient training of giant neural networks using pipeline parallelism*. Curran Associates Inc., Red Hook, NY, USA.

[28] 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 ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 968–981. https://doi.org/10.1109/ISCA45697.2020.00083

[29] Rishabh Jain, Vivek M. Bhasi, Adwait Jog, Anand Sivasubramaniam, Mahmut T. Kandemir, and Chita R. Das. 2024. Pushing the Performance Envelope of DNN-based Recommendation Systems Inference on GPUs. arXiv:2410.22249 [cs.AR] https://arxiv.org/abs/2410.22249

[30] Rishabh Jain, Scott Cheng, Vishwas Kalagi, Vrushabh Sanghavi, Samvit Kaul, Meena Arunachalam, Kiwan Maeng, Adwait Jog, Anand Sivasubramaniam, Mahmut Taylan Kandemir, and Chita R. Das. 2023. Optimizing CPU Performance for Recommendation Systems At-Scale. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 77, 15 pages. https://doi.org/10.1145/3579371.3589112

[31] Nan Jiang, Daniel U. Becker, George Michelogiannakis, James Balfour, Brian Towles, D. E. Shaw, John Kim, and William J. Dally. 2013. A detailed and flexible cycle-accurate Network-on-Chip simulator. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. https://doi.org/10.1109/ISPASS.2013.6557149

[32] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve

Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. *SIGARCH Comput. Archit. News* 45, 2 (jun 2017), 1–12. https://doi.org/10.1145/3140659.3080246

[33] 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 Annual International Symposium on Computer Architecture* (Virtual Event, Spain) *(ISCA '21)*. IEEE Press, 679–691. https://doi.org/10.1109/ISCA52012.2021.00059

[34] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim 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 *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 790–803. https://doi.org/10.1109/ISCA45697.2020.00070

[35] Liu Ke, 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 *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 141–154. https://doi.org/10.1109/HPCA53966.2022.00019

[36] Liu Ke, Xuan Zhang, Jinin So, Jong-Geon Lee, Shin-Haeng Kang, Sukhan Lee, Songyi Han, YeonGon Cho, Jin Hyun Kim, Yongsuk Kwon, KyungSoo Kim, Jin Jung, Ilkwon Yun, Sung Joo Park, Hyunsun Park, Joonho 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. https://doi.org/10.1109/MM.2021.3097700

[37] Mahmoud Khairy, Vadim Nikiforov, David Nellans, and Timothy G. Rogers. 2020. Locality-Centric Data and Threadblock Management for Massive GPUs. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1022–1036. https://doi.org/10.1109/MICRO50266.2020.00086

[38] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 473–486. https://doi.org/10.1109/ISCA45697.2020.00047

[39] Arpandeep Khatua, Vikram Sharma Mailthody, Bhagyashree Taleka, Tengfei Ma, Xiang Song, and Wen-mei Hwu. 2023. IGB: Addressing The Gaps In Labeling, Features, Heterogeneity, and Size of Public Graph Datasets for Deep Learning Research. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23)*. ACM, 4284–4295. https://doi.org/10.1145/3580305.3599843

[40] Mohammad Khavari Tavana, Yifan Sun, Nicolas Bohm Agostini, and David Kaeli. 2019. Exploiting Adaptive Data Compression to Improve Performance and Energy-Efficiency of Compute Workloads in Multi-GPU Systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 664–674. https://doi.org/10.1109/IPDPS.2019.00075

[41] Hyojong Kim, Jaewoong Sim, Prasun Gera, Ramyad Hadidi, and Hyesoon Kim. 2020. Batch-Aware Unified Memory Management in GPUs for Irregular Workloads *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 1357–1370. https://doi.org/10.1145/3373376.3378529

[42] 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* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 740–753. https://doi.org/10.1145/3352460.3358284

[43] 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* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 860–873. https://doi.org/10.1145/3470496.3527386

[44] Criterio Labs. 2014. Criteo display ad challenge. https://www.kaggle.com/c/criteo-display-ad-challenge

[45] Criterio Labs. 2014. Criteo display ad challenge. https://ailab.criteo.com/download-criteo-1tb-click-logs-dataset/

[46] Yejin Lee, Seong Hoon Seo, Hyunji Choi, Hyoung Uk Sul, Soosung Kim, Jae W Lee, and Tae Jun Ham. 2021. MERCI: efficient embedding reduction on commodity hardware via sub-query memoization. In *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems (ASPLOS)*. ACM, 302–313.

[47] Gang Liao, Hongsen Qin, Ying Wang, Alicia Golden, Michael Kuchnik, Yavuz Yetim, Jia Jiunn Ang, Chunli Fu, Yihan He, Samuel Hsia, Zewei Jiang, Dianshi Li, Uladzimir Pashkevich, Varna Puvvada, Feng Shi, Matt Steiner, Ruichao Xiao, Nathan Yan, Xiayu Yu, Zhou Fang, Roman Levenstein, Kunming Ho, Haishan Zhu, Alec Hammond, Richard Li, Ajit Mathews, Kaustubh Gondkar, Abdul Zainul-Abedin, Ketan Singh, Hongtao Yu, Wenyuan Chi, Barney Huang, Sean Zhang, Noah Weller, Zach Marine, Wyatt Cook, Carole-Jean Wu, and Gaoxiang Liu. 2026. KernelEvolve: Scaling Agentic Kernel Coding for Heterogeneous AI Accelerators at Meta. arXiv:2512.23236 [cs.LG] https://arxiv.org/abs/2512.23236

[48] Zhen Lin, Mohammad A. Alshboul, Yan Solihin, and Huiyang Zhou. 2019. Exploring Memory Persistency Models for GPUs. In *Proceedings of the 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 311–323.

[49] Michael Lui, Yavuz Yetim, Özgür Özkan, Zhuoran Zhao, Shin-Yeh Tsai, Carole-Jean Wu, and Mark Hempstead. 2021. Understanding Capacity-Driven Scale-Out Neural Recommendation Inference. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 162–171. https://doi.org/10.1109/ISPASS51385.2021.00033

[50] Meta. 2024. Embedding lookup production dataset. https://github.com/facebookresearch/dlrm_datasets

[51] Meta. 2024. Meta's 2024 AI infrastructure. https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/

[52] Meta. 2024. Pytorch Embedding bag kernel. https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/native/cuda/EmbeddingBag.cu

[53] Ugljesa Milic, Oreste Villa, Evgeny Bolotin, Akhil Arunkumar, Eiman Ebrahimi, Aamer Jaleel, Alex Ramirez, and David Nellans. 2017. Beyond the Socket: NUMA-Aware GPUs. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 123–135.

[54] Diksha Moolchandani, Joyjit Kundu, Frederik Ruelens, Peter Vrancx, Timon Evenblij, and Manu Perumkunnil. 2023. AMPeD: An Analytical Model for Performance in Distributed Training of Transformers. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 306–315. https://doi.org/10.1109/ISPASS57527.2023.00037

[55] MSFT. 2025. Microsoft Collective Communication Library (MSCCL). https://github.com/Azure/msccl

[56] 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, KR 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* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 993–1011. https://doi.org/10.1145/3470496.3533727

[57] Harini Muthukrishnan, Daniel Lustig, Oreste Villa, Thomas Wenisch, and David Nellans. 2023. FinePack: Transparently Improving the Efficiency of Fine-Grained Transfers in Multi-GPU Systems. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 516–529. https://doi.org/10.1109/HPCA56546.2023.10070949

[58] Harini Muthukrishnan, David Nellans, Daniel Lustig, Jeffrey A. Fessler, and Thomas F. Wenisch. 2021. Efficient Multi-GPU Shared Memory via Automatic Optimization of Fine-Grained Transfers. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 139–152. https://doi.org/10.1109/ISCA52012.2021.00020

[59] 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). arXiv:1906.00091 http://arxiv.org/abs/1906.00091

[60] NVIDIA. [n. d.]. Nsight compute profiling guide. https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html

[61] NVIDIA. 2022. Improving GPU Memory Oversubscription Performance. NVIDIA Developer Blog. https://developer.nvidia.com/blog/improving-gpu-memory-oversubscription-performance/ Accessed: 2023-04-11.

[62] NVIDIA. 2024. CUDA C++ Programming Guide. NVIDIA Developer Blog. https://docs.nvidia.com/cuda/cuda-c-programming-guide/#device-memory-l2-access-management

[63] NVIDIA. 2024. NVIDIA Ampere whitepaper. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

[64] NVIDIA. 2024. NVIDIA Blackwell. https://www.nvidia.com/en-us/data-center/technologies/blackwell-architecture/

[65] NVIDIA. 2024. NVIDIA CUDA samples. https://github.com/NVIDIA/cuda-samples/tree/master/Samples/5_Domain_Specific/p2pBandwidthLatencyTest

[66] NVIDIA. 2024. NVIDIA Hopper whitepaper. https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

[67] NVIDIA. 2024. NVIDIA Unified Virtual Addressing (UVA). https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__UNIFIED.html

[68] NVIDIA. 2024. NVIDIA's data-center NVLink and NVLink Switch. https://www.nvidia.com/en-us/data-center/nvlink/

[69] NVIDIA. 2024. NVSHMEM. https://developer.nvidia.com/nvshmem

[70] NVIDIA. 2025. NVIDIA Collective Communications Library (NCCL). https://developer.nvidia.com/nccl

[71] NVIDIA. 2025. NVIDIA Quantum InfiniBand Networking Solutions. https://www.nvidia.com/en-us/networking/products/infiniband/

[72] 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 *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 268–281. https://doi.org/10.1145/3466752.3480080

[73] Yoon-Joo Park and Alexander Tuzhilin. 2008. The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM conference on Recommender systems*. ACM, 11–18.

[74] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703 [cs.LG]

[75] Suchita Pati, Shaizeen Aga, Mahzabeen Islam, Nuwan Jayasena, and Matthew D. Sinclair. 2023. Tale of Two Cs: Computation vs. Communication Scaling for Future Transformers on Future Hardware. In *2023 IEEE International Symposium on Workload Characterization (IISWC)*. 140–153. https://doi.org/10.1109/IISWC59245.2023.00026

[76] Suchita Pati, Shaizeen Aga, Mahzabeen Islam, Nuwan Jayasena, and Matthew D. Sinclair. 2024. T3: Transparent Tracking & Triggering for Fine-grained Overlap of Compute & Collectives. arXiv:2401.16677 [cs.AR]

[77] Sarunya Pumma and Abhinav Vishnu. 2021. Semantic-Aware Lossless Data Compression for Deep Learning Recommendation Model (DLRM). In *2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*. 1–8. https://doi.org/10.1109/MLHPC54614.2021.00006

[78] Xiaowei Ren, Daniel Lustig, Evgeny Bolotin, Aamer Jaleel, Oreste Villa, and David Nellans. 2020. HMG: Extending Cache Coherence Protocols Across Modern Hierarchical Multi-GPU Systems. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 582–595. https://doi.org/10.1109/HPCA47549.2020.00054

[79] 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* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 344–358. https://doi.org/10.1145/3503222.3507777

[80] Brent Smith and Greg Linden. 2017. Two Decades of Recommender Systems at Amazon.com. *IEEE Internet Computing* 21, 3 (2017), 12–18. https://doi.org/10.1109/MIC.2017.72

[81] James Tuck, Luis Ceze, and Josep Torrellas. 2006. Scalable Cache Miss Handling for High Memory-Level Parallelism. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. 409–422. https://doi.org/10.1109/MICRO.2006.44

[82] Oreste Villa, Mark Stephenson, David Nellans, and Stephen W. Keckler. 2019. NVBit: A Dynamic Binary Instrumentation Framework for NVIDIA GPUs. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 372–383. https://doi.org/10.1145/3352460.3358307

[83] Yueqi Wang, Bingyao Li, Mohamed Tarek Ibn Ziad, Lieven Eeckhout, Jun Yang, Aamer Jaleel, and Xulong Tang. 2025. OASIS: Object-Aware Page Management for Multi-GPU Systems. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1678–1692. https://doi.org/10.1109/HPCA61900.2025.00124

[84] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga Behram, James Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin S. Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim Hazelwood. 2022. Sustainable AI: Environmental Implications, Challenges and Opportunities. arXiv:2111.00364 [cs.LG]

[85] Haojie Ye, Sanketh Vedula, Yuhan Chen, Yichen Yang, Alex Bronstein, Ronald Dreslinski, Trevor Mudge, and Nishil Talati. 2023. GRACE: A Scalable Graph-Based Approach to Accelerating Recommendation Model Inference. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 282–301. https://doi.org/10.1145/3582016.3582029

[86] Vinson Young, Aamer Jaleel, Evgeny Bolotin, Eiman Ebrahimi, David Nellans, and Oreste Villa. 2018. Combining HW/SW Mechanisms to Improve NUMA Performance of Multi-GPU Systems. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 339–351. https://doi.org/10.1109/MICRO.2018.00035

[87] Zhenkai Zhang, Tyler Allen, Fan Yao, Xing Gao, and Rong Ge. 2023. TunneLs for Bootlegging: Fully Reverse-Engineering GPU TLBs for Challenging Isolation Guarantees of NVIDIA MIG. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*.

[88] Zhenkai Zhang, Zhen Zhang, Zhenyu Shen, Hongyu Jin, and Yanan Guo. 2024. Invalidate+Compare: A Timer-Free GPU Cache Attack Primitive. In *Proceedings of the 33rd USENIX Security Symposium*. USENIX Association.

[89] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems* (Copenhagen, Denmark) *(RecSys '19)*. Association for Computing Machinery, New York, NY, USA, 43–51. https://doi.org/10.1145/3298689.3346997

[90] 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 and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence* (Honolulu, Hawaii, USA) *(AAAI'19/IAAI'19/EAAI'19)*. AAAI Press, Article 729, 8 pages. https://doi.org/10.1609/aaai.v33i01.33015941