HDPAT: Hierarchical Distributed Page Address Translation for Wafer-Scale GPUs

Daoxuan Xu†, Ying Li†, Yuwei Sun‡, Jie Ren†, Yifan Sun†
William & Mary†, UIUC‡
dxu05@wm.edu, yli81@wm.edu, yuweis2@illinois.edu, jren03@wm.edu, ysun25@wm.edu

***Abstract*—A Wafer-scale GPU connects a large number of chiplets via a high-bandwidth, low-latency interposer-based network, promising to overcome the communication bottleneck of traditional multi-GPU systems. While prior work has prototyped wafer-scale GPUs to demonstrate technical feasibility, scaling to massive chiplet counts creates new bottlenecks: virtual-to-physical address translation becomes severely constrained by massive concurrent requests and long multi-hop network latencies. We propose HDPAT, a hardware-accelerated distributed address translation system that addresses this challenge through three complementary techniques: (1) Concentric caching converts near-IOMMU chiplets into hierarchical translation caches based on their distance to the IOMMU. A lightweight rotation mechanism ensures that there is always a nearby chiplet that can provide translation caching. (2) The redirection table further reduces the burden of IOMMU by delegating translations to caching chiplets, and (3) Prefetching proactively delivers potentially needed address translation into the chiplet to improve translation cache hit rate. Experimental results on 14 representative workloads show that HDPAT improves overall performance by an average of 1.57×.**

I. Introduction

The increasing complexity of modern workloads demands increasingly higher GPU computing capabilities [11], [16], [24], [31]. Due to yield limitations, the transistor count on a single die cannot scale indefinitely. Multi-GPU systems can aggregate substantial computational power, however, their scalability is limited by slow interconnect networks. A promising alternative is the Multi-Chip-Module (MCM) GPU approach [1], [5], where multiple GPU modules (GPMs, typically 2 to 4) are interconnected on an interposer. Chiplet-based design overcomes single-die limitations, enhancing GPU performance by utilizing high-bandwidth, low-latency interposer-based networks to mitigate communication bottlenecks.

In theory, increasing the number of GPMs integrated on an interposer maximizes the benefit derived from the high-bandwidth, low-latency network. The ultimate extension of this concept is to use an entire wafer as the interposer, thus creating wafer-scale GPUs [9], [15], [21], [23], [30]. With recent advancements in packing, wafer-scale GPUs have become practically feasible, as demonstrated by prior prototype implementations [23]. TSMC has also proposed its first-generation wafer-scale technology [22], suggesting the commercialization of such devices is around the corner. However, conventional multi-GPU techniques fail to address the architectural challenges introduced by such massive scale in wafer-scale GPUs.

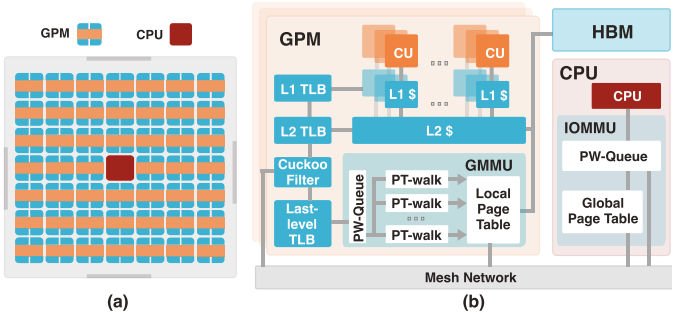


Fig. 1: An overview of (a) the wafer-scale GPU architecture and (b) the memory hierarchies of both the GPMs and the CPU, interconnected by the on-chip network.

In this work, we model the wafer-scale GPU architecture as a group of GPMs that are connected using a mesh network [9], [15] (see Figure 1(a)). Similar to mature products like Grace Hopper [25], we designate the center tile as the CPU, which serves as the system controller, dispatching tasks to GPMs and managing data allocation across the wafer. In such systems, when a GPM needs to access data residing in another GPM's memory, it must obtain the physical address information through the CPU's centralized Input-Output Memory Management Unit (IOMMU). While this design ensures coherent system-wide memory management, it creates a critical bottleneck as the number of GPMs scales.

Performance analysis reveals that while conventional IOMMUs handle 1-4 GPUs adequately, wafer-scale systems generate translation requests that overwhelm the IOMMU (see Figure 4). Physical constraints (e.g., die size, energy) prevent simply scaling up the number of page table walkers in proportion to the increased GPM count. To unleash the full computing potential of wafer-scale GPUs, we need a distributed solution that leverages the underutilized GMMUs within individual GPMs.

Guided by our extensive performance analysis (see III), we design Hierarchical Distributed Page Address Translation (HDPAT) for wafer-scale GPUs. The key insight of HDPAT is distributing the IOMMU's translation workload across the system's GMMUs, effectively transforming an underutilized resource into a solution for the translation bottleneck. HDPAT implements three complementary mechanisms:

First, hierarchical caching organizes GPMs into concentric layers based on their distance from the central IOMMU,

enabling inner-layer GPMs to serve as translation providers for their outer-layer peers. This hierarchical structure leverages the geometric properties of the wafer, routing translation requests through progressively inner layers until resolution, significantly reducing the path length for many translations compared to centralized approaches. Meanwhile, a simple rotation guarantees GPMs can always find an auxiliary caching GPM with a small number of hops.

Second, translation redirection enables the central IOMMU to delegate page table walks to distributed GPMs. By maintaining a lightweight redirection table that tracks recently walked translations, the IOMMU can quickly redirect requests to GPMs that already hold the needed translations, including proactively prefetched ones. This strategy effectively reduces redundant page table walks and translation latency.

Third, the prefetching mechanism provides more opportunities for GPMs to find the requested address in hierarchical caching GPMs. Prefetching proactively brings potentially needed address translations into caching GPMs before they are explicitly requested. This both alleviates the address translation pressure and the traffic at the IOMMU end.

Overall, we present a comprehensive solution for address translation in wafer-scale GPUs. Our contributions include:

- We conduct an extensive performance analysis of wafer-scale GPUs, revealing critical bottlenecks in the address translation process. Our characterization identifies unique challenges arising from the scale, network topology, and distributed nature of these architectures that are not present in conventional multi-GPU systems.
- Based on these insights, we develop Hierarchical Distributed Page Address Translation (HDPAT), an approach that distributes translation responsibilities across the wafer. HDPAT carefully balances translation efficiency and network communication overhead by minimizing unnecessary network traversals while maximizing translation parallelism.
- Our evaluation of HDPAT across 14 representative GPU applications demonstrates that HDPAT improves overall performance by an average of 1.57× compared to conventional centralized translation approaches.

II. Background

A. Wafer-scale GPU Architecture

Recent wafer-scale GPU architectures employ a mesh network topology to connect hundreds of processing units on a single wafer [9], [15], as illustrated in Figure 1(a). A central CPU, positioned at the network's center, assigns tasks and data to GPU Processing Modules (GPMs) while providing address translation services, thereby avoiding off-chip communication.

As shown in Figure 1(b), each GPM functions as an independent GPU with multiple Compute Units (CUs) and a two-level cache hierarchy. CUs have dedicated L1 caches and share a unified L2 cache within each GPM. Every GPM also includes its own DRAM stack. We assume HBM in this work.

Following the zero-copy memory management model adopted in multi-chip GPU systems [5], [14], [17], [19], each GPM contains a GMMU for translating addresses of local pages. Non-local address translations are handled by the central IOMMU at the CPU. Once translated, GPMs can directly access remote data at cacheline granularity.

We exclude GPU-to-GPU page migration from our scope due to the absence of mature page migration mechanisms tailored for wafer-scale GPU systems. No page migration means no page shutdown. The only necessity of TLB shootdown is freeing allocated memory, which has a negligible impact on the overall performance. Therefore, we do not evaluate the impact of TLB shootdown in this work.

Additionally, we assume the wafer-scale GPU is packaged and sold as a single integrated product, abstracting all GPMs under the unified interface of a single GPU. This design choice allows existing GPU applications to run without modifications, as the underlying runtime and driver manage the mapping of data and kernel threads across GPMs. Specifically, our experiments evenly partition both memory buffers and kernel threads among the GPMs. For example, in a hypothetical 48-GPM wafer-scale GPU, a memory allocation request for 480 pages results in pages 1–10 assigned to GPM 1, pages 11–20 to GPM 2, and so forth. Prior research has validated this approach as an effective method for minimizing inter-GPU memory access overhead [5], [19].

B. Address Translation

We adopt the address translation architecture from state-of-the-art multi-GPU systems [5], [19] as our baseline, illustrated in Figure 1(b). When a CU requires address translation, the system attempts to resolve it locally first: the request traverses the GPM's translation hierarchy consisting of L1 TLB, L2 TLB, Cuckoo filter, last-level TLB, and GMMU with its page table walkers (PT-Walkers). Each GPM's local page table contains mappings only for pages residing in its own HBM. If the translation cannot be resolved locally, the request must traverse the mesh network to the central IOMMU, which maintains the global page table for all system mappings.

Residing between the L2 TLB and the last-level TLB, a Cuckoo filter will quickly determine if the requested Virtual Page Number (VPN) might exist in the last-level TLB or the local page table. The Cuckoo filter [13] is a space-efficient probabilistic data structure that supports rapid element queries, insertion, and deletion operations while maintaining low false-positive rates even at high capacity. In multi-GPU address translation scenarios, Cuckoo filters can be used to efficiently check if page table entries (PTEs) exist in local page tables [19]. A negative lookup indicates that a translation request should bypass the local page table walk and proceed directly to a remote IOMMU, avoiding costly and unnecessary searches. HDPAT leverages Cuckoo filters for the same purpose in wafer-scale GPU systems.

There are three possible outcomes with the Cuckoo filter: (1) A negative response guarantees the VPN is absent from both the last-level TLB and local page table, allowing the request to bypass these structures entirely and proceed directly to the IOMMU. (2) A true positive response identifies that the VPN

exists locally, directing the request to the last-level TLB or GMMU where the translation completes successfully. (3) A false positive response incorrectly suggests the VPN resides locally, forcing the request through the complete local translation path before eventual forwarding to the IOMMU. This false positive case doubles the translation latency compared to a negative response from the Cuckoo filter.

When address translation cannot be resolved locally, the central IOMMU becomes a critical bottleneck. All remote translation requests converge at the IOMMU, which maintains the global page table. Hardware constraints limit the IOMMU to approximately 16 concurrent page table walkers, forcing incoming requests to queue during high translation traffic. As wafer-scale GPUs scale to more GPMs, this bottleneck increasingly limits system performance.

C. TLB Sharing

TLB sharing has long been studied in CPU systems as a means to reduce redundant address translations across cores. Notably, Bhattacharjee et al. [8] proposed sharing the last-level TLB among CPU cores to improve translation reuse. However, most prior work has focused exclusively on intra-CPU scenarios, typically within a limited number of CPU cores and under conventional workload assumptions. In contrast, modern GPUs already implement L2 TLBs shared among all CUs within a single GPU, making intra-GPU TLB sharing the default architecture. As GPU systems evolve toward wafer-scale integration with dozens of interconnected GPMs, the demand for translation reuse across GPUs (i.e., inter-GPU TLB sharing) becomes increasingly critical. Extending TLB sharing from within a GPU to across multiple GPMs introduces new challenges. We must decide (1) how to locate a translation that may be cached on a peer GPM and (2) where to cache translations to balance lookup latency and storage/traffic cost.

III. Performance Analysis for Wafer-Scale GPUs

The scale of wafer-scale GPUs introduces fundamental differences compared to Multi-Chip-Module (MCM) GPUs, particularly in communication characteristics. Wafer-scale GPUs rely on multi-hop message forwarding across a mesh network, leading to *geometry-dependent* access latency: adjacent tiles exhibit low-latency communication while cross-wafer requests suffer from both increased latency and reduced effective bandwidth as they consume multiple network links and fill intermediate buffers. These communication characteristics severely impact address translation performance. To quantify memory access latency in detail, we evaluate wafer-scale GPUs (shown in Table I) using MGPUSim with representative benchmarks (shown in Table II). Our analysis reveals four key observations:

O1: Centralized IOMMU limits address translation performance. Wafer-scale GPUs integrate thousands of compute units that generate massive volumes of concurrent address translation requests. To evaluate the performance bottleneck on the IOMMU, we compare the baseline system (shown in table I) against two types of IOMMU configurations with a 1-cycle latency with 16 parallel PW-walkers, and a 500-cycle latency with 4096 PW-walkers. Figure 2 demonstrates that IOMMU is a crucial bottleneck of the wafer-scale GPU. An ideal IOMMU configuration delivers a substantial 5.45× and 4.96× speedup over the baseline. The short latency and the high parallelism yield a similar speedup because both of the idealizations reduce dominating queueing time.

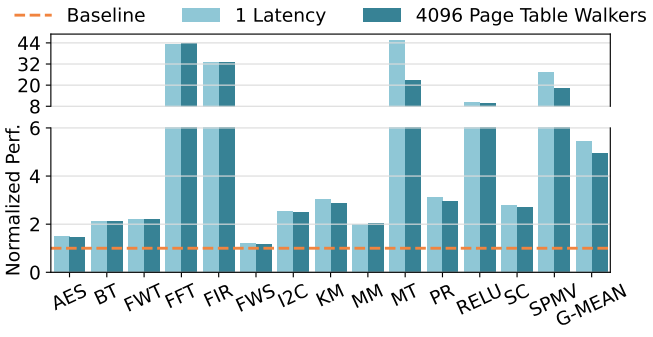


Fig. 2: Performance headroom analysis comparing the baseline MMU configuration (500-cycle translation latency, 16 page table walkers) with two different types of idealized IOMMU (1-cycle latency, 16 walkers and 500-cycle latency, 4096 walkers). The results highlight the upper-bound performance achievable with reduced translation latency and increased walker parallelism.

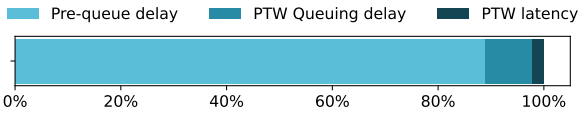


Fig. 3: Averaged latency breakdown per IOMMU translation request (percentage) for SPMV.

To further quantify the latency overhead of the centralized IOMMU, we examine the detailed breakdown of address translation time for the SPMV benchmark presented in Figure 3. This figure decomposes the total latency per request into three primary components: pre-queue latency represents the time awaiting service initiation; PTW queuing delay measures the time spent waiting within the internal request queue; and PTW latency represents the duration of the page table walk. In SPMV, pre-queue delay constitutes the largest single component of the average latency, primarily due to a persistent backlog of nearly 700 translation requests waiting for a limited number of available walkers, which is shown in Figure 4.

Simply adding more walkers is not a scalable solution. It introduces excessive area and power overhead on the CPU die. More fundamentally, the centralized design imposes a communication bottleneck because all translation requests must traverse the mesh network to reach the CPU, with latency increasing with Manhattan distance. To address these limitations, we propose a hierarchical distributed address translation architecture to alleviate IOMMU congestion and reduce translation latency.

O2: Execution imbalance due to geometric position of

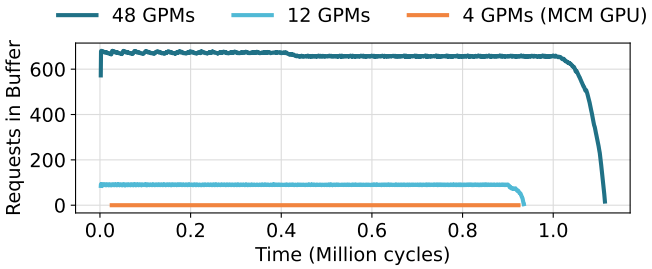
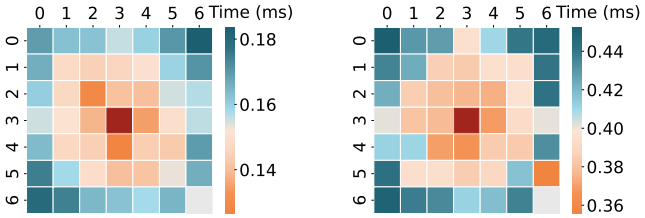


Fig. 4: Comparing the buffer pressure of the IOMMU in the SPMV benchmark between MCM-GPU (4 GPM) and wafer-scale GPU (48 GPMs). We intentionally set the IOMMU buffer to 4096 in this experiment to better demonstrate the load. The all-time high buffer pressure clearly demonstrates that IOMMU is a performance bottleneck.



(a) FWS (b) FWT

Fig. 5: GPM execution time (ms) variation based on geometric position within the wafer-scale GPU for two benchmarks. Centrally located GPMs exhibit lower execution times.

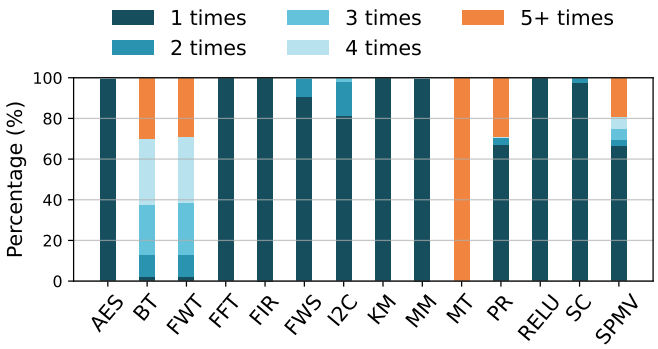


Fig. 6: Distribution of address translation counts by IOMMU, In most cases, addresses will be translated multiple times. This prompted us to store the translated address for further use.

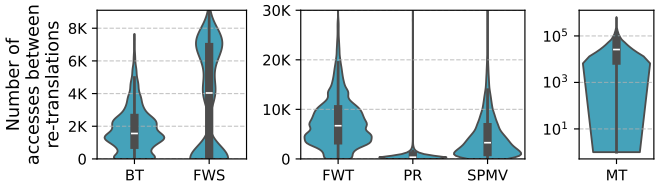


Fig. 7: Distribution of access counts between repeated address translation requests for selected benchmarks.

GPMs. While the driver evenly distributes workloads across GPMs, individual GPM execution times reveal imbalance (see Figure 5). GPMs located near the wafer center consistently complete their workloads faster than those at the periphery, leading to underutilization of compute resources near the wafer center. The performance disparity arises primarily from the geometric characteristics of wafer-scale architectures. First, central GPMs are physically closer to CPU-hosted IOMMU, resulting in faster address translation. Second, central GPMs experience fewer network hops on average when accessing memory across the wafer, while peripheral GPMs often incur significantly higher hop counts for remote memory accesses. This observation motivates leveraging underutilized central GPMs to assist peripheral GPMs with address translation.

O3: Repeated page translations are separated by different intervals. To understand address translation request reuse over time, we analyze traces collected at the IOMMU. As shown in Figure 6, address translation request patterns vary significantly across benchmarks. For benchmarks such as AES and RELU, each virtual page triggers only a single IOMMU request during execution, indicating that TLBs effectively filter repeated translations. In contrast, benchmarks such as BT and FWT exhibit repeat translation requests to the same virtual page, motivating a caching-based method.

As shown in Figure 7, the reuse distances in these cases vary widely, ranging from small values up to hundreds of thousands. The existence of a small reuse distance requires us to consider opportunities to combine multiple translations in one page table walk. Meanwhile, the existence of large reuse distances determines that using a least-recently-used (LRU) eviction method in a way-associative cache may not be the best option to capture translation reuse. A larger DRAM-based cache with less frequent eviction is desired.

O4: Spatial locality in address translation requests. As shown in Figure 8, we measure spatial locality by analyzing the virtual address distance between each translation request and the one that occurs immediately after it in the request stream. For each request, we record how close the next translated address is in terms of virtual page distance, such as within 1, 2, or 4 pages. This captures the tendency of consecutive translation requests to target nearby memory regions. The results demonstrate that 10% to 30% of future translation requests target virtual addresses in close proximity to recently translated addresses, particularly in compute-intensive benchmarks such as AES, FWS, and MM. This locality arises from algorithmic patterns that access adjacent memory regions, such as strided accesses in matrix operations and sliding windows in convolutional workloads. The presence of spatial reuse creates an opportunity for predictive translation mechanisms that preemptively translate nearby addresses, potentially reducing translation traffic to the centralized IOMMU.

Fig. 8: Distribution of address distance between consecutive translation requests. Each bar shows the fraction of next translation requests that access virtual pages within a given distance of the current request.

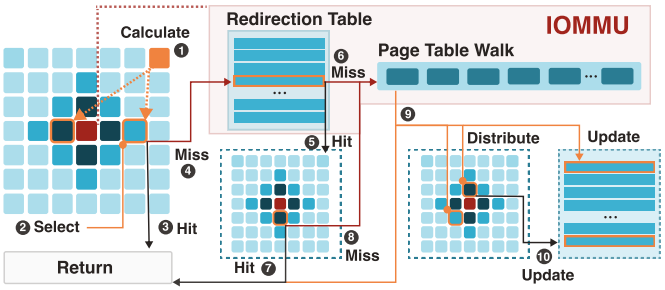


Fig. 9: Overview of the HDPAT mechanism. When a remote address translation request is issued, it is first redirected to a nearby auxiliary GPM. A Cuckoo filter is used to check whether the corresponding PTE exists quickly. If there is a TLB hit, the result is returned directly to the GPM. Otherwise, the request is forwarded to the IOMMU. Upon completing the address translation, the PT-walker additionally translates the next four addresses of the translated address. The translated addresses are then sent to the auxiliary GPMs and stored in the redirection table for future use; meanwhile, the cuckoo filter in the auxiliary TLBs is updated.

IV. HDPAT Design

To tackle the address translation challenge (**O1**), we introduce HDPAT, a comprehensive hierarchical distributed translation mechanism that utilizes the last-level TLB and GMMU distributed on the wafer to mitigate the address translation bottleneck on the IOMMU. While software-based solutions such as tiling and prefetching can improve memory locality, they require application-specific tuning and cannot adapt to dynamic runtime behavior. Therefore, HDPAT adopts a transparent hardware approach that accelerates address translation across diverse workloads without code modifications.

A. Summary

Before introducing the details, we walk through the HDPAT mechanism, as shown in Figure 9.

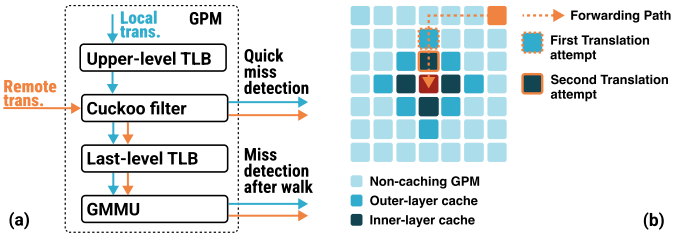


Fig. 10: PTE caching design in HDPAT. (a) The structure of a GPM's address translation hierarchy. (b) How addresses are translated in the concentric layers design ($C = 2$).

When a GPU Processing Module (GPM) needs to access a virtual address not found locally, HDPAT initiates a multi-layered resolution process. First, after a failed local translation, the GMMU calculates which auxiliary GPMs might hold the required PTE and sends the translation request to the nearest potential auxiliary GPMs. This hierarchical peer search progresses inward through layers (e.g., from middle to inner-layer GPMs), with Cuckoo filters quickly determining whether a translation might be available at each hop. If found, the PFN is returned directly to the requesting GPM, relieving the burden of the IOMMU.

If peer caching misses, the request reaches the IOMMU, where two mechanisms activate: (1) the redirection table is checked first, potentially redirecting the request to a GPM already holding the translation; and (2) if a page table walk becomes necessary, the walker both processes the request and revisits the queue of to-be-translated addresses to handle identical pending translations.

Once translation is completed, the IOMMU proactively fetches the PTE for the next sequential address, capitalizing on spatial locality. Both the requested and prefetched translations are selectively distributed to appropriate auxiliary GPMs based on access frequency thresholds, while the redirection table is updated to optimize future requests. This coordinated proactive delivery approach minimizes network hops, reduces IOMMU pressure, and exploits spatial locality to accelerate translation throughout the system.

B. Route-based Address Caching

An effective way to reduce the load on the centralized IOMMU is to introduce caches closer to the source of translation requests. Building upon this concept, we first examine a route-based caching strategy.

As translation requests travel toward the central IOMMU, each intermediate GPM checks its local last-level TLB and page table, effectively acting as a cache. If the page table entry (PTE) is found, it is immediately returned to the requester. If not, the request continues toward the IOMMU, with the intermediate GPM awaiting the returned translation. Once received, the GPM stores the PTE locally for future use.

The translation process within each GPM is illustrated in Figure 10. In our model, incoming translation requests, whether remote or local, first pass through a cuckoo filter.

The cuckoo filter quickly determines whether the requested page is definitely absent from the local page table, thus avoiding costly page table walks. Note that the use of cuckoo filters in GPU address translation systems has been previously introduced [14], [19], and our work does not claim novelty. If the cuckoo filter indicates a potential match, the translation proceeds through the standard path via the last-level TLB and GMMU. However, because cuckoo filters can yield false positives, a failed translation at the GMMU results in forwarding the request to the next hop toward the central IOMMU.

However, the route-based caching method has several drawbacks: (1) The system may repeatedly attempt to translate an address along the path. For instance, in a 7x7 wafer-scale GPU, translation attempts could occur up to five times before reaching the IOMMU, significantly increasing total latency. (2) Since any GPM can cache any PTE, this may cause high duplication and inefficient utilization of the limited caching space. Such redundancy could lead to thrashing, nullifying the benefits of caching. To address these challenges, we propose enhancements to the caching mechanism described below.

C. Concentric Address Caching

To address the excessive translation attempts, we introduce a concentric translation caching scheme. Concentric caching leverages observation **O2**, in which GPMs closer to the central CPU typically execute faster. It is therefore reasonable to balance the workload by requiring these central GPMs to handle additional translation responsibilities.

In this scheme, we define a maximum number of translation attempts, denoted as a constant C. For instance, in a typical 7x7 wafer-scale GPU, C might range from 0 (no caching) to 3 (up to three translation attempts before reaching the IOMMU). By default, we set $C = 2$, as being one step away from the border to maximize the caching GPMs and avoid wasting GPMs. The number of layers is tunable by drivers or firmware.

Translation attempts are limited to one per concentric layer. For example, if a peripheral GPM (outside the defined concentric layers) initiates a translation request, the first attempt occurs at a GPM in the outer layer. If unsuccessful, the request is forwarded inward to the next concentric layer (inner layer) for another attempt before ultimately being resolved by the IOMMU. GPMs within a concentric layer first attempt translation on the current layer, forwarding the request to inner layers upon a miss, thereby progressively centralizing translation tasks.

D. Clustering

While concentric caching significantly reduces the number of translation attempts, inefficiencies remain in caching space usage. For instance, a PTE could redundantly reside in all 8 GPMs within a given layer. To resolve this redundancy, HDPAT enforces that each PTE appears exactly once per concentric layer (illustrated in Figure 11 (a)). The caching process involves two steps: (1) The virtual page number (VPN) is taken modulo the number of clusters to determine the target cluster. Given the 2D mesh, we partition the wafer into four quadrant-based clusters to keep each caching layer within one hop of the next inner layer. (2) Within the chosen cluster, another modulo operation using the VPN selects the specific GPM responsible for caching the PTE. This process is formally described by the following equations:

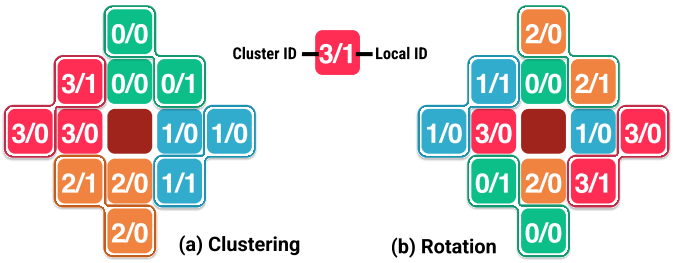


Fig. 11: Clustering and rotation scheme used in HDPAT ($C = 2$). (a) HDPAT first divides the wafer-scale GPU into quadrant-based clusters and allocates Page Table Entries (PTEs) to corresponding clusters. (b) To ensure translation requests encounter cached translations within short distances from any direction, HDPAT rotates each caching layer by redefining the starting points.

$$ID_{cluster} = \text{VPN} \mod N_c \tag{1}$$

$$ID_{local} = \left\lfloor \frac{\text{VPN}}{N_c} \right\rfloor \mod N_g \tag{2}$$

where N_c denotes the number of clusters and N_g denotes the number of GPMs per cluster for that layer.

This clustering approach implies that HDPAT no longer translates addresses along the forwarding path toward the central IOMMU. Instead, a requester directly computes and sends translation requests to the GPM that may cache the PTE in the outer layer. To reduce latency, requests are sent concurrently to all concentric layers, and the earliest response is returned. Generally, requests move inward to avoid costly cross-wafer communications.

E. Rotation

While clustering reduces redundant PTE copies and effectively expands caching capacity, quadrant-based clustering has the drawback that cached PTEs may cluster closely together. This proximity benefits requesters within the same quadrant, which may require only a few hops, but disadvantages those from opposite quadrants, which require additional hops, thereby increasing latency.

To mitigate this issue and ensure translation opportunities are consistently close to GPMs from all directions, HDPAT introduces a rotation mechanism. For example, when $C = 2$, the cluster and local ID counting begin from the GPM located 180 degrees from the original starting point (illustrated in Figure 11 (b)). This rotation ensures all GPMs, regardless of their quadrant, have at least one nearby caching GPM. Given that requests are dispatched concurrently, there is always a nearby GPM available to respond quickly.

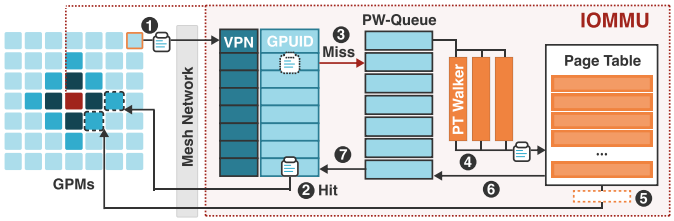


Fig. 12: Enhanced IOMMU architecture incorporating the redirection table.

Although the logic behind clustering and rotation might appear complex, its implementation is straightforward and low-cost. Vendors can predefine clustering and indexing strategies either before product deployment or as part of regular driver updates. Translation requesters perform simple modulo and rotation operations, typically reduced to straightforward bit operations and table lookups, making this approach highly practical.

F. Redirection

As observed in **O3 and Figure 7**, the same virtual address is often translated multiple times within a short interval. This scenario poses a unique challenge for wafer-scale caching, as an address may initially miss in the concentric-layer GPM caches simply because the IOMMU is currently handling its translation request or is in transit back to the requesting GPM. To address this, we need a mechanism at the IOMMU that redirects incoming translation requests to the correct concentric-layer GPM, where the requested Page Table Entry (PTE) has just been sent. This helps later requests quickly catch up to recently completed translations, preventing unnecessary repeated page walks.

Redirection Table. To efficiently redirect requests for recently translated or prefetched PTEs, HDPAT introduces a lightweight Redirection Table (1024 entries) at the IOMMU. This table records recently translated or prefetched VPNs along with the IDs of the corresponding GPMs in the concentric layers. Upon receiving an address translation request, the IOMMU first checks the Redirection Table. If the VPN is found, the request is immediately redirected to the indicated GPM instead of performing a redundant page-table walk at the IOMMU, thus significantly reducing translation overhead and alleviating the centralized translation bottleneck.

One might argue that the redirection table closely resembles a TLB, questioning the need to introduce a separate structure. However, we identify two key advantages of the redirection table over a traditional TLB for wafer-scale GPUs: (1) the redirection table is nearly twice as space-efficient, as it only stores process ID and VPN, whereas a TLB also stores additional metadata; (2) unlike a TLB, the redirection table does not require MSHRs, which constrain concurrency by holding translation requests that miss in the TLB. When all MSHRs are occupied, subsequent translation requests must stall, unable to proceed to the IOMMU's page-table walkers. Consequently, the redirection table is better suited for efficiently handling the high concurrency of translation requests in wafer-scale GPU systems.

Operational Flow. The operational flow of the redirection table is shown in Figure 12. When a translation request arrives ❶, the redirection table is checked first. If the request is a hit, it is immediately redirected to a peer GPM ❷, thereby avoiding an IOMMU page walk and reducing IOMMU load. (addressing **O1**). If it's a miss, the request is enqueued in the PW-Queue ❸. Once a PT walker processes a request ❹, it conditionally pushes the corresponding PTE to an auxiliary GPM ❺ (as detailed in the next paragraph) and then revisits the PW-Queue ❻ to handle any identical pending requests, which further accelerates the frequently accessed PTE translation process. Finally, the redirection table is updated ❼.

Due to the limited space of GMMU, GPM cannot afford remote page table replication. Leveraging observation **O3**, which shows that only some pages experience high re-translation counts, we make auxiliary caching selective. The IOMMU only considers pushing a PTE to an auxiliary GPM (Step ❺) if its access count (tracked using unused PTE bits) exceeds a threshold, ensuring that peer cache resources are used for frequently reused translations. To minimize duplication overhead, we allow only one copy per layer.

G. Prefetching

Leveraging the principle of spatial locality demonstrated in Observation **O4**, we introduce Proactive Page-Entry Delivery to preemptively fetch and distribute PTEs for addresses adjacent to those currently being translated, further reducing IOMMU workload (**O1**) and translation latency (**O4**).

Mechanism. When an IOMMU PT walker resolves VPN N, it also fetches the PTE from VPN N to VPN N+3. This prefetched PTE is pushed to the appropriate auxiliary GPMs in the inner or middle layers. Concurrently, the IOMMU updates its redirection table (see IV-F) for VPN N+1, enabling potential future redirection.

Benefits. By anticipating likely future requests based on observed spatial locality and proactively placing PTEs in distributed peer caches, this technique reduces the probability of subsequent requests needing a full IOMMU translation. The minimal overhead of fetching one extra PTE is offset by savings in latency and IOMMU load. This proactive approach works synergistically with the other HDPAT components.

V. Evaluation

A. Methodology

Simulation. We evaluate HDPAT using MGPUSim [26], as MGPUSim is tailored to evaluate multi-GPU systems and has built-in high-fidelity modeling of the address translation process. We implement the GMMU component and a mesh network in MGPUSim, as well as other HDPAT features.

We configure a wafer-scale GPU system with a mesh network. We use a baseline configuration of 32 CUs per GPM in this paper. Our GPMs' configuration aligns with the current AMD MI100 GPU configuration [28]. Due to the limitation of the simulator, we proportionally scale down the CU counts and

TABLE I: Configuration of wafer-scale GPUs.

Module	Configuration
CU	1.0 GHz, 32 per GPM
L1 Vector Cache	16 KB, 4-way, 16-MSHR
L1 Scalar Cache	16 KB, 4-way, 16-MSHR
L1 Inst. Cache	32 KB, 4-way, 16-MSHR
L2 Cache	4 MB, 16-way, 64-MSHR
L1 Vector TLB	1-set, 32-way, 4-MSHR, 4-cycle latency, LRU
L1 Scalar TLB	1-set, 32-way, 4-MSHR, 4-cycle latency, LRU
L1 Inst. TLB	1-set, 32-way, 4-MSHR, 4-cycle latency, LRU
L2 TLB	64-set, 32-way, 32-MSHR, 32-cycle latency, LRU
GMMU Cache	64-set, 16-way
GMMU	8 shared page table walkers 100 × 5 levels = 500 cycles in total [14], [19]
IOMMU	Host MMU 16 shared page table walkers, 100 × 5 levels = 500 cycles in total [14], [19]
Redirection Table	1024 entries, LRU
HBM	8 GB, 1.23 TB/s
Mesh Network	768 GB/s, 32-cycle latency per link [14]

L2 Cache size so that each GPM is approximately one-fourth of an MI100 GPU (consider using 4 GPMs to implement an MI100 GPU). We also test configurations of newer AMD GPUs in V-E. Each GPM has its own page table stored in its device memory that covers all the pages on the GPM. The detailed wafer-scale GPU setup is summarized in Table I. The mesh network provides 768 GB/s bandwidth per link, with a link traversal latency of 32 cycles.

We carefully model the interference between the local and remote translations. For the Cuckoo filter, the local and remote translations share the same set of ports (no extra ports are added), with local translations having higher priority. Both local and remote translation occupy the Cuckoo filter, L2 TLB, and GMMU spaces without priority differences.

Workloads. We use 14 benchmarks from Hetero-Mark [27], AMDAPPSDK [2], SHOC [10], and DNNMark [12] benchmark suites, listed in Table II. Such benchmarks are supported by MGPUSim and are also used in prior works [14], [18], [20], [29]. The suite covers a wide range of data access patterns, enabling a comprehensive evaluation of HDPAT under diverse scenarios. These patterns include random, partitioned, adjacent, and scatter-gather access.

We configure each benchmark with the largest problem size that can be supported by the simulation environment, constrained by the memory capacity of the host machine and the practical limits of simulation time. We consider that the problem sizes are sufficient to test the wafer-scale GPUs for the following two reasons.

(1) The wafer-scale system we configured has 48 GPMs with 32 compute units (a total of 1536 CUs). Other than AES, each workload can fill each compute unit at least 10 times, sufficient to reach a saturation point. AES is a highly iterative workload, with each workgroup staying in the CU for a long time, stressing the memory system with a steady memory request issuing rate. Therefore, the number of compute tasks is sufficient to evaluate wafer-scale GPUs.

(2) While modern LLM workloads dwarf our memory footprint, the selected workloads are still sufficient to stress the memory system. We measured the IOMMU pressure with different memory footprints (see Figure 13, using FIR as an example). Results suggest that the IOMMU pressure is steady for most of the time, regardless of the memory footprint. The size invariance allows us to use small configurations as a proxy for larger ones, maintaining accuracy and reasonable simulation time.

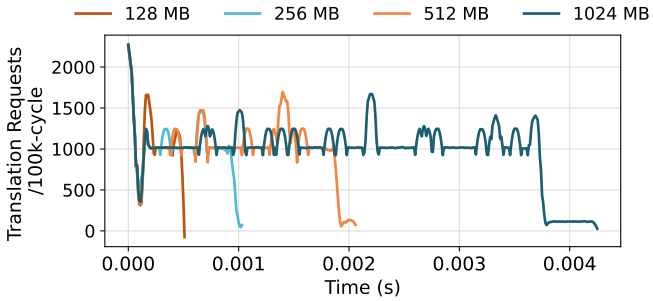


Fig. 13: Time series of IOMMU-served translation requests for FIR at different problem sizes. Counts are aggregated with a fixed time window (100000-cycle). The similar shapes indicate size-invariant translation behavior, making the small-size benchmark representative of the large-size configuration.

TABLE II: Benchmarks, corresponding workgroup counts, and memory footprint.

Abbr.	Benchmark	Workgroups	Memory FP
AES	Advanced Encryption Standard	4,096	8 MB
BT	Bitonic Sort	16,384	16 MB
FWT	Fast Walsh Transform	16,384	64 MB
FFT	Fast Fourier Transform	32,768	256 MB
FIR	Finite Impulse Response Filter	65,536	256 MB
FWS	Floyd-Warshall Shortest Paths	65,536	72 MB
I2C	Image to Column Conversion	16,384	32 MB
KM	KMeans	32,768	40 MB
MM	Matrix Multiplication	16,384	256 MB
MT	Matrix Transpose	524,288	2,048 MB
PR	PageRank	524,288	14 MB
RELU	Rectified Linear Unit	1,310,720	1,280 MB
SC	Simple Convolution	262,465	256 MB
SPMV	Sparse Matrix-Vector Multiplication	81,920	120 MB

Baselines. The naive implementation translates all non-local translations at the IOMMU. We further compare HDPAT with the state-of-the-art solutions designed for chiplet-based multi-GPU systems including Trans-FW [19], Valkyrie [7], and Barre [14]. We do not compare with Griffin [6] as Griffin mainly focuses on page migration.

We further evaluate a straightforward distributed-caching baseline. The same number of GPMs as in the concentric-caching setup are split into two equal groups placed symmetrically on the two sides of the CPU. For each translation request, a GPM first probes the nearest (in hop count) peer within its own group; if that probe misses, the request is forwarded directly to the IOMMU—no cross-group lookup, rotation, or redirection is performed. Comparing with the simple distributed caching approach allows us to better understand the value of concentric cache and rotation.

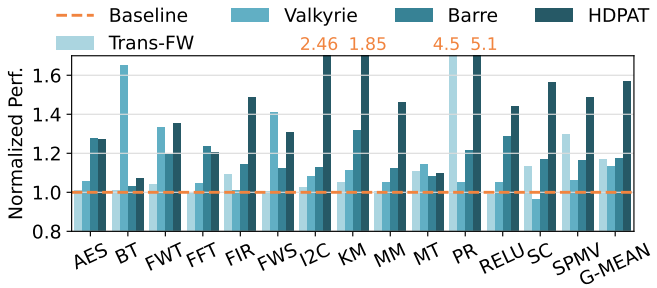


Fig. 14: Normalized performance of Trans-FW and HDPAT compared to the baseline configuration.

B. Overall Performance

We compare the performance of HDPAT with the baseline configuration and several state-of-the-art solutions: Trans-FW [19], Valkyrie [7], and Barre [14]. Results are shown in Figure 14. HDPAT achieves 1.57× performance improvement on average across all evaluated benchmarks. Since Valkyrie and Trans-FW focus on local address translation optimization, remote address translation requests still burden the IOMMU. Moreover, while Barre alleviates IOMMU pressure by finding address translation opportunities in the PW-queue, the size of the PW-queue limits the performance improvement.

C. Ablation Study

We analyze all proposed techniques. The peer caching techniques include routing-based caching, concentric caching, and cluster with rotation caching. The contributions of the redirection table and prefetching in the design of HDPAT have also been evaluated separately. The results are shown in Figure 15. Due to the penalty incurred by multiple address translation attempts, both routing-based caching and concentric caching do not achieve a noticeable performance improvement. Meanwhile, although distributed caching limits the number of address translation attempts, an uneven distribution of PTEs introduces additional communication overhead. Therefore, the distributed caching method only gets 1.08× performance improvement.

The clustering and rotation caching allow only one pre-IOMMU address translation attempt and minimize the search path, gaining 1.13× performance improvement. Enabling the redirection table yields a 1.18× performance improvement over the baseline. Benchmarks PR, MT, and SPMV see gains from the redirection table, consistent with previous observations on address remapping efficiency. The benefit from prefetching averages 1.17× across all benchmarks. Specifically, FIR and KM achieve greater performance gains than the others due to their iterative access with a small stride. Excluding cold-walk requests, the combination of the redirection table and prefetching still serves 58% of the remaining translation requests from peer GPM caches. When combining all methods together, HDPAT achieves an average performance gain of 1.57×.

The performance impact of HDPAT varies across benchmarks due to their distinct memory access patterns. PR benefits substantially from the combination of concentric caching, rotation, and redirection tables because it exhibits strong temporal locality, with multiple GPMs frequently requesting the same PFN repeatedly. In contrast, BT shows minimal improvement, as its inherent spatial locality enables the local GMMU to handle most address translation requests without remote lookups. MT presents a middle case: despite exhibiting high-frequency and long-range memory reuse patterns, it sees limited gains because the large reuse distances exceed the capacity of redirection tables and caches, causing valid entries to be evicted between reuses. This workload diversity validates our decision to combine complementary techniques, as each addresses distinct access-pattern characteristics that no single optimization could handle effectively.

We further analyze how address translation requests are distributed across three components in HDPAT and in IOMMU, as shown in Figure 16. Without HDPAT, all requests would be serviced by the IOMMU, which suffers from congestion and long service latency. In contrast, HDPAT offloads 42.1% of translations to distributed mechanisms, significantly reducing IOMMU pressure and alleviating the central bottleneck.

While the IOMMU still dominates in benchmarks with irregular or hard-to-predict access patterns, many workloads benefit from offloading, highlighting the effectiveness of exploiting translation locality and reuse. Benchmarks such as FFT, FWS, FWT, and SPMV show a balanced distribution across peer caching, redirection, and proactive delivery. Their structured but dynamic access patterns expose both spatial and temporal locality. Peer caching resolves translations distributed across the mesh, and proactive delivery anticipates near-future accesses. The average prefetch accuracy reaches 65.55%, and the redirection table redirects misses to recently active page managers. In BT and PR, peer caching contributes 38% and 65% of address translation, driven by strong spatial reuse across GPU nodes. Translations are often resolved before reaching the IOMMU, showing the value of leveraging mesh-locality through search-based translation. AES, FIR, KM, MM, ReLU, and SC rely more evenly on redirection and proactive delivery. Their repetitive, structured patterns make future accesses predictable, enabling effective reuse without a single dominant mechanism. HDPAT is less effective for MT, which generates frequent translations with long reuse distances. While some spatial locality exists, entries are often evicted before reuse, making caching less effective and resulting in heavy reliance on the IOMMU.

D. Remote Address Translation Response Time

We evaluate the round-trip time of address translation requests, from sending the translation request to search for a PTE in the next layer to the translation response back to the GPM. We compare the average round-trip time with the baseline configuration. The result in Figure 17 shows that the response time is dramatically reduced after implementing HDPAT, saving 41% of the time on average. We also quantify

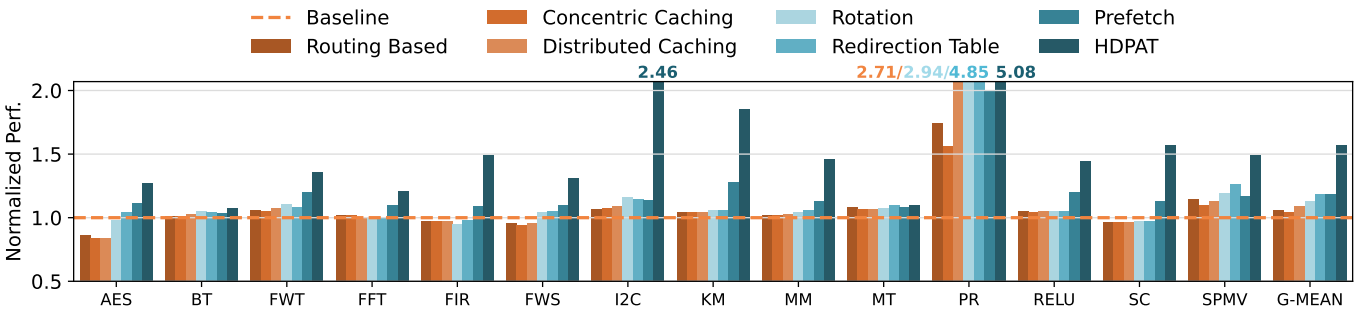


Fig. 15: The performance improvement for various techniques mentioned in IV. Routing-based caching and concentric caching have worse performance than the clustering and rotation design due to repeated translation and high PTE duplication. The clustering and rotation caching continue to reduce attempt times and network latency, gaining 1.13×. The redirection table and prefetching techniques provide 1.18× and 1.17× performance improvement, respectively. Combining all techniques, HDPAT has 1.57× performance improvement.

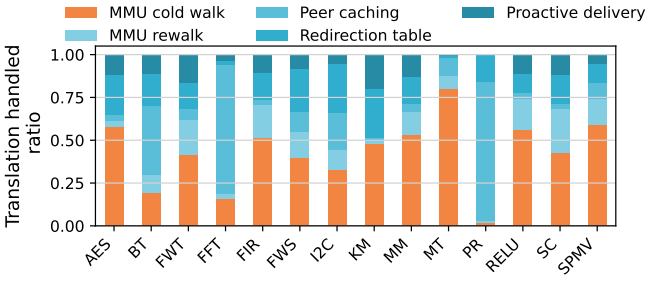


Fig. 16: Breakdown of how address translations are handled in HDPAT across benchmarks.

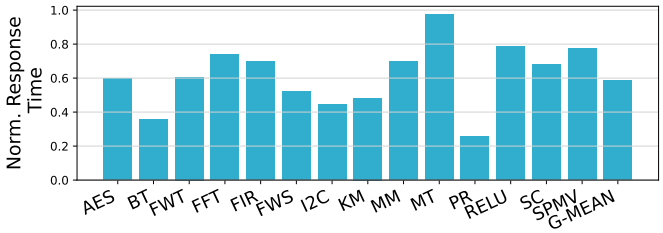


Fig. 17: Translation requests round-trip response time with HDPAT, normalized to baseline (without HDPAT).

the impact of NoC traffic when using HDPAT, which incurs only 0.82% additional traffic.

E. Sensitivity Study

Proactive delivery granularity. Figure 18 shows the performance impact of varying the number of contiguous PTEs delivered proactively per page table walk, normalized to the baseline without HDPAT. Delivering 1, 4, and 8 PTEs yields average performance improvements of 1.40×, 1.57×, and 1.59×, respectively. These gains demonstrate the spatial locality in GPU workloads (Observation O4), where proactively fetching nearby PTEs reduces future translation requests and improves GMMU cache utilization.

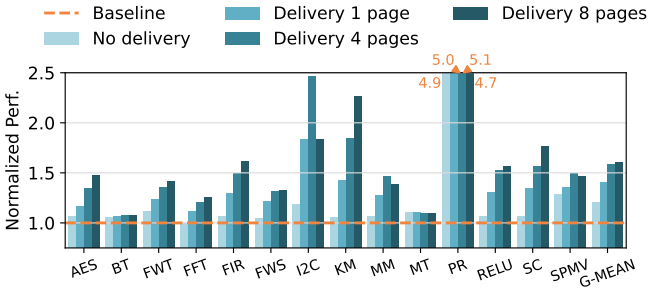


Fig. 18: Performance impact of proactive delivery granularity during GPU address translation across benchmarks. Performance is normalized to no-HDPAT.

Workloads with strong spatial locality benefit most: PR, I2C, and MM achieve speedups up to 5.07×, 1.84×, and 1.46×, respectively. Proactive delivery enables the system to anticipate sequential accesses, reducing translation latency. Conversely, BT and MT show minimal improvement ($< 10\%$) due to irregular access patterns.

We observe performance saturates at 4-PTE delivery for most benchmarks. While 8-PTE delivery provides marginal additional benefit (1.91% on average), it increases GMMU cache pressure and interconnect traffic. Based on this analysis, HDPAT adopts 4-PTE proactive delivery as the optimal configuration.

Redirection Table vs. TLB. We also test the performance difference between using the redirection table and the TLB on the IOMMU side. For a fair comparison, we replace the redirection table with a conventional TLB occupying an equivalent physical area on the IOMMU side. Figure 19 shows that redirection table has 1.27× improvement compared with using TLB. The main reason is that, in the same area of consumption, the TLB can hold fewer entries than the redirection table (512 entries for the TLB versus 1024 for the redirection table). The redirection table is more space-efficient because it stores only virtual addresses, with no

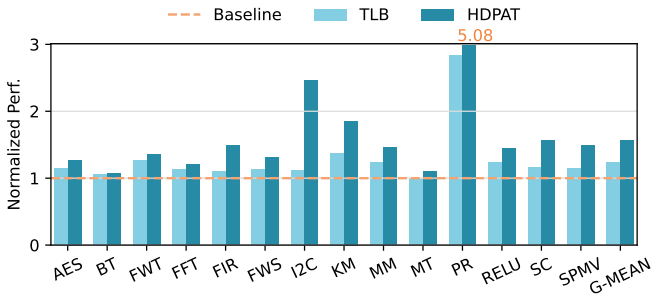


Fig. 19: Comparing the performance of using a TLB or a redirection table at the IOMMU.

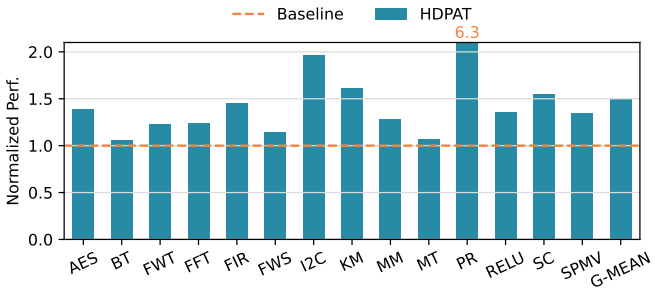


Fig. 22: The performance improvement of HDPAT in the 7×12 wafer-scale GPU configuration.

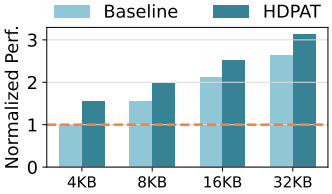


Fig. 20: Performance impact of system page size (geometric mean), normalized to the 4KB Baseline.

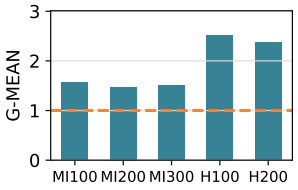


Fig. 21: Geometric mean of HDPAT performance improvement across configurations.

physical addresses required. The proactive page-entry delivery frequently flushes TLB entries, making it harder to hold older non-proactive page entries and preventing them from benefiting from temporal locality. Additionally, due to the MSHR mechanism in TLBs, translation requests that cannot be found in the TLB are stored in the TLB's MSHR. When the MSHR is full, incoming requests will be blocked outside of the TLB. They cannot benefit from the PW-Queue revisit mechanism that we mentioned in IV-F, and translation requests cannot be responded to immediately, especially if the proactive page-entry delivery mechanism has prefetched the corresponding PFN.

System page size. Figure 20 shows the geometric mean performance across evaluated workloads for different page sizes, normalized to the 4KB baseline configuration. Without HDPAT, larger pages provide performance gains by reducing translation requests. HDPAT maintains an average 50% performance advantage over the baseline across all page sizes, demonstrating that its hierarchical translation mechanism operates efficiently and orthogonally to page size optimizations.

Generalization across modern GPU configurations. We evaluate HDPAT using GPM configurations aligned with various commercial GPUs. In addition to our baseline MI100 configuration, we model GPMs with specifications matching AMD MI200, MI300, and NVIDIA H100, H200 memory storage system configurations. The NVIDIA configurations represent large-scale memory systems, featuring 256KB L1 cache per CU and 50MB L2 cache, paired with HBM2 (H100) and HBM3 (H200), respectively. In Figure 21, HDPAT consistently achieves performance improvements of 1.47× and 1.50× under these configurations. Notably, HDPAT shows even greater benefits on the larger-memory H100 and H200 configurations, achieving 2.52× and 2.36× speedups.

Generalization with different wafer size. We extend our evaluation beyond the baseline 7×7 wafer configuration to a larger 7×12 wafer. As shown in Figure 22, all workloads benefit from HDPAT on the larger wafer, with a geometric mean performance improvement of 1.49×. This demonstrates that HDPAT's distributed translation approach scales effectively as wafer dimensions increase.

F. Area Overhead and Power Consumption

We also estimate the hardware overhead of our HDPAT, including the 1024-entry redirection table and cuckoo filters. We use OpenRoad [3] with a 7nm node tech process. The redirection table occupies 0.034 mm^2 with 0.16W. Assuming the center tile is an AMD Ryzen 9 CPU (141.2 mm^2, 170 W TDP) [4], the redirection table adds 0.02% spatial overhead and 0.09% energy overhead.

VI. Conclusion

Wafer-scale GPUs promise significant scalability improvements by connecting multiple GPU chiplets (GPMs) through high-performance, interposer-based networks. However, architectural challenges, such as address translation in the centralized IOMMU, become critical performance bottlenecks. To address this, we propose HDPAT, a comprehensive hierarchical distributed translation mechanism integrating Hierarchical Caching, IOMMU-Initiated Translation Redirection, and Proactive Page-Entry Delivery. Our evaluation demonstrates that HDPAT significantly reduces translation overhead, achieving a 1.35× performance improvement over state-of-the-art methods and a 1.57× improvement over baseline configurations by effectively offloading 42.1% of translation requests. HDPAT not only resolves the address translation bottleneck but also opens pathways for future exploration in memory access optimization, intelligent page migration, and resource allocation, supporting the advancement of wafer-scale GPU architectures.

ACKNOWLEDGMENT

We would like to thank our anonymous reviewers for their valuable feedback. This work was partially supported by the United States' National Science Foundation (NSF) OAC-2402947, OAC-2505119, and CCF-2402804.

REFERENCES

[1] Advanced Micro Devices, "Amd accelerates pace of data center ai innovation and leadership with expanded amd instinct gpu roadmap," https://www.amd.com/en/newsroom/press-releases/2024-6-2-amd-accelerates-pace-of-data-center-ai-innovation-.html, 2024, accessed: 2025-04-09.

[2] Advanced Micro Devices, Inc., "Amd opencl programming optimization guide," https://www.amd.com/content/dam/amd/en/documents/radeon-tech-docs/programmer-references/AMD_OpenCL_Programming_Optimization_Guide2.pdf, 2012, accessed: 2025-04-11.

[3] T. Ajayi and D. Blaauw, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," in *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019.

[4] AMD. (2022) Amd ryzen 9 7900x desktop processor. TechPowerUp. Accessed: 2025-10-11. [Online]. Available: https://www.techpowerup.com/cpu-specs/ryzen-9-7900x.c2847

[5] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "Mcm-gpu: Multi-chip-module gpus for continued performance scalability," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 320–332, 2017.

[6] T. Baruah, Y. Sun, A. T. Dinçer, S. A. Mojumder, J. L. Abellán, Y. Ukidave, A. Joshi, N. Rubin, J. Kim, and D. Kaeli, "Griffin: Hardware-software support for efficient page migration in multi-gpu systems," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 596–609.

[7] T. Baruah, Y. Sun, S. A. Mojumder, J. L. Abellán, Y. Ukidave, A. Joshi, N. Rubin, J. Kim, and D. Kaeli, "Valkyrie: Leveraging inter-tlb locality to enhance gpu performance," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 455–466. [Online]. Available: https://doi.org/10.1145/3410463.3414639

[8] A. Bhattacharjee, D. Lustig, and M. Martonosi, "Shared last-level tlbs for chip multiprocessors," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 2011, pp. 62–63.

[9] S. Chen, S. Pal, and R. Kumar, "Waferscale network switches," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 215–229.

[10] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (shoc) benchmark suite," in *Proceedings of the 3rd workshop on general-purpose computation on graphics processing units*, 2010, pp. 63–74.

[11] J. Dean, "Large-scale deep learning for building intelligent computer systems," 2016.

[12] S. Dong and D. Kaeli, "Dnnmark: A deep neural network benchmark suite for gpus," in *Proceedings of the General Purpose GPUs*, 2017, pp. 63–72.

[13] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 75–88.

[14] Y. Feng, S. Na, H. Kim, and H. Jeon, "Barre chord: Efficient virtual memory translation for multi-chip-module gpus," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 834–847.

[15] Y. Hu, X. Lin, H. Wang, Z. He, X. Yu, J. Zhang, Q. Yang, Z. Xu, S. Guan, J. Fang *et al.*, "Wafer-scale computing: Advancements, challenges, and future perspectives [feature]," *IEEE Circuits and Systems Magazine*, vol. 24, no. 1, pp. 52–81, 2024.

[16] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[17] M. Khairy, V. Nikiforov, D. Nellans, and T. G. Rogers, "Locality-centric data and threadblock management for massive gpus," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1022–1036.

[18] B. Li, Y. Guo, Y. Wang, A. Jaleel, J. Yang, and X. Tang, "Idyll: Enhancing page translation in multi-gpus via light weight pte invalidations," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1163–1177.

[19] B. Li, J. Yin, A. Holey, Y. Zhang, J. Yang, and X. Tang, "Trans-fw: Short circuiting page table walk in multi-gpu systems via remote forwarding," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 456–470.

[20] B. Li, J. Yin, Y. Zhang, and X. Tang, "Improving address translation in multi-gpus via sharing and spilling aware tlb design," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1154–1168.

[21] S. Lie, "Wafer-scale ai: Gpu impossible performance," in *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society, 2024, pp. 1–71.

[22] S. K. Moore, "Tsmc's advanced packaging: The key to more-than-moore," *IEEE Spectrum*, February 2024, accessed: 2025-06-20. [Online]. Available: https://spectrum.ieee.org/tsmc-advanced-packaging

[23] S. Pal, D. Petrisko, M. Tomei, P. Gupta, S. S. Iyer, and R. Kumar, "Architecting waferscale processors-a gpu case study," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 250–263.

[24] G. Rossi, F. Catani, L. Leoni, S. Segoni, and V. Tofani, "Hiresss: a physically based slope stability simulator for hpc applications," *Natural Hazards and Earth System Sciences*, vol. 13, no. 1, pp. 151–166, 2013.

[25] N. A. Simakov, M. D. Jones, T. R. Furlani, E. Siegmann, and R. J. Harrison, "First impressions of the nvidia grace cpu superchip and nvidia grace hopper superchip for scientific workloads," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops*, 2024, pp. 36–44.

[26] Y. Sun, T. Baruah, S. A. Mojumder, S. Dong, X. Gong, S. Treadway, Y. Bao, S. Hance, C. McCardwell, V. Zhao *et al.*, "Mgpusim: Enabling multi-gpu performance modeling and optimization," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 197–209.

[27] Y. Sun, X. Gong, A. K. Ziabari, L. Yu, X. Li, S. Mukherjee, C. McCardwell, A. Villegas, and D. Kaeli, "Hetero-mark, a benchmark suite for cpu-gpu collaborative computing," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2016, pp. 1–10.

[28] TechPowerUp. AMD Radeon Instinct MI100. TechPowerUp. [Online]. Available: https://www.techpowerup.com/gpu-specs/radeon-instinct-mi100.c3496

[29] Y. Wang, B. Li, M. T. I. Ziad, L. Eeckhout, J. Yang, A. Jaleel, and X. Tang, "Oasis: Object-aware page management for multi-gpu systems." HPCA, 2025.

[30] D. Xu, L. Xu, J. Ren, and Y. Sun, "Exploring the wafer-scale gpu," 2025.

[31] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model (llm) security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, p. 100211, 2024.