V-PWC: Accelerating Page Table Walk for Multi-Chip-Module GPUs

Sen Yang
College of Computer Science and Technology
National University of Defense Technology
Changsha, Hunan, China
yangsen_@nudt.edu.cn

Wei Chen*
College of Computer Science and Technology
National University of Defense Technology
Changsha, Hunan, China
chenwei@nudt.edu.cn

Xinjin Gao
College of Computer Science and Technology
National University of Defense Technology
Changsha, Hunan, China
xinjin.gaocs@nudt.edu.cn

Mingxin Tang
College of Computer Science and Technology
National University of Defense Technology
Changsha, Hunan, China
tangmingxin16@nudt.edu.cn

Qiong Li
National Innovation Institute of Defense Technology
Academy of Military Sciences
Beijing, China
517066092@qq.com

Xia Zhao*
National Innovation Institute of Defense Technology
Academy of Military Sciences
Beijing, China
xiazhao@nudt.edu.cn

Abstract

With the slowdown of Moore's law and the advancement of processor packaging technologies, multi-chip-module (MCM) GPUs have emerged as a promising architecture for sustaining performance scaling. At the same time, applications targeting MCM-GPUs continue to exhibit rapidly growing memory footprints, significantly increasing pressure on the GPU virtual memory subsystem. Modern MCM-GPUs rely on translation lookaside buffers (TLBs) and page table walk caches (PWCs) to accelerate virtual-to-physical address translation. However, we observe that for locality-sparse applications, the TLB often experiences near-zero hit rates, while the PWC frequently misses at higher-level page table indices. As a result, page table walks must traverse multiple GPU modules to complete multi-level translation, incurring substantial performance degradation due to the inherent non-uniformity of MCM-GPU architectures.

In this paper, we propose Victim Page Table Walk Cache (V-PWC), a novel design that improves address translation performance for locality-sparse workloads by effectively increasing the usable PWC capacity. V-PWC repurposes the L2 TLB as a victim cache for PWC entries evicted due to capacity limitations, while leveraging a pressure-aware static re-reference interval prediction (SRRIP) based replacement policy to balance the hit rates of the L2 TLB and the PWC. V-PWC slightly extends TLB entries to store page table walk information, enabling evicted PWC entries to be retained and reused. For applications with high PWC pressure, these victim PWC entries are protected in the L2 TLB, as the native L2 TLB hit rate is already low even when fully dedicated to direct translations. Conversely, for applications with low PWC pressure, victim PWC entries are few while not protected, ensuring that L2 TLB performance remains unaffected. Our experimental results using 21 GPU workloads demonstrate that V-PWC improves performance by 41% on average (up to 75%) for locality-sparse applications while maintaining the performance for locality-dense applications.

CCS Concepts

• **Software and its engineering** → **Virtual memory**; • **Hardware** → *Multi-chip modules*; • **Computer systems organization** → *Parallel architectures.*

Keywords

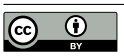
Multi-chip-module GPU, Address translation, Page walk cache, Translation lookaside buffers

1 Introduction

As Moore's law slows, modern GPUs increasingly rely on advanced packaging to scale performance beyond monolithic integration. MCM-GPUs integrate multiple GPU chiplets within a single package, increasing aggregate compute capability and memory bandwidth while reducing the yield and cost challenges of large dies [3, 38, 58]. Meanwhile, large-scale graph analytics, scientific simulations, and deep learning workloads continue to expand their memory footprints [55, 56], placing growing pressure on GPU virtual memory systems [41]. Consequently, address translation efficiency has become a critical factor for MCM-GPU performance [19].

Modern MCM-GPU virtual memory systems rely on a combination of TLBs and PWCs to accelerate virtual-to-physical address translation. TLBs store recently used virtual-to-physical page translations and are highly effective at capturing translation locality for applications with small or moderate memory footprints. However, when a TLB miss occurs, the GPU must initiate a multi-level page table walk to resolve the translation. To mitigate the cost of these

*Corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License.
PACT '26, Chicago, IL, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2915-7/2026/10
https://doi.org/10.1145/3838684.3846831

page table walks, PWCs cache intermediate page table entries accessed during translation, enabling subsequent walks to skip one or more levels of the page table hierarchy [6, 10]. The original design philosophy of this two-level translation mechanism is to exploit the TLB to efficiently serve applications with high translation locality, while relying on the PWC to accelerate page table walks when TLB misses occur.

However, growing memory footprints in MCM-GPUs increasingly undermine this conventional translation hierarchy. Many emerging workloads span hundreds of gigabytes, far exceeding the reach of on-chip translation structures. In particular, locality-sparse workloads rapidly touch many pages across a wide address range, reducing translation reuse and stressing both the TLB and PWC. Under these conditions, the TLB often exhibits near-zero hit rates, while the PWC frequently fails to retain useful upper-level page-table entries. Consequently, many translations fall back to full page table walks that traverse multiple GPU modules. Because page tables and memory resources are distributed across chiplets [51, 52], these cross-module walks introduce high latency and consume interconnect bandwidth, making address translation a major bottleneck for locality-sparse applications.

In this paper, we propose V-PWC, a novel address translation mechanism designed to improve performance for locality-sparse workloads on MCM-GPUs by effectively increasing the usable capacity of the page table walk cache. Rather than simply expanding PWC storage or introducing new translation structures that cause unacceptable hardware cost, V-PWC repurposes the existing L2 TLB as a victim cache for PWC entries evicted due to capacity limitations, allowing these entries to be retained and reused. V-PWC augments TLB entries with lightweight metadata to encode page table walk information, allowing the translation hierarchy to preserve intermediate translation states that would otherwise be discarded. To balance the competing demands of TLB translations and page table walk entries, V-PWC proposes a pressure-aware replacement policy based on Static Re-Reference Interval Prediction (SRRIP) [25], which dynamically manages different entries in L2 TLB according to the translation behavior of the workload.

V-PWC integrates page table walk information into the address translation pipeline by extending the L2 TLB entry format and supporting parallel lookup paths for TLB translations and page walk entries. In particular, each L2 TLB way is augmented with lightweight metadata to distinguish whether an entry represents a direct virtual-to-physical translation or an intermediate page table entry corresponding to an upper level of the page table. During address translation, the L2 TLB lookup is extended to simultaneously probe both types of entries, enabling the hardware to identify the longest valid match available. If a conventional TLB entry is found, translation completes immediately as in traditional designs. Otherwise, if a redirected page table walk entry is hit, the hardware skips the corresponding upper-level page table accesses and resumes the walk from a lower level, effectively short-circuiting the translation process. By integrating page table walk reuse directly into the L2 TLB lookup path, V-PWC reduces redundant multi-level page walks across GPU modules while preserving the fast translation latency of standard TLB hits.

V-PWC further adapts L2 TLB replacement policy to account for reuse characteristics of terminal translations and page table walk entries. Building on the widely used SRRIP policy, V-PWC introduces lightweight, semantic-aware priority control to selectively protect victim PWC entries when they are most beneficial. For applications with high PWC miss rates, where L2 TLB provides limited benefit for direct translations, V-PWC prioritizes retaining victim PWC entries in the L2 TLB, preserving page table walk progress and reducing redundant multi-level traversals. In contrast, for applications with low PWC miss rates and small memory footprints, both TLB and PWC naturally achieve high hit rates, resulting in few victim PWC entries. In these cases, V-PWC does not apply protection, ensuring that L2 TLB replacement behavior and hit rate remain unaffected. This adaptive SRRIP-based policy allows V-PWC to dynamically balance translation and page walk reuse, improving performance for large-memory-footprint workloads without compromising efficiency for smaller working sets.

In summary, we make the following major contributions:

- We identify address translation as a critical performance bottleneck for locality-sparse workloads on MCM-GPUs, and demonstrate that conventional TLB and page table walk cache designs are ineffective under such conditions due to near-zero TLB hit rates and limited reuse of high-level page table entries.
- We propose V-PWC, a unified translation design that repurposes the L2 TLB as a victim cache for evicted page table walk entries, effectively increasing usable PWC capacity without introducing new hardware structures.
- We design a lightweight L2 TLB extension that enables parallel lookup and pressure-aware replacement of direct virtual-to-physical translations and page table walk entries, achieving low-latency address translation while dynamically balancing translation and page walk reuse across various workloads.
- We quantitatively evaluate V-PWC and show that it improves performance by 41% on average (up to 75%) for locality-sparse applications, while maintaining performance for locality-dense applications. Across all evaluated workloads, V-PWC delivers an average performance improvement of 20%.

2 Background

2.1 MCM-GPU Architecture

Figure 1 illustrates the typical architecture of an MCM-GPU with 4 modules, with the components of a single module shown in detail. MCM-GPU comprises multiple GPU Modules (GPMs) linked through a high-speed internal interconnect, for which we assume a ring-based topology in this paper. Each GPM is equipped with its own local memory and can access the remote GPM's memory. This architecture introduces non-uniform memory access (NUMA) effects, where remote memory accesses incur higher latency, while local memory accesses have lower latency [3, 4]. Internally, a GPM contains several Compute Units (CUs), i.e. Streaming Multiprocessors (SMs) in NVIDIA's terminology. Each CU is composed of several Single Instruction Multiple Data (SIMD) units. SIMD units in a CU share an L1 Cache and an L1 TLB. A coalescer combines memory access requests that fall on the same cache line before looking up the L1 cache. Furthermore, each GPM has an L2 Cache and an L2 TLB that are shared across all CUs. The private L1 TLBs

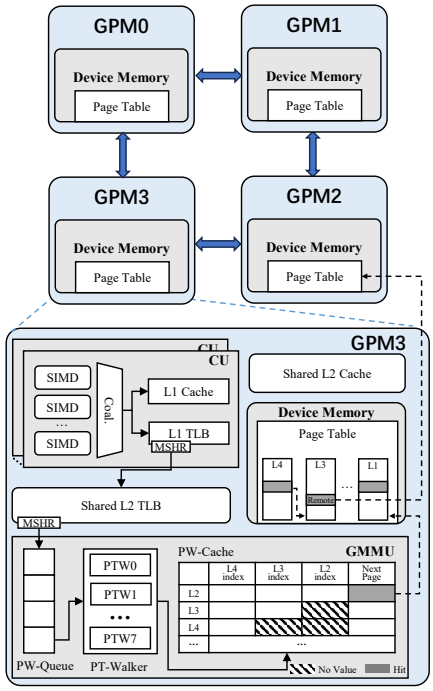


Figure 1: MCM-GPU architecture. *MCM-GPU comprises multiple GPU modules integrated together with in-package interconnects and all modules work together as a single GPU.*

and shared L2 TLB are equipped with a Miss Status Holding Register (MSHR) to combine the translation of the same virtual address. The GPU Memory Management Unit (GMMU) is responsible for handling page table walks from local translation requests [49]. The GMMU consists of: (i) a PW-Queue to buffer incoming address translation requests; (ii) a PT-Walker that supports multi-threaded page table walking; and (iii) a PW-Cache for caching recently used upper-level page table entries.

2.2 Virtual Memory Management

Address Translation Workflow in MCM-GPU. Figure 1 also illustrates the address translation workflow within the MCM-GPU. Address translation requests generated by SIMDs are first coalesced and then used to access the L1 Cache and L1 TLB in parallel (assuming that the L1 Cache is virtually indexed). Upon an L1 TLB miss, the request checks the MSHR; if the request misses in the MSHR, it is forwarded to the shared L2 TLB. Similarly, requests that miss in L2 TLB are also filtered by an MSHR. If a request misses in both L2 TLB and MSHR, it is sent to GMMU and triggers a Page Table Walk (PTW). Address translation requests are initially buffered in the PW-Queue and wait for the available PT-Walker thread. During the PTW, the PT-Walker first checks the PWC with the address translation request. If there are no matches in the PWC, the PT-Walker traverses the page table hierarchy with multiple memory accesses. In MCM-GPUs, page table pages can be mapped to the physical memory of a remote GPM, requiring cross-chiplet interconnect

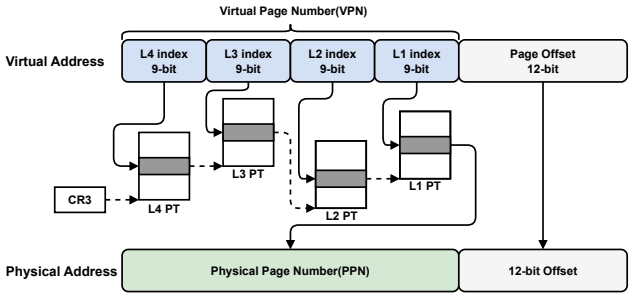


Figure 2: Four-level radix-tree page table walk with 4KB pages. *The VPN is partitioned into four 9-bit index fields (L4 to L1) to traverse the page table hierarchy and get the PPN combined with a 12-bit page offset.*

traffic. These remote accesses drastically increase the latency of address translation, exacerbating the performance bottleneck caused by TLB misses.

Page Table Walk. In multi-level page table structures, each level contains page table entries (PTEs) that store pointers to the subsequent level. Although this hierarchical design significantly optimizes memory space utilization, it inherently sacrifices address translation efficiency. Memory management units (MMU) must execute the costly PTW operation to traverse a whole page table. Figure 2 illustrates the virtual address decomposition and the mechanics of a four-level radix-tree PTW in a system with 4KB pages. In the relevant sections of this paper, we use a 48-bit virtual address. The virtual address is partitioned into a Virtual Page Number (VPN) and a 12 bit page offset. The VPN is further segmented into four 9-bit indices ($L4$ to $L1$). The walk commences using the base address of the top-level table, which is stored in the CR3 register. The $L4$ index serves as an offset within this table to retrieve the PTE pointing to the base address of the $L3$ table. This procedure is repeated iteratively: the $L3$ index indexes the second-level table, followed by the $L2$ and $L1$ levels. Upon reaching the leaf-level PTE, the Physical Page Number (PPN) is extracted. Finally, the complete physical address is constructed by concatenating the PPN with the original page offset.

Page Walk Cache. In virtual memory management, PWC is exploited to reduce the latency incurred by page table walks. PWC caches the upper-level (L4-L2) page table entries to reduce the number of long-latency page table walks. In this paper, we adopt the Unified Translation Cache (UTC) [23] as shown in Figure 1. In UTC, entries are tagged by indices in the virtual address and different level entries are mixed in the same cache. Specifically, an L4 entry is tagged by L4 index, an L3 entry is tagged by L4 and L3 indexes and an L2 entry is tagged by L4, L3 and L2 indexes. The PWC will return the longest prefix match of the virtual address. The Next Page filed in UTC stores the PPN that can be used to generate the physical address of next level page table. A hit within the PWC allows the translation logic to bypass one to three intermediate page table levels, reducing the number of memory accesses. In the ideal case, a PTW requires only a single memory access (i.e. if a

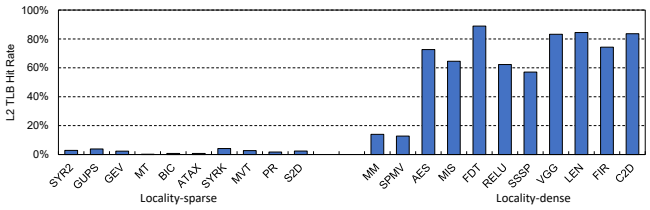


Figure 3: L2 TLB Hit Rate. *12 workloads exhibit L2 TLB hit rates close to 0%, indicating that most cached virtual-to-physical translations are never reused.*

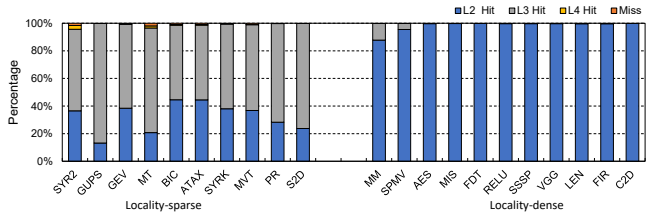


Figure 4: PWC Hit Rate. *The PWC L2 hit rate of 10 applicationsin falls below 50%, implying that more than half of the PTWs must perform at least two levels of memory-resident page table lookups.*

VPN matches an L2 entry in PWC, the walker can directly accesses the leaf-level page table).

3 Motivational Study

3.1 Insufficient Translation Reach

Memory-intensive GPU workloads place unprecedented pressure on the virtual memory subsystem. To characterize address translation efficiency for large-memory-footprint applications, we measure the L2 TLB hit rate and the hit distribution across different PWC levels. We use MGPUSim [59] to simulate an MCM-GPU with four GPU modules. The detailed configuration is described in Section 6. Our evaluation shows a sharp decline in L2 TLB effectiveness for different applications. Figure 3 plots the L2 TLB hit rates of the evaluated applications. Notably, 12 workloads exhibit L2 TLB hit rates close to 0%, indicating that most cached virtual-to-physical translations are never reused. These results suggest that, for large-scale workloads, the active translation working set far exceeds the reach of the L2 TLB, rendering its capacity largely ineffective.

To mitigate the substantial latency penalty incurred by these frequent L2 TLB misses, translation requests traverse the PWC to match intermediate PTEs. This mechanism aims to short-circuit the hierarchical traversal by bypassing one to three levels of sequential memory accesses. Figure 4 displays the hit rates for each PWC tier used for our applications. While workloads such as MM, and SPMV demonstrate low L2 TLB hits, they achieve around 90% hit rates at the L2 tier of the PWC, meaning the majority of their PTWs are resolved with just a single memory access. However, for the remaining 10 applications among the 12 with low L2 TLB hit rates, the L2 hit rate in PWC falls below 50%. This implies that more than half of the PTWs in these workloads must perform at least two levels of memory-resident page table lookups. In the context of

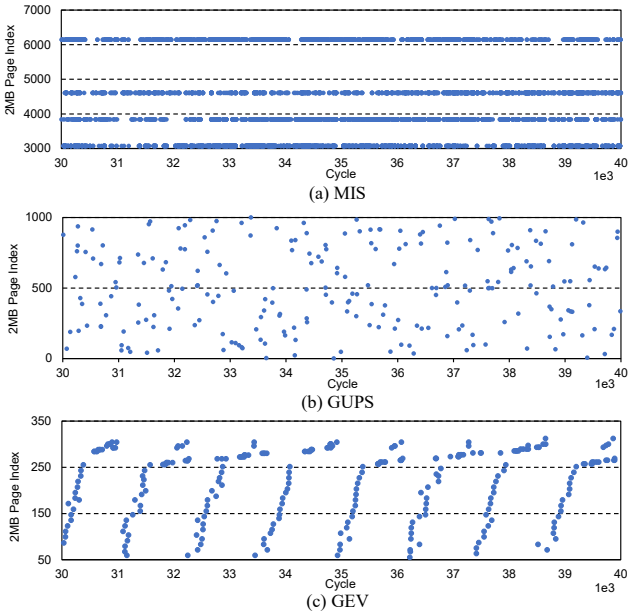


Figure 5: Memory access patterns of MIS, GUPS, and GEV. *MIS exhibits a locality-dense access pattern,repeatedly accessing a small set of 2MB regions. In contrast, GUPS and GEV display a locality-sparse pattern, rapidly touching many 2MB regions across a wide address range.*

MCM-GPU architectures, these memory accesses are particularly costly as they frequently involve high-latency remote page table traversals across chiplet interconnects. Characterization analysis of address translation efficiency in large-scale GPU applications reveals that conventional L2 TLB and PWC architectures are increasingly inadequate for MCM-GPUs.

> ***Insight #1.*** *For many large-scale GPU applications, the L2 TLB and PWC are insufficient to sustain efficient virtual-to-physical address translation.*

3.2 Memory Access Patterns

As shown above, GPU applications exhibit widely varying L2 TLB and PWC hit rates. Interestingly, these hit rates are not solely determined by memory footprint. For example, compared to S2D, MIS has a much larger memory footprint (12 GB vs. 512 MB) yet achieves a substantially higher L2 TLB hit rate. To understand the underlying reasons, we examine the memory access characteristics of GPU applications. It is well known that GPUs achieve high throughput by concurrently executing multiple threads. Threads within the same warp execute the same instruction on different data elements, necessitating simultaneous access to multiple addresses. Ideally, these addresses target contiguous and localized data, allowing the hardware to perform address coalescing and access only a single page [14, 53]. However, the increasing memory footprint and irregular access patterns of large-scale workloads can undermine coalescing efficiency. Together, these factors expand each warp's dynamic working set, increase the number of distinct

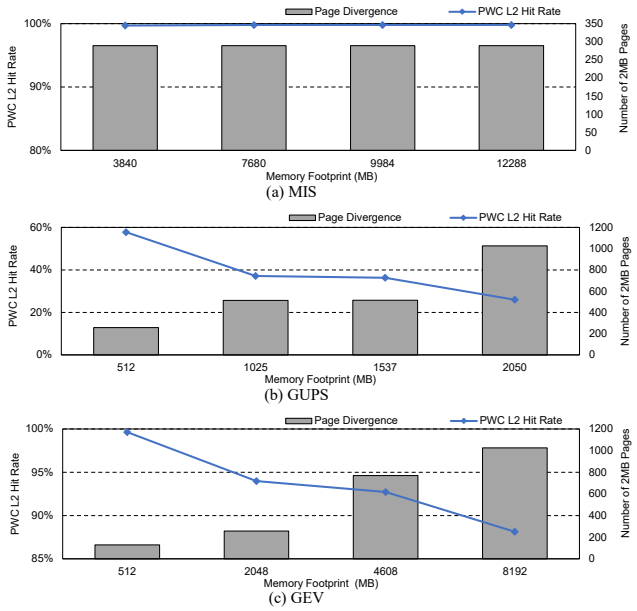


Figure 6: Performance of MIS, GUPS, and GEV for different memory footprints. *As the memory footprint increases, the PWC L2 hit rate decreases and page divergence increases in locality-sparse applications (GUPS and GEV). In contrast, locality-dense applications (MIS) remain largely insensitive to memory footprint scaling.*

pages touched, and trigger more frequent address translations[21]. For these workloads, this warp-level page divergence significantly degrades PWC L2 entry hit rates.

To analyze the impact of various memory access patterns on PWC L2 hit rates under large memory footprint conditions, we analyzed the address translation requests generated by three representative benchmarks. Figure 5 visualizes the memory access behavior of MIS, GUPS and GEV within a single GPM in an MCM-GPU, at a granularity of 2MB (i.e., the coverage of a single PWC L2 entry). The x-axis shows an execution window from 30,000 to 40,000 cycles, and the y-axis denotes the accessed 2MB region index during this interval. MIS exhibits a locality-dense access pattern, repeatedly accessing a small set of 2MB regions over an extended period. This sustained locality allows the PWC to exploit temporal and spatial reuse, resulting in high L2 hit rates. In contrast, GUPS and GEV display a locality-sparse pattern, rapidly touching many 2MB regions across a wide address range. Such broad access easily exceeds the PWC's reach, causing L2 entries with long reuse distances to be evicted prematurely. Consequently, these workloads experience frequent PWC L2 misses and significantly higher translation overhead. Overall, these results suggest that locality-sparse access patterns, common in irregular GPU workloads, are a primary driver of diminished PWC L2 hit rates under large working sets.

To further analyze how memory footprint affects virtual memory system performance, we vary the memory footprints of the three representative applications by scaling their input sizes and measure the resulting PWC behavior. We define page divergence as the number of unique 2MB pages accessed by a GPM. Figure 6

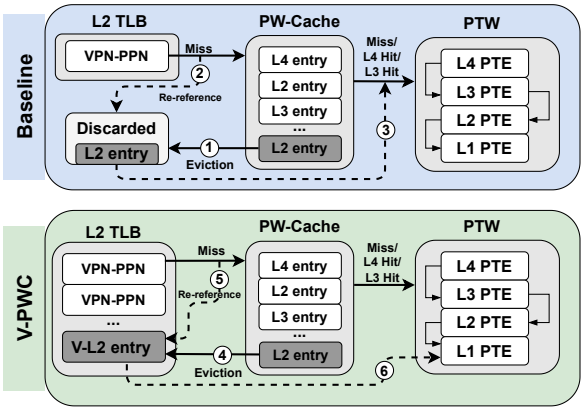


Figure 7: Workflow in baseline and V-PWC. *V-PWC converts the L2 entries evicted from the PWC into Victim L2 (V-L2) entries which are stored in L2 TLB. Subsequent translation requests that would otherwise miss in the PWC can instead hit in the L2 TLB and reuse these entries.*

reports the PWC L2 hit rate and page divergence for GUPS, GEV, and MIS across different memory footprints. As memory footprint increases, the locality-sparse workloads, i.e., GUPS and GEV, exhibit a pronounced increase in page divergence, which causes a substantial drop in the PWC L2 hit rate. In contrast, the locality-dense workload MIS shows nearly unchanged page divergence and PWC L2 hit rate as the footprint scales. This difference arises because, for regular workloads like MIS, the effective working set observed by each GPU module is largely determined by the number of concurrently active thread blocks, which is bounded by on-chip resources. Although the total application footprint grows with larger inputs, each module executes a fixed number of thread blocks at a time, so the per-module active working set and thus page divergence remains stable. These results show that a larger memory footprint does not necessarily degrade virtual memory performance. Instead, the impact of footprint on PWC effectiveness depends strongly on the workload's memory access pattern.

> ***Insight #2.*** *Memory footprint and memory access patterns jointly determine the PWC hit rate.*

4 Design Overview of V-PWC

In this paper, we propose V-PWC, a mechanism that leverages underutilized resources within the address translation hierarchy to expand the translation reach of PWC. Based on the key observation that L2 TLB is severely under-utilized with large memory footprint applications, V-PWC repurposes the L2 TLB to house the victim entries of the PWC. This design allocates additional capacity for caching lower-level page table entries, thereby significantly reducing the frequency of high-latency page table walks, especially in MCM-GPUs.

In particular, our observations reveal a significant resource mismatch. While the L2 TLB capacity remains underutilized, the PWC suffers from frequent L2 entry misses due to being capacity constrained, ultimately triggering prohibitive page table walk penalties.

To exploit this, we redirect the L2 entries evicted from PWC to the L2 TLB. Furthermore, we introduce a pressure-aware replacement policy to adapt translation pressure and dynamically protect victim L2 entries evicted from PWC within the L2 TLB. To enhance the lookup efficiency for redirected L2 entries, our design augments the L2 TLB to support parallel matching.

Figure 7 illustrates the workflow both in the baseline and V-PWC. Here, we focus on the virtual-to-physical translation requests that miss in L2 TLB. In the baseline, translation requests that miss in the L2 TLB traverse the PWC to retrieve intermediate page table entries, thereby reducing the latency of a full PTW. However, entries evicted from the PWC are simply discarded ①. As illustrated by the dashed line, any subsequent re-reference ② to these evicted entries necessitates a costly, multi-level PTW ③. As shown before, this behavior is prevalent in GPU applications with locality-sparse characteristics, where the limited PWC capacity is insufficient to retain L2 entries that cover the working set.

In contrast, V-PWC converts the L2 entries evicted from the PWC into Victim L2 (V-L2) entries and stores them in the L2 TLB ④. To support this, V-PWC slightly extends the L2 TLB entry format so that each way can hold either a conventional VPN-to-PPN translation or a redirected V-L2 entry encoding PWC L2 information. We focus on caching PWC L2 entries because our analysis shows that, for the target workloads, existing PWC L3 entries already provide sufficient translation reach, whereas PWC L2 entries suffer frequent capacity-induced evictions. Intuitively, an L3 entry covers a much larger region than an L2 entry, e.g., 1 GB vs. 2 MB, making L2 entries more vulnerable under large working sets. As application memory footprints continue to grow, V-PWC can be naturally extended to cache other PWC entry types such as L3 or even L4 using the same principle. By caching these victim entries in the L2 TLB, our design effectively improves the PWC capacity. Therefore, subsequent translation requests requiring these previously evicted L2 entries can now hit in the L2 TLB ⑤, effectively short-circuiting the translation process and significantly reducing redundant remote memory accesses in MCM-GPUs ⑥.

5 Detailed Design of V-PWC

This section first describes the V-L2 entry design, which enables the L2 TLB to store either conventional TLB entries or redirected PWC entries, and then presents the pressure-aware replacement policy and the parallel matching scheme proposed for L2 TLB.

5.1 V-L2 Entry Design

In this subsection, we describe the V-L2 entry format, which slightly extends a conventional L2 TLB entry to store either an intermediate page-table entry (V-L2) or a terminal virtual-to-physical translation (V-TLB). Figure 8 illustrates (i) the structure of traditional TLB entry and (ii) the structure of new TLB entry and two types of entries that conform to the new TLB structure. To simplify, each index level is assumed to be 9 bits. A traditional TLB entry comprises a tag field and a corresponding physical address field. To distinguish between V-TLB and V-L2 entries in new TLB entry structure, we augment each L2 TLB entry with a 1-bit type-indicator flag, i.e., a value of 0 denotes a TLB entry, while 1 identifies a redirected V-L2 entry. As mentioned in subsection 2.2, entries of PWC are tagged by prefix of

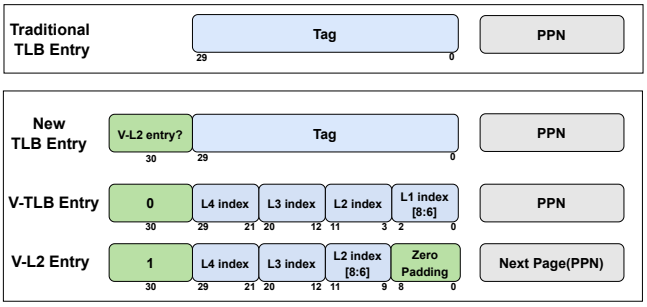


Figure 8: Traditional TLB entry and new TLB entry structure. *An additional type indicator bit is appended to each entry to distinguish between V-TLB and V-L2 entries. For a V-L2 entry, the set index is derived from the lower bits of the L2 index (i.e., [5:0]), while the remaining bits are zero-padded and stored in the tag field.*

VPN (L4 index, L3 index and L2 index) to store PPN, which is similar to a standard TLB using the VPN for indexing PPN. Exploiting this structural similarity, V-PWC converts evicted PWC entries into the TLB-compatible V-L2 format by zero-padding the unused fields. A key challenge is that PWC L2 entries do not include a valid set index. If we applied the standard TLB set-indexing scheme, these redirected entries would alias to a small number of sets, causing severe contention. We address this with a simple index-remapping scheme that derives the set index for V-L2 entries from the lower bits of the L2 index, while padding the missing portion of the tag with zeros. In particular, in our 512-entry, 8-way L2 TLB (64 sets), conventional TLB indexing uses the lower 6 bits of the VPN to select a set. For V-L2 entries, we instead use the lower 6 bits of the PWC L2 index to select the set. The remaining upper bits form the tag for the V-L2 lookup. This remapping distributes redirected entries across the L2 TLB sets and enables concurrent matching of V-TLB and V-L2 entries during translation.

5.2 Pressure-aware Replacement Policy

To optimize the retention of redirected metadata, we extend the state-of-the-art SRRIP policy. Our objective is to improve the residency of V-L2 entries in the L2 TLB when page table walks become a dominant source of translation latency. V-PWC therefore introduces a lightweight protection mechanism that is enabled only when recent PWC behavior indicates sustained capacity pressure. Each GPM monitors the reuse behavior of PWC L2 entries and identifies phases in which these entries are frequently missed and evicted. During such phases, redirected V-L2 entries are treated as protected entries in the L2 TLB, allowing them to remain resident longer and serve future page walks. When PWC pressure is low, V-L2 entries receive no special protection and follow the normal SRRIP replacement behavior, preserving L2 TLB capacity for conventional virtual-to-physical translations. The SRRIP policy maintains a re-reference prediction value (RRPV) for each entry, where a smaller RRPV indicates a higher likelihood of near-term reuse. Empty entries are initialized to the maximum RRPV value, denoted as MAX.

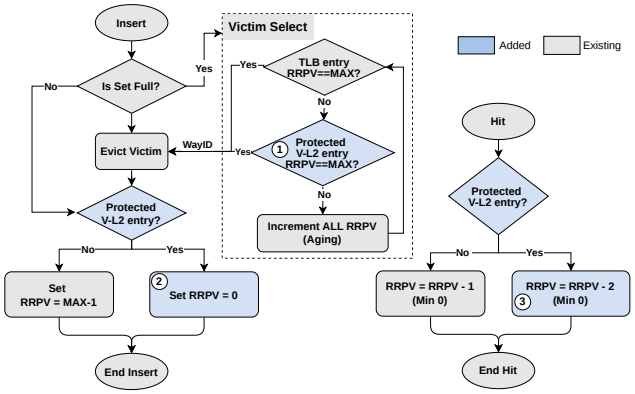


Figure 9: Flowchart of TLB replacement policy. *To extend the retention time of protected V-L2 entries, the victim selection, entry insertion, and RRPV update policies upon a hit within the SRRIP framework are modified.*

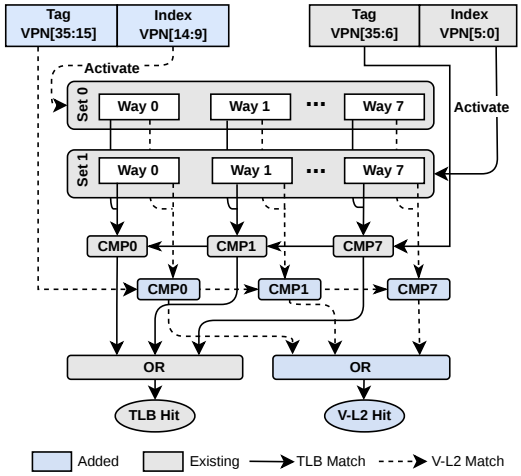


Figure 10: Parallel Matching Structure. *The VPN is partitioned to index V-L2 entries, and an additional comparator array is introduced to enable parallel matching of V-L2 entries alongside conventional TLB lookups (CMP denotes the comparator).*

Figure 9 illustrates how the SRRIP policy handles insertions and hits in our approach, with the specific changes highlighted in blue. There are three major modifications as follows:

- **Victim selection**: During the victim selection process, the traditional SRRIP algorithm defaults to evicting the first candidate encountered with the MAX RRPV. If there is no entry with an RRPV of MAX, RRPV for each entry will be incremented, which is called aging. In contrast, our policy incorporates a protection-aware bypass logic. The replacement unit intentionally skips V-L2 entries currently in the protected state when searching for a replacement candidate. A protected V-L2 entry is only considered for eviction as a last resort, specifically when no TLB entries are available for replacement within the same set ①.
- **Entry insertion**: In standard SRRIP policy, the RRPV of a new entry is initialized to MAX - 1. When V-PWC inserts a V-L2 entry while protection is enabled, its RRPV is initialized to 0 ②. This near-immediate re-reference probability ensures the entry is placed at the top of the replacement hierarchy, thereby delaying its eviction.
- **RRPV update upon hit**: Upon a hit, the RRPV of a protected V-L2 entry is decremented by 2 ③ (as opposed to the standard decrement of 1). This RRPV update logic effectively prioritizes V-L2 entries for long-term residency, allowing them to remain in L2 TLB across multiple eviction cycles.

5.3 Parallel Matching Scheme

In GPUs, translation requests that miss the TLB are forwarded to the PWC to search for upper-level PTEs. If a request misses the PWC's L2 entries, the page table walker must traverse multiple PT levels, which in MCM-GPUs often incurs cross-chiplet accesses over the interconnect to reach remote page table pages. In V-PWC, a straightforward approach would be to perform an additional L2 TLB check for a corresponding V-L2 entry after consulting the PWC. However, for workloads with high PWC L2 miss rates, this serialized lookup sequence would increase the translation critical path and impose extra access pressure on the L2 TLB. To avoid this, V-PWC enhances the L2 TLB to support a parallel matching scheme, as illustrated in Figure 10.

We first introduce the matching workflow in a traditional TLB. Upon an address translation request, the VPN is logically partitioned into a tag and an index. The index bits are decoded to activate a specific set. The incoming tag is then compared against all stored tags within that set simultaneously via a comparator (CMP). Specifically, for each way, a bit-wise comparison is performed using XNOR gates. The outputs of these XNOR gates for a given way are fed into a multi-input AND gate tree; if all bits match, the CMP asserts a '1' for that way. Subsequently, the outputs from all way-comparators are aggregated by an OR gate. If any match is detected, the overall hit signal is asserted, and the corresponding PPN stored in the matching way is returned. In V-PWC, we replicate and modify the VPN decomposition logic to derive the tag and index for V-L2 entry matching. By augmenting the activation circuitry, the corresponding sets for V-L2 lookups can be activated simultaneously with the standard TLB sets. In scenarios where the V-L2 index differs from the TLB index, both target sets are activated concurrently for parallel lookup. To facilitate this, we incorporate a secondary array of XNOR-based comparison logic with specific modifications. Specifically, the incoming tag is matched against the stored tag prefixes across all ways within the activated set. These comparison results, augmented by a type-indicator flag, are then processed by an additional multi-input AND gate to determine the V-L2 match status. Furthermore, to handle cases where distinct indices map to the same physical set, our design enables the newly added XNOR arrays to access the stored tags in parallel with the original logic. It is important to note that this supplementary logic operates independently of the baseline circuitry, ensuring no interference with the critical path of standard TLB lookups. Similar to PWC, V-PWC implements a hit-priority logic that favors the virtual-physical translation hit. If a TLB hit occurs, the hardware

extracts the PPN and any simultaneous V-L2 hit is ignored to avoid redundant processing.

5.4 TLB Shootdowns

MCM-GPUs initiate TLB shootdown operations to maintain translation coherence during events such as page migrations. In V-PWC, if a TLB shootdown occurs, the L2 TLB can perform TLB entry lookups and invalidation operations just as in the baseline implementation. If invalidation of the intermediate page table is required, V-PWC forwards invalidation requests to the L2 TLB to invalidate the intermediate page table entries stored in the TLB. The additional TLB invalidation operations can be performed in parallel with the PWC shootdowns, resulting in no additional latency.

5.5 Hardware Overhead

In V-PWC, we augment an existing MCM-GPU with three lightweight components, i.e., a 1-bit flag in each L2 TLB entry to distinguish V-TLB entries from V-L2 entries, a per-GPM monitor that tracks the PWC L2 hit rate, and simple logic to enable parallel matching. Adding the V-L2 flag incurs only a 1.25% storage overhead for an 80-bit entry (512 additional bits in total for a 512-entry L2 TLB, i.e., 5KB). The PWC monitor computes the fraction of L2 hits among PWC accesses which requires only two 10-bit counters and a comparator. Finally, as discussed in Section 5.3, the parallel matching logic is straightforward and only takes 1.4% area overhead compared to a GPU L2 TLB according to CACTI [5] measurement. Our CACTI evaluation shows a power increase of 4.4 mW, or 0.025% of the AMD R9 Nano's TDP (175 W). However, V-PWC's system-level power savings offset this increase. By improving the intermediate page table hit rate, V-PWC reduces high-power cross-chip and DRAM accesses, which are energy-intensive in MCM-GPUs.

6 Experimental Setup

Table 1: Baseline MCM-GPU Configuration

Module	Configuration
Number of GPU modules	4
Number of CUs	64 per module. 256 in total
L1 Vector Cache	16 KB, 4-Way, 16 MSHRs
L1 Inst Cache	32 KB, 4-Way, 16 MSHRs
L1 Scalar Cache	16 KB, 4-Way, 16 MSHRs
L2 Cache	4 MB, 16-way, 64 MSHRs
DRAM	1TBps, 100ns latency
L1 TLB	64 entries, fully assoc., 4 MSHRs. 1 cycle lookup latency, private to CU.
L2 TLB	512 entries, 8-Way, GPU chip-shared. 10 cycle lookup latency, 64 MSHRs. SRRIP replacement policy
GMMU Page Table Walker	8 shared PTWs. 100-cycle latency per level.
GMMU PW-Queue	64 entries.
GMMU Page Walk Cache	32 entries fully assoc. 10-cycle latency, LRU.
Inter-chip bandwidth	768 GB/s ring, 32 cycle latency

Simulated System. In the evaluation, we simulate an AMD GCN3-like MCM-GPU and integrate our proposed V-PWC using a cycle-level simulator MGPUSim [59]. Similar to previous work [35, 51], to model the full address translation process, we modified and extended MGPUSim by adding GMMU with PW-Queue and Page Walk Cache. Our simulated MCM-GPU comprises four GPMs, each equipped with its own device memory and capable of accessing another module's memory. Table 1 details the parameters for each GPU module. Each module contains 64 CUs, totaling 256 CUs across the entire system. Each CU contains a private, 64-entry fully associative L1 TLB with a 1-cycle lookup latency and the LRU replacement policy. A larger, chip-shared 512-entry 8-way L2 TLB handles L1 misses with a 10-cycle latency, utilizing the SRRIP placement policy and 64 MSHRs to manage concurrent requests. The GMMU employs 8 shared hardware PTWs [35, 50] and a 64-entry PW-Queue is utilized to buffer outstanding walk requests. We adopt a 4-level page table and 4KB memory page size and each level of the page table walk incurs a significant penalty of 100 cycles [22, 35]. The GMMU is equipped with a 32 entry fully associative page walk cache with a 10-cycle lookup latency[51] and different sizes are evaluated as the sensitivity analysis. In this paper, we adopt the Unified Translation Cache (UTC) design for PWC, because of the similar structures of UTC and L2 TLB. All GPMs are interconnected via a low-latency, high-bandwidth network forming a ring topology [46]. This interconnect provides a total bandwidth of 768 GB/s with a 32-cycle transfer latency per direction[3, 51].

Table 2: List of Applications

Category	Abbr.	Application	Benchmark Suite	Memory Footprint
Locality-Sparse	SYR2	syr2k	Polybench	12288 MB
	GUPS	gups	MAFIA	2050 MB
	GEV	gesummv	Polybench	8192.38 MB
	MT	matrixtranspose	AMD SDK	4608 MB
	BIC	bicg	Polybench	6400.63 MB
	ATAX	atax	Polybench	6400.47 MB
	SYRK	syrk	Polybench	3200 MB
	MVT	mvt	Polybench	6400.63 MB
	PR	pagerank	Hetero-Mark	4096.38 MB
	S2D	stencil 2-D	SHOC	512.02 MB
Locality-Dense	MM	matrixmultiplication	AMD SDK	3072.01 MB
	SPMV	spmv	SHOC	8508.61 MB
	AES	aes	Hetero-Mark	2400 MB
	MIS	mis	Pannotia	12288 MB
	FDT	fdtd2d	Polybench	3072.01 MB
	RELU	relu	DNN-Mark	5120 MB
	SSSP	sssp	Pannotia	5120.01 MB
	VGG	vgg16	DNN-Mark	2125.54 MB
	LEN	lenet	DNN-Mark	4845.45 MB
	FIR	fir	Hetero-Mark	9216 MB
	C2D	convolution2d	Polybench	8192 MB

Applications. Table 2 shows the details of the applications and their memory footprints. We select large memory footprint applications from Polybench [42], SHOC [16], Hetero-Mark [60], DNN-Mark [17], AMD SDK [20], Pannotia [15], and MAFIA [28], and classify them as locality-sparse or locality-dense based on their memory access patterns. We use the MGPUSim implementations of these applications, consistent with prior work [19, 27, 35, 51]. To provide a holistic evaluation, we include workloads spanning a wide range of memory footprints, access patterns, L2 TLB hit rates, and PWC L2 hit rates.

7 Evaluation

This section compares the performance of following designs:

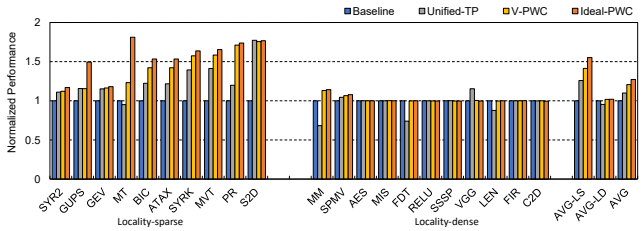


Figure 11: Overall performance normalized to baseline. *V-PWC improves performance by 41% on average for the locality-sparse applications and maintains the performance of the locality-dense applications.*

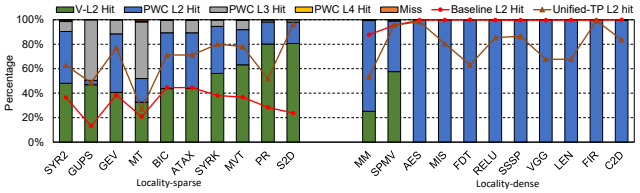


Figure 12: PWC hit rate in system with V-PWC. *V-PWC achieves an average improvement of* 51% *(reaching up to* 74%*) in PWC L2 hit rate in locality-sparse applications.*

- **Baseline:** An MCM-GPU with a conventional virtual memory system, consisting of per-CU L1 TLBs, a per-module L2 TLB, and a PWC that caches upper-level PT entries to reduce page walk latency.
- **V-PWC:** Our proposed design, which repurposes the L2 TLB as a victim cache for PWC entries evicted due to capacity limitations, enabling evicted L2-tier walk entries to be retained and reused to short-circuit future page table walks.
- **Ideal-PWC:** The MCM-GPUs with ideal PWC, i.e., all L2 TLB misses hit in the PWC L2 entries, effectively reducing each page table walk to a single memory access. This represents an optimistic upper bound.
- **Unified-TP:** An MCM-GPU using a unified structure that co-locates PWC and TLB entries in a single shared capacity pool. An application-aware replacement policy is proposed to dynamically adjust the allocation between TLB entries and PWC entries, adapting to workload behavior [1].

7.1 Overall Performance

We compare the V-PWC performance with Ideal-PWC and Unified-TP. Figure 11 shows speedups of V-PWC, Ideal-PWC, and Unified-TP normalized to the baseline. On average, V-PWC achieves a 1.20× performance improvement across all evaluated applications, capturing a significant portion of the 1.27× gain offered by the Ideal-PWC, while Unified-TP only improves by 9%. The performance benefits are particularly pronounced in locality-sparse workloads with low PWC L2 hit rates. For instance, S2D, SYRK, PR, and MVT attain speedups exceeding 1.5× over the baseline. Notably, V-PWC delivers an average performance boost of 41% for the 10 locality-sparse applications and Unified-TP achieves an average 25% performance improvement. Conversely, V-PWC maintains baseline performance for locality-dense applications, despite the additional write traffic generated by V-L2 entries. However, Unified-TP simply co-locates TLB and PWC entries within the same structure, leading to capacity contention between TLB entries and PWC entries. In cases where frequently updated TLB entries encroach upon the PWC's available capacity, an overall performance improvement may still occur if the TLB's gains outweigh the PWC's losses, as observed in VGG. However, in a broader range of applications, this resource contention restricts potential speedups or even leads to net performance degradation. Our mechanism effectively enhances the translation efficiency of workloads with high PWC miss rates while avoiding performance degradation in locality-dense scenarios. Furthermore, while V-PWC's performance approaches the ideal PWC for most benchmarks, a performance gap persists in GUPS and MT, where V-PWC recovers less than half of the potential gains. A detailed root-cause analysis of this disparity is provided in the subsequent sections.

[1]We adapt the Unified-TP design, originally proposed for CPUs, to our MCM-GPU setting [39].

7.2 Impact on PWC Hit Rate

To better understand the source of V-PWC's performance gains, we measure the hit distribution across PWC levels with V-PWC enabled, as shown in Figure 12. The figure separates V-L2 hits in the L2 TLB from regular PWC L2 hits, and reports the baseline and Unified-TP L2 hit rates for comparison. Compared with the baseline in Figure 4, V-PWC shifts many L3 hits into more efficient L2 hits, including both V-L2 and PWC L2 hits. On average, V-PWC improves the L2 hit rate of locality-sparse applications by 51%, with a maximum improvement of 74% in S2D. GUPS and MT still achieve less than 60% L2 hit rate under V-PWC, explaining their remaining gap from Ideal-PWC. These workloads have long re-reference distances for L2-level page-table entries and require substantially larger capacity to retain their working sets. Scaling the PWC to 1024 entries raises their L2 hit rates to about 90%, indicating that the L2 TLB capacity used for victim V-L2 entries becomes the main bottleneck. Unified-TP also improves PWC hit rates under high PWC pressure, but its unified capacity pool allows TLB entries to displace PWC entries and reduce effective PWC capacity. Conversely, V-PWC preserves the original PWC capacity and opportunistically uses the L2 TLB only to retain evicted walk entries.

7.3 Impact on L2 TLB Hit Rate

Since V-L2 entries are stored within the L2 TLB, they inevitably compete for capacity with standard virtual-to-physical mappings, potentially compromising the L2 TLB hit rate. To evaluate this trade-off, Figure 13 presents the L2 TLB hit rates under the V-PWC, normalized to the baseline. For applications such as GUPS, MT and SYR2, the L2 TLB hit rates exhibit significant degradation, falling below 60% of the baseline. Similarly, S2D, SYRK, and PR drop to approximately 80%. However, despite these relative reductions, the absolute impact on performance is marginal given that their baseline hit rates are near zero (as shown in Figure 3). The performance

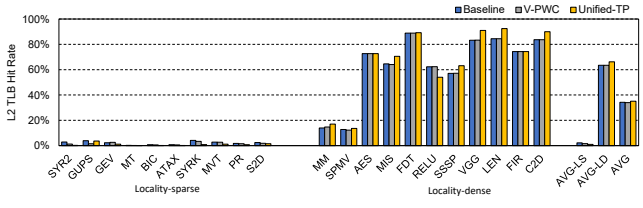


Figure 13: L2 TLB hit rate. *V-PWC causes L2 TLB hit rate degradation in locality-sparse applications with near zero baseline hit rates, while maintains the high hit rate in locality-dense applications.*

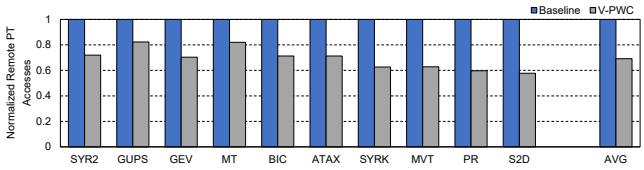


Figure 14: Remote PT accesses of locality-sparse applications. *V-PWC reduces remote page table accesses by approximately* 15% *on average, with a peak reduction of approximately* 40% *in workloads such as PR and S2D.*

gains obtained by V-PWC far outweigh the overhead of increased L2 TLB misses, rendering this contention about capacity acceptable.

Interestingly, MM and GEV show a slight improvement in the hit rate. We attribute this to the reduced PTW latency under V-PWC, which allows pending requests in the MSHR to be committed to the L2 TLB faster, converting potential MSHR hits into L2 TLB hits for subsequent requests. Note that we employ a strict definition of 'L2 TLB hit rate', accounting only for actual virtual-to-physical translation matches, even though MSHR hits also effectively avoid full page table walks.

7.4 Impact on Remote Page Table Accesses

Figure 14 illustrates the normalized remote page table accesses of the proposed V-PWC mechanism across various applications. V-PWC significantly reduces inter-GPM translation traffic during PTW, achieving an average reduction of approximately 15% in remote PT accesses compared to the baseline. The performance gains are particularly pronounced in locality-sparse benchmarks. For instance, in workloads such as PR and S2D, V-PWC curtails remote PT transactions by as much as 40% and 42%, respectively. The underlying reason is that these applications exhibit irregular memory access patterns, which often lead to frequent cache thrashing within the conventional L2 PWC. By repurposing underutilized L2 TLB entries as a victim cache for evicted PWC metadata, V-PWC effectively expands the reach of the translation cache. This extended capacity allows the system to retain more page table entries locally, thereby filtering out a substantial portion of requests that would otherwise necessitate expensive cross-GPM interconnect traversals.

7.5 Reuse of V-L2 Entries

Figure 15 quantifies the reuse rate of the evicted PWC L2 entries that are cached within the L2 TLB under the V-PWC design. This

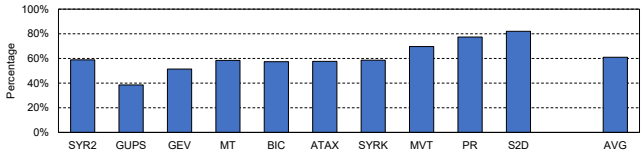


Figure 15: Reuse rates of V-L2 entries of locality-sparse applications. *V-PWC achieves an average reuse rate of* 60% *for evicted L2 entries cached in the L2 TLB in locality-sparse applications.*

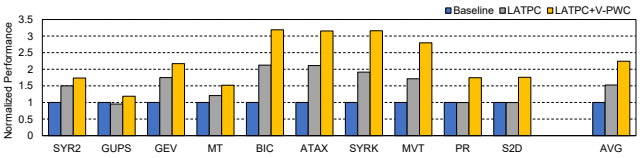


Figure 16: Speedups of V-PWC integrated with LATPC. *V-PWC achieves an average speedup of 2.25× over the baseline when combined with LATPC.*

metric is crucial for validating the effectiveness of utilizing the underutilized L2 TLB as a victim cache. As illustrated, a substantial portion of the redirected entries are successfully reused across all locality-sparse benchmarks. Specifically, S2D and PR exhibit the highest reuse rates, exceeding 82% and 77% respectively, indicating that the majority of L2 entries evicted from the primary PWC still possess significant temporal locality. Even for the most irregular workload, GUPS, nearly 40% of the cached entries are accessed again before being replaced. These high reuse percentages provide direct evidence that V-PWC effectively extends the practical capacity of the page walk metadata storage, thereby converting potential multi-level page table walks into single memory accesses and driving the overall performance gains observed in our evaluation.

7.6 Compatibility with Other Techniques

To evaluate the orthogonality of V-PWC, we integrated it with LATPC[21], a state-of-the-art TLB prefetching technique for GPUs. While LATPC mitigates TLB miss penalties via prefetching, V-PWC focuses on enhancing intermediate PWC hit rates. Figure 16 illustrates the performance gains. For locality-sparse applications, LATPC alone achieves a 1.53× speedup, which V-PWC further elevates to 2.25×. These results demonstrate that V-PWC can seamlessly synergize with existing prefetching mechanisms to further minimize the overhead of complex page table walks in modern GPU architectures.

7.7 Sensitive Study

Page Walk Cache Capacity. We evaluate V-PWC under three PWC capacities, i.e., 0.5×, 1× and 2× of the baseline. V-PWC achieves larger speedups with a smaller PWC, where frequent evictions create more opportunities to reuse the underutilized L2 TLB as a victim cache. Although the benefit decreases with a 2× PWC, most locality-sparse benchmarks still show clear speedup .

L2 TLB Capacity. We evaluate V-PWC under different L2 TLB sizes. As L2 TLB capacity increases from 1× to 4×, the average

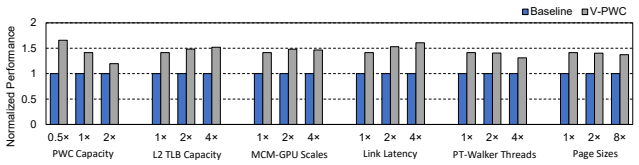


Figure 17: Sensitive Study. *V-PWC achieves consistent performance gains across various architectural configurations, including PWC and L2 TLB capacities, MCM-GPU scales, link latencies, PT-Walker thread counts, and page sizes.*

speedup for locality-sparse applications improves from 41% to 51%, as more evicted PWC entries can remain resident. This shows that V-PWC effectively converts additional L2 TLB capacity into lower page-walk overhead.

MCM-GPU scales. We use a 4-GPM MCM-GPU as the baseline. V-PWC achieves 11.41×, 1.48×, and 1.47× speedups as the system scales, by reducing long-latency remote page table walks. The speedup slightly drops from 2× to 4× scaling because the workloads remain unchanged, while larger systems provide more aggregate translation resources, reducing the relative opportunity for improvement.

Link latency. Following prior work [3, 19, 51], we assume a 32-cycle on-package interconnect latency similar to NVIDIA's GRS and evaluate sensitivity to link latency. V-PWC's average speedup increases from 1.42× at 1× latency to 1.61× at 4× latency, as higher inter-GPM latency amplifies the cost of remote page table walks. By caching translation metadata, V-PWC reduces these expensive cross-chiplet accesses and delivers larger gains under higher communication latency.

PT-Walker Threads. We evaluate V-PWC with different numbers of PT-Walker threads. As the thread count scales from 1× to 4×, V-PWC's normalized speedup gradually decreases from 1.41× to 1.31×, because higher walker parallelism improves the baseline's ability to handle concurrent translations. Nevertheless, V-PWC remains effective across all configurations.

Page Size. Many modern GPUs use larger base pages, such as 16KB or 64KB, to improve address translation. We evaluate V-PWC under these page sizes and observe that its speedup slightly decreases from 1.41× to 1.37×. Larger pages improve baseline TLB hit rates and reduce PWC pressure, thereby slightly narrowing V-PWC's relative benefit.

L2 TLB Replacement Policy. We further evaluate V-PWC with the proposed pressure-aware SRRIP policy and compare it against a conventional SRRIP replacement policy in the L2 TLB. The pressure-aware policy improves performance by 1.4% on average, with a maximum improvement of 4.9% for MT. The figure is omitted due to space constraints.

8 Related Works

TLB Optimizations. Past studies have considered optimizations such as novel TLB organizations [12], prefetching [13, 21, 29], sharing [8, 36, 57], speculation [7, 47], and direct segments [9] to optimize TLB. Pham et al. [48] and Park et al. [44] improved the TLB reach by merging TLB entries based on the contiguity of operating system page allocations. Lee et al. [33] presented an address translation scheme in GPUs named SnakeByte that can dynamically manage variable-sized pages and maximize TLB reach by recursively merging contiguous pages. Jaleel et al. [24] , Kotra et al. [31], and Kanellopoulos et al. [30] discovered underutilized space on the chip and utilized these areas to store TLB entries. Li et al. [34] reduced L3 TLB sharing interference in multi-instance GPU. In contrast to prior efforts that primarily target TLB hit rate enhancement, V-PWC focuses on optimizing the page table walk process. Furthermore, our approach is orthogonal to these existing techniques and can be deployed synergistically to further improve address translation efficiency.

Page Table Walk Optimizations. To minimize the latency incurred by TLB misses, researchers have explored a wide range of methodologies to accelerate page table traversals [1, 2, 18, 32, 43, 50, 54]. A growing body of research has been dedicated to optimizing page table walks within MCM-GPU and multi-GPU architectures [19, 35, 37, 51]. Jang et al. [26] proposed to dispatch software threads running on GPU cores to handle TLB misses requiring page table walks. Extensive research has focused on optimizing the PWC to further minimize the performance penalty of multi-level page table traversals. Margaritov et al. [40] introduced prefetching technology to PWC, enabling direct accessing of leaf-level page tables during the first-level page table access. However, our observations indicate that root-level page table traversals are exceptionally infrequent in the context of MCM-GPUs. Bhattacharjee [11] proposed a coalescing and sharing technique to reduce MMU cache misses. Park et al. [45] reduced the number of indirections required to page table walk by flattening page table. All aforementioned approaches necessitate operating system modifications for support, whereas V-PWC is transparent to both applications and system software. Ma et al.

9 Conclusion

MCM-GPUs offer a promising path to sustain performance scaling, but the rapidly growing memory footprints of workloads increasingly stress GPU address translation. Our analysis shows that for locality-sparse applications, conventional translation mechanisms break down. L2 TLB hit rates collapse and PWC misses at upper levels trigger expensive multi-level page table walks that traverse multiple GPU modules, amplifying latency under MCM non-uniformity. To address this challenge, we propose V-PWC, a software-transparent mechanism that repurposes underutilized L2 TLB capacity as a victim store for evicted PWC entries, effectively enlarging the usable reach of page-walk metadata and reducing memory accesses during page table walks. Our evaluation demonstrates that V-PWC improves performance by 20% on average across all evaluated workloads and by 41% on average for locality-sparse applications, while preserving performance for locality-dense applications.

Acknowledgments

We thank the anonymous reviewers for their thoughtful and valuable feedback. This work was supported in part by NSFC under Grants 62572481.

References

[1] Reto Achermann, Ashish Panwar, Abhishek Bhattacharjee, Timothy Roscoe, and Jayneel Gandhi. 2020. Mitosis: Transparently self-replicating page-tables for large-memory machines. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 283–300.

[2] Neha Agarwal and Thomas F Wenisch. 2017. Thermostat: Application-transparent page management for two-tiered main memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. 631–644.

[3] 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)*.

[4] Akhil Arunkumar, Evgeny Bolotin, David Nellans, and Carole-Jean Wu. 2019. Understanding the Future of Energy Efficiency in Multi-Module GPUs. In *IEEE International Symposium on High Performance Computer Architecture*.

[5] Rajeev Balasubramonian, Andrew B Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 2 (2017), 1–25.

[6] Thomas W Barr, Alan L Cox, and Scott Rixner. 2010. Translation caching: skip, don't walk (the page table). *ACM SIGARCH Computer Architecture News* 38, 3 (2010), 48–59.

[7] Thomas W Barr, Alan L Cox, and Scott Rixner. 2011. SpecTLB: A mechanism for speculative address translation. *ACM SIGARCH Computer Architecture News* 39, 3 (2011), 307–318.

[8] Trinayan Baruah, Yifan Sun, Saiful A Mojumder, José L Abellán, Yash Ukidave, Ajay Joshi, Norman Rubin, John Kim, and David Kaeli. 2020. Valkyrie: Leveraging inter-tlb locality to enhance gpu performance. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. 455–466.

[9] Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D Hill, and Michael M Swift. 2013. Efficient virtual memory for big memory servers. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 237–248.

[10] Ravi Bhargava, Benjamin Serebrin, Francesco Spadini, and Srilatha Manne. 2008. Accelerating two-dimensional page walks for virtualized systems. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*. 26–35.

[11] Abhishek Bhattacharjee. 2013. Large-reach memory management unit caches. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. 383–394.

[12] Abhishek Bhattacharjee, Daniel Lustig, and Margaret Martonosi. 2011. Shared last-level TLBs for chip multiprocessors. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 62–63.

[13] Abhishek Bhattacharjee and Margaret Martonosi. 2010. Inter-core cooperative TLB for chip multiprocessors. *ACM Sigplan Notices* 45, 3 (2010), 359–370.

[14] Niladrish Chatterjee, Mike O'Connor, Gabriel H Loh, Nuwan Jayasena, and Rajeev Balasubramonia. 2014. Managing DRAM latency divergence in irregular GPGPU applications. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 128–139.

[15] Shuai Che, Bradford M. Beckmann, Steven K. Reinhardt, and Kevin Skadron. 2013. Pannotia: Understanding irregular GPGPU graph applications. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 185–195.

[16] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. 2010. The Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU-3)*. Association for Computing Machinery, New York, NY, USA, 63–74. doi:10.1145/1735688.1735702

[17] Shuo Dong and David Kaeli. 2017. DNNMark: A deep neural network benchmark suite for GPUs. In *Proceedings of the 10th Workshop on General Purpose GPUs (GPGPU-10)*. Association for Computing Machinery, New York, NY, USA, 63–72. doi:10.1145/3055443.3055447

[18] Yuan Feng, Yuke Li, Jiwon Lee, Won Woo Ro, and Hyeran Jeon. 2025. Heliostat: Harnessing Ray Tracing Accelerators for Page Table Walks. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 122–136.

[19] Yuan Feng, Seonjin Na, Hyesoon Kim, and Hyeran Jeon. 2024. Barre Chord: Efficient Virtual Memory Translation for Multi-Chip-Module GPUs. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 834–847.

[20] GPUOpen-LibrariesAndSDKs. [n. d.]. Releases · GPUOpen-LibrariesAndSDKs/OCL-SDK. https://github.com/GPUOpen-LibrariesAndSDKs/OCL-SDK/releases. Accessed: 2026-02-01.

[21] Yeonan Ha, Jiho Park, Hanna Cha, Jiwon Lee, Joonsung Kim, Won Woo Ro, and Youngsok Kim. 2025. LATPC: Accelerating GPU Address Translation Using Locality-Aware TLB Prefetching and MSHR Compression. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*. 418–431.

[22] Bongjoon Hyun, Youngeun Kwon, Yujeong Choi, John Kim, and Minsoo Rhu. 2020. NeuMMU: Architectural Support for Efficient Address Translations in Neural Processing Units. In *ASPLOS '20: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[23] Intel Corporation. 2008. *TLBs, Paging-Structure Caches, and Their Invalidation*. Intel Corporation, Santa Clara, CA, USA. Technical White Paper.

[24] Aamer Jaleel, Eiman Ebrahimi, and Sam Duncan. 2019. Ducati: High-performance address translation by extending tlb reach of gpu-accelerated systems. *ACM Transactions on Architecture and Code Optimization (TACO)* 16, 1 (2019), 1–24.

[25] Aamer Jaleel, Kevin B Theobald, Simon C Steely Jr, and Joel Emer. 2010. High performance cache replacement using re-reference interval prediction (RRIP). *ACM SIGARCH computer architecture news* 38, 3 (2010), 60–71.

[26] Sungbin Jang, Junhyeok Park, Yongho Lee, Osang Kwon, Donghyun Kim, Juyoung Seok, and Seokin Hong. 2025. SoftWalker: Supporting Software Page Table Walk for Irregular GPU Applications. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*. 401–417.

[27] WET Jin-hui, LI Chen, and LU Jian-zhuang. 2021. Research on virtual-physical address translation architecture of multi-GPU system. *Computer Engineering & Science* (2021), 228–234. Issue 2.

[28] Adwait Jog, Onur Kayiran, Tuba Kesten, Ashutosh Pattnaik, Evgeny Bolotin, Niladrish Chatterjee, Stephen W. Keckler, Mahmut T. Kandemir, and Chita R. Das. 2015. Anatomy of GPU memory system for multi-application execution. In *Proceedings of the 2015 International Symposium on Memory Systems*. 223–234. doi:10.1145/2818950.2818976

[29] Gokul B Kandiraju and Anand Sivasubramaniam. 2002. Going the distance for TLB prefetching: An application-driven study. *ACM SIGARCH Computer Architecture News* 30, 2 (2002), 195–206.

[30] Konstantinos Kanellopoulos, Hong Chul Nam, Nisa Bostanci, Rahul Bera, Mohammad Sadrosadati, Rakesh Kumar, Davide Basilio Bartolini, and Onur Mutlu. 2023. Victima: Drastically increasing address translation reach by leveraging underutilized cache resources. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 1178–1195.

[31] Jagadish B Kotra, Michael LeBeane, Mahmut T Kandemir, and Gabriel H Loh. 2021. Increasing gpu translation reach by leveraging under-utilized on-chip resources. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 1169–1181.

[32] Jiwon Lee, Gun Ko, Myung Kuk Yoon, Ipoom Jeong, Yunho Oh, and Won Woo Ro. 2025. Marching Page Walks: Batching and Concurrent Page Table Walks for Enhancing GPU Throughput. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1662–1677.

[33] Jiwon Lee, Ju Min Lee, Yunho Oh, William J Song, and Won Woo Ro. 2023. Snakebyte: A tlb design with adaptive and recursive page merging in gpus. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1195–1207.

[34] Bingyao Li, Yueqi Wang, Tianyu Wang, Lieven Eeckhout, Jun Yang, Aamer Jaleel, and Xulong Tang. 2024. STAR: Sub-Entry Sharing-Aware TLB for Multi-Instance GPU. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 309–323.

[35] Bingyao Li, Jieming Yin, Anup Holey, Youtao Zhang, Jun Yang, and Xulong Tang. 2023. 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, 456–470.

[36] Bingyao Li, Jieming Yin, Youtao Zhang, and Xulong Tang. 2021. Improving address translation in multi-gpus via sharing and spilling aware tlb design. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 1154–1168.

[37] Chen Li, Jianzhuang Lu, Jinhui Wei, Qi Yu, and Yunping Zhao. 2020. Dynamic GMMU Bypass for Address Translation in Multi-GPU Systems. In *17th IFIP WG 10.3 International Conference on Network and Parallel Computing, NPC 2020*.

[38] Gabriel H Loh, Samuel Naffziger, and Kevin Lepak. 2021. Understanding chiplets today to anticipate future integration opportunities and limits. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 142–145.

[39] Zhulin Ma, Yujuan Tan, Hong Jiang, Zhichao Yan, Duo Liu, Xianzhang Chen, Qingfeng Zhuge, Edwin Hsing-Mean Sha, and Chengliang Wang. 2020. Unified-tp: A unified tlb and page table cache structure for efficient address translation. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 255–262.

[40] Artemiy Margaritov, Dmitrii Ustiugov, Edouard Bugnion, and Boris Grot. 2019. Prefetched address translation. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 1023–1036.

[41] Collin McCurdy, Alan L Cox, and Jeffrey Vetter. 2008. Investigating the TLB behavior of high-end scientific applications on commodity microprocessors. In *ISPASS 2008-IEEE International Symposium on Performance Analysis of Systems and software*. IEEE, 95–104.

[42] Ohio State University. [n. d.]. PolyBench. https://sourceforge.net/projects/polybench/. Accessed: 2026-02-01.

[43] Ashish Panwar, Reto Achermann, Arkaprava Basu, Abhishek Bhattacharjee, K Gopinath, and Jayneel Gandhi. 2021. Fast local page-tables for virtualized numa servers with vmitosis. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 194–210.

[44] Chang Hyun Park, Taekyung Heo, Jungi Jeong, and Jaehyuk Huh. 2017. Hybrid tlb coalescing: Improving tlb translation coverage under diverse fragmented memory allocations. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 444–456.

[45] Chang Hyun Park, Ilias Vougioukas, Andreas Sandberg, and David Black-Schaffer. 2022. Every walk'sa hit: making page walks single-access cache hits. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 128–141.

[46] Junhyeok Park, Sungbin Jang, Osang Kwon, Yongho Lee, and Seokin Hong. 2025. Leveraging Chiplet-Locality for Efficient Memory Mapping in Multi-Chip Module GPUs. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*. 1040–1057.

[47] Junhyeok Park, Osang Kwon, Yongho Lee, Seongwook Kim, Gwangeun Byeon, Jihun Yoon, Prashant J Nair, and Seokin Hong. 2024. A case for speculative address translation with rapid validation for gpus. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 278–292.

[48] Binh Pham, Viswanathan Vaidyanathan, Aamer Jaleel, and Abhishek Bhattacharjee. 2012. Colt: Coalesced large-reach tlbs. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 258–269.

[49] Bharath Pichai, Lisa Hsu, and Abhishek Bhattacharjee. 2014. Architectural support for address translation on GPUs: designing memory management units for CPU/GPUs with unified address spaces. In *ASPLOS '14: Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*.

[50] B Pratheek, Neha Jawalkar, and Arkaprava Basu. 2021. Improving gpu multi-tenancy with page walk stealing. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 626–639.

[51] Bharadwaj Pratheek, Neha Jawalkar, and Arkaprava Basu. 2022. Designing virtual memory system of mcm gpus. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 404–422.

[52] Hongliang Qu and Zhibin Yu. 2024. WASP: Workload-Aware Self-Replicating Page-Tables for NUMA Servers. In *ASPLOS '24: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*.

[53] Minsoo Rhu, Michael Sullivan, Jingwen Leng, and Mattan Erez. 2013. A locality-aware memory hierarchy for energy-efficient GPU architectures. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. 86–98.

[54] Seunghee Shin, Guilherme Cox, Mark Oskin, Gabriel H Loh, Yan Solihin, Abhishek Bhattacharjee, and Arkaprava Basu. 2018. Scheduling page table walks for irregular GPU applications. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 180–192.

[55] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[56] Addison Snell and Laura Segervall. 2017. HPC application support for GPU computing. *Intersect365 Research* (2017).

[57] Shekhar Srikantaiah and Mahmut Kandemir. 2010. Synergistic TLBs for high performance address translation in chip multiprocessors. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 313–324.

[58] Lisa T Su, Samuel Naffziger, and Mark Papermaster. 2017. Multi-chip technologies to unleash computing performance gains over the next decade. In *2017 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 1–1.

[59] Yifan Sun, Trinayan Baruah, Saiful A Mojumder, Shi Dong, Xiang Gong, Shane Treadway, Yuhui Bao, Spencer Hance, Carter McCardwell, Vincent Zhao, et al. 2019. Mgpusim: Enabling multi-gpu performance modeling and optimization. In *Proceedings of the 46th International Symposium on Computer Architecture*. 197–209.

[60] Yifan Sun, Xiang Gong, Amir Kavyan Ziabari, Leiming Yu, Xinnian Li, Saoni Mukherjee, Carter Mccardwell, Alejandro Villegas, and David Kaeli. 2016. Hetero-mark, a benchmark suite for CPU-GPU collaborative computing. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–10.