A Case for Speculative Address Translation with Rapid Validation for GPUs

Junhyeok Park†, Osang Kwon†, Yongho Lee†, Seongwook Kim†, Gwangeun Byeon†, Jihun Yoon‡, Prashant J. Nair§, and Seokin Hong†

†*Dept. of Electrical and Computer Engineering*, ‡*Dept. of Semiconductor Convergence Engineering*, *Sungkyunkwan University*
§*Dept. of Electrical and Computer Engineering*, *University of British Columbia*
Email: {vzx00770, osang915, jhyn205, su8939, kebyun, head06}@skku.edu, *prashantnair@ece.ubc.ca*, *seokin@skku.edu*

2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO) | 979-8-3503-5057-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/MICRO61859.2024.00029

***Abstract*—A unified address space is vital for heterogeneous systems as it enables efficient data sharing between CPUs and GPUs. However, GPU address translation faces challenges due to high TLB pressure, particularly with irregular and memory-intensive applications. Compared to an ideal scenario, we observe that address translation overheads cause a slowdown of up to 34.5% in modern heterogeneous systems.**

This paper introduces *Avatar*, a novel framework to accelerate address translation in GPUs. Avatar comprises two key components: Contiguity-Aware Speculative Translation (CAST) and In-Cache Validation (CAVA) mechanisms. Avatar identifies the potential for predicting virtual-to-physical address mapping by monitoring contiguous pages that lie in both virtual and physical address spaces. Leveraging this insight, CAST speculatively translates virtual addresses into physical addresses. This speculative address translation enables immediate data fetching into GPUs while addressing translation occurs in the background, reducing TLB-miss overhead.

Unfortunately, modern GPUs lack support for speculative execution, which limits CAST's performance gain. Data fetched from speculated physical addresses is unusable until validation. CAVA addresses this limitation by quickly validating speculated physical addresses. To this end, CAVA embeds page mapping information into each 32B sector of 128B cache lines. Thus, CAVA enables fetching a sector block from memory for a speculated address and *rapidly* validating the speculative translation using the embedded mapping information. Our experiments show that Avatar achieves a 90.3% (high) speculation accuracy and improves GPU performance by 37.2% (on average).

***Index Terms*—GPU, virtual memory, address translation, speculation, data compression**

I. Introduction

The GPU community has become increasingly interested in virtual memory support in recent years. This interest is primarily driven by the need for seamless data sharing and efficient memory management in CPU-GPU systems [60], [61]. One notable technique in this area is Unified Virtual Memory (UVM) [3], [4], [50]. UVM creates a unified address space accessible by both CPU and GPU, simplifying programming by eliminating manual data copying between heterogeneous compute units and enabling GPU memory oversubscription.

Address translation poses a significant challenge for virtual memory support. This process often involves traversing multi-level TLBs (Translation Lookaside Buffers) or walking a multi-level page table when TLB misses occur, resulting in substantial memory access delays [12], [27], [28], [36], [67], [68]. As depicted in Figure 1, this challenge is particularly pronounced in commodity GPUs, where numerous threads concurrently generate memory requests requiring address translation. Our observations indicate that commodity GPUs can experience up to 1.96× higher memory access latency (nearly 950 cycles) due to page walks. Compounding the issue, the GPU pipeline frequently stalls if its TLBs cannot handle these translation requests. Given the simultaneous translation requests from a significant number of threads, GPUs cannot mitigate this prolonged memory access latency using thread-level parallelism [9], [67].

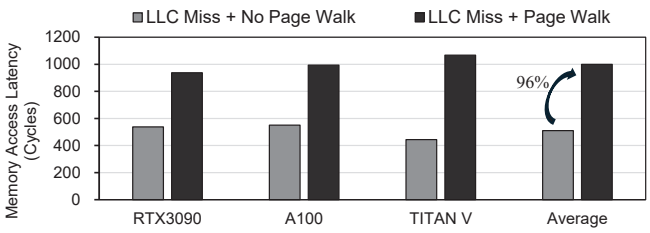


Fig. 1: The latency of page walks on memory access latency in commodity GPUs [48], [53], [54]. On average, by using micro-benchmarks, we see that commodity GPUs have up to 1.96× higher memory access latency (nearly 1000 cycles) due to page walks.

GPU Address Translation – Challenges and Solutions: Prior work have explored techniques to enhance address translation in GPUs [7], [9], [18], [26], [35], [37], [39]–[41], [62], [63], [67], [68], [78]. These efforts generally focus on two approaches. The first approach aims to improve page walk performance to reduce TLB miss penalties [18], [40], [62], [67], [68]. The second approach seeks to increase TLB reach using large page utilization [7], TLB coalescing [37], or resource optimization [25], [35]. While effective, these approaches do not completely eliminate address translation overhead from the memory access path.

Increasing the reach of TLBs helps minimize translation overheads. This is because they tend to increase the lookups that result in TLB hits. We observed that, without these trans-

979-8-3503-5057-9/24/$31.00 ©2024 IEEE
DOI 10.1109/MICRO61859.2024.00029

lation overheads, one can achieve an ideal speedup of 53%. However, prior works such as Page Promotion [7], CoLT [58], and SnakeByte [37] achieve speedups of 22.2%, 27.6%, and 21.8% – short of the ideal attainable speedup. We expect this difference between ideal and achievable speedup to increase as memory sizes increase. This is because the overheads of prior work are tied to the memory capacity. A *key* feature of UVM is to enable GPUs to over-subscribe memory allocation to have access to large host memories. Thus, it would be useful to develop a new scheme that *completely conceals* the address translation overheads to make them amenable to memory over-subscription. In practice, this scheme should be independent of TLB designs and page walk systems.

Accelerated Address Translation in GPUs: This paper introduces ***Avatar*** (**A**ccelerated **V**irtual **A**ddress **T**ranslation with **A**ddress Speculation and **R**apid Validation), a framework for reducing address translation overhead on GPUs. Avatar comprises two components: Contiguity-Aware Speculative Translation (CAST) and In-Cache Validation (CAVA). Broadly, it uses insights from inverted page tables and compression [23], [24] to store virtual translations on physical pages.

CAST leverages page continuity and spatial locality in GPU accesses. Page continuity implies that contiguous pages in the virtual address space are also contiguous in the physical address space – a typical behavior in several memory allocation strategies for GPUs [3], [4], [32]. CAST tracks this contiguity in a chunk by exploiting spatial locality in GPU memory requests. When a TLB miss occurs, CAST *speculates* a physical address for a given virtual address, enabling immediate physical memory access. Simultaneously, CAST initiates an address translation to obtain the accurate address mapping for the missed virtual address on the TLB.

However, GPUs lack support for speculative execution. Thus, they cannot use the data fetched from speculated physical addresses until the speculation is confirmed. This limits the potential performance gains of CAST. To overcome this, Avatar augments CAST with CAVA, an address validation mechanism that *rapidly* verifies speculative address translations. CAVA attempts to compress each 32B sector of the 128B cache lines to 22 bytes for the data located in GPU memory by leveraging memory compression opportunities in modern GPU workloads [15], [33]. It does so by reallocating a portion of the remaining space (10 bytes) to store address mapping information, including the virtual page number (VPN). When the sector fetched from a speculated physical address is compressed and contains the mapping information, CAVA validates the address speculation by directly comparing the VPN embedded within the fetched data with the requested virtual address. This rapid validation enables GPU cores to use the data fetched by CAST immediately.

In rare cases where the data sectors are uncompressible, or CAST mispredicts, Avatar reverts to the baseline strategy of obtaining the address mapping information through slower page walks. We show that Avatar effectively conceals the overall memory latency 44.5% of the time and enhances GPU performance by 37.2%. Moreover, Avatar requires simple changes, incurring low-cost overheads that do not scale with memory sizes. Also, as Avatar *always* checks for the embedded VPN and uses it to match with the requested VPN, it does not affect the correctness of the page translation process.

Contributions: This paper makes four key contributions:

- **GPU-Based Speculative Address Translation**: We propose Avatar, a low-cost framework that enables GPUs and CPUs to share the unified virtual address space while using speculative address translation without affecting correctness.
- **Exploiting Data Mapping**: We propose CAST, a mechanism that leverages page continuity and spatial locality within GPU accesses. By exploiting these properties, CAST enables immediate physical memory access, thus reducing latency associated with address translation.
- **Exploiting Data Content**: We propose CAVA, a rapid validation mechanism to use the data fetched from the speculated physical address once it is loaded into the on-chip cache to verify the speculated address. To this end, CAVA compresses data sectors and embeds the page translation information within 32-byte data sectors.
- **Repurposing Retrieved Translation**: We also show that, once speculation is confirmed using CAVA, it can be repurposed for additional performance enhancements.

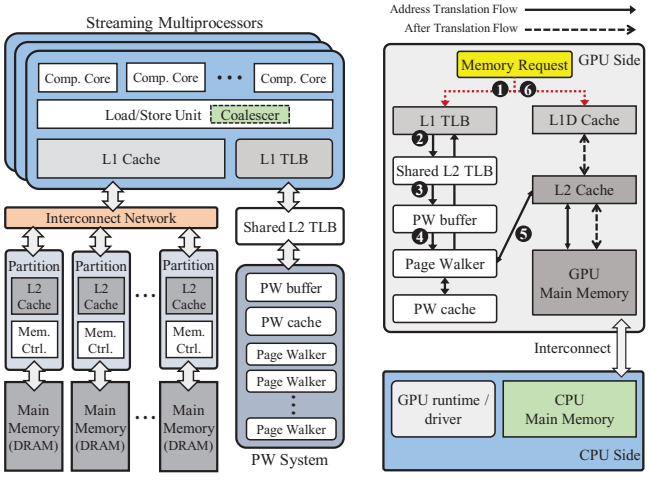
Our experiments show that Avatar achieves an average speedup of 37.2% while providing 90.3% speculative address translation accuracy.

II. Background and Motivation

A. Baseline GPU Architecture

Figure 2a illustrates the baseline GPU architecture based on a disclosed NVIDIA design[1] [49]. A discrete GPU connects to the host (CPU) via PCIe or NVLink. It comprises multiple streaming multiprocessors (SMs), each with compute cores. Within an SM, a load/store unit handles memory accesses from

[1]We use NVIDIA's terminology in this paper. However, our proposed technique is general and applicable to other GPU architectures.



(a) GPU architecture. (b) Address translation flow.

Fig. 2: Architecture of the baseline system.

multiple threads. Coalescers merge adjacent memory requests to reduce traffic. Each SM has a private L1 cache, and all SMs share L2 caches through an interconnect network. Modern GPUs use a sector-based design for both L1 and L2 caches, breaking cache lines into smaller sectors [1], [31], [72]. Each sector can store a smaller part of the memory address space. This design allows precise control over the data fetched and stored in the cache.

B. Virtual Memory in GPUs

Address Translation Hierarchy: Enabling virtual memory support in GPUs is challenging due to the high memory bandwidth demands in parallel processing. To mitigate this, GPUs use hierarchical address translation, such as multi-level TLBs and a multi-threaded page walk system [62], [67], [68].

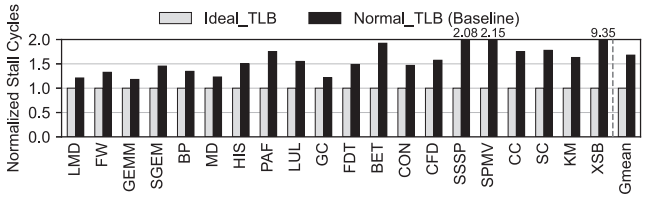
Figure 2b shows the address translation flow in the baseline GPU architecture. The baseline employs a virtually indexed physically tagged (VIPT) cache for the L1 cache. Initially, the memory request searches the per-SM private L1 TLB and L1 cache concurrently (❶). If there is a miss in the L1 TLB, the request goes to the shared L2 TLB, which is accessible by all SMs in the GPU (❷). If the translation is not found there, it is queued in the L2 TLB MSHR and PW buffer (❸). A shared and highly threaded page walk system handles the translation request. When idle, a walker picks up a request from the PW buffer (❹). In a four-level page table, the walker accesses the page table four times to obtain a page table entry (PTE) (❺). A PW cache stores recent page structure entries to speed up this process. Once the translation is complete, the memory request retries the lookup to find the required data (❻).

Unified Memory: Modern GPU systems support a unified virtual address space shared among the host CPU and other GPUs, as seen in NVIDIA UVM [3], [4], [50]. This system handles GPU page faults by dynamically allocating required data pages on-demand [19], [65], [80]. The GPU starts the address translation process when a global memory access is made. The GPU Memory Management Unit (GMMU) triggers a page fault if the corresponding PTE is missing or invalid. Once the GPU Runtime resolves the fault, the requested page is migrated from CPU to GPU memory.

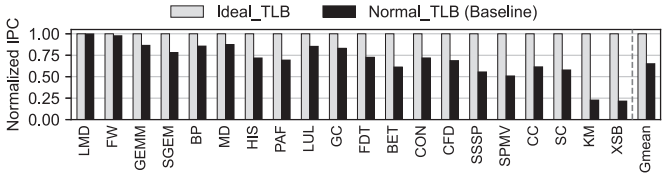
C. Leveraging Page Contiguity in UVM environment

Page contiguity refers to the property where adjacent pages (typically 4KB) in the virtual address space are also adjacent in the physical address space. Various factors contribute to page contiguity, including memory allocation strategies like the buddy memory allocation in the Linux kernel [57]. In GPU systems, the memory allocation strategy employed by runtime systems such as NVIDIA's CUDA Runtime plays a significant role in promoting page contiguity.

In the UVM environment, the CUDA Runtime manages memory in 2MB logical chunks to optimize memory allocation and the page fault handling [3], [4], [32]. Page faults are buffered and grouped into 2MB logical chunk regions based on their virtual addresses. Each virtual 2MB logical chunk reserves a corresponding physical 2MB logical chunk, and the pages within the virtual chunk are migrated to the mapped physical chunk. This allocation strategy, leveraging contiguity within chunks, simplifies memory management while effectively supporting other virtual memory techniques such as page-size promotion and chunk-based page prefetching [3], [19], [47], [65].



(a) Normalized stall cycles due to waiting for memory accesses. On average, address translation leads to 1.7× more cycles as compared to an ideal TLB baseline. In workloads like SSSP, SPMV, and XSB, this can cause 2.08×, 2.15×, and 9.35× more cycles as compared to an ideal baseline.



(b) Performance degradation due to the address translation. On average, address translation leads to 34.5% decrease in performance as compared to an ideal baseline. This overhead is only bound to become worse with increasing memory sizes.

Fig. 3: Impact of address translation on GPU performance.

D. Address Translation: A Key Bottleneck

Despite leveraging page continuity, GPU systems with UVM can experience significant slowdowns due to address translation. GPUs achieve high throughput through multi-threading, which effectively hides memory latencies. When a warp (a basic unit of thread execution) stalls due to long-latency memory operations, the warp scheduler quickly switches to another ready warp with the help of low-cycle context switching [44]. However, address translation adds *substantial* latency in the memory operations, hindering the warp scheduler's ability to hide long memory latencies.

Figure 3a illustrates this impact, showing more frequent SM stalls with the baseline TLB compared to an ideal TLB[2]. This leads to significant performance degradation, especially in workloads like SPMV, SSSP, and XSB, where SM stalls are $>2\times$ compared to the ideal TLB scenario. This underutilization of GPU cores reduces performance, as shown in Figure 3b, where the baseline performs 34.5% worse (on average) than the ideal TLB configuration. Thus, mitigating address translation overhead is key to improving GPU performance.

E. Rethinking Approaches that Exploit Large Pages and TLBs

Prior research has used the page contiguity to improve performance in virtual memory environments across CPU and GPU domains [5], [7], [10], [21], [29], [37], [56], [58], [59].

[2]The details about the simulation environment are described in Section IV

TABLE I: Comparison with Prior Techniques Exploiting Page Contiguity.

Techniques	Goal	Approach	Scalable	No TLB changes	Supported contiguity	Management granularity for contiguous regions	Where to store contiguity info.?	Suitable for oversubscription?
Mosaic [7]	Increasing TLB reach	Page promotion	No	No	4KB, 2MB	Coarse-grained (2MB)	Page table	No
CoLT [58]	Increasing TLB reach	TLB coalescing	No	No	4KB to 64KB	Coarse-grained (64KB)	TLB	No
SnakeByte [37]	Increasing TLB reach	Page promotion + TLB coalescing	No	No	4KB to 8GB	Coarse-grained (32KB to 8GB)	Page table + TLB	No
Avatar (Ours)	Reducing TLB miss penalty	Speculative address translation	Yes	Yes	No limit	Fine-grained (4KB)	N/A	Yes

However, some methods may not be suitable for discrete GPUs due to reliance on frequent OS support [21], [29], [56], which can degrade GPU performance. They may also rely on eager paging [21], [29] which is incompatible with GPU demand paging. This section focuses on GPU-compatible approaches.

Large Pages: Using larger page sizes significantly reduces address translation overhead by improving TLB reach. However, large pages increase internal fragmentation where a portion of the large page is not used. Moreover, the direct adoption of large pages in the UVM environment is impractical as each page fault requires a 2MB migration, significantly increasing demand paging overhead [3].

Page Promotion: An alternative approach to achieve the benefit of the large page is page promotion [7], which increases page size when all subpages in a logical chunk are valid. This approach can increase the TLB reach while keeping demand paging overhead low. However, it may not yield benefits until small-size pages are promoted to larger ones. It may also provide a small gain if the application only uses a partial set of pages within the logical chunk without promotion.

TLB Coalescing: TLB coalescing [58] can be a favorable approach to increase the TLB reach. This approach coalesces multiple address mappings for contiguous pages into one TLB entry. Figure 4 shows an example of TLB coalescing. To compact such mappings, each TLB entry has multiple valid bits to indicate the validity of particular pages within a set of contiguous pages mapped to the same TLB entry. As shown in the figure, a coalesced TLB can maintain the translation information with a small number of TLB entries. However, the address range covered by the coalesced TLB entry is limited to a small set of contiguous pages [56].

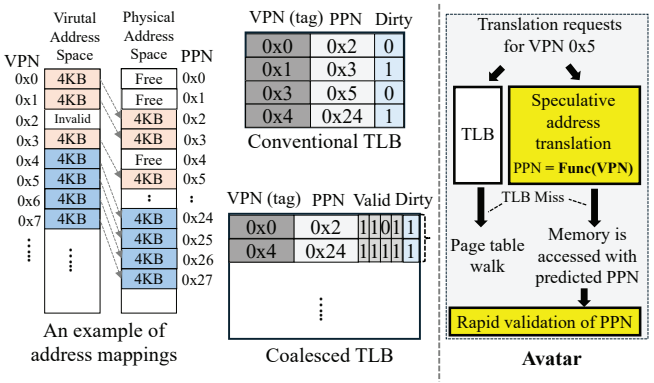


Fig. 4: An example of TLB coalescing (left) and speculative address translation (right).

Page Promotion with TLB Coalescing: Recent prior work [37] synergistically combines the page promotion and the TLB coalescing. In the proposed scheme, the contiguous pages are promoted to various large-sized pages depending on the page contiguity level. In addition, the translation information for the enlarged pages is coalesced in each TLB entry. Even if a coalesced entry can cover a larger contiguous memory region in this approach, it requires additional memory references (in-memory page table accesses) to maintain the contiguity information in each page table entry and to merge small pages into a larger one. The additional memory references may decrease the performance gain if the page contiguity level is not significantly high or the merged page is splintered back into multiple small pages [7].

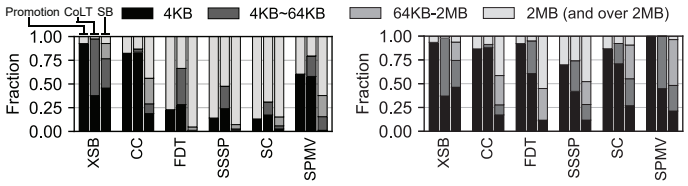
Common Limitations of Prior Techniques: Table I compares the prior techniques exploiting the page contiguity. These techniques have five common limitations.

Firstly, they are fundamentally limited in scalability as they rely on the TLB to reduce the address translation overheads. Even if these techniques increase TLB reach, their benefits will be reduced as the workload's working set size increases.

Secondly, all these techniques necessitate modifications in TLB design, such as including the validity of contiguous pages within a TLB entry or applying different TLB indexing methods depending on the page size of the referenced address. Given that the TLB, especially the L1 TLB, is on the critical path, these modifications can potentially reduce the performance of the GPU pipeline, a scenario that demands immediate attention.

Thirdly, they support only a limited number of page contiguity. For example, Mosaic [7] only supports 4KB and 2MB page sizes. Thus, this technique can not benefit the workloads with intermediate level (e.g., 64KB-128KB) of page contiguity as shown in Figure 5a.

Fourthly, page metadata (e.g., access and dirty bits) is managed coarsely. For instance, when four pages are merged into one TLB entry, their dirty bits are also combined into a single bit, as illustrated in Figure 4 (left). This coarse management can lead to inefficient memory operations, such as unnecessary writebacks for clean 4KB pages. Moreover, due to the page metadata, merging contiguous small pages into a larger one can be challenging [8]. This issue can be mitigated by increasing metadata bits in each TLB entry or by immediately propagating metadata updates to the in-memory page table in the background [8], [16]. However, the former approach incurs extra hardware costs, which increase with the



(a) No oversubscription. (b) Oversubscription (130%).

Fig. 5: Coverage breakdown of accessed TLB entries. The memory oversubscription scenario can reduce the effective TLB coverage of the prior techniques due to frequent TLB shootdowns.

degree of coalescing, while the latter can introduce additional memory references for updating the page table.

Finally, the benefits of the prior techniques can be reduced in the memory oversubscription scenario. Memory oversubscription, a key feature of UVM, occurs when an application's working set exceeds GPU memory capacity, leading to multiple page (chunk) evictions from GPU memory [11]. This eviction process impacts the benefits of page promotion and TLB coalescing. Similarly, TLB coalescing can suffer when TLB shootdowns flush merged TLB entries covering large regions. As shown in Figure 5b, under memory oversubscription, the fraction of hits in entries with large coverage is significantly reduced when compared to the no-oversubscription scenario.

F. A Case for Speculative Address Translation

One promising approach to mitigate the address translation overhead in the virtual memory is speculative address translation [5], [10], [59], which predicts the physical address corresponding to a given virtual address by leveraging page contiguity, as shown in Figure 4 (right). This method enables the pipeline to continue processing with the predicted address while validating the address speculation in the background. Once the data is loaded from the predicted address, the compute units in the processor pipeline immediately use it to execute instructions. This method can effectively eliminate the address translation latency if the speculation proves correct.

However, in case of incorrect speculation (mis-speculation), all incorrectly executed instructions must be flushed from the processor pipeline. This mis-speculation is more problematic on GPUs than CPUs since the GPU architecture fundamentally does not support speculative execution [45] and cannot roll back erroneously executed instructions. Furthermore, even if we make GPUs to support speculative execution, the expensive roll-back process would essentially diminish the effectiveness of the speculative address translation. Consequently, despite its potential benefits, adopting speculative address translation in GPUs is quite challenging without proper support.

In this paper, we aim to propose *a speculative address translation scheme equipped with a rapid validation mechanism* to mitigate the address translation overheads in GPUs while tackling GPU's architectural limitations that hinder the adoption of the speculative approach. Note that the goal of our proposed scheme is to reduce the TLB miss penalty, and thus, it can be synergistically integrated with the prior techniques developed to decrease TLB misses.

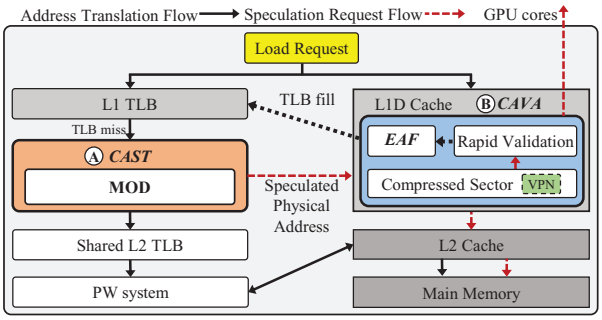


Fig. 6: An overview of Avatar.

III. Avatar Framework

This section presents Avatar, a novel framework that enhances GPU performance by leveraging speculative address translation with rapid validation. Figure 6 provides an overview of *Avatar*. *Avatar* includes two components: Ⓐ ***Contiguity-aware Speculative Translation (CAST)*** and Ⓑ ***In-Cache Validation (CAVA)***.

Figure 7a illustrates a timeline example for a memory request in the baseline system. This example assumes that the memory request misses on both the TLBs and caches, resulting in highly serialized memory access. This serialization significantly increases the memory access latency and is generally observed in most workloads because memory accesses missed in the L1 TLB are likely missed in the L1 cache [12].

Speculative Address Translation in GPUs: To mitigate the long memory access latency caused by the address translation, we propose a novel address translation scheme called CAST. CAST speculates the physical address by tracking the continuous virtual-to-physical mapping region. Figure 7b illustrates a timeline of the memory request in the system with CAST. When a load request misses in the L1 TLB (①), the address translation request is forwarded to the L2 TLB. Simultaneously, CAST speculates the physical address and performs a lookup in the L1 cache with this address. If the memory request misses in the L1 cache (②), the data is fetched from the L2 cache or main memory. Since GPUs do not support speculative execution, the fetched data remains in the L1 cache unusable until the address translation confirms it.

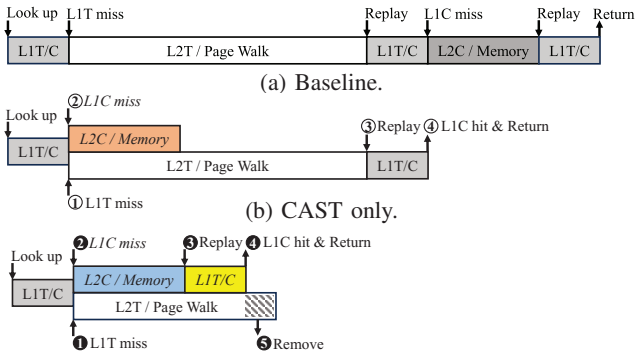


Fig. 7: Timelines of a memory request in three configurations. T and C indicate TLB and Cache, respectively.

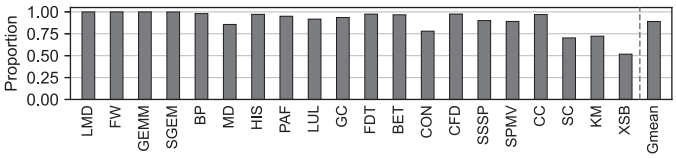


Fig. 8: The proportion of memory accesses from the same load instruction that accesses pages within a 2MB memory chunk. Memory accesses from the same load instruction exhibit high spatial locality within a chunk, with an average of 89%.

Upon successful speculation, the memory request replays the L1 TLB\L1 cache lookup (③), and it hits in the L1 cache as the data has already been fetched using the speculated address (④). CAST allows GPUs to overlap address translation with memory access, thereby reducing the TLB miss penalty.

While CAST can enhance performance, its benefit is constrained because speculatively fetched data remains unusable until validated. This necessary validation process limits performance gains, especially when address translation involves a slow page walk. Thus, quickly validating the speculative translation is essential to fully realize CAST's potential.

Rapid Validation: For this purpose, we propose the second component: CAVA. CAVA enables rapid validation of the speculative address translation in the L1 cache, allowing immediate use of the data fetched with the speculated physical address. To this end, we embed page information (i.e., VPN) into data (i.e., sector) using memory compression. Once the data (i.e., sector) is brought from the lower-level memory, CAVA utilizes the embedded information for rapid validation. Figure 7c illustrates the timeline of a memory request in the system with CAST and CAVA. The processes ❶ and ❷ are the same as the system with only CAST. However, once the memory request fetches the data into the L1 cache, CAVA validates the speculation by comparing the embedded VPN in the data with the requested virtual address (❸). On correct speculation, GPU cores immediately use the fetched data (❹).

Eliminating & Propagating Verified Address Translation: While CAVA performs validation for speculative address translations, there may still be an ongoing address translation process (e.g., page walk). This ongoing translation request consumes constrained memory resources, such as memory bandwidth or TLB MSHRs. In particular, a page walk generally takes a long execution time, preventing other requests from utilizing those resources during that period. Furthermore, other SMs may also require translation for the verified page. To this end, CAVA employs the *Early-TLB-Fill (EAF)* scheme. Once a speculated address is verified with CAVA, EAF generates a TLB entry with the embedded page information and stores it in the TLBs, eliminating the ongoing address translation process (⑤ in Figure 7c). Moreover, this TLB entry is sent to other SMs, enhancing the efficiency of one rapid validation across the system.

A. CAST: Contiguity-aware Speculative Translation

Observation - Page Access Pattern of Load: Warps originating from the same GPU kernel share the same program code. As a result, threads belonging to these warps tend to exhibit similar instruction and data access patterns [34], [55]. When executing the same load instruction, we analyze the page region accessed by multiple warps allocated to the same SM. As shown in Figure 8, the memory requests from the load instructions with the same PC are highly likely to access the same 2MB memory chunk, a physically contiguous memory region. On average, 89.0% of memory accesses from the same instruction fall within a 2MB memory chunk.

Mapping Offset Detection Table (MOD): Based on this observation, CAST employs a Mapping Offset Detection Table (MOD) that dynamically identifies continuous regions. MOD maintains the offset between a virtual address and its corresponding physical address for each load instruction to monitor contiguity in 2MB chunks. Each MOD entry has three fields: PC, state counter, and V2P offset. The PC field contains a load instruction's program counter used as a tag. The state counter is a 2-bit saturating counter indicating prediction confidence. The V2P offset provides the difference between the virtual and physical addresses. MOD is fully associative and uses an LRU replacement policy.

When an instruction encounters a miss in the L1 TLB, the required translation is retrieved from either the L2 TLB or the page table, and a new TLB entry is inserted into the L1 TLB. During this process, a MOD entry corresponding to the instruction is updated with a new V2P offset calculated by subtracting the Virtual Page Number (VPN) from the Physical Page Number (PPN). When the new offset is identical to the existing one, the state counter is incremented by 1. Otherwise, it is decremented by 2; the state counter is decremented quickly to catch changes in address mapping. A higher value of the state counter indicates that the current V2P offset is fairly reliable. The V2P offset is only updated to a new value when the state counter value is zero. When the V2P offset is updated, the state counter is initialized to 1. If no MOD entry corresponds to the instruction, a new entry is allocated and initialized with the new V2P offset.

CAST Operation with MOD: Figure 9 shows the whole process of CAST. A load request missing in the L1 TLB accesses the MOD using its PC (Ⓐ). If a corresponding entry exists, CAST checks if the state counter exceeds a threshold (Ⓑ). In this study, we set the threshold to 2. When the counter value exceeds the threshold, CAST generates a speculated PPN by adding the MOD entry's V2P offset to the VPN (Ⓒ). It then

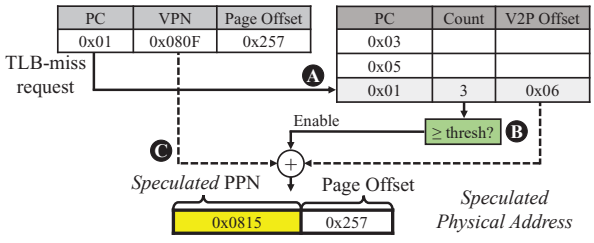


Fig. 9: Speculative address translation with MOD. A virtual-to-physical address (V2P) offset is added to the VPN of the requested virtual address.

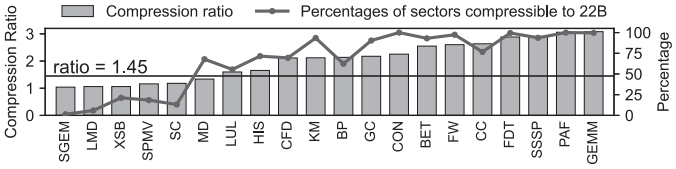


Fig. 10: Compression ratio (BPC [33]) on 32B sectors and percentage of 32B sectors compressible to 22B.

constructs a speculated physical address by appending the page offset to this PPN and forwards this address to the load-store unit and the L1 data cache.

To record and manage memory requests with speculative address translation, we employ an additional bit, called the speculation bit, in each entry of the pending table within the load-store unit [1]. The pending table holds the speculated PPN and the speculation bit representing the speculative translation status. This bit is set when speculative translation is initiated and unset when the speculation is validated.

B. CAVA: In-Cache Validation

Memory Compression Opportunities: CAVA utilizes a data compression technique to embed the page information (e.g., VPN and Permission bits) into each sector. Several memory compression schemes are used in prior works, particularly for GPU workloads [6], [15], [33], [42], [75]. In this paper, we employ BPC [33] (Bit-Plane Compression) algorithm to compress the sectors. To analyze the compressibility of 32B sectors in GPU workloads, we attempt to compress each 32B sector into 22 bytes. We utilize NVBit [76] to collect the memory dump of each sector for various GPU workloads. Figure 10 shows the compression ratio and the percentage of 32B sectors that can be compressible to 22 bytes. Most of the benchmarks show a compression ratio above 1.45, which is the ratio needed to compress a 32B sector to 22 bytes. This result is consistent with observations of the compressibility of GPU workloads in prior works [15], [33], and it arises from the inherent homogeneity of GPU workloads across various data types. Our experimental result shows that about 67.5% of sectors can be compressed to 22 bytes.

Embedding Page Information into Sectors: Figure 11 shows the process of embedding page information into sectors. The GPU runtime and driver prepare for a page migration when a page fault occurs. As the runtime manages the whole unified memory space, it provides the page information about the demanded page ahead of the migration. This is possible because page migration is triggered only after memory mapping is completed [4]. The transferred page information includes the VPN and permission bits (e.g., read-only), which are essential for fast validation [52]. When the demanded page is transferred to the GPU, the page information is transferred together (ⓐ). On the GPU side, the individual sector of the transferred page is compressed, and the page information is appended to the compressed sector (ⓑ). To this end, we add a (de)compression engine to each GPU memory controller for compressing a 32B sector to 22 bytes, allowing the 8B page information to fit

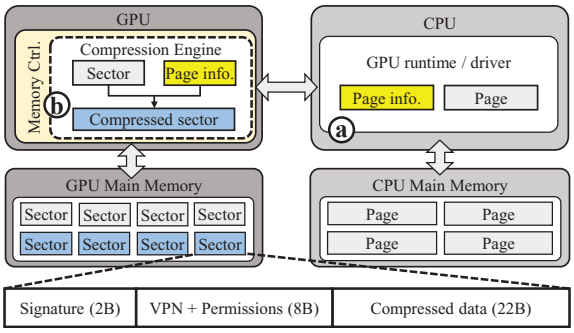


Fig. 11: Embedding page information into each sector.

within the remaining 10B region. The remaining 2B space is utilized for a signature that indicates the compression status of the sector.

Managing Compressed Sectors: CAVA must identify whether the sector fetched from the GPU main memory is compressed and includes page information. To accomplish this, CAVA employs Attaché [24] framework, a specialized metadata encoding scheme. Attaché utilizes the top 15 bits of the 2B signature to store a predefined value called the Compression ID (CID), which is used for marking compressed sectors. When the top 15 bits of a sector match the CID, it is considered compressed. This straightforward approach assumes that the top 15 bits of the uncompressed sector are different from the CID. However, there is a possibility (0.003%) that some uncompressed sectors may naturally contain the CID, leading to invalid CID collisions. To address this issue, Attaché introduces the Exclusive ID (XID), which signifies that the sector has the CID even though it is uncompressed. When a sector is incompressible, and its top 15 bits are identical to the CID, the compressor replaces the top 16th bit of the sector with XID and stores the replaced bit in a reserved region of the GPU main memory.

Fetching Sectors from GPU Main Memory: When a sector is fetched from the GPU main memory, the memory controller identifies whether the sector is compressed by comparing the top 15 bits of the sector with the CID. To support CAVA, we add two additional bits to each sector tag of the L2 cache: the compression bit (C) and the guarantee bit (G), as shown in Figure 12. The compression bit distinguishes compressed sectors in the cache, while the guarantee bit specifies whether speculation for the sector is validated. The sector tag of

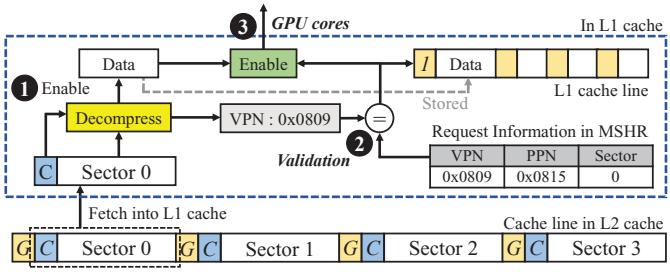


Fig. 12: In-Cache Validation for memory request with predicted physical address.

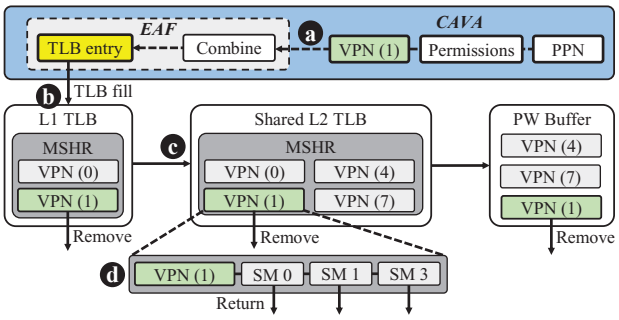


Fig. 13: Early TLB Fill process. EAF repurposes the page information embedded in the sector to reduce traffic on the address translation hierarchy.

the L1 cache only needs the guarantee bit because data is decompressed before being stored in the L1 cache.

Validating Speculative Translation: Figure 12 depicts the rapid validation process. When data is fetched from the lower memory hierarchy with a speculated address, CAVA checks the compression bit of the fetched sector. If the sector is compressed, CAVA obtains the VPN by decompressing the sector (❶) and compares the VPN with that of the requested virtual address (❷). If it turns out that the speculation was correct, the GPU cores can immediately use the data (❸). Consequently, the verified data is stored in the L1 cache just like normal data fetch. Once the memory request with the speculated address is validated, CAVA performs a lookup in the load-store unit's pending table to find the corresponding entry registered by the memory request. If the speculation bit is set, it indicates that the background address translation is currently in progress within the TLB hierarchy. In such a case, CAVA removes the corresponding entry from the table and completes the memory request.

Handling Mis-Speculation: Since Avatar enables access to memory with speculatively translated addresses before validating the speculations, effective handling of mis-speculation is essential to ensuring accurate and trusted program executions. Avatar renders unverified sectors invisible in the cache to prevent malfunctions or security threats, similar to a prior technique [77]. When a compressed sector fails validation due to VPN or permissions mismatch, Avatar promptly invalidates the fetched sector to prevent any inappropriate usage. For uncompressed sectors that are speculatively fetched, Avatar unsets the guarantee bit in the sector tag. If the guarantee bit is unset, the sector remains unseen (valid but not usable) in the cache. Once the translation of the background address is completed, Avatar will check the pending table in the load-store unit. When the speculated PPN does not match with the actual PPN, Avatar invalidates the fetched sector from the cache. If the speculation is accurate, Avatar sets the guarantee bit of the fetched sector to mark it as valid and usable.

C. EAF: Early TLB Fill

Constructing a TLB Entry: When CAVA successfully validates a speculative address translation by using the page information embedded in the fetched sector, we can use the information to construct a new TLB entry. Figure 13 illustrates Early-TLB-Fill (EAF) process. CAVA provides both the page information and PPN to EAF, and EAF then utilizes this information to construct a TLB entry (ⓐ).

Releasing Constrained Resources: EAF stores the new TLB entry in L1 TLB (ⓑ) and deallocates its corresponding MSHR entry. Subsequently, EAF finds a page walk request in the L2 TLB MSHR and PW buffer registered for the TLB entry (ⓒ). If a match is found, the new TLB entry is stored in the L2 TLB, releasing resources in the L2 TLB MSHR and PW buffer. By quickly releasing resources, EAF reduces TLB miss penalties. Furthermore, EAF forwards the new TLB entry to the L1 TLB of different SMs that require the translation, ensuring the desired translation is efficiently prefetched across SMs (ⓓ).

D. Discussion

Compression Bandwidth: Avatar employs (de)compression engines in each GPU memory controller as in prior works [6], [15], [33], [64], [66], [75]. Therefore, the compression scheme of Avatar does not affect the memory channel bandwidth, which is crucial for efficiently handling data-intensive workloads on GPUs. To validate the hardware overhead of the (de)compression engines, we implement a Verilog model for the BPC algorithm and synthesize it with 28nm UMC standard cells [74]. The total area of the (de)compressor engines in GPU memory controllers (total 16 memory controllers in our baseline) is 0.314mm^2, which is quite small compared to a single GPU die size (∼800mm^2 [48]).

TLB Shootdown: When a page table entry (PTE) is modified, the GPU driver triggers a TLB shootdown operation to clear outdated translation data from the GPU's TLBs. Since Avatar stores the page information alongside the data within each sector, the TLB shootdown operation also necessitates invalidating the in-sector page information from caches or DRAMs, increasing the cost of the TLB shootdown. The primary cause of the TLB shootdown in the CPU-GPU systems is the page migration between a GPU and a host CPU or between GPUs [2], [11], [38]. Figure 14 illustrates a timeline of the page migration process. To migrate a page, the GPU driver flushes the page's contents from the on-chip caches (①) and invalidates the PTE of the page from the TLBs (②). After that, the page's contents are transferred to the CPU (or another GPU) through an inter-processor link (③). Finally, the driver updates both the local page tables on the GPUs and a global page table (⑤). During this process, it is not necessary to explicitly invalidate in-sector page information in the cache because all the contents of the migrated page are automatically

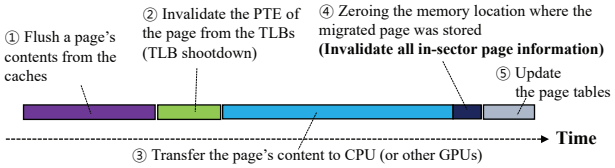


Fig. 14: Timeline of page migration process with invalidation of in-sector page information.

flushed from the caches (①). Furthermore, updating the in-sector page information for the migrated page is unnecessary. This is because the information contains a VPN rather than a PPN, and the VPN remains unchanged even after the page is remapped to a new physical address.

The only additional operation introduced by Avatar during the page migration process is clearing the in-sector information from the GPU DRAM. Even after a page is migrated to the CPU or another GPU, the page's contents, including the in-sector page information, remain in the original location within the GPU DRAM. This leftover data poses a risk of generating an accurate speculation for an invalid page. To address this issue, Avatar must zero out the DRAM columns where the page was previously located (④). The migration process inherently involves reading data from the DRAM, allowing for the zeroing operation to be seamlessly integrated by overwriting the opened row with zeros just after reading data from the row.

Speculation on Invalid Pages (False-Speculation): Since pages are migrated to GPU memory on-demand in UVM, the physical address speculated by Avatar may be invalid or not present in GPU memory. We refer to this case as a false speculation. CAVA can handle false speculation because a speculatively fetched invalid sector is not compressed and lacks page information. After the page fault is serviced, the memory requests to the fault pages are replayed [80]. Avatar then offers speculative translation again on this replay.

Page Migration Scheme: Some modern GPU systems use a static access counter threshold-based migration [38], [65]. This approach monitors memory accesses on pages using counters and selectively migrates those pages once their access counter reaches a specific threshold. As a result, some cold pages remain unmapped to the GPU memory. In those GPU systems, updates to MOD are conducted only for GPU-mapped addresses. This strategy prevents speculation translation from being used for non-GPU-mapped regions.

Multi-tenancy: When a GPU is spatially shared to support multi-tenancy [7], [9], [41], [62], Avatar needs to differentiate between multiple active virtual address spaces. Avatar embeds the Address Space ID (ASID) within the page information to achieve this. In the case of page sharing, where multiple virtual address spaces share a page, multiple VPNs are mapped to the same PPN. Avatar does not store the page information within the data to prevent translation collisions for that page.

Cache Designs: Avatar can work with the VIPT cache design (the baseline) and other cache designs, such as the PIPT (Physically Indexed Physically Tagged) cache and the VIVT (Virtually Indexed Virtually Tagged) cache. In the PIPT design, a memory request performs data cache lookups after the address translation is completed. This process does not require major changes in Avatar's operation compared to the VIPT design. In contrast, in a fully virtual L1 cache design, the address translation hierarchy is located between the L1 cache and the shared L2 cache. Thus, address translation is only needed on an L1 cache miss. When a memory request misses the L1 cache and L1 TLB, CAST performs speculative address translation and sends a memory request using the speculated physical address to the L2 cache.

Security Implications: Avatar maintains a security level comparable to conventional GPUs by effectively addressing potential information leakage through two primary mechanisms. First, the CAVA makes the sectors speculatively fetched invisible in the cache until their speculative translations are validated, as described in Section III-B, which inherently blocks direct unauthorized data access by attackers. Secondly, the CAST mechanism uses historical address translation records to guide speculative translations, preventing unauthorized access to arbitrary addresses. Additionally, current GPUs support isolated security domains, such as NVIDIA Multi-Instance GPU (MIG) [48], [51]. MIG provides isolation on computing resources and the entire memory system. In such cases, unauthorized data access through mis-speculation can be initially prevented at the memory system isolation boundary.

IV. Evaluation

A. Experimental Methodology

We use GPGPU-sim v4.0 [31] with a system configuration similar to NVIDIA Ampere (RTX3070) [49]. We enhance the framework to implement hierarchical address translation. The baseline system use the UVM demand paging [3], [4] and page fault handling mechanism [19], [80]. For efficient memory allocation, we employ a tree-based neighborhood

TABLE II: Simulated Baseline Configuration.

GPU core	46 SMs, 1132MHz, LRR scheduler Max 48 warps per SM
L1 TLB	32 entries (4KB, base), 16 entries (2MB) 25 cycle latency, fully associative 4 ports, 32 MSHR entries
L2 TLB	1024 entries (4KB, base), 128 entries (2MB) 90 cycle latency, 8-way associative 8 ports, 128 MSHR entries
L1 cache	128KB, 39 cycle latency, 128B line (sectored)
L2 cache	4MB, 187 cycle latency, 128B line (sectored)
DRAM	1750MHz (GDDR6), 16 channels 28GB/s per channel, DRAM page size = 4KB t_{RCD} = 13.7ns, t_{CL} = 13.7ns, t_{WL} = 4.6ns t_{RTW} = 6.3ns, t_{RP} = 15.3ns
Page table	4KB page (base), 2MB page (promotion), 4-level
Page walker	16 walkers, 128 page walk buffer entries
Page walk cache	64 entries, 8 ports
Page Prefetcher	TBN prefetcher [19]
CAST	32-entry MOD 2 for the state counter threshold
CAVA	(de)compression engine based on BPC [33] 7 cycles for decompression in L2 cache

TABLE III: Workload Categorization.

Class	Benchmark	Abbr.	Data Type	Benchmark	Abbr.	Data Type
L	fw [13]	FW	I	lavaMD [14]	LMD	D
	gemm [20]	GEMM	F	sgemm [70]	SGEM	F
M	backprop [14]	BP	F	shoc-MD [17]	MD	I
	histo [70]	HIS	UI	pathfinder [14]	PAF	I
H	lulesh [30]	LUL	F	color_max [13]	GC	I
	fdtd2d [20]	FDT	F	betweenness [43]	BET	UI
	conv.Sepa [46]	CON	F	cfd [14]	CFD	F
	sssp [13]	SSSP	I	spmv [70]	SPMV	I/F
	connected [43]	CC	UI	s.cluster [14]	SC	F
	kmeans [14]	KM	F	XSBench [73]	XSB	I/D

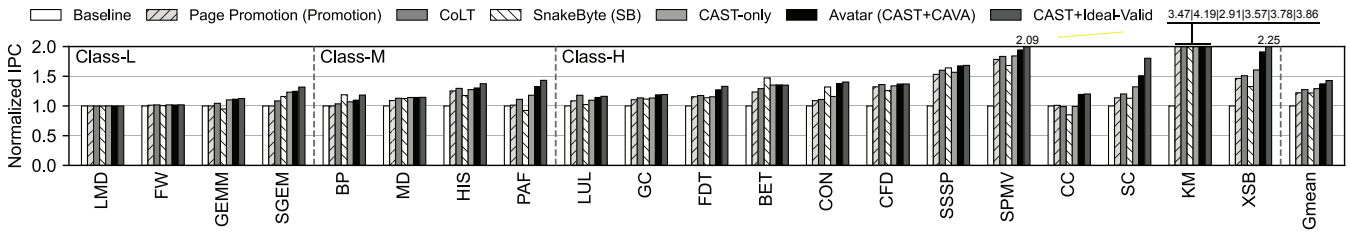


Fig. 15: Performance comparison. Results are normalized to the baseline.

page prefetcher [19] on our system. Table II summarizes detailed system configurations for the baseline.

For the Avatar mechanism, we incorporate CAST, which consists of a 32-entry MOD. A GPU kernel generally does not have diverse load instructions (PCs) [34], so a 32-entry size is sufficient for tracking the page contiguity. For CAVA, we add a compression engine based on BPC [33] and extend the whole memory system to support sector compression. As Avatar cooperates with the memory compression scheme, we append additional latency to the data cache access path. We assume seven cycles of decompression latency [33] for the shared L2 cache.

We evaluate 20 workloads from various benchmark suites [13], [14], [17], [20], [30], [43], [46], [70], [73] as listed in Table III. The workloads are selected based on their data types (for diverse compressibility), working set sizes, and memory access patterns (for TLB sensitivity). The dominant data type for each workload is denoted as I/UI/F/D for int/unsigned int/float/double, respectively. We classify the workloads into three classes based on their L2 TLB misses per million instructions (L2 TLB MPMI); under 10 for class-L, between 10 and 60 for class-M, and over 60 for class-H. The working set size of the workloads ranges from 4MB to 2.24GB, on average 14.5MB for class-L, 80.4MB for class-M, and 701.7MB for class-H.

B. Performance

Figure 15 shows the overall performance of seven different configurations. We compare the baseline, Page Promotion (Promotion) [7], CoLT [58], SnakeByte (SB) [37], and Avatar (CAST+CAVA). We assume that CoLT can merge up to 16 contiguous pages since a 128B cache line can accommodate 16 8B-PTEs. Additionally, we compare Avatar with its two variants: using only CAST *(CAST-only)* and using CAST with ideal validation *(CAST+Ideal-Valid)*. This experiment highlights the individual contributions of CAST and CAVA to performance improvements and explores the potential benefits of integrating CAST with ideal validation. The main distinction between CAVA and ideal validation is that ideal validation confirms all speculative translations done by CAST before CAST fetches the data from the speculated addresses. We adopt Page Promotion for other configurations, as it can work orthogonally with them. In the experiment, we do not consider page fault handling latency to focus on the address translation overheads.

1) Impact of CAST: CAST-only achieves an average performance improvement of 29.1% compared to the baseline, outperforming Page Promotion by 6.8%, CoLT by 1.5%, and SnakeByte by 7.3%. This improvement comes from CAST's ability to overlap address translation with memory access by fetching data from a speculated address. This approach is particularly effective for workloads such as GEMM and MD, where the address translation traffic is relatively low; thus the speculative translation is quickly validated for most memory requests. On the other hand, workloads such as FDT and CC show relatively low-performance improvements with CAST-only. This is because the address translation takes longer for these workloads due to multiple in-memory page table walks. Without CAVA, the fetched data from the speculated address is not usable before validation, reducing the potential benefit from CAST for these workloads. Similarly, PAF, CON, and XSB exhibit a performance gap between CAST-only and Avatar due to this limitation.

2) Avatar Performance: Avatar achieves a 37.2% performance improvement on average compared to the baseline. Compared to prior works, Avatar outperforms Page Promotion by 14.9%, CoLT by 10.1%, and SnakeByte by 16.3%. To analyze the impact of speculative translation, we track memory accesses that get speculative translation in Avatar. For clarity, we track only accurate speculation. Figure 16 quantifies the fraction of those memory-access results. There are four terms in Figure 16, each associated with various situations that can occur in Avatar. The term 'L1D_hit' signifies that rapid validation has failed (no page information in a fetched sector), but after background translation is completed, the original memory access uses the prefetched sector. The term 'L1D_merge' signifies that rapid validation has failed, and the background translation is completed before the sector is fetched. In this case, the original memory access is merged with the speculative access in the cache MSHR. These two cases partially hide the memory latency, similar to what is shown in Figure 7b, accounting for 59.0% of the results on average. GEMM, SGEM, and BET mainly benefit from this partial latency hiding.

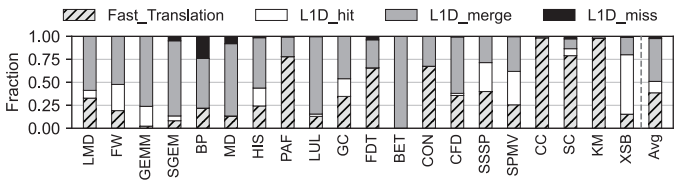
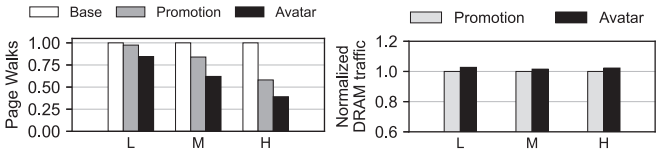


Fig. 16: Fraction of the memory-access results which get speculation in Avatar.



(a) The number of page walks. (b) DRAM traffic.

Fig. 17: The impact of EAF on (a) page walks and (b) DRAM traffic

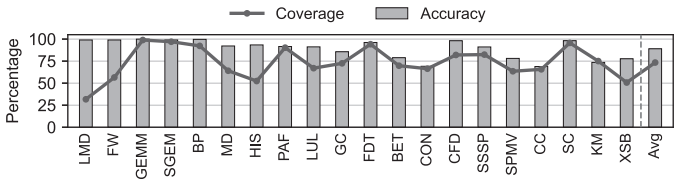


Fig. 18: Speculation accuracy and coverage.

The term 'Fast_Translation' represents the memory accesses affected by rapid validation and EAF. When a rapid validation occurs, EAF combines a TLB entry with the page information obtained from the sector and propagates it throughout the whole address translation hierarchy. As a result, a single rapid validation can influence numerous address translation requests. The 'Fast_Translation' provides much more benefit than the 'L1D_hit' and the 'L1D_merge' since it effectively eliminates TLB-miss overhead. On average, the 'Fast_Translation' accounts for 38.6% of the results. In particular, PAF, CON, CC, SC, and KM show a high portion of 'Fast_Translation', demonstrating outstanding performance gains in Avatar. Remarkably, SC shows a relatively high 'Fast_Translation' ratio (78.9%) despite its low data compressibility (13.5%). This result suggests that Avatar can achieve relatively high performance for low-compressible workloads with the help of EAF.

Lastly, the term 'L1D_miss' signifies that the speculatively-fetched sector has failed to perform rapid validation and is evicted from the cache before the original request uses it (resulting in no benefit from the speculation). This early eviction leads to cache pollution and memory bandwidth waste. However, the 'L1D_miss' only accounts for 2.3% of the results on average, meaning that Avatar does not cause cache pollution despite aggressive data fetching.

Since Avatar aggressively fetches data from speculated addresses, DRAM traffic may increase. However, Avatar only fetches 'a sector', reducing the potential impact on memory bandwidth. Furthermore, Avatar reduces the burden on memory bandwidth by cooperating with EAF. Figure 17a shows the impact of EAF on page walks, indicating a 19.1% decrease in the number of page walks for class-H workloads compared to Promotion. Removing a single page walk can have a similar effect to eliminating multiple memory accesses, countering potential drawbacks of aggressive speculation that may saturate memory bandwidth. Consequently, Avatar can preserve DRAM traffic similar to the non-speculation case (2.2% increased on average, as shown in Figure 17b).

3) Speedup with Ideal Validation: CAST+Ideal-Valid outperforms Avatar by 5.8% on average. For GEMM, MD, BET, and CFD, all variants (i.e., CAST-only, Avatar, and CAST+Ideal-Valid) achieve similar performance. These workloads perform well even when only using CAVA, as mentioned in Section IV-B1. For CON and CC, the performance difference between Avatar and CAST+Ideal-Valid is minimal because of their high data compressibility. In contrast, the low data compressibility of some workloads, such as SC and XSB, results in a noticeable performance gap between Avatar and CAST+Ideal-Valid. Nevertheless, the performance improvement achieved by Avatar is significantly higher compared to CAST-only and other prior techniques for these workloads. This is because the translation information provided by CAVA can be used to validate the speculation and construct a TLB entry for multiple SMs. By constructing a TLB entry before the page walk is completed, Avatar can reduce TLB misses and pipeline stalls, even if a few sectors are compressible. This highlights Avatar's superior capability to enhance performance, even for workloads with low data compressibility.

4) Speculation Accuracy and Coverage: Figure 18 shows the accuracy and coverage of speculations achieved by MOD. The PC-based contiguity detection mechanism results in an average speculation accuracy of 90.3%. Speculation coverage refers to the percentage of correct speculations over all L1 TLB misses. MOD achieves an average speculation coverage of 73.4%. Overall, MOD effectively supports the translation of speculative addresses.

5) Comparison to Prior Works: Page Promotion [7] effectively reduces TLB pressure for workloads exhibiting locality in memory access patterns within a 2MB logical chunk, such as SSSP and SPMV. With the support of a page prefetcher [19], these workloads rapidly fill a logical chunk and promote it to a large page. However, the benefits of Promotion only materialize after such a promotion occurs. In scenarios where a workload utilizes only a small portion of a chunk, it fails to build a large page and remains underperforming, as observed in SGEM and CC). CoLT complements Page Promotion by providing intermediate TLB reach between base and large pages. Workloads like KM greatly benefit from CoLT, as the increased coverage from coalesced TLB entries effectively meets the required translation range. Nonetheless, the intermediate TLB reach offered by CoLT has its limitations. Specific workloads, such as SC and XSB, derive little benefit from CoLT or exhibit a notable performance gap compared to Avatar. These workloads feature irregular memory access patterns, challenging CoLT's ability to meet the translation range they demand. Moreover, the TLB reach CoLT offers comes after a page table walk, lacking the immediacy of solutions like Avatar. While SnakeByte introduces varying degrees of TLB reach through recursive merging (effective for BET and CON), certain workloads such as PAF and CC show decreased performance compared to CoLT. It is because the recursive merging of SnakeByte fundamentally performs best with a specific paging scheme [5], the benefits from recursive merging in a UVM-like paging scheme may not effectively cover the overhead of additional memory references for merg-

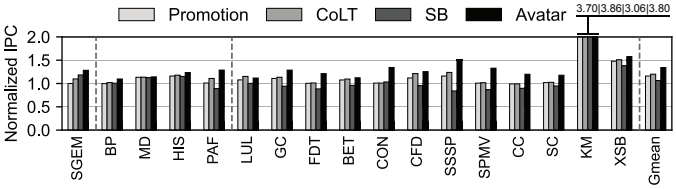
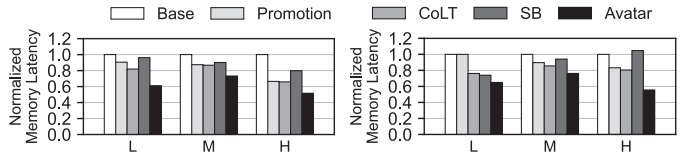


Fig. 19: Performance comparison under memory oversubscription (130%). Results are normalized to the baseline.



(a) No oversubscription. (b) Oversubscription (130%).

Fig. 20: Average memory access latency comparison.

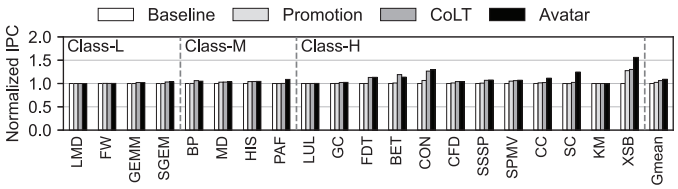
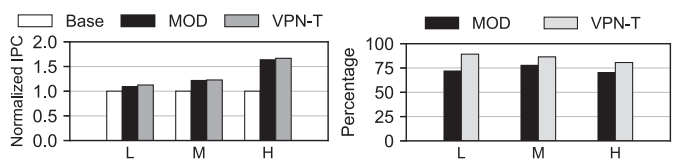


Fig. 21: Performance comparison with 64KB base page. Results are normalized to the baseline.



(a) Performance improvement. (b) Speculation coverage.

Fig. 22: Comparison of MOD and L-TLB.

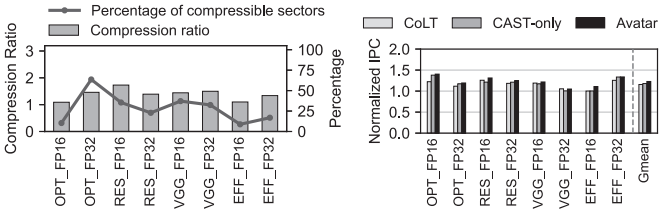
ing. Figure 20a compares average memory access latency on those techniques. Promotion and CoLT adequately reduce the memory latency by releasing TLB overhead. Snakebyte shows slightly higher memory latency than CoLT because of the overhead for recursive merging. On the other hand, Avatar provides immediate translation compared to TLB coalescing. Consequently, Avatar effectively reduces overall memory latency and exhibits an average performance gap of 10.1% or more compared to other techniques.

6) Memory Oversubscription: As mentioned in Section II-E, memory oversubscription can break page contiguity and further impact the benefits of prior works. To analyze the impact of oversubscription, we adjust the main memory size to incur oversubscription (130%) for each workload, as in prior research [19]. We exclude LMD, FW, and GEMM due to their small working sets. As shown in Figure 19, Promotion and CoLT are still effective for some workloads (such as HIS and KM) under oversubscription. However, SSSP, SPMV, and SC show less effectiveness with Promotion and CoLT. For those workloads, page evictions on TLB-hot pages occur frequently. As a result, page evictions significantly reduce TLB reach by flushing TLB entries. For SnakeByte, an eviction reduces TLB reach and initiates more recursive merging, exacerbating the overhead of additional memory references. Figure 20b compares average memory access latency under oversubscription (130%). Compared to the non-oversubscription case, Promotion, CoLT, and Snakebyte are less effective in reducing memory latency on class-H workloads. On the other hand, Avatar provides more oversubscription-resilient benefits compared to prior approaches (performance gap of 14.3% or more). Although oversubscription increases the probability of false speculation when a chunk is reused after an eviction, the replay mechanism (discussed in Section III-D) compensates for the losses caused by false speculation.

C. Sensitivity Analysis

1) Increasing Base Page Size: Page prefetching [3], [4], [19] increases the minimum page fault handling unit from 4KB to 64KB, thereby the base page size also can be increased from 4KB to 64KB. We evaluate Avatar and other prior works with a 64KB base page, adopting the promotion to a 2MB large page in the same manner. We exclude SnakeByte from this evaluation since a 64KB page does not align with SnakeByte's basic operation. Figure 21 indicates that Avatar achieves a 13% average performance improvement over the baseline, outperforming Promotion by 7.2% (9.8% for class-H) and CoLT by 3.0% (4.9% for class-H). The performance gap between CoLT and Avatar is reduced compared to a 4KB base page. This reduction is attributed to the large base page size, which is beneficial for CoLT as it increases the maximum coverage of coalesced TLB entries (up to 1MB). However, certain workloads like SC and XSB still perform better with Avatar. These irregular workloads still touch regions uncovered by CoLT, so they benefit less from CoLT even though TLB coverage is increased. In contrast, direct speculation with Avatar covers any contiguous region within 2MB, resulting in better performance for these workloads.

2) Address Prediction Technique of CAST: Avatar employs a PC-based contiguity tracking mechanism (MOD) to ensure compatibility with other paging schemes. An alternative approach is to utilize a simple VPN-based contiguity tracking method, similar to that used in a previous speculation study [59]. We compare the performance of Avatar using a 32-entry MOD and a 32-entry VPN-based tracking table (VPN-T). Figure 22a shows that VPN-T outperforms MOD by 2.8% due to its capability for direct speculation, unlike MOD, which relies on building confidence via the state counter. As a result, VPN-T offers higher speculation coverage when the entry size is adequate, as depicted in Figure 22b. However, VPN-based speculation is less adaptable to different paging schemes due to its reliance on static prediction based on VPN. Changes in the contiguity range require updating the table entries to maintain accurate coverage of the contiguity region. On the other hand, PC-based MOD is more flexible because it tracks contiguity based on the characteristics of GPU load instructions.



(a) Compressibility (b) Performance

Fig. 23: Compressibility and performance for ML workloads.

3) ML Workloads: We evaluate ML workloads such as opt-LLM (OPT) [79], ResNet50 (RES) [22], VGG16 (VGG) [69], and EfficientNet (EFF) [71] in different precisions (i.e., FP16 and FP32). Figure 23a shows the compressibility of each workload. The average compression ratio is 1.38x, and 28.4% of 32-byte sectors are compressible to 22 bytes. Compression ratios vary across the ML models and tend to be high for the models with high precision (i.e., FP32). We note that the compression ratios measured in our study are lower than the ratio (1.85x) reported in the prior work [15] as we exclude the memory requests loading all zeros in measuring the compression ratio to evaluate the performance gain of the proposed technique conservatively. Figure 23b shows the performance normalized to the baseline across different configurations. For the ML workloads, Avatar outperforms CoLT (best performing among prior approaches) by 7.1% on average despite relatively low data compressibility. This considerable performance gain of Avatar for workloads with limited data compressibility is mainly due to its ability to overlap data fetching with address translation, as discussed in Section IV-B1.

V. Conclusion

This paper presents Avatar, a technique that reduces address translation overheads for GPUs. Avatar speculates the physical address by exploiting page contiguity and rapidly validates the speculated address using the page information stored within data blocks. This novel speculation scheme is orthogonal to various TLB designs and page walk systems in GPUs. It helps break GPUs' architectural limit and the problem of flush on mis-speculation, essentially overcoming a fundamental hurdle for adopting speculative translation. Our experiments show that Avatar achieves an average performance improvement of 37.2% across various workloads.

Acknowledgement

We thank the anonymous reviewers for their valuable comments and feedback that helped improve the quality of our paper. This work was partly supported by the National Research Foundation of Korea (NRF) grant [No. 2022R1C1C1012154, 80%], Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [No. RS-2023-00228970, 20%], the Natural Sciences and Engineering Research Council of Canada (NSERC) [funding number RGPIN-2019-05059], and Samsung Electronics Co., Ltd [No. IO220902-02337-01]. Seokin Hong is the corresponding author.

References

[1] T. M. Aamodt, W. W. L. Fung, T. G. Rogers, and M. Martonosi, *General-Purpose Graphics Processor Architectures*. Springer, 2018.

[2] N. Agarwal, D. Nellans, M. O'Connor, S. W. Keckler, and T. F. Wenisch, "Unlocking Bandwidth for GPUs in CC-NUMA Systems," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 354–365.

[3] T. Allen and R. Ge, "Demystifying GPU UVM Cost with Deep Runtime and Workload Analysis," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 141–150.

[4] ——, "In-Depth Analyses of Unified Virtual Memory System for GPU Accelerated Computing," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021, pp. 1–14.

[5] C. Alverti, S. Psomadakis, V. Karakostas, J. Gandhi, K. Nikas, G. Goumas, and N. Koziris, "Enhancing and Exploiting Contiguity for Fast Memory Virtualization," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 515–528.

[6] A. Arunkumar, S.-Y. Lee, V. Soundararajan, and C.-J. Wu, "LATTE-CC: Latency Tolerance Aware Adaptive Cache Compression Management for Energy Efficient GPUs," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 221–234.

[7] R. Ausavarungnirun, J. Landgraf, V. Miller, S. Ghose, J. Gandhi, C. J. Rossbach, and O. Mutlu, "Mosaic: A GPU Memory Manager with Application-Transparent Support for Multiple Page Sizes," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, p. 136–150.

[8] R. Ausavarungnirun, T. Merrifield, J. Gandhi, and C. J. Rossbach, "PRISM: Architectural Support for Variable-Granularity Memory Metadata," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2020, p. 441–454.

[9] R. Ausavarungnirun, V. Miller, J. Landgraf, S. Ghose, J. Gandhi, A. Jog, C. J. Rossbach, and O. Mutlu, "MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPOLOS)*, 2018, p. 503–518.

[10] T. W. Barr, A. L. Cox, and S. Rixner, "SpecTLB: A Mechanism for Speculative Address Translation," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 307–317.

[11] 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)*, 2020, pp. 596–609.

[12] A. Bhattacharjee, "Translation-Triggered Prefetching," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPOLOS)*, 2017, p. 63–76.

[13] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron, "Pannotia: Understanding Irregular GPGPU Graph Applications," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*, 2013, pp. 185–195.

[14] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[15] E. Choukse, M. B. Sullivan, M. O'Connor, M. Erez, J. Pool, D. Nellans, and S. W. Keckler, "Buddy Compression: Enabling Larger Memory for Deep Learning and HPC Workloads on GPUs," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 926–939.

[16] G. Cox and A. Bhattacharjee, "Efficient Address Translation for Architectures with Multiple Page Sizes," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPOLOS)*, 2017, p. 435–448.

[17] 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 (GPGPU)*, 2010, p. 63–74.

[18] 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)*, 2024, pp. 834–847.

[19] D. Ganguly, Z. Zhang, J. Yang, and R. Melhem, "Interplay between Hardware Prefetcher and Page Eviction Policy in CPU-GPU Unified Virtual Memory," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 224–235.

[20] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a High-Level Language Targeted to GPU Codes," in *2012 Innovative Parallel Computing (InPar)*, 2012, pp. 1–10.

[21] S. Haria, M. D. Hill, and M. M. Swift, "Devirtualizing Memory in Heterogeneous Systems," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPOLOS)*, 2018, p. 637–650.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016, pp. 770–778.

[23] S. Hong, B. Abali, A. Buyuktosunoglu, M. B. Healy, and P. J. Nair, "Touché: Towards ideal and efficient cache compression by mitigating tag area overheads," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 453–465. [Online]. Available: https://doi.org/10.1145/3352460.3358281

[24] S. Hong, P. J. Nair, B. Abali, A. Buyuktosunoglu, K.-H. Kim, and M. Healy, "Attaché: Towards Ideal Memory Compression by Mitigating Metadata Bandwidth Overheads," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 326–338.

[25] A. Jaleel, E. Ebrahimi, and S. Duncan, "DUCATI: High-performance Address Translation by Extending TLB Reach of GPU-accelerated Systems," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 16, no. 1, pp. 1–24, 2019.

[26] S. Jang, J. Park, O. Kwon, Y. Lee, and S. Hong, "Rethinking Page Table Structure for Fast Address Translation in GPUs: A Fixed-Size Hashed Page Table," in *2024 International Conference on Parallel Architecture and Compilation (PACT)*, 2024.

[27] K. Kanellopoulos, R. Bera, K. Stojiljkovic, F. N. Bostanci, C. Firtina, R. Ausavarungnirun, R. Kumar, N. Hajinazar, M. Sadrosadati, N. Vijaykumar, and O. Mutlu, "Utopia: Fast and Efficient Address Translation via Hybrid Restrictive & Flexible Virtual-to-Physical Address Mappings," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, p. 1196–1212.

[28] K. Kanellopoulos, H. C. Nam, N. Bostanci, R. Bera, M. Sadrosadati, R. Kumar, D. B. Bartolini, and O. Mutlu, "Victima: Drastically Increasing Address Translation Reach by Leveraging Underutilized Cache Resources," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, p. 1178–1195.

[29] V. Karakostas, J. Gandhi, F. Ayar, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. Ünsal, "Redundant Memory Mappings for Fast Access to Large Memories," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, p. 66–78.

[30] I. Karlin, "Lulesh Programming Model and Performance Ports Overview," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2012.

[31] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 473–486.

[32] H. Kim, J. Sim, P. Gera, R. Hadidi, and H. Kim, "Batch-Aware Unified Memory Management in GPUs for Irregular Workloads," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, p. 1357–1370.

[33] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-Plane Compression: Transforming Data for Better Compression in Many-Core Architectures," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, 2016, p. 329–340.

[34] G. Koo, Y. Oh, W. W. Ro, and M. Annavaram, "Access Pattern-Aware Cache Management for Improving Data Utilization in GPU," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, p. 307–319.

[35] J. B. Kotra, M. LeBeane, M. T. Kandemir, and G. H. Loh, "Increasing GPU Translation Reach by Leveraging Under-Utilized On-Chip Resources," in *54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021, p. 1169–1181.

[36] O. Kwon, Y. Lee, and S. Hong, "Pinning Page Structure Entries to Last-Level Cache for Fast Address Translation," *IEEE Access*, vol. 10, pp. 114 552–114 565, 2022.

[37] J. Lee, J. M. Lee, Y. Oh, W. J. Song, and W. W. Ro, "SnakeByte: A TLB Design with Adaptive and Recursive Page Merging in GPUs," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 1195–1207.

[38] 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 (MICRO)*, 2023, p. 1163–1177.

[39] B. Li, Y. Wang, and X. Tang, "Orchestrated Scheduling and Partitioning for Improved Address Translation in GPUs," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[40] 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)*, 2023, pp. 456–470.

[41] 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 (MICRO)*, 2021, p. 1154–1168.

[42] C. Li, R. Ausavarungnirun, C. J. Rossbach, Y. Zhang, O. Mutlu, Y. Guo, and J. Yang, "A Framework for Memory Oversubscription Management in Graphics Processing Units," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPOLOS)*, 2019, p. 49–63.

[43] L. Nai, Y. Xia, I. G. Tanase, H. Kim, and C.-Y. Lin, "GraphBIG: Understanding Graph Computing in the Context of Industrial Solutions," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2015, pp. 1–12.

[44] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving GPU performance via large warps and two-level warp scheduling," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011, pp. 308–317.

[45] NVIDIA, "CUDA C++ Programming Guide," 2023. [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[46] ——, "CUDA C/C++ SDK Code Samples," 2023. [Online]. Available: https://developer.nvidia.com/cuda-code-samples.

[47] ——, "Maximizing Unified Memory Performance in CUDA," 2017. [Online]. Available: https://developer.nvidia.com/blog/maximizing-unified-memory-performance-cuda/.

[48] ——, "NVIDIA A100," 2020. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

[49] ——, "NVIDIA Ampere GA102 GPU Architecture," 2021. [Online]. Available: https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.1.pdf.

[50] ——, "NVIDIA Linux Open GPU Kernel Module Source," 2023. [Online]. Available: https://github.com/NVIDIA/open-gpu-kernel-modules.

[51] ——, "NVIDIA Multi-Instance GPU," 2024. [Online]. Available: https://www.nvidia.com/en-us/technologies/multi-instance-gpu/.

[52] ——, "NVIDIA Pascal MMU," 2019. [Online]. Available: https://nvidia.github.io/open-gpu-doc/pascal/gp100-mmu-format.pdf.

[53] ——, "NVIDIA RTX3090," 2022. [Online]. Available: https://www.nvidia.com/content/geforce-gtx/GEFORCE_RTX_3090_USER_GUIDE_v02.pdf.

[54] ——, "NVIDIA TITAN V," 2017. [Online]. Available: https://nvidianews.nvidia.com/news/nvidia-titan-v-transforms-the-pc-into-ai-supercomputer.

[55] Y. Oh, K. Kim, M. K. Yoon, J. H. Park, Y. Park, W. W. Ro, and M. Annavaram, "APRES: Improving Cache Efficiency by Exploiting Load Characteristics on GPUs," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, 2016, p. 191–203.

[56] C. H. Park, T. Heo, J. Jeong, and J. Huh, "Hybrid TLB Coalescing: Improving TLB Translation Coverage under Diverse Fragmented Memory Allocations," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, p. 444–456.

[57] S. Park, M. Kim, and H. Y. Yeom, "GCMA: Guaranteed Contiguous Memory Allocator," *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 390–401, 2019.
[58] B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee, "CoLT: Coalesced Large-Reach TLBs," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012, pp. 258–269.
[59] B. Pham, J. Veselý, G. H. Loh, and A. Bhattacharjee, "Large Pages and Lightweight Memory Management in Virtualized Environments: Can You Have it Both Ways?" in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 1–12.
[60] B. Pichai, L. Hsu, and A. Bhattacharjee, "Architectural Support for Address Translation on GPUs: Designing Memory Management Units for CPU/GPUs with Unified Address Spaces," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPOLOS)*, 2014, p. 743–758.
[61] J. Power, M. D. Hill, and D. A. Wood, "Supporting x86-64 Address Translation for 100s of GPU Lanes," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 568–578.
[62] B. Pratheek, N. Jawalkar, and A. Basu, "Improving GPU Multi-tenancy with Page Walk Stealing," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 626–639.
[63] ——, "Designing Virtual Memory System of MCM GPUs," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 404–422.
[64] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, "Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 78–91.
[65] N. Sakharnykh, "UNIFIED MEMORY ON PASCAL AND VOLTA," *NVIDIA GTC*, 2017.
[66] V. Sathish, M. J. Schulte, and N. S. Kim, "Lossless and lossy memory I/O link compression for improving performance of GPGPU workloads," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 325–334.
[67] S. Shin, G. Cox, M. Oskin, G. H. Loh, Y. Solihin, A. Bhattacharjee, and A. Basu, "Scheduling Page Table Walks for Irregular GPU Applications," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 180–192.
[68] S. Shin, M. LeBeane, Y. Solihin, and A. Basu, "Neighborhood-Aware Address Translation for Irregular GPU Applications," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 352–363.
[69] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[70] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, vol. 127, p. 27, 2012.
[71] M. Tan and Q. Le, "Efficientnet: Rethinking Model Scaling for Convolutional Neural Networks," in *International conference on machine learning (ICML)*, 2019, pp. 6105–6114.
[72] G. Thmoas-Collignon and V. Mehta, "Optimizing CUDA Appications for NVIDIA A100 GPU," *NVIDIA GTC*, 2020.
[73] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz, "XSBench-the development and verification of a performance abstraction for Monte Carlo reactor analysis," *The Role of Reactor Physics toward a Sustainable Future (PHYSOR)*, 2014.
[74] UMC, "28 Nanometer," [Online]. Available: https://www.umc.com/upload/media/05_Press_Center/3_Literatures/Process_Technology/28nm_Brochure.pdf.
[75] N. Vijaykumar, G. Pekhimenko, A. Jog, A. Bhowmick, R. Ausavarungnirun, C. Das, M. Kandemir, T. C. Mowry, and O. Mutlu, "A Case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 41–53.
[76] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "NVBit: A Dynamic Binary Instrumentation Framework for NVIDIA GPUs," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019, p. 372–383.
[77] M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. Fletcher, and J. Torrellas, "InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 428–441.
[78] H. Yoon, J. Lowe-Power, and G. S. Sohi, "Filtering Translation Bandwidth with Virtual Caching," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPOLOS)*, 2018, p. 113–127.
[79] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "OPT: Open Pre-trained Transformer Language Models," *arXiv preprint arXiv:2205.01068*, 2022.
[80] T. Zheng, D. Nellans, A. Zulfiqar, M. Stephenson, and S. W. Keckler, "Towards High Performance Paged Memory for GPUs," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 345–357.