

# Boosting LLC Bandwidth Utilization in GPUs through Adaptive Fine-Grained Data Migration

Jihun Yoon\*, Sungbin Jang<sup>†</sup>, and Seokin Hong<sup>†</sup>

\**Department of Semiconductor Convergence Engineering*, <sup>†</sup>*Department of Electrical and Computer Engineering*  
Sungkyunkwan University, Suwon, Republic of Korea  
{head06, sunbi3361, seokin}@skku.edu

**Abstract**—Modern server-grade GPUs (e.g., NVIDIA A100) integrate hundreds of cores and tens of memory partitions, providing massive compute capability and memory bandwidth. However, the increased scale amplifies interconnect overhead between cores and memory partitions. To mitigate this, NVIDIA A100 clusters multiple cores and memory partitions into two large groups, thereby simplifying interconnect complexity. Unfortunately, this partitioning introduces a new limitation: remote partition accesses. A core accessing a remote memory partition incurs higher latency and lower bandwidth compared to local accesses.

In this paper, we propose a cache-line migration mechanism across partitions to alleviate remote memory access overhead. Our design is motivated by two key observations: (1) conventional GPUs employ limited and often ineffective optimizations for remote access handling, and (2) GPU applications typically exhibit high temporal locality, where a specific partition of cores makes frequent memory accesses for the same data within short time intervals. Leveraging these insights, we dynamically migrate cache-lines to the local partition where the requesting core resides. Experimental results demonstrate that our approach achieves up to 1.24× speedup over the baseline with NVIDIA A100-like replication, highlighting its effectiveness in reducing remote access penalties.

**Index Terms**—GPGPU, Network-on-Chip (NoC), Last-Level Cache (LLC)

## I. INTRODUCTION

The rapid growth of Artificial Intelligence (AI) workloads [1], [8], [15] has driven the evolution of GPUs toward higher compute throughput and memory bandwidth [7], [20]. To meet the demands of increasingly large model sizes [3], [6], [27], recent GPUs have expanded their Last-Level Cache (LLC), implemented as L2 cache, as it plays a crucial role in exploiting data reuse and reducing off-chip traffic. However, increasing the total L2 capacity often requires scaling the number of L2 cache banks, which in turn exacerbates the complexity of on-chip interconnects [10], [28]. To address this, vendors have adopted a partitioned architecture that groups multiple Streaming Multiprocessors (SMs) and the memory partitions (including L2 banks) into GPU partitions [17]. This organization enables higher local L2 bandwidth with reduced interconnect overhead.

While effective in improving local access throughput, this partitioning introduces non-uniform memory access (NUMA) behavior within the GPU. In modern server-grade GPUs, such as NVIDIA A100 and H100, hardware coherence protocols are employed to ensure consistency across partitions, thereby alleviating NUMA effects [17]. However, these protocols are largely undocumented, making it difficult to evaluate their

effectiveness or limitations. Through microbenchmarking on the NVIDIA A100 GPU, we observe that its hardware replication mechanism utilizes a time-based self-invalidation policy. Specifically, after a short delay, a remote cache-line is replicated into the local GPU partition, but is automatically invalidated after a fixed time interval regardless of access frequency. While intended to avoid stale data, this behavior often evicts useful replicas before they can be utilized, thereby degrading performance in workloads with short-term locality.

Furthermore, our analysis reveals that this conservative approach fundamentally conflicts with actual application behavior, primarily due to widespread biased access patterns where data is requested far more frequently from a remote partition than from its native home partition. This phenomenon is significantly amplified at finer granularities, as evidenced by the proportion of biased access rising from 51.7% to 80.4% when evaluated at the 128B cache-line level rather than a coarser 4KB page-level granularity. These findings necessitate a fine-grained migration policy that selectively localizes data based on observed access trends, thereby addressing the inefficient reuse and remote traffic caused by A100’s time-based invalidation.

To address these limitations, we propose *AFM (Adaptive Fine-grained Migration)*, a lightweight hardware mechanism that dynamically migrates data at the cache-line granularity based on short-term access bias. AFM tracks the dominant access GPU partition (local GPU partition or remote GPU partition) for individual cache-lines using a 3-bit saturating counter and migrates the data to the requester’s GPU partition when remote access exceeds a threshold. Unlike replication-based approaches, AFM’s fine-grained management avoids unnecessary duplication, alleviates cache pressure, and improves local L2 bandwidth utilization. While replication blindly copies remote data regardless of actual reuse trends, AFM selectively migrates only when short-term access bias is detected, thereby preserving cache capacity for useful data. This pattern-aware behavior not only reduces remote traffic but also enhances data locality at the GPU partition level. As a result, AFM improves overall performance by an average of 7% (geometric mean), with peak gains reaching 24% across the evaluated benchmarks.

## II. BACKGROUND AND MOTIVATION

### A. Partitioned GPU Architecture

The server-grade General-Purpose Graphics Processing Units (GPGPUs) have significantly increased in scale with each

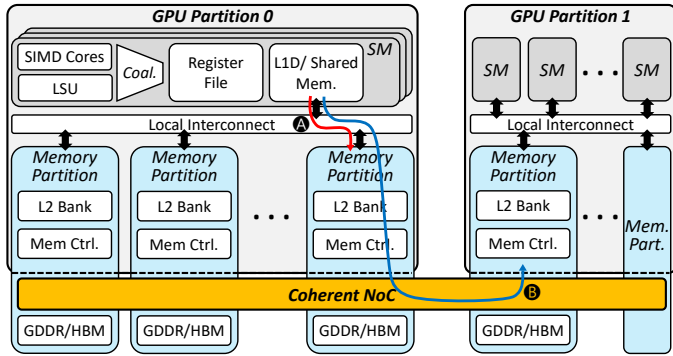


Fig. 1: Baseline GPU Architecture. A single GPU chip consists of two GPU partitions. Path A (red arrow) represents local L2 access, while Path B (blue arrow) represents remote L2 access.

generation. This growth, characterized by an increasing number of Streaming Multiprocessors (SMs) and larger L2 cache capacities, enhances computational throughput and reduces off-chip memory traffic. While these trends boost GPGPU performance, they also pose significant design challenges, particularly in on-chip network complexity. A primary challenge is the escalating complexity of the interconnect. For instance, a monolithic crossbar network scales quadratically  $O(N^2)$  with the number of endpoints ( $N$ ) [22], [28].

To address these scalability bottlenecks, modern GPU designs have shifted toward a partitioned architecture, as illustrated in Figure 1. This design groups multiple SMs and memory partitions, each including L2 cache banks and a memory controller, into distinct GPU partitions. Such an organization enhances effective L2 bandwidth by reducing network contention for local accesses, which is critical for enhancing the overall throughput of parallel applications [17]. A hardware coherence protocol, operating over a Network-on-Chip (NoC), maintains consistency across L2 banks in different GPU partitions. When a request from an SM in one partition targets data, it initially searches its local L2 cache partition (A). If a valid copy of the target data exists (a local hit), the data is returned directly. Otherwise (a local miss), the request traverses the coherence NoC to fetch the data from the remote L2 partition where it resides (B). The coherence NoC ensures the invalidation of stale data copies, allowing requests to always access the latest version of the data [4].

### B. Characteristics of NVIDIA A100 GPU

**Non-Uniform Memory Access.** Recent studies [9], [10] discovered that there is Non-Uniform Memory Access (NUMA) when accessing L2 cache in modern server-grade GPGPUs (e.g., NVIDIA A100). The access to the near L2 cache takes around 200 cycles; on the other hand, it takes around 400 cycles for the remote L2 cache. This NUMA behavior arises because the interconnected L2 partitions function as a distributed set of memory nodes, each with a different access latency depending on its physical distance from the SM. To hide the latency of traversing this on-chip network, data from a remote partition is often replicated into the local partition of the requesting SM. However, this strategy consumes valuable cache capacity,

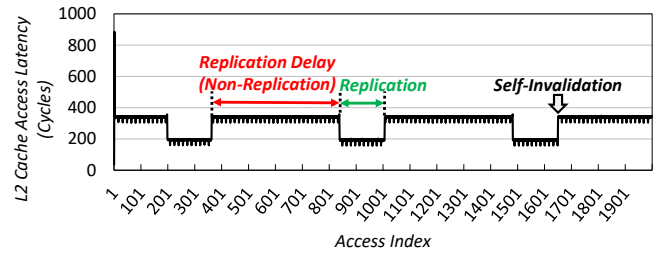


Fig. 2: Behavior of replication in A100 GPU.

which can be detrimental to performance. This necessitates a mechanism to manage the lifecycle of these replicated data blocks efficiently; without timely invalidation or eviction, they can persist unnecessarily in the cache, intensifying pressure and forcing out other useful resident data.

**Time-Based Self-Invalidation.** To investigate this lifecycle management, we characterize the data replication policy of the NVIDIA A100 GPU—and the corresponding policy for invalidating the replicated data—using a targeted synthetic kernel. This kernel forces an SM to repeatedly access a sector (32 bytes of the cache-line) located in a remote L2 cache partition, while using the `.cg` load modifier [18] to bypass the L1 data cache and isolate L2 behavior. Our initial experiment, probing a remote address from a single SM, revealed a bimodal latency distribution (Figure 2). An initial high remote L2 latency (about 400 cycles) would transition to a low, local L2 latency (about 200 cycles). This raises a critical question: Is this behavior caused by data replication (copying the data) or data migration (moving the data)? To distinguish replication from migration, we do additional experiments with an extended kernel that generates local and remote accesses to the same address. The definitive result was twofold: the remote SM’s latency dropped from high to low as expected, while the local SM’s access latency remained consistently low. The latter observation is the key evidence that rules out migration—which would have invalidated the original data—and confirms the mechanism is data replication.

Having established the mechanism as replication, we can now present our key observations regarding its detailed policy. This reveals a sophisticated, time-based protocol unlike traditional access-counter-based schemes [19], [24], [26]. Our analysis uncovers three key properties. First, replication is not immediate but occurs after a noticeable delay. Second, the replica self-invalidates after a fixed duration without external intervention. Finally, we verify the time-based nature by inserting computational delays between accesses; the replica’s lifespan correlated with elapsed time, not access frequency, confirming a mechanism independent of SM execution state.

**Replicated Data Size Awareness.** A critical challenge arising from hardware coherence in NUMA GPUs is managing data replication. While essential for performance, it can lead to a significant drawback: excessive replication of remote data exacerbates cache pressure on the local L2 partition, causing the premature eviction of useful resident data. However, how state-of-the-art GPUs like the NVIDIA A100 resolve this replication-induced pressure at the hardware level remains undocumented and poorly understood.

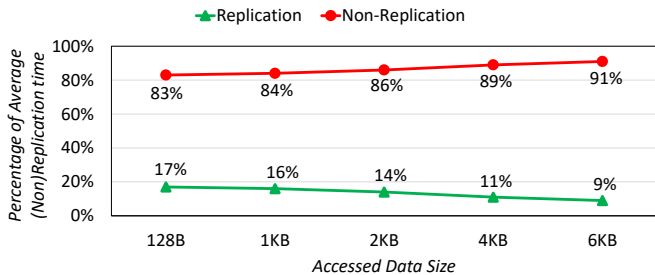


Fig. 3: Replication vs. Non-Replication with various accessed data sizes.

To uncover this mechanism, we develop an extended set of our targeted synthetic kernels. By systematically varying the size of the memory footprint accessed by the SM—ranging from a single cache-line to a large, multi-line region—we can observe how the hardware policy adapts to different levels of induced cache pressure. Figure 3 shows that as the replicated data size grows, the retention cycles of the replica shorten, leading to a faster self-invalidation. These experiments demonstrate that the replication and self-invalidation behavior is not static; instead, it changes dynamically in response to the memory access pattern, demonstrating a clear element of size awareness. Uncovering this previously unknown mechanism not only provides new insights into modern GPU architecture but also establishes an accurate, state-of-the-art baseline. This enables a fair and direct comparison for the enhanced cache management strategy we propose in Section III. These results also imply a trade-off between replication granularity and coherence metadata overhead, revealing the need for fine-grained, resource-aware coherence management.

### C. Detailed Analysis on Memory Access Pattern

Building on this baseline characterization, we further investigate how memory access patterns vary with granularity. To this end, we perform a per-kernel classification, grouping each cache-line based on its remote-to-local access ratio. Figure 4 shows the resulting distribution. A pattern is considered **Biased** if this ratio exceeds 2:1, and **Uniform** if it is more balanced, such as when remote access is less than twice that of local access. The third category, **Streaming Access**, identifies patterns characterized by little to no data reuse, where a cache-line or sector is typically accessed only once. Such patterns exhibit poor temporal locality, rendering traditional caching strategies ineffective. At the fine-grained, 128B cache-line level, the workload is overwhelmingly biased. Our results show that 80.4% of actively accessed lines exhibit a Biased Access pattern, with only 19.6% classified as Uniform. In contrast, a coarse-grained 4KB page-level analysis yields a significantly different distribution: the proportion of Biased Access drops to 51.7%, while Uniform Access appears to increase to 32.5%.

This finding raises questions about the effectiveness of the replication-based strategies employed by the A100 GPU. As characterized in Section II-B, A100’s policy attempts to mitigate cache pressure by replicating only small data blocks (Replicated Data Size Awareness) and by invalidating replicas after a fixed time period (Time-based Self-Invalidation). How-

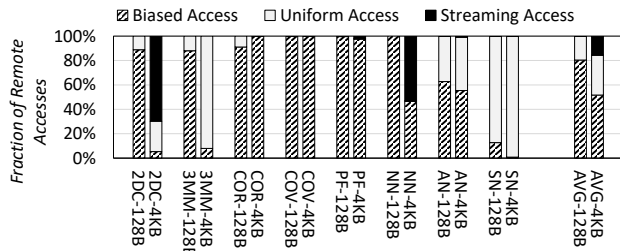


Fig. 4: Breakdown of remote access patterns.

ever, our analysis reveals that these heuristics are insufficient, as the observed access pattern is fundamentally biased regardless of data size. Under such bursty and skewed patterns, replicas are often created but quickly invalidated before they can be reused, leading to unnecessary remote accesses and lost performance opportunities.

These limitations provide a strong motivation for a fundamental shift in strategy from replication to data migration. For the vast majority of cache-lines exhibiting biased access, migration presents a superior performance trade-off. While creating a replica can satisfy the small number of local accesses, this benefit is outweighed by the cache pressure it creates and its inability to serve the high-frequency remote requests. Migration resolves this imbalance by serving the dominant, high-frequency access stream locally at the requesting partition. The few original local accesses can then be affordably rerouted as remote requests—a small cost for the significant gains of eliminating replica-induced cache pressure and maximizing overall L2 bandwidth utilization.

## III. ADAPTIVE FINE-GRAINED MIGRATION

### A. Migration Requirements and Challenges

While migration offers a more effective alternative to replication under biased access patterns, implementing such a mechanism in practice requires overcoming two key challenges. First, the system must be able to detect temporal access trends in hardware to decide when migration is beneficial. Second, the architecture must support redirecting future accesses to the new location after migration has occurred.

Our proposed design, **Adaptive Fine-grained Migration (AFM)**, illustrated in Figure 5, addresses these challenges using two simple but effective components: a per-line 3-bit saturating

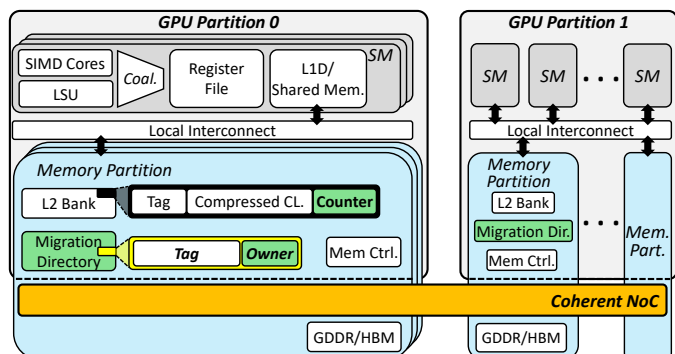


Fig. 5: The AFM GPU architecture.

counter to track access bias, and a lightweight migration directory to locate migrated data.

### B. Migration Directory

Previous studies [16], [23] employed a directory to hold coherence metadata, including the tag, state (e.g., Modified, Shared), and sharer list for each block. Such directory-based coherence protocols must track multiple copies of data, generating significant traffic to maintain consistency.

A migration-based approach, however, fundamentally simplifies this process. It enforces a *single-copy invariant* across the shared L2 cache hierarchy. This guarantees that only one valid copy of a data block exists across all L2 banks at any given time. This L2-level invariant is what obviates the need for complex coherence states (e.g., Modified, Shared) and sharer lists in the directory. Consequently, our migration directory entry is lightweight, holding only the block’s tag, the current owner’s partition ID, and a valid bit. Coherence is still necessary, and the problem boils down to ensuring the directory remains accurate. This requires a targeted invalidation message in two key scenarios: (1) when the owner GPU partition evicts the migrated data, it must invalidate the corresponding directory entry at the home GPU partition. (2) Conversely, when the home directory itself evicts a valid entry (e.g., due to a conflict), it must invalidate the corresponding data block at the owner partition. This simple, symmetric protocol prevents both stale directory pointers and data copies, ensuring correctness with minimal traffic.

### C. 3-bit Saturating Counter

The AFM policy requires a mechanism to dynamically track per-cache-line access patterns. Our design achieves this efficiently with a small 3-bit saturating counter. This low-cost approach is effective because the goal is not to compute perfect, long-term statistics, but to quickly detect the recent dominant access pattern. A 3-bit counter provides sufficient history to track this short-term trend. This allows the system to determine if a data block is currently biased toward remote or local access, which in turn enables a proactive migration decision.

To implement this pattern detection in hardware, the counter employs an asymmetric update policy specifically calibrated to the 2:1 latency disparity between remote and local L2 accesses (400 vs. 200 cycles), ensuring migration targets the primary performance bottleneck. For each access to a cache-line: A remote access increments the counter by one (saturating at the maximum value of 7). A local access decrements the counter by two (saturating at the minimum value of 0). Migration is triggered only when the counter saturates at its maximum value (7). This condition represents that remote accesses have consistently exceeded local accesses by a ratio greater than 2:1 over the recent history captured by the counter. Upon a migration decision, the counter is reset to a neutral state.

**Hardware Overhead and Implementation.** The AFM policy requires storing a 3-bit saturating counter with each L2 cache-line. To implement this 3-bit metadata with minimal overhead, we leverage Bit-Plane Compression (BPC) [14]. The feasibility and low cost of this approach are well-supported by prior

work [21]. Specifically, prior work has shown not only that BPC achieves a sufficient compression ratio on GPU workloads to accommodate the 3-bit metadata, but also that the required (de)compressor engines incur a negligible area overhead. Prior work reported a total area of 0.314mm<sup>2</sup> for a 16-controller configuration [21]. Linearly scaling this to our 40-controller GPU model yields an estimated area of 0.785mm<sup>2</sup>, which is still less than 0.1% of an 800mm<sup>2</sup> GPU die. To ensure robustness, if a cache-line cannot be compressed (an infrequent event), our AFM policy simply disables migration for that specific line, treating it with the baseline’s standard non-migratory policy. This fallback mechanism guarantees that AFM only provides performance gains, preventing any potential degradation in these corner cases.

### D. Overview of the AFM Mechanism

In the AFM architecture, decompression adds a small latency overhead when accessing compressed cache-lines. Nonetheless, local L2 cache hits are serviced in the same manner as in the baseline design, without any change to the request flow. However, the key challenge—and the primary optimization target of AFM—arises when the requested data resides in a remote partition. To address this, AFM introduces a fine-grained migration policy for dynamically managing remote accesses. The following sections detail the two key phases of this mechanism: triggering a migration and locating data post-migration.

**Migration Decision and Execution.** Figure 6 illustrates the process of deciding upon and executing a data migration. When a request from a remote GPU partition arrives at the data’s home GPU partition (e.g., GPU Partition 0, which contains the data with tag 0x001 and 0x003) (A), the L2 cache controller checks a per-cache-line 3-bit saturating counter. **Case 1: No Migration.** If the counter is below the migration threshold (as depicted in the data with tag 0x003), the request is treated as a standard remote access. The data is sent back to the requester but is not cached in the requester’s L2 cache (B). **Case 2: Trigger Migration.** If the counter has reached its saturation threshold (as depicted in the data with tag 0x001), the AFM policy initiates a migration. The home GPU partition’s controller updates a local migration directory with the new owner’s identity (the requester’s partition ID) (C). Subsequently, it sends the data block with a special migrated reply, which is then cached in the requesting GPU partition’s L2 cache (D). Once cached in the L2 cache, the migrated data block (with tag 0x001) is managed by the memory partition’s standard cache

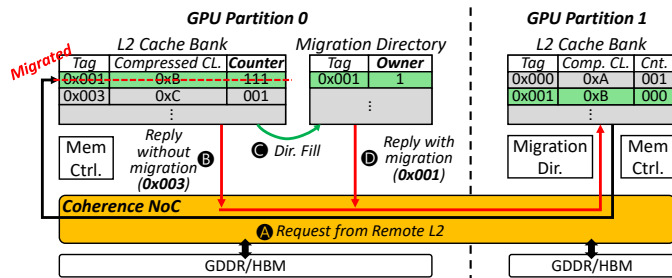


Fig. 6: The AFM migration decision and execution flow.

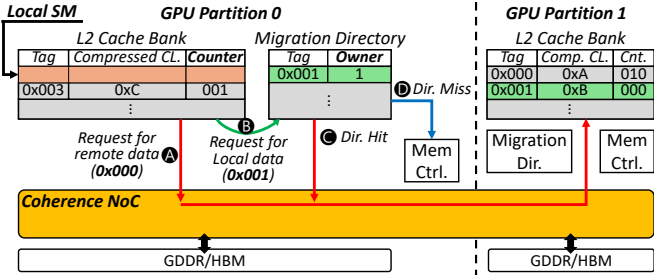


Fig. 7: Lookup process for migrated data.

replacement policy (e.g., LRU). This interaction is one factor that can influence overall performance (Section IV-B).

**Locating and Accessing Migrated Data.** The process of locating data in AFM critically depends on whether the requested address is mapped to the local or a remote GPU partition (and its memory partition), as determined by the system’s address mapping scheme [9]. Figure 7 illustrates the handling of requests for data originally mapped to the remote GPU partition and the local GPU partition. We describe the two distinct cases below. **Case 1: Accessing Remotely Mapped Data.** If an L2 miss occurs for data (with tag 0x000) that is natively mapped to a remote GPU partition, the request is forwarded to that destination GPU partition via the NoC (A). This path handles standard remote accesses for data that has not been migrated into the current GPU partition. It follows the same protocol as the baseline architecture and does not involve the local migration directory. **Case 2: Accessing Locally Mapped Data.** If a request is for data (with tag 0x001) originally mapped to the SM’s own local partition, the local L2 cache is checked first. A cache hit is served directly. However, an L2 miss implies the data is either not on-chip or has been migrated elsewhere. This specific event—an L2 miss for a locally-homed address—is what triggers a lookup in the migration directory (B). A directory hit indicates the data has been migrated, providing the new owner’s location (e.g., GPU Partition 1) to which the request is then forwarded (C). A directory miss implies the data is not resident in the L2 cache hierarchy, prompting the memory controller to fetch it from off-chip memory (HBM/GDDR) (D). When this data returns from memory to its home partition, the migration directory is checked again. If the entry is valid, the data is forwarded directly to the current owner partition, ensuring the migrated copy remains the single coherent version on-chip.

We also account for the specific case of a sector miss on data that has already been migrated. In this scenario, a request for the corresponding sectors traverses the NoC as a remote access to the memory controller of the original home GPU partition. The home controller then fetches the required sectors from its adjacent off-chip memory. Upon the return of the data from DRAM, the home partition queries the migration directory. If a valid entry for the migrated line exists, the sectors are forwarded directly to the current owner partition. While this data path incurs additional NoC traversal, we found this sequence of events to be infrequent in our evaluated workloads, resulting in a negligible performance impact.

TABLE I: Simulated GPU configuration.

Parameter	Value
Core	98 SMs, 4 warp schedulers per SM, Greedy-then-oldest (GTO) scheduler
Clock Frequency	Core/Interconnect/L2 @ 1410MHz
L1 Data Cache	192KB, 37 cycles, 128B line (sectored), adaptive, write-through, write-no-allocate
L2 Cache	40MB in total, 2 L2 partitions, 80 L2 banks, 128B line (sectored), write-back
L2 Access Latency	(Local) 200 cycles (Remote) 388 cycles Including 7 cycles for decompression
L2 Miss Penalty	240 cycles
Interconnect	Partitioned Crossbar (3.6 TB/s for each)
Migration Directory	4K entries, 16-way
HBM	1512MHz, 40 channels, dual bus interface, FR-FCFS
HBM Timing	RCD=22, RC=72, RP=22, CL=22, WL=4, RAS=50, RRD=7, CCD=1, CDLR=5, CCDL=4

TABLE II: Tested benchmarks.

Benchmark Suite	Benchmarks
PolyBench [5]	2D Convolution (2DC), 3 Matrix Multiplications (3MM), Correlation (COR), Covariance (COV)
Rodinia [2]	Particlefilter-Float (PF), K-Nearest Neighbors (NN), LU Decomposition (LUD)
Tango [11]	AlexNet (AN), ResNet (RN), SqueezeNet (SN), CifarNet (CFN)
Parboil [25]	Breadth-First Search (BFS)

## IV. EVALUATION

### A. Methodology

**Simulation Setup.** We perform our evaluation using Accel-Sim, a trace-driven GPGPU framework [13] integrated with the cycle-accurate GPGPU-Sim v4.2 [12]. We extend the simulator to model our partitioned GPU architecture, including inter-partition communication, with the detailed parameters specified in Table I.

We compare the following three architectural models:

- **Baseline:** Partitioned GPU architecture with A100-like replication, which incorporates the features described in Section II-B.
- **AFM:** The Baseline architecture enhanced with our proposed Adaptive Fine-grained migration (AFM) policy.
- **Migration Cache:** AFM with a dedicated migration cache. This configuration is used to isolate the performance of the migration policy itself from the effects of migration-induced cache pressure.

**Benchmarks.** We use several memory-intensive applications from Polybench [5], Rodinia [2], and Parboil [25] suites, in addition to several Deep Neural Network (DNN) models [11]. Table II lists the specific applications and their abbreviations. The majority of our benchmarks are simulated to completion. However, for long-running workloads, such as DNN benchmarks and graph processing workloads, we limit the maximum number of instructions in simulations to 100 million.

### B. Performance Analysis

To understand how AFM improves GPU performance, we analyze the following contributing factors: instruction throughput

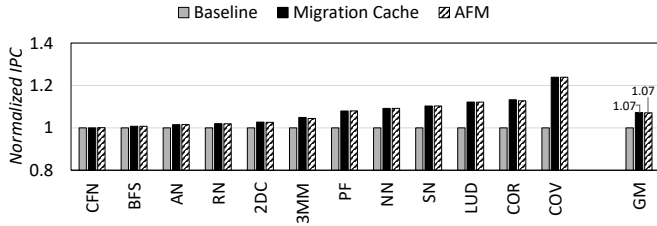
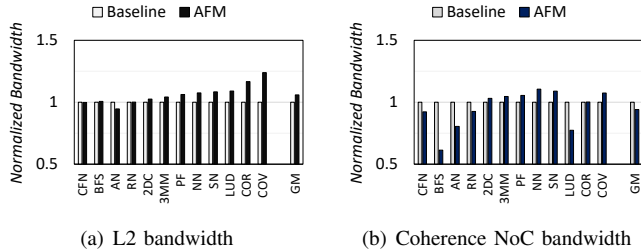


Fig. 8: Normalized GPU speedup.



(a) L2 bandwidth

(b) Coherence NoC bandwidth

Fig. 9: Impact of AFM on L2 and coherence NoC bandwidth.

(as measured by IPC), local L2 bandwidth utilization, reduction in remote accesses through migration compared to replication, and changes in overall cache behavior.

**Instruction Throughput (IPC).** Figure 8 presents the normalized IPC across all evaluated workloads. The proposed AFM policy improves performance by 7% on average (geometric mean) over the baseline. A migration cache—with dedicated storage—achieves similar gains, as both reduce redundant remote accesses. AFM achieves performance nearly equivalent to a dedicated migration cache by seamlessly integrating with the standard L2 cache replacement policy. This synergy allows the system to dynamically adapt to shifting access patterns while minimizing cache pressure, ensuring high efficiency within the standard memory hierarchy. Among all benchmarks, the COV kernel exhibits the highest performance gain, achieving a  $1.24\times$  speedup over the baseline, demonstrating AFM’s potential under highly biased access patterns.

**Local L2 Bandwidth and NoC Utilization.** Figure 9 illustrates the local L2 bandwidth and coherence NoC bandwidth utilization, both normalized to the corresponding baseline bandwidth. Compared to the baseline, AFM improves local L2 bandwidth utilization by 5.9% on average (geometric mean), with the COV benchmark showing the largest gain of  $1.24\times$ . This improvement results from reducing remote accesses and more effectively utilizing on-partition data locality. As a consequence, AFM also reduces coherence NoC traffic, lowering overall network contention and access latency. On average, AFM utilizes only 94% of the baseline’s NoC bandwidth, demonstrating improved efficiency in inter-GPU partition communication.

**Replication/Migration Hit Rate.** To further analyze the source of performance gains, we evaluate the replication/migration hit rate, as shown in Figure 10. AFM achieves a 5.88% replication/migration hit rate (the fraction of L2 accesses satisfied by relocated data), significantly outperforming the baseline’s 0.61%. While the baseline’s aggressive self-invalidation evicts replicas prematurely, AFM selectively migrates biased lines to enable meaningful reuse without unnecessary duplication.

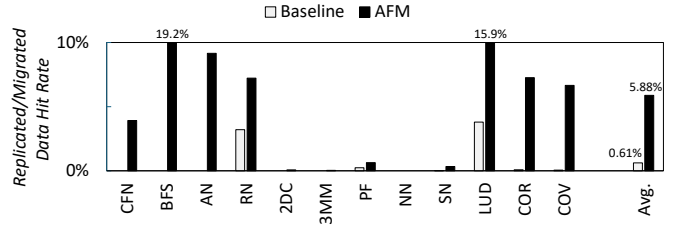


Fig. 10: Replicated/Migrated data hit rate in L2 cache.

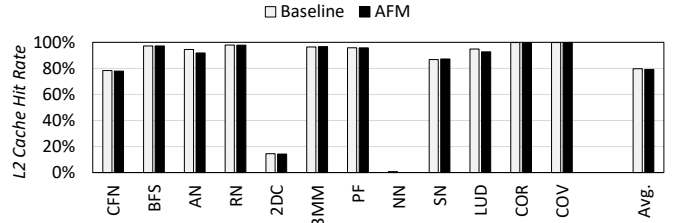


Fig. 11: Overall L2 cache hit rate.

However, workloads with high L1 locality exhibit marginal gains because L2 improvements have a diminished impact on their overall execution time.

**L2 Hit Rate and Cache Pressure.** Figure 11 shows the overall L2 hit rate across all evaluated benchmarks. AFM maintains a comparable hit rate to the baseline, with an average difference of only 1.0 percentage points and a maximum difference of less than 3.0 points in the worst case. This indicates that AFM’s selective migration strategy does not significantly discard useful local data. Instead, it preserves the effective L2 capacity while simultaneously improving access locality.

## V. CONCLUSION

This paper proposes *AFM (Adaptive Fine-grained Migration)*, a hardware-efficient mechanism that improves GPU memory performance by replacing NVIDIA A100’s inefficient replication strategy with selective migration for temporally biased workloads. By leveraging a migration directory and 3-bit saturating counters, AFM achieves up to  $1.24\times$  speedup and a 5.88% higher hit rate for migrated data while maintaining overall L2 hit rates with minimal cache pressure. These results demonstrate that AFM effectively enhances data locality and local L2 bandwidth utilization, offering a scalable and practical direction for future GPU cache architectures.

## ACKNOWLEDGMENT

This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.10692981, 50%), the Korea Institute for Advancement of Technology(KIAT) grant funded by the Korea Government(MOTIE)(No.P0023704, 25%), and the IITP(Institute of Information & Communications Technology Planning & Evaluation)-ITRC(Information Technology Research Center) (RS-2021-II212052, 25%) grant funded by the Korea government(Ministry of Science and ICT). Seokin Hong is the corresponding author.

## REFERENCES

- [1] G. Byeon, S. Kim, H. Kim, S. Han, J. Kim, P. Nair, T. Kang, and S. Hong, "Avalanche: Optimizing cache utilization via matrix reordering for sparse matrix multiplication accelerator," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1746–1759.
- [2] 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)*. IEEE, 2009, pp. 44–54.
- [3] H. Cho, Y. Jang, H. Kim, S. Kim, K. Kwon, G. Kim, and S. Hong, "Librapim: Dynamic load rebalancing to maximize utilization in pim-assisted llm inference systems," in *2025 34th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2025, pp. 43–56.
- [4] W. A. GANDHI, T. Mandal, R. K. Manyam, and S. S. Rao, "Coherent caching of data for high bandwidth scaling," Feb. 9 2021, uS Patent 10,915,445.
- [5] 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)*. IEEE, 2012, pp. 1–10.
- [6] S. Han, S. Kim, G. Byeon, J. Yoon, and S. Hong, "Zebra: Leveraging diagonal attention pattern for vision transformer accelerator," in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–7.
- [7] S. Jang, J. Park, Y. Lee, O. Kwon, D. Kim, J. Seok, and S. Hong, "Softwalker: Supporting software page table walk for irregular gpu applications," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, 2025, pp. 401–417.
- [8] Y. Jang, H. Cho, Y. Ryu, J. Kim, and S. Hong, "Pimpal: Accelerating llm inference on edge devices via in-dram arithmetic lookup," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2025, pp. 1–7.
- [9] Z. Jia, P. Sandt, M. Maggioni, J. Smith, and D. Scarpazza, "Dissecting the ampere gpu architecture through microbenchmarking," *GTC*. <https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s33322>, 2021.
- [10] Z. Jin, C. Rocca, J. Kim, H. Kasan, M. Rhu, A. Bakhoda, T. M. Aamodt, and J. Kim, "Uncovering real gpu noc characteristics: Implications on interconnect architecture," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 885–898.
- [11] A. Karki, C. P. Keshava, S. M. Shivakumar, J. Skow, G. M. Hegde, and H. Jeon, "Tango: A deep neural network benchmark suite for various accelerators," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 137–138.
- [12] M. Khairy, J. Akshay, T. Aamodt, and T. G. Rogers, "Exploring modern gpu memory system design challenges through accurate modeling," *arXiv preprint arXiv:1810.07269*, 2018.
- [13] 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)*. IEEE, 2020, pp. 473–486.
- [14] 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*, 2016, pp. 329–340.
- [15] S. Kim, G. Byeon, S. Kim, H. Kim, and S. Hong, "Conveyor: Towards asynchronous dataflow in systolic array to exploit unstructured sparsity," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 2023, pp. 423–431.
- [16] G. Ko, J. Lee, H. Kal, H. Lee, and W. W. Ro, "Rec: Enhancing fine-grained cache coherence protocol in multi-gpu systems," *Journal of Systems Architecture*, vol. 160, p. 103339, 2025.
- [17] *NVIDIA A100 Tensor Core GPU Architecture*, NVIDIA, 2020. [Online]. Available: <https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf>
- [18] *Parallel Thread Execution ISA Version 9.0*, NVIDIA, 2025. [Online]. Available: <https://docs.nvidia.com/cuda/parallel-thread-execution/>
- [19] M. Oh, K. Kim, D. Choi, H.-J. Lee, and E.-Y. Chung, "Per-operation reusability based allocation and migration policy for hybrid cache," *IEEE Transactions on Computers*, vol. 69, no. 2, pp. 158–171, 2019.
- [20] J. Park, S. Jang, O. Kwon, Y. Lee, and S. Hong, "Leveraging chiplet-locality for efficient memory mapping in multi-chip module gpus," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, 2025, pp. 1040–1057.
- [21] J. Park, O. Kwon, Y. Lee, S. Kim, G. Byeon, J. Yoon, P. J. Nair, and S. Hong, "A case for speculative address translation with rapid validation for gpus," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 278–292.
- [22] G. Passas, M. Katevenis, and D. Pnevmatikatos, "Vlsi micro-architectures for high-radix crossbar schedulers," in *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, 2011, pp. 217–224.
- [23] X. Ren, D. Lustig, E. Bolotin, A. Jaleel, O. Villa, and D. Nellans, "Hmg: Extending cache coherence protocols across modern hierarchical multi-gpu systems," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 582–595.
- [24] G. Schieffer, J. Wahlgren, J. Ren, J. Faj, and I. Peng, "Harnessing integrated cpu-gpu system memory for hpc: a first look into grace hopper," in *Proceedings of the 53rd International Conference on Parallel Processing*, 2024, pp. 199–209.
- [25] 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, no. 7.2, 2012.
- [26] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "Llc-guided data migration in hybrid memory systems," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 932–942.
- [27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [28] X. Zhao, M. Jahre, Y. Tang, G. Zhang, and L. Eeckhout, "Nuba: Non-uniform bandwidth gpus," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 544–559.