UNIVERSITY OF CALIFORNIA
RIVERSIDE

The MALL is Open: Exploring Shared Caches and Latency in AMD CDNA™ 3 GPUs

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Electrical Engineering

by

Andrew Tee

December 2025

Thesis Committee:

Dr. Daniel Wong, Chairperson
Dr. Hung-Wei Tseng
Dr. Elaheh Sadredini

The Thesis of Andrew Tee is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to express my deepest gratitude to my advisor, Dr. Daniel Wong, for his guidance, support, and encouragement throughout my graduate studies. His mentorship has been instrumental to my growth as a researcher, and this work would not have been possible without his continual support.

This work was carried out in collaboration with AMD. I am especially grateful to my mentors, Dr. Noah Wolfe and Dr. Nicholas Curtis, for their invaluable guidance, technical insights, and constant support throughout this project. I want to thank Noah for providing me with the opportunity to work on this research, and both Noah and Nick for the countless discussions that shaped the direction and rigor of this thesis.

To my parents and my sister, for their unwavering support and encouragement throughout this journey.

ABSTRACT OF THE THESIS

The MALL is Open: Exploring Shared Caches and Latency in AMD CDNA™ 3 GPUs

by

Andrew Tee

Master of Science, Graduate Program in Electrical Engineering
University of California, Riverside, December 2025
Dr. Daniel Wong, Chairperson

Understanding GPU memory hierarchy is essential for achieving high performance in scientific and machine learning workloads. This thesis analyzes memory latency on AMD Instinct™ MI300A, MI300X, and MI250X GPUs using a fine-grained pointer chasing microbenchmark. We characterize the scalar L1 (sL1), L2, AMD Infinity Cache™ (MALL), and HBM, revealing clear latency levels and architectural differences. The MI300A and MI300X, based on the AMD CDNA™ 3 architecture, show similar behavior, while the MI250X (AMD CDNA™ 2) differs due to the absence of a MALL. Compute partitioning has negligible impact on latency, but NUMA Partitioning per Socket (NPS) reduces latency by up to 1.42× in MALL and 1.31× in HBM. Notably, we find that NPS4 not only partitions the HBM stacks but also divides the 256 MB MALL into four 64 MB slices. We further analyze Translation Lookaside Buffer (TLB) behavior under varying parallelism levels and identify conditions where MALL latency rises. These findings provide actionable guidance for optimizing memory-bound workloads on AMD GPUs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern GPUs rely on deep and complex memory hierarchies to sustain the massive parallelism required by scientific computing and machine learning workloads. AMD Instinct™ MI300A and MI300X, built on the AMD CDNA™ 3 architecture [4], introduce a shared MALL, which adds a new layer to the traditional scalar and vector L1, L2, and HBM memory hierarchy. This thesis investigates the latency behavior of these memory layers using a fine-grained pointer chasing benchmark, which is sensitive to cache and TLB behavior.

To evaluate whether compute partitioning influences memory latency, we systematically characterized all available compute modes on the MI300A and MI300X. We then compare the MI300A and MI300X against the MI250X, which lacks a MALL, to highlight architectural differences and their performance implications. Our analysis reveals four distinct latency levels in MI300A/X and three in MI250X, with the MALL providing up to a 1.27× speedup under certain buffer sizes.

However, this performance benefit is highly sensitive to TLB capacity, memory contention, and locality. Our heatmap analysis and pointer chasing results show that latency increases before the MALL reaches its nominal 256 MB capacity. We attribute this early onset of latency to TLB limitations. Additionally, as more processes compete for the shared MALL capacity, cache contention increases, leading to capacity misses and further performance degradation.

In NPS4 mode, we discover that the 256 MB MALL is not unified, but is instead partitioned into four 64 MB slices. HBM is similarly partitioned. This partitioning scheme reduces interconnect traversal and enhances memory locality, resulting in latency reductions of up to 1.42× and 1.31× for MALL and HBM.

With the exception of the analysis presented in Section 4.2, all experiments were conducted in NPS1 mode. Results are expected to vary under different NPS configurations that alter MALL and HBM partitioning. Additionally, because the pointer chasing benchmark was single-threaded and uniform, the compiler scalarized the workload, resulting in accesses to primarily exercise the scalar cache hierarchy. As a result, memory accesses targeted the scalar cache hierarchy.

This study aims to provide users of AMD systems with actionable insights into memory latency behavior across buffer sizes, compute modes, and parallel configurations. By identifying the thresholds at which MALL becomes saturated, we offer practical guidelines for workload tuning and system design on CDNA3-based GPUs.

Chapter 2

Background and Related Work

In this section, we provide the architectural background needed to interpret our latency measurements on the AMD Instinct™ MI300A, MI300X, and MI250X GPUs. We introduce the AMD CDNA™ 3 chiplet architecture, outline the memory hierarchy (sL1, vL1, L2, and MALL), and summarize the compute and NUMA partitioning modes. We also introduce the MI250X (AMD CDNA™ 2), which lacks a MALL.

2.1 AMD CDNA™ 3 Chiplet Architecture

The AMD Instinct™ MI300A and MI300X both leverage the AMD CDNA™ 3 architecture and advanced 3D packaging technologies [4]. These processors integrate multiple chiplets, specifically Accelerator Complex Dies (XCDs) for GPU compute and CPU Complex Compute Dies (CCDs) for CPU compute in the case of MI300A, interconnected via AMD Infinity Fabric™. Figure 2.1 illustrates the chiplet architecture for both MI300X and MI300A.

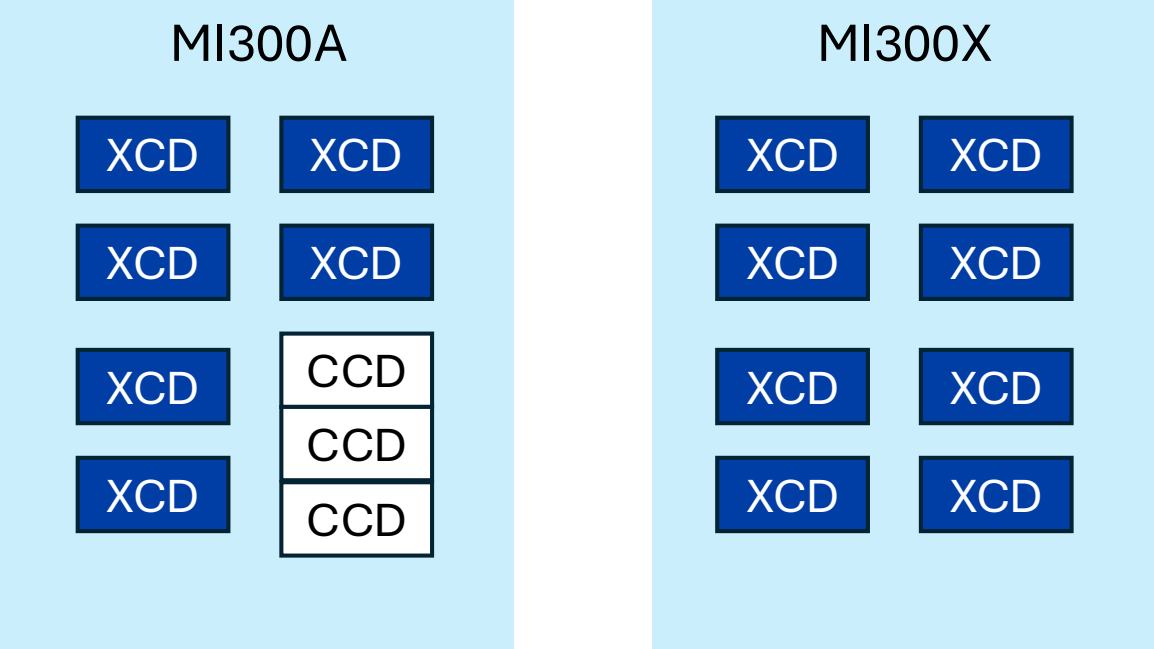


Figure 2.1: AMD Instinct™ MI300X and MI300A chiplet architecture.

- **MI300X** comprises **8 XCDs**, each representing a GPU chiplet with a local **24 GB HBM3 stack**, totaling **192 GB** of high-bandwidth memory.
- **MI300A** integrates **6 XCDs** and **3 CCDs** (Zen 4 CPU chiplets), with each XCD paired with a local **16 GB HBM3 stack**, totaling **128 GB** of unified memory shared across CPU and GPU.

2.2 AMD CDNA™ 2 Chiplet Architecture and Memory Hierarchy

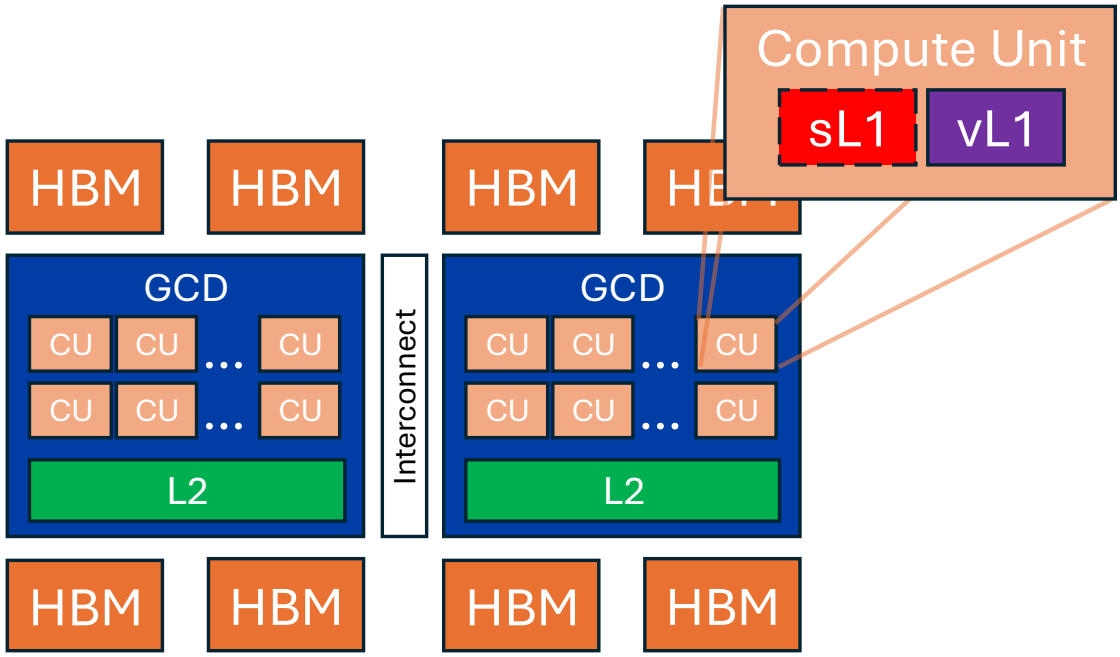


Figure 2.2: Simplified AMD Instinct™ MI250X chiplet and memory hierarchy.

The MI250X consists of two Graphics Compute Dies (GCDs), each with an 8 MB L2 cache and four HBM2e stacks. Unlike AMD CDNA™ 3 devices, the AMD CDNA™ 2 architecture has no shared last level cache (MALL), making L2 the final on-chip cache before HBM. Each GCD includes multiple Compute Units (CUs) with a sL1 shared across adjacent CUs, private vL1, and a L2 shared across all compute units. The larger 8 MB L2 and absence of a MALL are key architectural differences influencing MI250X latency behavior.

2.3 AMD CDNA™ 3 Memory Hierarchy

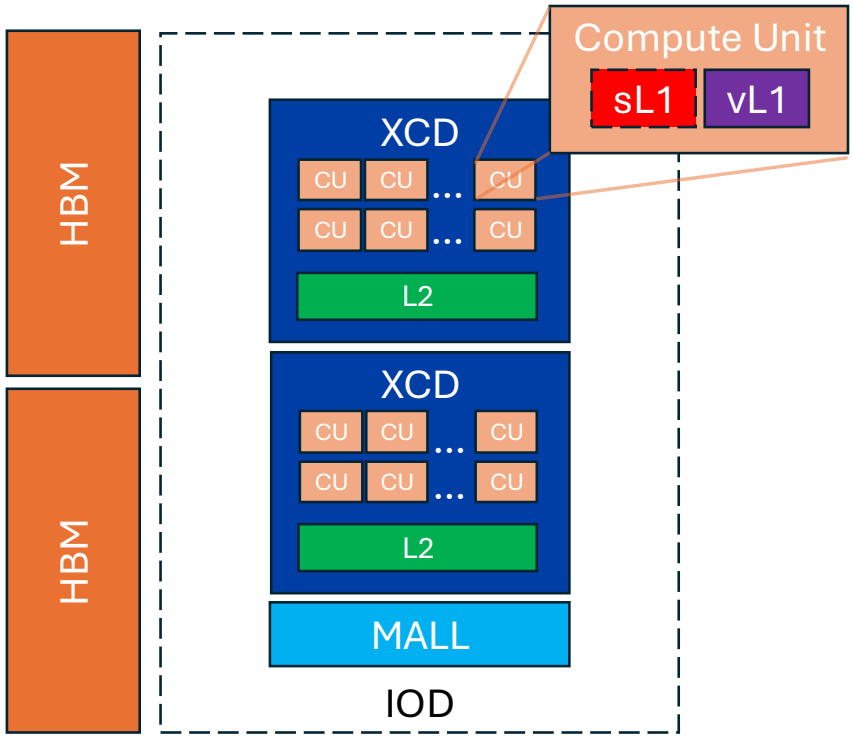


Figure 2.3: Simplified memory hierarchy for the AMD Instinct™ MI300 series GPUs, illustrating the scalar and vector L1 caches, L2 cache, MALL, and HBM.

The AMD CDNA™ 3 architecture uses a multi-level memory hierarchy shown in Figure 2.3. The major components are summarized below:

- **Vector L1 Cache (vL1):** A private L1 cache for each Compute Unit (CU), servicing vector memory operations.
- **Scalar L1 Cache (sL1):** Shared among a group of adjacent CUs, used for scalar memory operations.
- **L2 Cache:** All CUs in an XCD share a single L2.

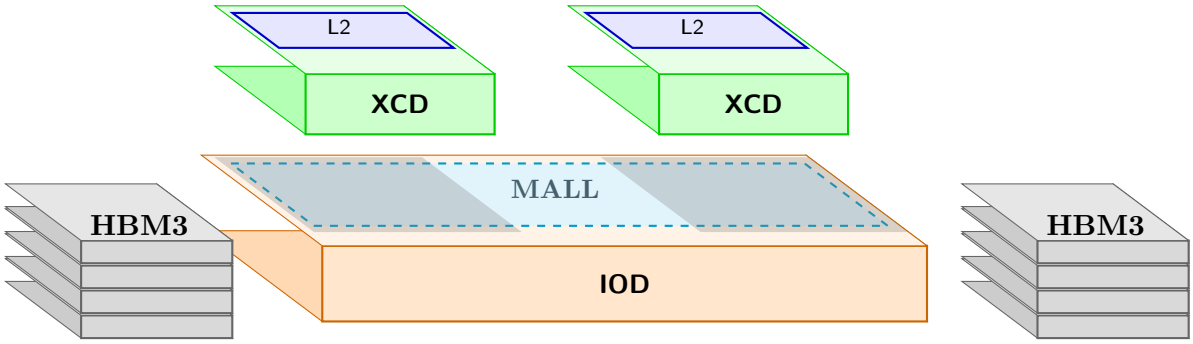


Figure 2.4: The MALL resides on the IOD, with XCD stacked above it.

- **MALL (Memory Attached Last Level Cache):** last level cache situated on the I/O Die (IOD) see Figure 2.4. MALL is shared across all XCDs.
- **High-Bandwidth Memory (HBM):** The final level of the hierarchy, consisting of multiple HBM3 stacks shared across all XCDs.

Table 2.1 summarizes the cache and memory configurations for MI300A, MI300X, and MI250X. While all three GPUs feature identical scalar L1 cache sizes, notable differences emerge deeper in the memory hierarchy. MI250X includes an 8 MB L2 cache per Graphics Compute Die (GCD), whereas MI300A and MI300X provide 4 MB per XCD. Additionally, MI300A and MI300X introduce the MALL (256 MB), a third-level cache absent in MI250X. MI300X also offers a larger HBM capacity (192 GB), compared to 128 GB on both MI250X and MI300A.

Cache Type	**MI300A**	**MI300X**	**MI250X**
Scalar L1 Cache (sL1)	16 KB	16 KB	16 KB
L2 Cache	4 MB	4 MB	8 MB
MALL (Total)	256 MB	256 MB	None
HBM (Total)	128 GB	192 GB	128 GB

Table 2.1: Cache and memory sizes for AMD Instinct™ MI300A, MI300X, and MI250X.

2.3.1 Compute Partitioning Modes

The MI300 series supports compute partitioning, allowing flexible configuration of logical GPU devices [8] [4]:

- **MI300X** allows:
  - **SPX**: All 8 XCDs unified as one logical GPU.
  - **DPX**: Two logical GPUs, each comprising 4 XCDs.
  - **QPX**: Four logical GPUs, each with 2 XCDs.
  - **CPX**: Eight independent logical GPUs, one per XCD.
- **MI300A** allows:
  - **SPX**: Unified CPU GPU configuration.
  - **TPX**: Three GPU partitions (2 XCDs each) and one CPU partition.
  - **CPX**: Six independent GPU partitions and one CPU.

Figure 2.5 summarizes the MI300X partitioning configurations, and Figure 2.6 summarizes the MI300A configurations.

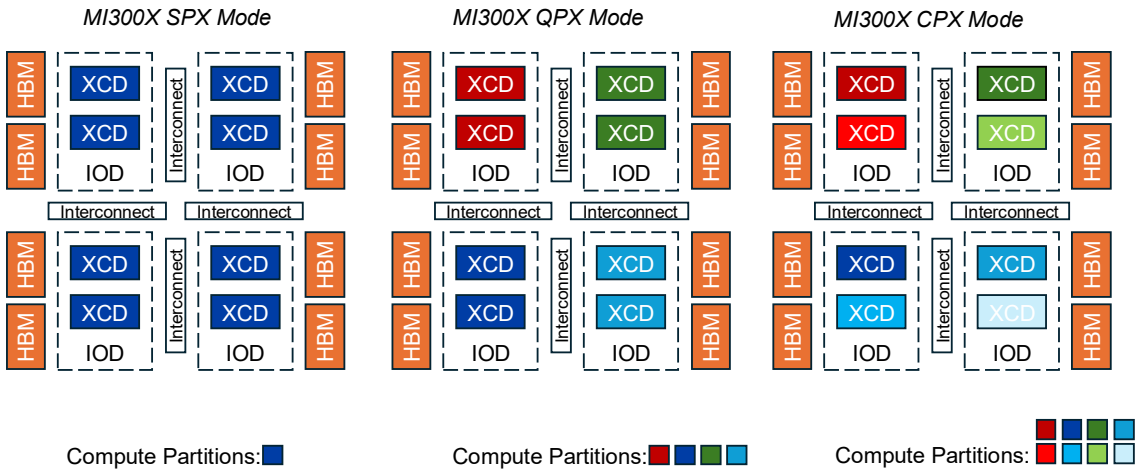


Figure 2.5: Compute partitioning modes supported on AMD Instinct™ MI300X.

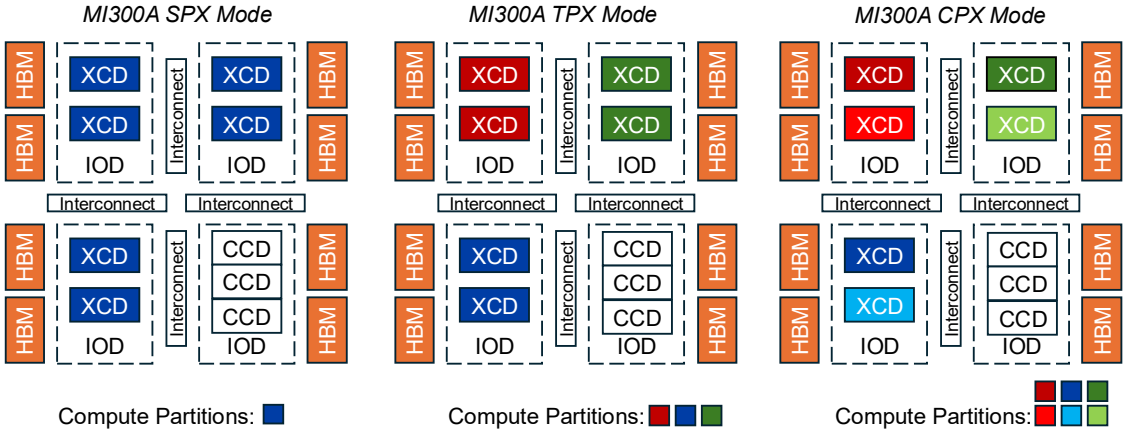


Figure 2.6: Compute partitioning modes supported on AMD Instinct™ MI300A.

2.3.2 NUMA Partitioning (NPS Modes)

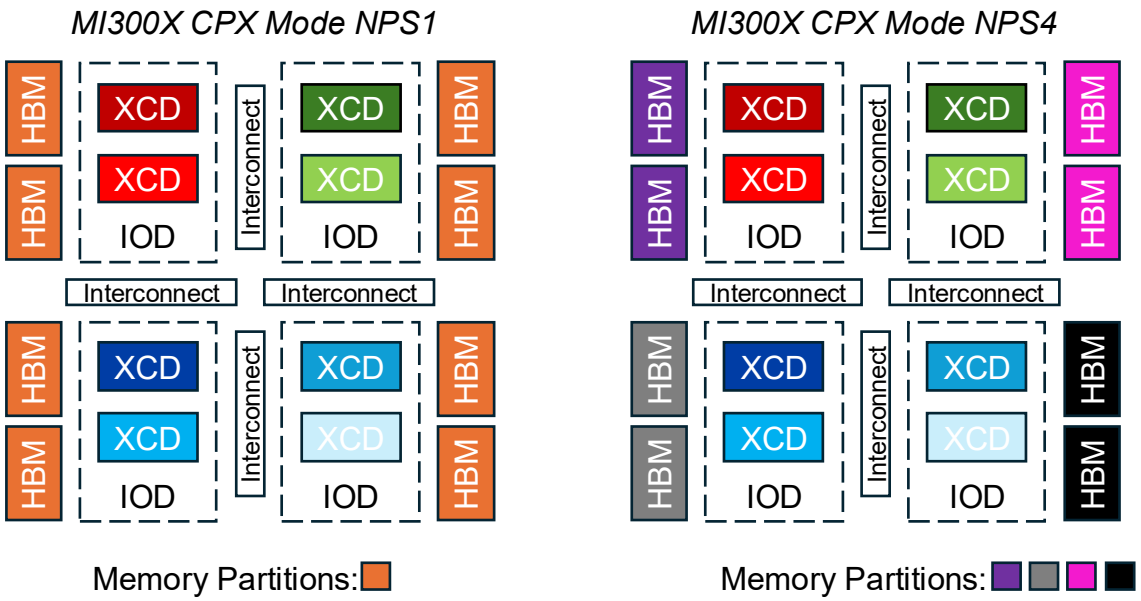


Figure 2.7: NUMA Partitioning (NPS) modes supported on AMD Instinct™ MI300X.

Additionally, AMD CDNA™ 3 supports memory partitioning via NPS modes. Figure 2.7 compares CPX operation under NPS1 and NPS4, illustrating how compute and memory resources interact under different NUMA configurations. In the CPX NPS1 mode, compute resources are divided into eight independent partitions, while memory remains unified and fully interleaved. This configuration provides isolated compute domains while maintaining shared access to a single memory space.

In contrast, the CPX NPS4 mode shows a fully partitioned configuration in which compute resources are still divided into eight partitions, but memory is split into four NUMA nodes. In this mode, pairs of compute partitions share a local HBM (as indicated by the dashed boxes). This improves memory locality by ensuring that each compute partition primarily accesses its associated local memory, thereby reducing cross-die traffic.

2.4 Related Work

Prior research has extensively explored GPU memory hierarchy and microarchitectural behavior through microbenchmarking. Mei and Chu [7] introduced a fine-grained pointer chasing microbenchmark that was the first to expose cache properties of NVIDIA's Kepler and Maxwell architectures, contributing significantly to the understanding of GPU memory behavior. Wong et al. [9] further advanced this area by analyzing instruction-level behavior and cache interactions to demystify GPU microarchitecture. More recently, Abdelkhalik et al. [1] and Luo et al. [6] applied similar techniques to Nvidia's Ampere and Hopper architectures, respectively, highlighting architectural evolution and its impact on performance.

In the context of AMD GPUs, the AMD CDNA™ 2 and AMD CDNA™ 3 architectures have been documented in official white papers [2, 4] and Instruction Set Architecture (ISA) manuals [3, 5], which detail chiplet-based packaging, memory hierarchy, and compute partitioning. The AMD ROCm™ blog [8] provides practical insights into AMD Instinct™ MI300's memory and compute modes, complementing the architectural documentation with usage-level perspectives.

Unlike prior pointer chasing methodologies [7, 1, 9], which rely on read-after-write (RAW) dependencies to serialize memory accesses and implicitly enforce memory completion, our approach employs the `s_waitcnt lgkmcnt(0)` instruction to explicitly synchronize scalar memory (smem) operations and ensure load completion. This method eliminates the need for additional dependency chains between the start and end of the timing interval.

Chapter 3

Methodology

This chapter details the experimental methodology, hardware isolation techniques, and the microbenchmarking approach used to characterize memory hierarchy latency on AMD GPUs. We employ a fine-grained, single-threaded pointer chasing microbenchmark to measure the latency of the GPU memory hierarchy, as shown in Algorithm 1.

Data Preparation and Execution

The experimental setup consists of several data-preparation steps. First, an array of size N is created on the host and populated with 4-byte unsigned indices $[0, N-1]$. This array is then randomly permuted to form a single pointer chasing cycle, ensuring that each access is strictly data-dependent and maximally cache-unfriendly. The permuted array is transferred to device memory as `ptr`, and the kernel `pChase` (see Algorithm 1) is launched with execution parameters `<<<1, 32>>>`. To force the kernel to be single-threaded, an if statement is placed to mask off all threads except thread 0. This configuration guarantees that only a single thread executes the loop. Because the compiler can statically determine

that the kernel is effectively single-threaded (due to the conditional in Line 4), it routes memory operations through the scalar pipeline. Strict serialization of memory accesses is enforced explicitly by `s_waitcnt lgkmcnt(0)`, which guarantees that each scalar load completes before the timestamp is captured. Latency for each access is measured using `s_memtime` (start/stop), and the resulting per-access values are stored in `timerArray`.

We verified the placement and correctness of the timing instructions by inspecting the compiled assembly using `--save-temps`. While low-level timing instructions such as `s_memtime` and `s_waitcnt` add a small, fixed overhead to the absolute latency values, this overhead does not affect the relative comparison to our analysis.

To evaluate the vector memory pipeline, memory operations must be issued as vector loads rather than scalar loads. This can be achieved using inline assembly (e.g., `global_load_dword`) to explicitly force the compiler to emit vector instructions. We demonstrate this technique in Chapter 5 on a AMD Radeon RX™ 7900XT, but the methodology generalizes across AMD architectures. Importantly, scalar and vector loads differ only in their private L1 hit latencies; their miss penalties through the rest of the memory hierarchy are identical. Thus, the scalar-based measurements used in this work still correctly capture the behavior of the subsequent levels of the memory hierarchy.

3.0.1 Pointer Chasing Microbenchmark

Algorithm 1 Pointer Chasing Kernel

```
1: procedure __GLOBAL__ PCHASE(ptr, timerArray, arraySize, runtime_limit)
2:     tid ← threadIdx.x
3:     index ← 0
4:     if tid < 1 then
5:         for i = 0 to arraySize - 1 do
6:             start ← s_memtime
7:             index ← ptr[index]                    ▷ Compiler emitted scalar load
8:             s_waitcnt lgkmcnt(0)
9:             stop ← s_memtime
10:            s_waitcnt lgkmcnt(0)
11:            timerArray[i] ← stop - start
12:        end for
13:    end if
14: end procedure
```

Experimental Parameters and Data Analysis

To characterize the entire memory hierarchy, the buffer size is swept in powers of two, from 2 KB up to 8 GB (e.g., 2 KB $\rightarrow$ 4 KB $\rightarrow$ 8 KB $\rightarrow$ ...). This range captures latency behavior when the working set fits within the on-chip caches (L1 and L2), as well as when it spills into the MALL and HBM.

Latency is measured in GPU clock cycles, and the median of each distribution is used to mitigate the influence of statistical outliers. Unless otherwise noted, all results are normalized to the scalar L1 latency obtained from the baseline configuration. An exception is Figure 4.5, which uses the 64 MB `Isolated` configuration as its baseline.

3.0.2 Hardware Isolation

To ensure performance isolation and reproducible measurements, we enforce strict hardware affinity controls for both the GPU and the host CPU. We use the environment variable `ROCR_VISIBLE_DEVICES` to restrict GPU visibility, and `numactl` with the `--physcpubind` flag to pin the host process to a specific CPU core.

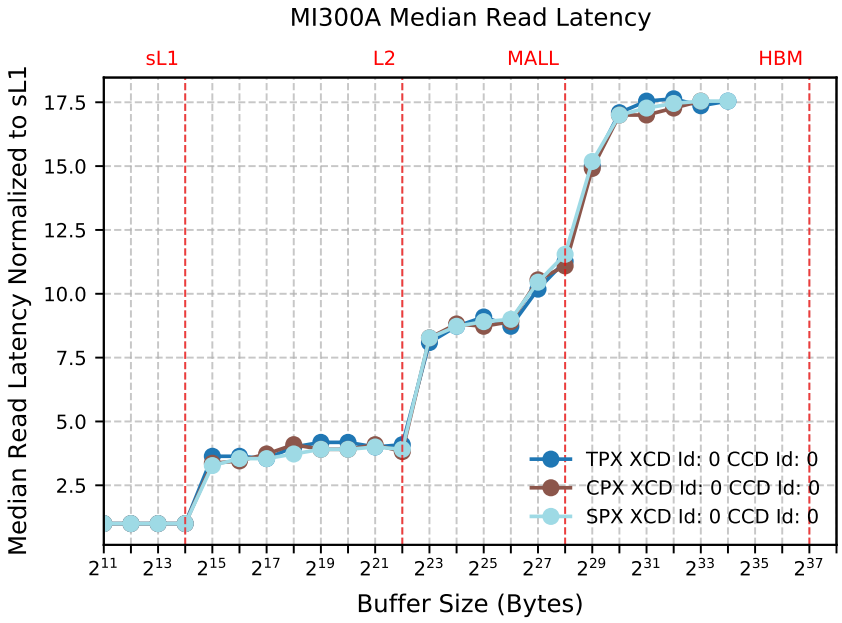
For experiments involving the AMD Instinct™ MI300A (see Figure 4.1a), the host process is bound to Core 1 on CCD0. The GPU is isolated to Device 0. For the discrete AMD Instinct™ MI300X experiments (see Figure 4.1b), the workload runs on the dedicated AMD EPYC™ host CPU, and the process is likewise pinned to Core 1.

Chapter 4

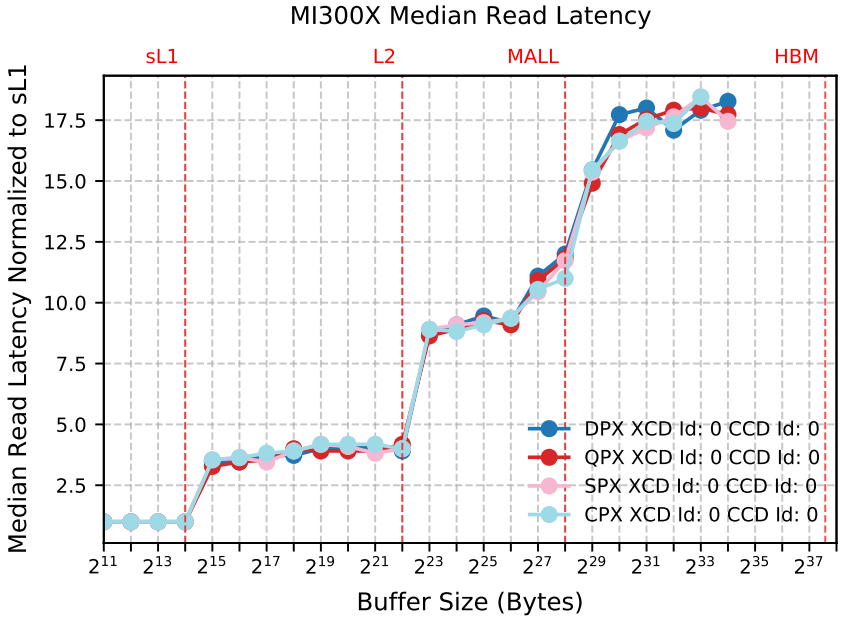
Results

4.1 Memory Hierarchy Median Read Latency

The AMD Instinct™ MI300A and MI300X GPUs share the same AMD CDNA™ 3 architecture, including identical sL1, vL1, L2, MALL, and HBM configurations (see Table 2.1). As a result, both GPUs exhibit nearly identical pointer chasing latency profiles, as shown in Figure 4.1a and Figure 4.1b. These profiles reveal four distinct steps corresponding to the memory hierarchy. The first level (2 KB to 16 KB) corresponds to the scalar L1 cache and serves as the baseline with a normalized latency of 1. The second level (16 KB up to 4 MB) is the L2 cache, with latency increasing to around 3.86x. The third level (4 MB to 256 MB) corresponds to MALL, showing a latency increase of roughly 9.09×. Beyond 256 MB, accesses fall into the HBM region, with a latency increase of approximately 17.5×.



(a) Median pointer chasing latency on AMD Instinct™ MI300A across buffer sizes in CPX (brown), TPX (dark blue), and SPX (light blue). Measured on XCD 0, CCD 0 in NPS1 mode.



(b) Median pointer chasing latency on AMD Instinct™ MI300X across buffer sizes in DPX (dark blue), QPX (red), SPX (pink), and CPX (light blue). Measured on XCD 0, CCD 0 in NPS1 mode.

Figure 4.1: Median pointer chasing read latency for MI300A and MI300X.

These results highlight two key observations. First, comparable latency performance is observed across all compute modes, with no single mode consistently outperforming the others. Second, the MALL begins to exhibit signs of pressure before reaching its nominal capacity (256 MB), as indicated by a noticeable increase in latency at the 2^{27} (128 MB) buffer size. We refer to this behavior as the "early spill", and we analyze its root cause in Section 4.1.1. We attribute this behavior to TLB, but acknowledge that cache evictions and other architectural effects may also contribute.

Given the pointer chasing pattern accesses elements in a randomized order, it frequently touches new pages throughout the traversal. As the working set grows, this increases the likelihood of TLB misses, particularly near and beyond the TLB's coverage threshold, causing the observed latency increase.

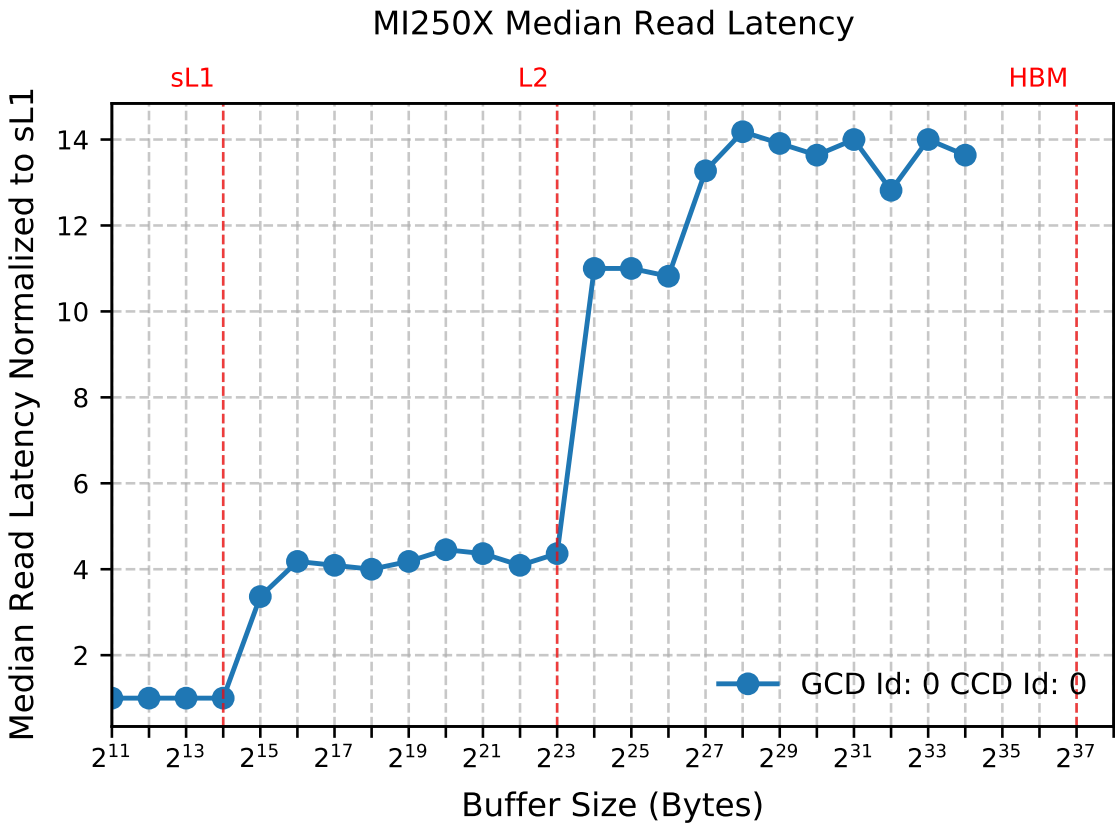


Figure 4.2: Pointer chasing median latency on AMD Instinct™ MI250X across buffer sizes. Single curve representing AMD Instinct™ MI250X taken from GCD 0 CCD 0.

As shown in Figure 4.2, the pointer chasing results for MI250X reveal three levels of memory hierarchy comprising the scalar L1, L2, and HBM memory. Notably, the MI250X lacks a MALL. The scalar L1 cache (baseline = 1), L2 cache (4.09x latency increase compared to sL1), and HBM memory (13.64x latency increase). A slight reduction in latency is observed between 16 MB and 64 MB, despite this range exceeding the typical L2 capacity. We hypothesize that this behavior is due to residual L2 effects, where some accesses are still hitting L2.

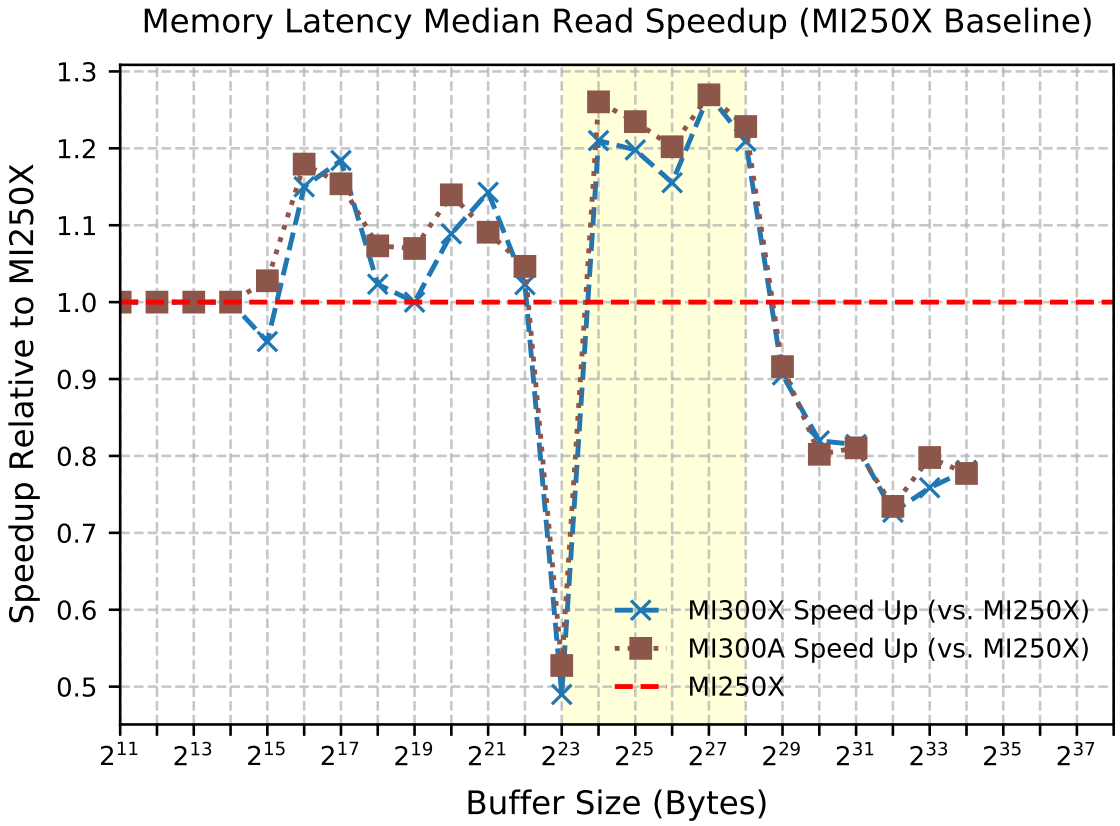


Figure 4.3: Speedup of memory read latency for AMD Instinct™ MI300X and MI300A relative to MI250X across varying buffer sizes. AMD Instinct™ MI250X serves as the 1.0x baseline. AMD Instinct™ MI300A/MI300X data were collected in SPX NPS1 mode. The yellow-shaded region highlights the MALL area.

Memory latency comparison between MI250X and MI300X/MI300A reveals several architectural trade-offs across the cache hierarchy. As shown in Figure 4.3, at the sL1 level (0 to 16 KB), both GPUs exhibit identical latency (speedup of 1.0), indicating equivalent scalar L1 cache performance. Interestingly, a modest performance improvement is observed in the L2 region (greater than 16 KB to 4 MB) for MI300X compared to MI250X. The cause of this improvement is not yet fully understood, but may be attributed to differences in cache associativity, replacement policy, or memory controller optimizations. However,

a notable slowdown occurs at 2^{23} bytes (8 MB). MI250X features an 8 MB L2 cache per Graphics Compute Die (GCD) [2], while MI300X includes 4 MB per XCD (see Table 2.1). As a result, memory accesses that remain within L2 on MI250X begin spilling into the MALL on MI300X, incurring a latency penalty. This leads to a performance degradation of up to 2 times in this region for MI300X relative to MI250X indicating that the MALL is approximately twice as slow as the L2 cache.

The performance speed up observed in 2^{24} to 2^{28} byte range corresponds to the utilization of the MALL on MI300X and MI300A. In this region, memory accesses benefit from MALL's lower latency compared to HBM on MI250X, resulting in up to 1.27x speedup.

However, because MALL introduces a third level in the memory hierarchy, accesses that miss in MALL and fall through to HBM incur additional latency due to the extra lookup stage. Furthermore, in `NPS1` mode, HBM reads are interleaved across all memory channels of all HBM stacks, leading to NUMA effects that further increase access latency. This results in performance degradation, with speedup dropping from approximately 1.2× to between 0.9× and 0.7× beyond 2^{28} bytes (256 MB), where more accesses hit in HBM.

Note that latency is measured in clock cycles, and MI300's compute units operate at a slightly higher frequency than those in MI250X (2.1 GHz vs. 1.7 GHz), which slightly underestimates MI300X's performance in this comparison.

4.1.1 MALL Analysis and Usage Guidelines

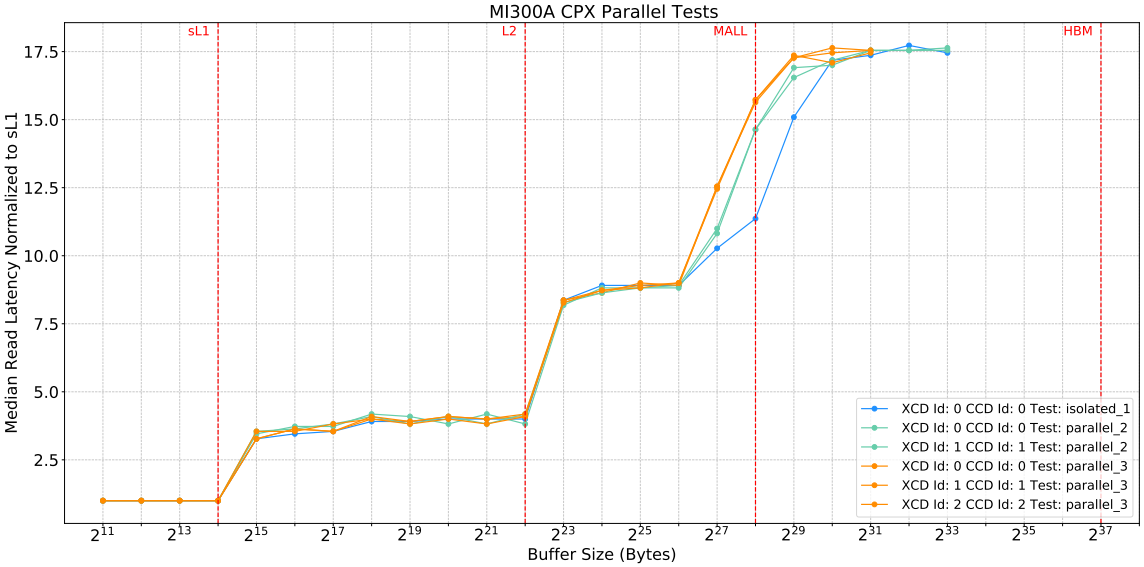


Figure 4.4: Impact of Parallelism on MALL Latency in AMD Instinct™ MI300A CPX mode. Parallelism is scaled from 1–3 processes, and buffer size is measured per process. This highlights the effects of MALL contention and TLB overhead under increasing parallel load.

Figure 4.4 and Figure 4.5 illustrate the impact of MALL and HBM contention, as well as TLB effects, under increasing parallelism on the AMD Instinct™ MI300A. In this experiment, each process is pinned to a distinct XCD and CCD. The degree of parallelism is varied from 1 (denoted as `Isolated`) to 3 concurrent processes. The graph shows median read latency across a wide range of buffer sizes, while the table summarizes key latency values at representative points. In all parallel tests, the buffer size refers to memory allocated **per process**. For example, in the `Parallel 3` configuration, a 256 MB buffer implies a total of 768 MB across three processes.

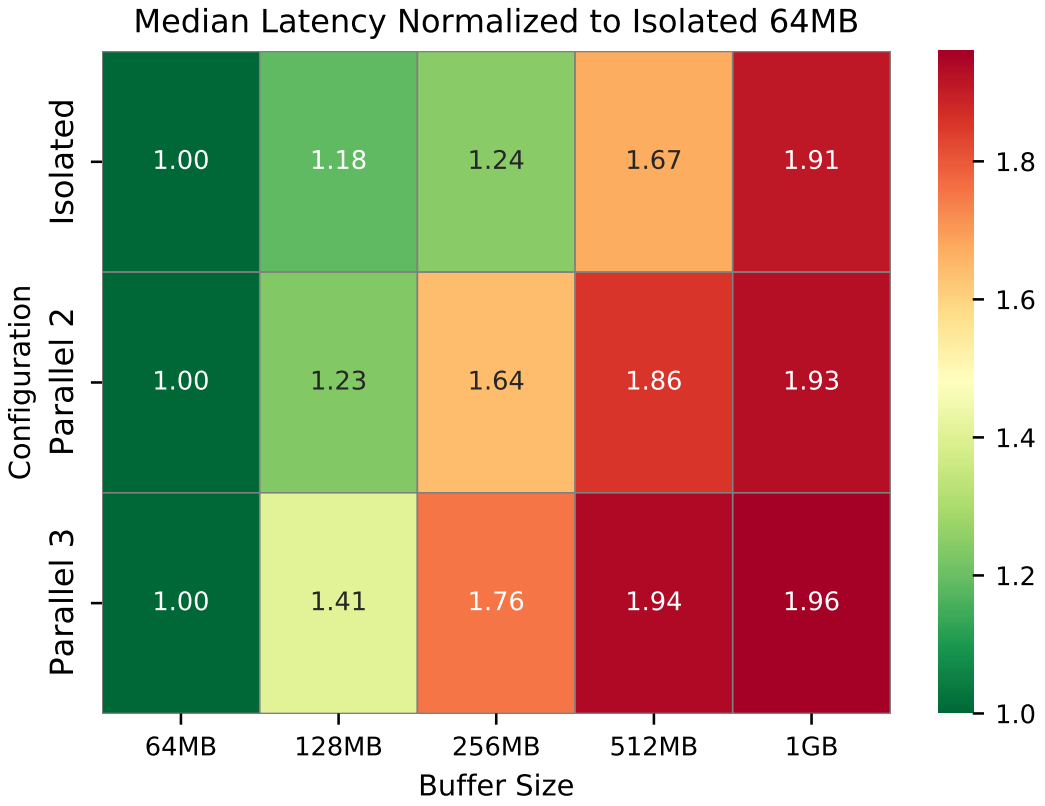


Figure 4.5: Heatmap of Normalized Median Latency across various Parallel Configurations and MALL to HBM Buffer Sizes.

Key Observations

MALL Baseline and Capacity Thresholds. MALL latency at 64 MB `Isolated` (i.e., single-process configuration) is used as the baseline, with a normalized value of 1. At 64 MB per process, the `Isolated`, `Parallel 2`, `Parallel 3` configurations remain within MALL's nominal 256 MB capacity.

TLB Effects. When comparing `Parallel 2` at 64 MB per process (128 MB total) with `Isolated` at 128 MB, we observe an 18% increase in normalized latency (1.18× vs. 1×; see Figure 4.5), suggesting TLB overhead despite both configurations fitting within

MALL. This result suggest that distributing memory across multiple smaller processes per XCD can be more efficient than allocating a large buffer to a single process, highlighting the benefit of parallelism in mitigating TLB overhead.

This behavior is only possible under compute partitioning (e.g, CPX mode), where the user has fine grained control over which XCD a process is assigned to. In CPX mode, each XCD is exposed as an independent device, allowing a workload to be decomposed into multiple smaller processes, each restricted to a single XCD. This reduces the memory footprint per process and per workgroup, thereby lowering TLB pressure since each XCD sees a smaller set of virtual pages.

In contrast, under SPX mode, workgroups are round-robined across all XCDs. A single large process therefore exposes every XCD to the full memory footprint of its workgroups, which can increase TLB miss rates relative to the CPX multi-process configuration. Thus, distributing work across multiple CPX partitioned processes may provide better TLB behavior than running one large SPX scheduled process.

At 128 MB per process, `Parallel 2` shows a latency increase of 1.23x, only slightly below the 1.24 increase latency of `Isolated` at 256 MB, indicating that both configurations are affected by TLB misses. This aligns with the earlier observation that TLB effects begin to appear at 128 MB, even for a single process.

We speculate that these effects may be due to TLB limitations. For example, when comparing `Isolated` at 64 MB to `Parallel 3` at 64 MB per process (192 MB total), we observe no increase in latency. In contrast, an `Isolated` configuration at 128 MB shows an 18% increase compared to `Isolated` at 64 MB. Despite the presence of three parallel

processes, potential cache and channel contention in `Parallel 3`, the latency remains unchanged. This suggests that TLB overhead may be more significant at buffer sizes of 128 MB and may have a greater impact on latency than the presence of parallel processes. While we acknowledge that channel effects could contribute to the observed behavior when comparing parallel and isolated configurations, each kernel is single-threaded and executes a single wavefront, making significant channel contention unlikely. Nonetheless, we recognize that such effects may still play a role.

Transition to HBM. At 512 MB and 1 GB, latency rises sharply across all configurations, indicating a transition from MALL to HBM. The total memory footprint quickly exceeds MALL capacity (e.g., 1 GB for `Parallel 2`, 1.5 GB for `Parallel 3`), causing all accesses to fall back to HBM. Configuration differences diminish, suggesting TLB effects are masked by the dominant HBM latency.

These thresholds and their implications are summarized in Table 4.1, which outlines recommended buffer sizes to avoid contention and TLB overhead. Note that these TLB thresholds reflect a worst case access pattern because pointer chasing maximizes TLB pressure. For real workloads with more spatial locality, the effective thresholds for TLB effects may be higher.

Table 4.1: Summary of MALL Usage Recommendations

Buffer Size per Process	**Recommendation**
≤ 64 MB	Safe zone — minimal TLB and MALL contention
128 MB	TLB overhead likely — monitor for early contention
256 MB	MALL capacity saturation — high contention risk
≥ 512 MB	HBM-dominant — MALL benefit negligible

4.2 Effect of NPS4 on MALL and HBM Latency

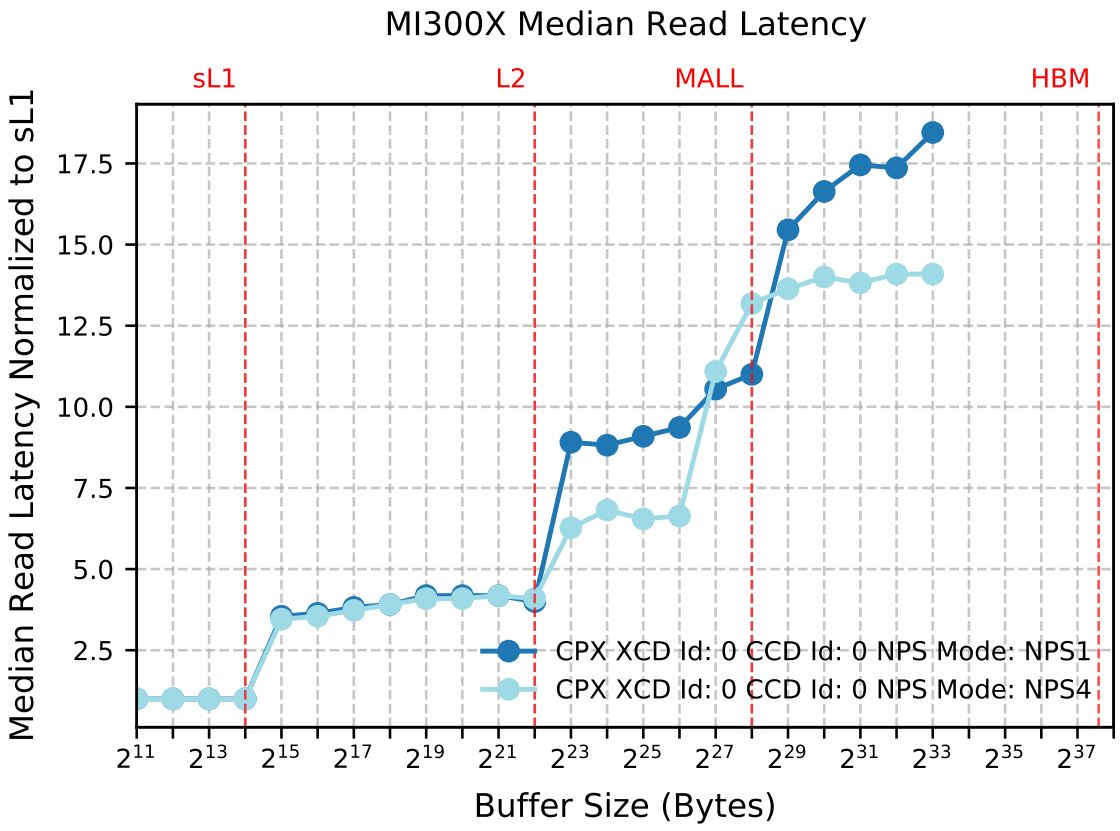


Figure 4.6: Pointer chase latency on AMD Instinct™ MI300X: CPX NPS1 results shown in dark blue, CPX NPS4 results shown in light blue.

Since the MALL sits on the IOD and below each XCD (see Figure 2.4), we questioned whether memory partitioning would also partition the MALL. To investigate this, we performed pointer chasing benchmarks in CPX compute mode under both NPS1 and NPS4 configurations on the AMD Instinct™ MI300X (see Figure 4.6).

With NPS4 enabled, cache latencies in the private regions of each XCD (sL1 and L2) remain consistent with NPS1. This is expected, as these caches are local to their respective XCDs and do not traverse the AMD Infinity Fabric™.

The most significant latency improvements occur in the shared memory regions: MALL and HBM. A key finding of this work is that NPS4 partitions not only HBM, but also the MALL. As shown in Figure 4.6, the light-blue curve (NPS4) exhibits an earlier transition at 64 MB, indicating that the 256 MB MALL is divided into four 64 MB slices one per XCD quadrant. This partitioning reduces cross-die traffic and improves memory locality. Under NPS4, MALL latency decreases by up to 1.42× at an 8 MB buffer size, while HBM latency decreases by 1.31× at 8 GB.

However, this same partitioning introduces trade-offs at larger buffer sizes. At 128 MB and 256 MB, NPS4 performance degrades because each XCD can access only 64 MB of MALL (256 MB ÷ 4), whereas under NPS1 the full 256 MB is available to any XCD. As a result, memory accesses that would remain within MALL under NPS1 spill into HBM under NPS4, producing a slowdown (speedup of only 0.83×). These results provide clear evidence that the MALL is partitioned under NPS modes and that this partitioning directly shapes the effective latency profile.

In NPS4 mode, MALL accesses (ranging from 2^{23} or 8 MB to 2^{26} or 64 MB) show an average latency decrease of 1.38× compared to NPS1. Similarly, HBM accesses (from 2^{29} or 512 MB to 2^{33} or 8 GB) exhibit an average latency decrease of 1.22×. These improvements result from eliminating interconnect traversal, as each XCD accesses only its local memory region.

Chapter 5

Methodological Validation: Vector Memory Hierarchy on AMD Radeon RX™ 7900XT

This chapter presents a methodological validation of the pointer chasing microbenchmark under vector memory operations. The goal of this analysis is twofold: (1) to demonstrate how the benchmark can be extended to explicitly exercise the vector memory hierarchy, and (2) to confirm the generalizability of the scalar-based AMD Instinct™ MI300A and MI300X latency results presented in Chapter 4. As shown in this chapter, the scalar and vector pipelines show identical latency trends through the L2 cache, the MALL, and HBM, differing only by a small constant offset due to the difference in their scalar and vector cache hit latencies.

Algorithm 2 Pointer Chasing Kernel with Explicit Scalar and Vector Inline Assembly

```
1: procedure __GLOBAL__ PCHASE(ptr, timerArray, arraySize)
2:     tid ← threadIdx.x
3:     if tid < 1 then
4:         index ← 0
5:         for i = 0 to arraySize - 1 do
6:             addr ← &ptr[index]
7:             if SCACHE then                                  ▷ Scalar pipeline
8:                 asm_block:
9:                     start ← READ_TIMER()
10:                    index ← SCALAR_LOAD(addr)
11:                    WAIT_FOR_SCALAR()                       ▷ s_waitcnt lgkmcnt(0)
12:                    stop ← READ_TIMER()
13:            else                                            ▷ Vector pipeline
14:                asm_block:
15:                    start ← READ_TIMER()
16:                    index ← VECTOR_LOAD(addr)
17:                    WAIT_FOR_VECTOR()                       ▷ s_waitcnt vmcnt(0)
18:                    stop ← READ_TIMER()
19:            end if
20:            timerArray[i] ← stop - start
21:        end for
22:    end if
23: end procedure
```

Implementation Note. The symbolic operations `SCALAR_LOAD` and `VECTOR_LOAD` map to different inline assembly instructions depending on the architecture. On MI300s scalar and vector loads use `s_load_dword` and `global_load_dword`, while on the Radeon the corresponding instructions are `s_load_b32` and `global_load_b32`. Timer instructions and wait counters also differ, but these naming differences do not affect the methodology: the inline assembly block always provides explicit control over the timing instruction, the executed load, and the necessary wait counter.

The flag `SCACHE` is a compile time macro used with `if constexpr` to select either the scalar or vector inline assembly path. Unlike the MI300 experiments in Chapter 3 where the load instruction was not written in inline assembly and was therefore scalarized by the compiler the experiments in this chapter use fully explicit inline assembly for the timer, load instruction, and wait counter. This ensures that the pointer chasing kernel executes exactly the intended scalar or vector memory operation.

5.1 Scalar vs. Vector Pointer Chase on Radeon

On AMD Radeon RX™ 7900XT, the private scalar and vector caches appear at the L0 level referred to as the L0 Scalar Cache (sL0) and L0 Vector Cache (vL0) followed by the L1 cache. This differs from AMD Instinct™ MI300A/X, where the private caches are instead designated as L1 (sL1/vL1). We adopt the Radeon naming convention (sL0/vL0) throughout this section for clarity.

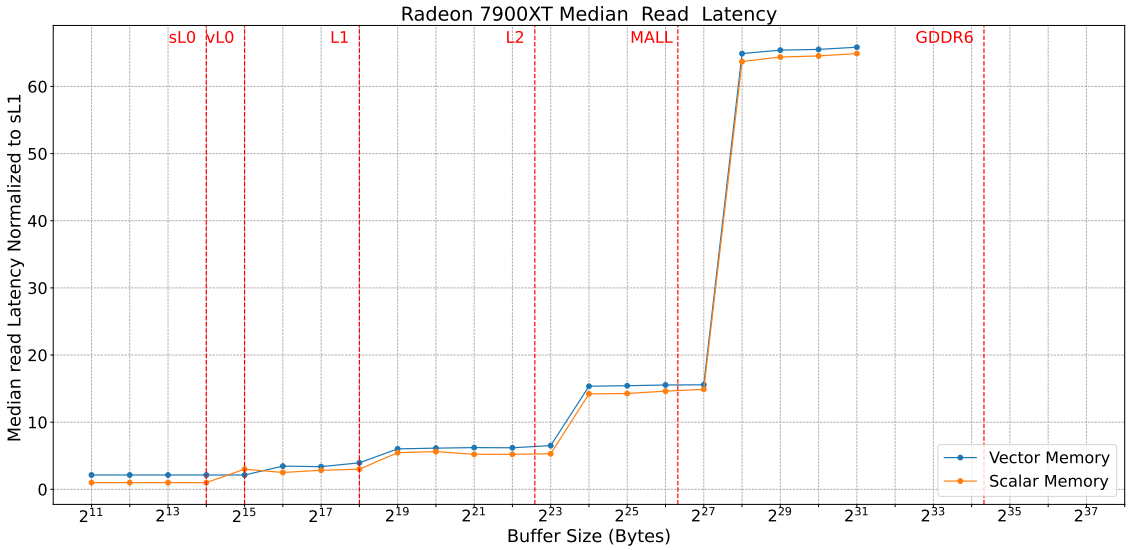


Figure 5.1: Pointer chasing median read latency on the AMD Radeon RX™ 7900XT, normalized to the sL0 latency. The figure compares the scalar memory pipeline (orange) and the vector memory pipeline (blue) across increasing buffer sizes.

Figure 5.1 shows the pointer chasing latency curves for scalar and vector memory operations on the AMD Radeon RX™ 7900XT. The orange curve corresponds to the scalar pipeline and exhibits an sL0 hit region from 0 to 16 KB, after which the latency increases as the working set spills into the L1 cache. The blue curve corresponds to the vector pipeline and shows a vL0 hit region from 0 to 32 KB before likewise transitioning into L1. Beyond these private L0 regions, both curves follow the same sequence of latency transitions, including L1 → L2, L2 → MALL, and MALL → GDDR6, demonstrating that the two pipelines share an identical downstream memory hierarchy.

At small buffer sizes, the vector pipeline incurs slightly higher latency due to the inherent difference between vL0 and sL0 hit times. This appears as a small constant vertical offset between the two normalized curves. This offset does not grow with larger working sets: the curves remain nearly parallel across all cache and memory regions, indicating that the difference between scalar and vector accesses is confined entirely to the L0 level and does not affect any subsequent stages of the memory hierarchy. Following the standard Average Memory Access Time (AMAT) formulation,

$$\text{AMAT} = \text{Hit Time} + \text{Miss Penalty},$$

the difference between the scalar and vector curves is determined entirely by the difference in their L0 hit times. Once the buffer size exceeds the L0 capacity, both pipelines incur the same miss penalties at each subsequent cache and memory level, resulting in two latency curves that differ by a small constant offset while remaining parallel across all memory regions.

Because the scalar and vector curves share identical inflection points, slopes, and latency growth through L1, L2, the MALL, and GDDR6, this experiment confirms that the scalar-based AMD Instinct™ MI300A/X results presented in Chapter 4 accurately reflect the behavior of the memory hierarchy beyond the scalar cache. Therefore, the use of scalar loads introduced by the compiler does not alter or bias the pointer chasing latency trends.

Chapter 6

Conclusions

This thesis presented a comprehensive characterization of the memory hierarchy of the AMD Instinct™ MI300A and MI300X GPUs using a fine-grained pointer chasing microbenchmark. By comparing results across MI300A, MI300X, and MI250X, this work provides new insights into the behavior of AMD's latest GPU memory systems and highlights architectural features that influence latency, locality, and overall performance.

The first key contribution of this work is a detailed characterization of the MI300A/X memory hierarchy, including the role of the MALL as a third-level cache and its impact on latency across a wide range of working set sizes. Our results show that the MALL substantially reduces effective memory latency compared to HBM, particularly for working sets up to 256 MB per process. However, at 128 MB we observe latency increases of up to 18%, which we attribute to TLB pressure. As shown in Chapter 4, this early MALL spill occurs whenever a single process or XCD reaches a 128 MB working set footprint, regardless of the total memory footprint across all processes.

Crucially, this bottleneck can be effectively mitigated under CPX mode, where each XCD is exposed as an independent device. This allows the programmer to decompose the workload into multiple smaller processes, each pinned to a specific XCD. This decomposition reduces the per-process working set footprint, lowering TLB pressure since each XCD manages a smaller set of virtual pages. In contrast, under SPX mode, workgroups are distributed round robin across all XCDs. A single large process can expose every XCD to the full memory footprint, thereby increasing TLB pressure. Beyond 256 MB, memory latency converges toward HBM performance, indicating that the benefit of MALL diminishes once its capacity is exceeded. We further find that compute partitioning itself has no measurable impact on memory latency, simplifying tuning decisions for latency-sensitive workloads.

A second key contribution of this work is the discovery that NPS4 partitions not only the HBM stacks but also the 256 MB MALL, dividing it into four 64 MB slices. This partitioning significantly improves memory locality: MALL latency decreases by up to 1.42×, and HBM latency by up to 1.31× for pointer chasing workloads. However, this configuration introduces trade-offs at larger buffer sizes. Once a process exceeds its local 64 MB MALL region in NPS4, memory accesses spill into HBM earlier than in NPS1, reducing performance relative to the unified configuration.

The third contribution of this thesis is the methodological validation that pointer chasing is robust across scalar and vector memory pipelines. Using fully explicit inline assembly on a AMD Radeon RX™ 7900XT, we showed that scalar and vector memory accesses differ only in their private L0 hit latencies while sharing an identical miss path through L1, L2, the last-level cache, and DRAM. As a result, the scalar and vector latency

curves remain parallel across all working set sizes. This confirms that the scalar-based MI300A/X measurements produced by the compiler in our single-threaded kernel accurately capture the behavior of the broader memory hierarchy.

These findings provide practical guidance for optimizing memory bound workloads on AMD CDNA™ 3 GPUs: (i) MALL and HBM latency depend strongly on NPS mode, (ii) TLB effects emerge at working set sizes around 128 MB per process or per XCD, and (iii) and reducing the working set footprint per XCD lowers TLB pressure. This can be achieved by using compute partitioning to bind processes to separate XCDs.

Limitations

Pointer chasing represents a worst-case access pattern with minimal spatial locality. Real applications may exhibit delayed TLB onset or different MALL/HBM utilization characteristics due to prefetching or higher locality. Additionally, this work focuses on single-threaded median latency. Multithreaded contention and bandwidth-saturated behaviors remain unexplored. Finally, the conclusions regarding TLB pressure are inferred from latency trends rather than direct hardware TLB measurements.

Future Work

Future work includes using hardware TLB performance counters to directly measure the 128 MB MALL spill, evaluating how page size (4 KB vs. 2 MB hugepages) affects TLB behavior, and examining energy and power characteristics across NPS and Compute partitioning modes. Additional directions include extending pointer chasing to multi-wavefront scenarios, exploring pointer chasing with configurable stride to study spatial locality effects, and characterizing bandwidth saturation and contention under different NPS and Compute partitioning configurations.

Bibliography

[1] Hamdy Abdelkhalik, Yehia Arafa, Nandakishore Santhi, and Abdel-Hameed Badawy. Demystifying the nvidia ampere architecture through microbenchmarking and instruction-level analysis. *arXiv preprint arXiv:2208.11174*, 2022.

[2] AMD. AMD CDNA 2 Architecture White Paper, 2022.

[3] AMD. AMD Instinct MI200 Instruction Set Architecture, 2022.

[4] AMD. AMD CDNA 3 Architecture White Paper, 2024.

[5] AMD. AMD Instinct MI300 Instruction Set Architecture, 2025.

[6] Weile Luo, Ruibo Fan, Zeyu Li, Dayou Du, Hongyuan Liu, Qiang Wang, and Xiaowen Chu. Dissecting the nvidia hopper architecture through microbenchmarking and multiple level analysis. *arXiv preprint arXiv:2501.12084*, 2025.

[7] Xinxin Mei and Xiaowen Chu. Dissecting gpu memory hierarchy through microbenchmarking. *arXiv preprint arXiv:1509.02308*, 2015.

[8] Muhammad Osama, Ryan Swann, Karthik Sangaiah, Sonali Singh, Ganesh Dasika, Rajneesh Bhardwaj, and AMD. Deep dive into the MI300 compute and memory partition modes, 2025.

[9] Henry Wong, Misel-Myrto Papadopoulou, Maryam Sadooghi-Alvandi, and Andreas Moshovos. Demystifying gpu microarchitecture through microbenchmarking. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2010.