Predicting Modern GPU Architecture Execution Time with Microbenchmarks and Analytical Models

Preliminary Technical Report

Aaron Jarmusch
Department of Computer & Information Sciences
University of Delaware
email: jarmusch@udel.edu

Advisor:
Dr. Sunita Chandrasekaran
Dept. CIS/University of Delaware

Committee Member:
Dr. Dong Dai
Dept. CIS/University of Delaware

November 5^{th}, 2025

Abstract

Modern GPU architectures exhibit unprecedented complexity, combining hierarchical memory systems, specialized tensor compute engines, and tightly integrated CPU–GPU interconnects. Understanding these architectural behaviors is essential for optimizing performance in high-performance computing (HPC) and artificial intelligence (AI) workloads. This report presents a comprehensive characterization and analytical performance modeling study of NVIDIA's **Blackwell** and AMD's **CNDA 3** architectures. We design a suite of low-level microbenchmarks to measure key aspects of GPU performance, including memory subsystem throughput, tensor compute units utilization, and execution pipeline efficiency. Building on the microbenchmark results, we construct an analytical model to predict the execution time of applications using the distinct characteristics of the NVIDIA Blackwell B200 GPU and the AMD MI300A APU. In conclusion, this preliminary work offers actionable guidance of performance tuning for application developers and architectural co-design across heterogeneous platforms.

Keywords: GPU Architecture, Microbenchmark, Analytical Model, Performance Analysis, Tensor Core, Matrix Core

1 Introduction

GPUs, once designed primarily for graphics rendering [1], now serve as the computational backbone for applications ranging from climate modeling [2] to large language models (LLM(s)) [3]. This popularity sparked escalation in architectural innovation to provide the best performance across multiple domains. For example, modern GPUs incorporate multi-level memory hierarchies with high-bandwidth memory, specialized tensor processing units, and a hardware decompression engine, just to name a few. Each

new architectural generation creates a widening gap of theoretical peak performance and achievable application efficiency with added layers of complexity. Offering a trade-off between architectural complexity and the best performance for specialized workloads.

Unfortunately, as GPUs become more intricate the usability to optimize application performance diminishes. Developers contend with scheduling policies and instruction sets that change between generations, which is compounded by limited public documentation. Consequently, various characterization approaches, from microbenchmarking individual subsystems to constructing analytical models, aim to understand the complexity of the microarchitectures. This work focuses on capturing the performance characteristics unique to modern GPU architectures to improve application performance and provide architectural insights that improve accuracy of analytical models.

This preliminary research addresses this gap through systematic performance characterization of modern accelerator architectures, specifically NVIDIA Blackwell (B200) GPU [4] and AMD MI300A APU [5] accelerators. We develop a mathematical based analytical model that captures both traditional performance factors (memory bandwidth, compute throughput) and specialized architectural features (TMEM double-buffering, decompression pipelines, unified execution units) to predict the execution time of applications. Based on results from our custom low-level microbenchmark suite, we designed mathematical equations to provide actionable guidance for both application developers and hardware designers. As GPU architectures continue evolving toward specialized accelerators for AI, quantum simulation, and other emerging domains, this preliminary work establishes a foundation that becomes increasingly critical for achieving computational efficiency at scale.

2 Related Work

Understanding GPU performance has evolved from simple compute-bound models to sophisticated frameworks capturing architectural complexity. The roofline model [6] pioneered performance visualization, with GPU-specific adaptations [7] broadening its reach. However, these frameworks fundamentally assume bottlenecks arise from either SM throughput or memory bandwidth, overlooking microarchitectural subtleties. Profiler tools such as NVIDIA Nsight and AMD ROCm Profiler [8, 9] provide empirical runtime diagnosis but lack predictive power and architectural generality. Analytical models, by contrast, aim to explain and predict performance based on hardware specifications, offering interpretability and cross-platform insights. Though, they require validated microarchitectural knowledge and continuous updates as hardware evolves.

Early analytical models [10] partitioned execution into compute-bound and memory-bound phases, predicting runtime as their maximum plus overhead. Subsequent refinements added detailed core modeling [11], memory contention schemes [12], and frameworks like MDM [13] and GPUMech [14], which integrate microbenchmark-driven characterization of cache state, interconnect contention, and CTA scheduling. Specialized accelerators introduced new modeling challenges: Chowdhury et al [15] formalized tensor core throughput and data movement constraints, while early characterization work [16,17] revealed underutilization for small matrices. Machine learning approaches [18,19] achieve high accuracy but sacrifice the interpretability that makes analytical models valuable for optimization guidance. Despite these advances, opacity and scheduling necessitate ongoing reverse-engineering through instruction-level timing analysis [20], creating a perpetual gap between hardware capabilities and modeling frameworks.

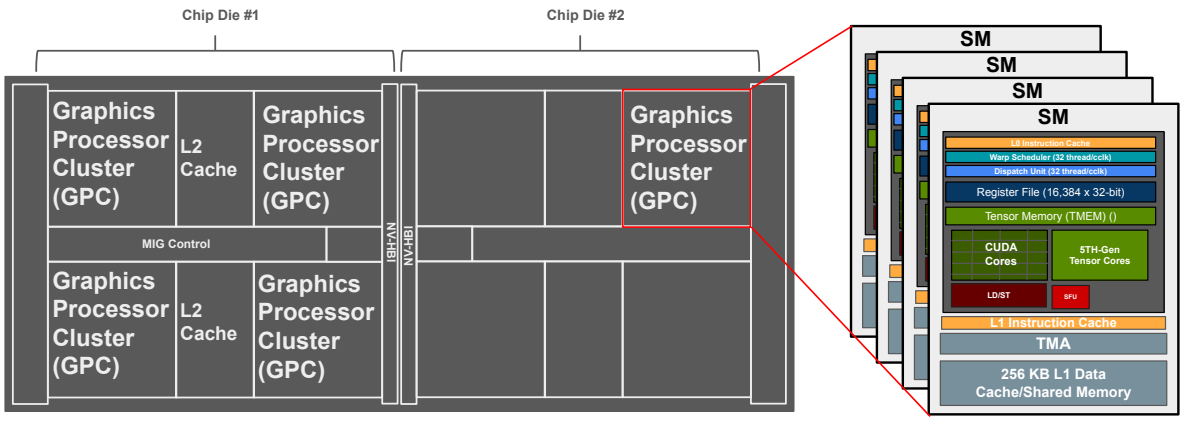

Figure 1: Block diagram of the NVIDIA Blackwell data center GPU architecture, highlighting the dual-chip multi-die design and the SM structure.

Cross-vendor insights remain limited despite recent progress. Recent studies systematically profile AMD CDNA 3 unified memory, matrix core performance, and power tradeoffs, though lack modeling for the full AMD CDNA 3 architecture [21, 22]. Microbenchmarking methodologies [23, 24] established cache and bandwidth analysis techniques, while recent work on scheduling complexity [25, 26] shows that resource-aware schedulers and thread block clusters introduce multi-dimensional performance tradeoffs that existing models fail to capture.

Gaps in Current Approaches. Most microbenchmark suites predate recent innovations: tensor cores lack characterization for newer formats (FP4, FP6, FP8), novel features like tensor memory remain unstudied, and expanded cache hierarchies lack systematic performance analysis. AMD's CDNA 3 architecture focuses on Unified Physical Memory. Existing analytical models miss new architectural additions, and profilers provide limited cross-platform predictive capability. Our work addresses these gaps through a comprehensive characterization of modern accelerators and validated analytical models applicable across NVIDIA (B200) and AMD (MI300A) architectures.

3 GPU Architecture Overview

Modern GPUs function as massively parallel throughput-oriented processors, achieving high computational density through thousands of concurrent threads executing across streaming multiprocessors (SMs) in NVIDIA terminology or compute units (CUs) in AMD nomenclature. Each SM/CU contains general-purpose arithmetic pipelines, specialized matrix-processing units (Tensor or Matrix Cores), and a multi-level memory hierarchy spanning registers, shared memory, L1/L2 caches, and global high-bandwidth memory (HBM). We focus on NVIDIA's Blackwell (B200) and AMD's CDNA 3 (MI300A) architectures, representing state-of-the-art designs in heterogeneous integration, precision scaling, and matrix-multiplication acceleration.

3.1 NVIDIA Blackwell (B200)

As illustrated in Figure 1, the Blackwell B200 GPU [4] adopts a dual-die architecture with 208 billion transistors connected via NVIDIA's High-Bandwidth Interface (NV-HBI) offering 10 TB/s inter-die bandwidth. Despite physical disaggregation, the two dies present

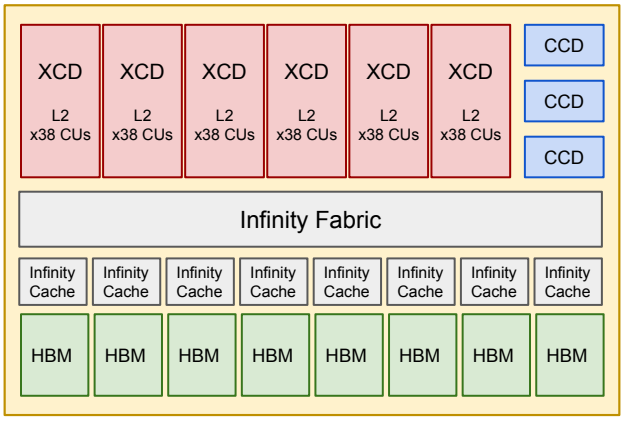

Figure 2: An overview of the MI300A APU architecture.

a unified 192 GB HBM3e memory space and cache-coherent programming model. **Fifth-generation Tensor Cores** extend support to FP4 and FP6 precisions, achieving up to 9,000 TFLOPS of FP4 throughput. The Transformer Engine introduces improved low-precision stability and dynamic loss-scaling mechanisms, while the **decompression engine** enables real-time model weight decompression within the memory pipeline. **Tensor Memory (TMEM)** provides a dedicated 256 KB on-chip buffer per SM for tensor operations, reducing shared-memory contention and improving matrix-core utilization.

Blackwell inherits Hopper's asynchronous pipeline architecture: the Tensor Memory Accelerator (TMA) performs hardware-managed data transfers from global to shared memory. Though, the new tcgen05.mma instruction integrates TMEM operations and weight-stationary dataflows to maximize on-chip reuse and minimize memory traffic.

3.2 AMD MI300A

The MI300A [5] represents the first heterogeneous APU combining GPU and CPU compute on a unified package. Shown in Figure 2, its design integrates six GPU chiplets (GCDs) and three CPU chiplets (Zen 4) in a 3D-stacked configuration, offering 24 high-performance CPU cores with direct, cache-coherent access to GPU memory. Unlike NVIDIA's unified virtual memory, MI300A implements **Unified Physical Memory (UPM)**, enabling true hardware-coherent memory sharing between CPU and GPU without explicit data copies. The enhanced **Matrix Cores** support FP8 arithmetic, delivering up to 1,307 TFLOPS (exceeding MI250X's FP16 throughput by 3×), while maintaining strong FP64 performance at 61.3 TFLOPS. A massive 256 MB L2 cache that reduces global memory access latency for workloads with large working sets.

AMD's CDNA architecture operates at the wavefront level (64 threads) rather than NVIDIA's 32-thread warps. Each MFMA (Matrix Fused Multiply-Add) instruction performs D=A×B+C with operands distributed across vector registers. The CDNA3 architecture features 304 compute units (38 CUs per XCD × 8 XCDs) with native TF32 and 2:4 structured sparsity support. MI300A's multi-XCD topology creates NUMA-like behavior where cross-XCD memory access incurs 50-100 ns penalties, requiring affinity-aware data placement for optimal performance.

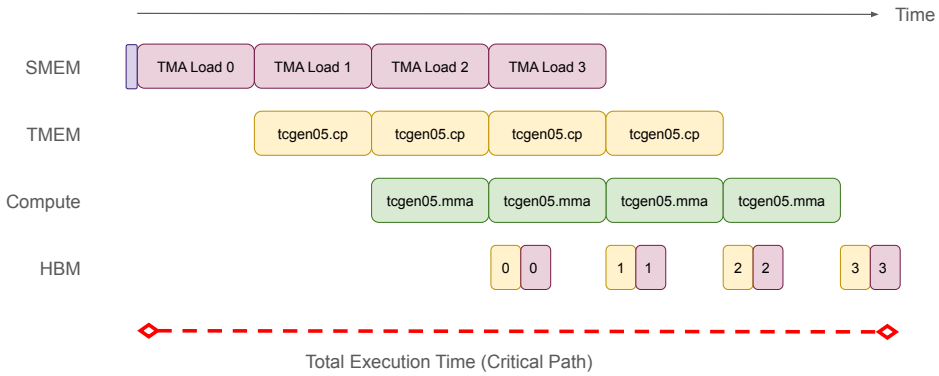

Figure 3: Overview of the per-CTA execution model showing major pipeline stages, overlap regions, and the critical path determining total execution time.

3.3 Architectural Implications for Modeling

These architectural distinctions directly influence performance modeling. Blackwell's explicit pipeline stages (TMA → TMEM → Tensor Core → Synchronization) enable stage-centric modeling with measurable hardware unit latencies and bandwidths. MI300A's wavefront-centric model relies on implicit overlap through occupancy and cache behavior, requiring occupancy-dependent latency hiding factors. Blackwell stores accumulators in dedicated TMEM (256 KB/SM) with explicit read/write operations, while MI300A uses general-purpose VGPRs, creating trade-offs between tile size and occupancy. These design choices enable decomposing GPU performance into measurable parameters—instruction throughput, data movement, and tensor utilization—laying the foundation for our analytical framework Section 4.

4 Analytical Model

We develop analytical frameworks for NVIDIA Blackwell and AMD MI300A, grounded in microbenchmarking. Blackwell employs explicit stage-centric pipelines with specialized hardware (TMA, TMEM, tensor cores), while MI300A uses implicit wavefront-centric scheduling. We adopt the Hong-Kim framework [10], estimating execution time as Equation 1, recognizing modern GPUs overlap compute and memory through deep pipelining.

$$T_{\text{exec}} = \max(T_{\text{compute}}, T_{\text{memory}}) + T_{\text{overhead}} \tag{1}$$

The term $T_{overhead}$, accounts for serialization points such as synchronization barriers and kernel launch latency.

4.1 NVIDIA Blackwell Analytical Model

Blackwell processes each output tile accumulates partial results across $K_{tiles} = [K/K_t]$ iterations involving: (1) TMA load, (2) TMEM management, (3) tcgen05.mma compute, (4) synchronization, and (5) writeback to HBM. Figure 3 shows this pipeline emphasizing explicit overlap regions between memory, TMEM, and compute stages.

4.1.1 Tensor Core Compute with TMEM

Blackwell's Tensor Memory (TMEM) provides dedicated 256 KB per SM exclusively for tensor operations, decoupling accumulators from the register file. Unlike prior architectures where accumulators resided in registers, TMEM requires explicit read/write operations with measurable bandwidth. For a single K-step tile iteration, the time for TMEM access is given by Equation 2, where D_{accum} is the accumulator size in bytes, $BW_{\text{TMEM\_read}}$ and $BW_{\text{TMEM\_write}}$ are measured TMEM bandwidths, L_{mma} is the `tcgen05.mma` instruction execution latency.

$$T_{\text{TMEM\_per\_tile}} = \frac{D_{\text{accum}}}{BW_{\text{TMEM\_read}}} + L_{\text{mma}} + \frac{D_{\text{accum}}}{BW_{\text{TMEM\_write}}} \tag{2}$$

Because TMEM imposes a hard 256 KB capacity limit, exceeding this threshold results in register or SMEM spillage, severely impacting pipeline efficiency. The per-CTA compute time is described by Equation 3.

$$T_{\text{compute}} = \frac{2b_M b_N b_K}{R_{TC}^{SM} \times S_{\text{mode}}} + T_{\text{TMEM}} + T_{\text{TMEM\_mgmt}} \tag{3}$$

In Equation 3, R_{TC}^{SM} represents the Tensor Core throughput per SM, and S_{mode} accounts for 2-SM cooperative execution.

TMEM usage strategies. Figure 4 illustrates two approaches for utilizing TMEM in the execution pipeline. In configuration (A), TMEM stores only accumulators while input tiles (A/B) reside in SMEM. TMA loads inputs to SMEM, tensor cores source operands from SMEM during computation, and accumulators are read from and written back to TMEM each iteration. This approach minimizes TMEM bandwidth requirements since only accumulator traffic (typically smaller than input tiles) flows through TMEM, and TMEM operations can be pipelined with computation.

In configuration (B), TMEM stores both input tile A and accumulators, using the `tcgen08.cp` instruction to copy data from SMEM to TMEM. This reduces SMEM pressure and may enable larger tile sizes when SMEM capacity is constrained, but increases TMEM bandwidth demand and adds staging latency before computation begins. The choice between configurations depends on the relative bottlenecks: configuration (A) is preferred when SMEM capacity is sufficient and TMEM bandwidth is limited, while configuration (B) is advantageous when SMEM becomes the capacity bottleneck. Our model accounts for both configurations through the T_{TMEM} term in Equation 3, with measured TMEM bandwidth parameters determining which configuration yields better performance for a given tile size.

4.1.2 Memory Operations

TMA enables hardware-managed asynchronous transfers with multicast support. Each participating CTA loads a portion of the shared tile, with P participants collectively loading complete data (Equation 4).

$$bytes_{perCTA} = \frac{bytes(T)}{P} \tag{4}$$

The corresponding TMA transfer time per CTA is shown in Equation 5, for tile size $bytes(T)$ multicast to P participants.

$$T_{tma} = L_{TMA} + \frac{bytes_{perCTA}}{B_{TMA}} = L_{TMA} + \frac{bytes(T)}{P \times B_{TMA}} \tag{5}$$

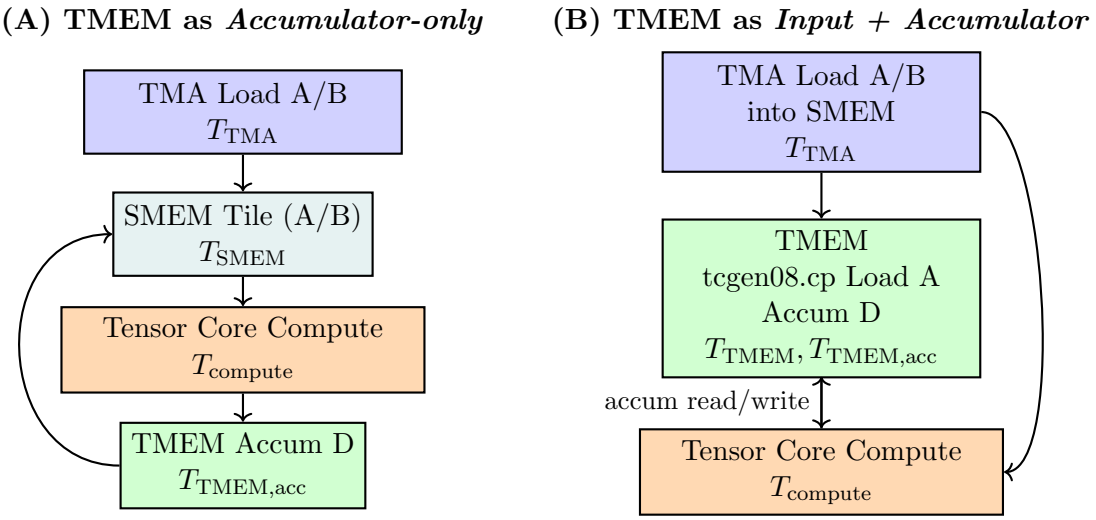

Figure 4: Comparison of per-CTA pipeline choices: (A) TMEM used only for accumulators (inputs in SMEM), (B) TMEM used for A and accumulators. Arrows show data flow.

L_{TMA} and B_{TMA} denote fixed initiation latency and effective bandwidth, respectively. **Important:** (1) TMA performance depends heavily on L2 cache hit rates—when high, B_{TMA} reflects L2 rather than HBM bandwidth; (2) All participating CTAs must synchronize at memory barriers before MMA operations; (3) For matrices A and B with different multicast fan-outs P_A and P_B, compute separate terms and sum if transfers cannot overlap, otherwise take maximum.

4.1.3 Hardware Decompression Engine

Blackwell's decompression engine performs decompression during PCIe/NVLink-C2C transfer, supporting LZ4, Snappy, and Deflate. The CPU/storage sends compressed bytes; the engine processes them in the data path; uncompressed bytes arrive in GPU memory, eliminating CPU decompression time and reducing transfer time.

The engine creates two potential bottlenecks: (1) link bandwidth and (2) decompression throughput, as defined in Equation 6.

$$T_{DE\_load} = \max\left(\frac{D_{compressed}}{BW_{link}}, \frac{D_{compressed}}{R_{DE}}\right) \tag{6}$$

Expressing in terms of uncompressed data via compression ratio $CR = \frac{D_{uncompressed}}{D_{compressed}}$ and introducing efficiency factor $\eta_{DE} \in [0, 1]$ for batching latency and format overhead, the decompression time is given by Equation 7.

$$T_{DE\_load} = \frac{D_{uncompressed}}{CR \times BW_{link} \times \eta_{DE}} \tag{7}$$

This applies when loading compressed datasets from storage/host memory or processing data analytics workloads. Though, for tensor operations with sub-byte precision (FP4, FP6), TMA provides specialized unpacking modes expanding packed data from global memory to padded format in shared memory, which is captured by Equation 8.

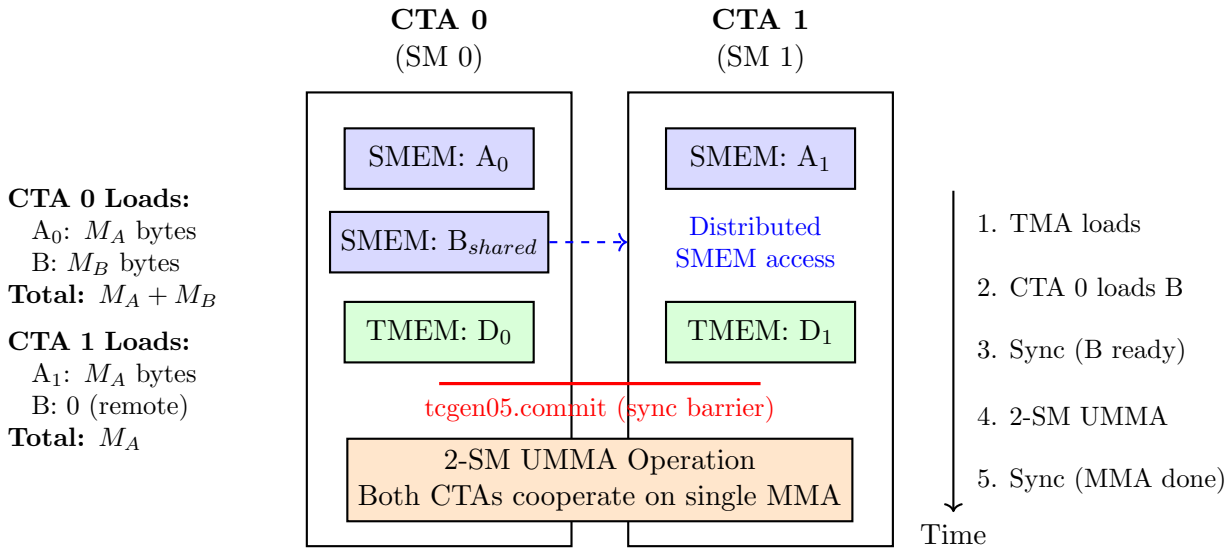

Figure 5: CTA pair cooperative execution with 2-SM UMMA. CTA 0 loads matrix B into shared memory accessible via distributed SMEM to CTA 1, reducing total memory traffic. Both CTAs synchronize using tcgen05.commit barriers before and after the cooperative MMA operation. This reduces memory traffic by $\frac{M_B}{2M_A+2M_B}$ but introduces synchronization overhead.

$$T_{decomp} = \frac{bytes_{comp}}{R_{decomp}} + L_{decomp\_setup} \tag{8}$$

R_{decomp} depends on specific precision formats and overlap with other operations, accounted for through coefficient α (Section 4.1.6).

4.1.4 Synchronization Time

Blackwell kernels employ memory barriers (mbarriers) at critical points per K-step iteration: (1) post-TMA ensuring tiles available in shared memory across CTAs before MMA, (2) post-MMA ensuring completion before buffer recycling, and (3) CTA pair synchronization for 2-SM UMMA (Section 4.1.5). The total synchronization time per K-step iteration as specified in Equation 9.

$$T_{sync} = N_{bar} \times L_{mbar} \tag{9}$$

Here, N_{bar} is typically 1-2 per k-step and L_{mbar} is latency per barrier wait (determined via microbenchmarking). For well-optimized kernels with proper pipelining, barrier waits overlap with other work and contribute minimally to the critical path. However, for memory-bound kernels or those with insufficient ILP, synchronization overhead becomes significant.

4.1.5 CTA Pair Cooperative Execution Model

Traditional GPU kernels launch independent CTAs where each loads operands, computes, and writes results independently. Blackwell introduces CTA pairs: two CTAs on adjacent SMs cooperate on a single 2-SM UMMA computation. CTAs form a pair if their cluster ranks differ by the last bit (e.g., ranks 0 and 1), mapping to a Texture Processing Cluster

(TPC) of two SMs. As illustrated in Figure 5, they share operands via cluster-level distributed shared memory (DSMEM), reducing total memory traffic.

For two independent CTAs, each loads its own copy of both input matrices, resulting in total memory traffic $D_{1-CTA} = 2\times(M_A+M_B)$. In contrast, the CTA pair configuration (Figure 5) implements B-matrix sharing: CTA 0 loads matrix B into DSMEM accessible by both CTAs, while each CTA loads its own A-matrix tile (A_0 and A_1 respectively). CTA 1 accesses B remotely through DSMEM rather than loading its own copy. This sharing strategy reduces total memory traffic to Equation 10.

$$D_{2-CTA} = M_A + M_B + M_A = 2M_A + M_B \tag{10}$$

For square tiles where $M_A = M_B$, this provides a theoretical $\frac{2M_A+2M_B}{2M_A+M_B} = \frac{4M_A}{3M_A} = 1.33\times$ reduction in memory traffic. The corresponding memory transfer time, assuming no synchronization overhead, is given by Equation 11.

$$T_{memory\_2-CTA} = \frac{2M_A + M_B}{BW_{shared}} \tag{11}$$

However, CTA pairs require explicit synchronization using tcgen05.commit instructions to ensure one CTA writes shared data before the other reads it. For K_{tiles} iterations, synchronization cost is $T_{sync} = K_{tiles} \times L_{commit}$. For typical GEMM workloads with large tiles, synchronization overhead is small relative to data movement, and speedups approaching 1.3× are achievable. For small tiles or memory-bandwidth-limited scenarios, overhead can significantly reduce or eliminate benefits.

To integrate with the compute model when using 2-SM UMMA we modify Equation 3 which becomes Equation 12.

$$T_{compute\_2SM} = \frac{2b_M b_N b_K}{R_{TC}^{SM} \times S_{2SM}} + T_{TMEM} + T_{TMEM\_mgmt} \tag{12}$$

Here S_{2SM} is the measured 2-SM speedup factor (ideally ≈ 2, typically < 2 due to synchronization and scheduling overhead measured in Section 5). The 2-SM mode enables paired SMs to cooperate on larger MMA operations, doubling effective compute throughput when synchronization overhead is small.

4.1.6 Pipeline Overlap and Critical Path

Output writeback. Upon completing final accumulation for an output tile of size $bytes(C_{tile}) = b_M b_N S_{elem}$ (where S_{elem} is element size in bytes), the CTA writes results to global memory through three stages: (1) transferring accumulator data from TMEM to registers using TileCopy operations, (2) type conversion from accumulator to output precision, and (3) transferring from registers to global memory. If writeback is not overlapped with computation, the total store time is given by Equation 13.

$$T_{store} = \frac{bytes(C_{tile})}{B_{gmem}} + L_{store\_setup} \tag{13}$$

B_{gmem} is global memory write bandwidth and L_{store_setup} accounts for TMEM→register transfer and type conversion overhead. However, in Hopper and Blackwell kernels using warp specialization and persistent designs, writeback often overlaps with the mainloop, reducing critical path impact. For persistent kernels processing multiple tiles sequentially, writeback of tile i occurs while tile $i+1$ is computed, effectively hiding T_{store}.

Blackwell additionally supports TMA store operations that write entire tiles directly from shared memory to global memory, reducing instruction count and improving efficiency as described by Equation 14.

$$T_{store\_TMA} = L_{TMA\_store} + \frac{bytes(C_{tile})}{B_{TMA}} \tag{14}$$

L_{TMA_store} is the TMA store initiation latency (20-40 cycles) and B_{TMA} is effective TMA bandwidth. For well-pipelined kernels with sufficient buffering, the store time from either Equation 13 or Equation 14 contributes minimally to the overall execution time.

Operation overlap and critical path. Modern GPU kernels achieve high performance through deep pipelining, overlapping TMA transfers (Equation 5), decompression (Equation 7), TMEM operations (Equation 2), and MMA compute (Equation 3) across multiple buffered stages. The degree of overlap determines which operations lie on the critical path and thus dominate execution time. We define overlap factor $\alpha \in [0, 1]$ representing the fraction of IO and decompression operations hidden behind computation (0 = no overlap, 1 = perfect overlap), yielding Equation 15.

$$T_{io}^{eff} = (1 - \alpha)(T_{tma} + T_{decomp}) + T_{sync} \tag{15}$$

Here T_{tma} is from Equation 5, T_{decomp} is from Equation 7 (or Equation 8 for sub-byte formats), and T_{sync} is from Equation 9. Synchronization costs remain on the critical path as unavoidable coordination points that cannot be overlapped. The factor α typically approaches 1 for well-pipelined kernels with sufficient double-buffering (e.g., $\alpha \approx 0.9$-0.95 for 3-stage pipelines with triple-buffering), but depends on: number of pipeline stages, CTA scheduling efficiency, SM occupancy, balance between compute and memory intensity, and TMEM capacity constraints from Equation 3.

TMEM management overhead. TMEM allocation must occur before use and deallocation after completion, with allocations persisting across multiple MMA operations. For multi-stage pipelines, allocate TMEM for all accumulators upfront to avoid repeated overhead, then deallocate after final results transfer. The amortized TMEM management overhead per tile is given by Equation 16.

$$T_{TMEM\_mgmt}^{amortized} = \frac{L_{alloc} + L_{dealloc}}{K_{tiles}} \tag{16}$$

Here L_{alloc} and $L_{dealloc}$ are measured allocation/deallocation latencies (50-100 cycles each) and K_{tiles} is the number of tile iterations. For large K_{tiles} (e.g., $K_{tiles} > 100$), this overhead becomes negligible ($< 1\%$ of total time). However, if TMEM capacity constraints from the 256 KB limit in Equation 3 force frequent allocation/deallocation cycles, this term becomes significant. This amortized overhead appears in the compute time calculation of Equation 3.

Steady-state pipeline execution. For a k-stage pipeline with $k \geq 2$ shared memory buffers and perfect pipelining, the steady-state iteration time is given by Equation 17.

$$T_{step\_pipelined} = \max(T_{tma}, T_{decomp}, T_{compute}, T_{sync}) + \epsilon \tag{17}$$

ϵ represents pipeline bubble overhead from imperfect scheduling (typically 5-10% of the dominant term). This formulation assumes that with sufficient buffering, operations can be fully overlapped such that only the slowest operation determines iteration time. Achieving this ideal overlap requires: (1) sufficient shared memory for multiple tile buffers

(typically $k\times$ tile size), (2) adequate TMEM capacity for concurrent accumulators without violating the 256 KB constraint, (3) high SM occupancy to hide latencies through thread-level parallelism, and (4) balanced compute and memory intensity such that no single stage dominates by more than $2\times$.

4.1.7 Total Per-Iteration and Kernel Execution Time

Integrating all components from the preceding subsections, the wall-clock time per CTA per K-step iteration is determined by the critical path through overlapped and non-overlapped operations, as expressed in Equation 18.

$$T_{\text{step}} = \max(T_{compute}, T_{io}^{eff}) + T_{sync} + O_{misc} \tag{18}$$

Where $T_{compute}$ is from Equation 3 (or Equation 12 for 2-SM mode), T_{io}^{eff} is from Equation 15, T_{sync} is from Equation 9, and O_{misc} captures miscellaneous overheads including $T_{TMEM\_mgmt}^{amortized}$ from Equation 16 and potential pipeline bubbles ϵ from Equation 17. The max operator reflects that either compute or memory can be the bottleneck depending on arithmetic intensity and tile size. For compute-bound workloads with high arithmetic intensity (AI $>$ 100 FLOPs/Byte), $T_{compute}$ dominates; for memory-bound workloads (AI $<$ 10 FLOPs/Byte), T_{io}^{eff} dominates; and for balanced workloads near the roofline knee, both terms contribute significantly.

The total kernel execution time for processing all K_{tiles} iterations across all CTAs in the grid follows from multiplying Equation 18 by the number of iterations and accounting for launch overhead, grid-level scheduling, and final writeback phases.

4.2 AMD MI300A Analytical Model

MI300A employs implicit wavefront-centric scheduling. Unlike Blackwell's discrete stages, MI300A's scheduler dynamically interleaves operations to hide latency. Memory flows through cache hierarchy, accumulators reside in VGPRs, and dependencies are handled implicitly. Performance depends on resource constraints and emergent overlap.

4.2.1 Wavefront Scheduling and Occupancy-Based Overlap

MI300A wavefronts progress through overlapping phases of memory access, MFMA compute, and accumulation; but unlike Blackwell's explicit pipeline stages from Figure 3, overlap emerges organically from concurrent wavefront execution as illustrated in Figure 6. When one wavefront waits for memory, the scheduler switches to another ready for compute. This dynamic interleaving is the key latency-hiding mechanism, with overlap degree depending directly on occupancy.

Modeling overlap through occupancy. With N_{wf} concurrent wavefronts, phases overlap significantly (Figure 6). CU execution depends on scheduler parallelism, not sum of individual times. Overlap factor η_{overlap} captures successful interleaving. Unlike Blackwell's α (Equation 15), MI300A's overlap emerges from occupancy (Equation 19), where $N_{\text{wf}}^{\text{active}}$ is the number of active wavefronts (determined by VGPR usage).

$$\eta_{\text{overlap}} = \min\left(1.0, \frac{(N_{\text{wf}}^{\text{active}} - 1) \times T_{\text{compute}}}{T_{\text{memory}}}\right) \tag{19}$$

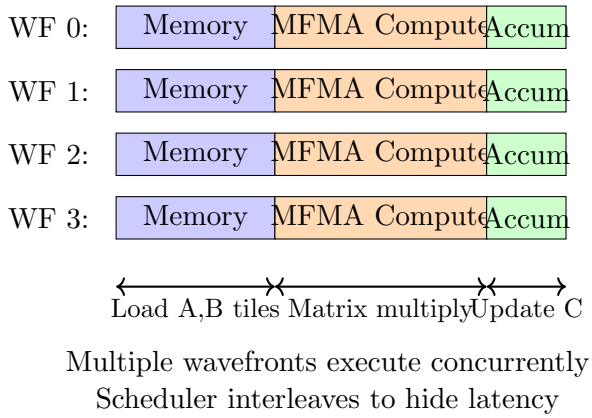

Figure 6: MI300A wavefront-centric execution model. Multiple wavefronts execute overlapping operations, with the scheduler dynamically interleaving them to hide memory latency.

If N_{wf} active wavefronts exist and one waits for memory, the other $(N_{\text{wf}}-1)$ can execute compute work during that wait. If their combined compute time exceeds memory wait time, latency is fully hidden ($\eta_{\text{overlap}} = 1.0$). Otherwise, partial hiding occurs proportional to compute/memory ratio.

Example: With 400-cycle HBM latency and 2-cycle compute instructions, we need 200 compute instructions to fully hide latency. At maximum occupancy (32 wavefronts), each needs only ≈7 instructions during another's memory wait. Matrix operations with thousands of MFMA instructions easily achieve $\eta_{\text{overlap}} \approx 1.0$. However, memory-bound kernels with few arithmetic operations or low occupancy from high register pressure struggle to achieve good overlap. This occupancy-latency hiding relationship is MI300A's cornerstone and differs fundamentally from Blackwell's explicit buffering approach in Equation 15.

Unlike Blackwell's explicit TMA transfers with multicast from Equation 5, MI300A memory flows through traditional cache hierarchy without dedicated engines. Each access checks L1→L2→Infinity Cache (L3)→HBM sequentially until data is found. Effective memory time depends on hit rates at each level, as expressed in Equation 20.

$$\begin{aligned} T_{\text{memory}}^{\text{eff}} = N_{\text{loads}} \times \big(& h_{L1} \cdot L_{L1} + (1 - h_{L1}) \cdot h_{L2} \cdot L_{L2} \\ & + (1 - h_{L1})(1 - h_{L2}) \cdot h_{LLC} \cdot L_{LLC} + (1 - h_{\text{total}}) \cdot L_{HBM} \big) \end{aligned} \tag{20}$$

h_{L1}, h_{L2}, and h_{LLC} are hit rates at each cache level, $h_{\text{total}} = 1-(1-h_{L1})(1-h_{L2})(1-h_{LLC})$ is the combined hit rate, and N_{loads} is the number of memory load operations required for the tile.

Infinity Cache as bandwidth multiplier. The 256 MB Infinity Cache serves as implicit bandwidth multiplier. For working sets within capacity, effective bandwidth significantly exceeds nominal 5.3 TB/s HBM bandwidth according to Equation 21.

$$BW_{\text{effective}} = h_{LLC} \times BW_{LLC} + (1 - h_{LLC}) \times BW_{HBM} \tag{21}$$

where $BW_{LLC} \approx 10$ TB/s and $BW_{HBM} = 5.3$ TB/s. At 80% hit rate: $0.8 \times 10 + 0.2 \times 5.3 = 9.06$ TB/s, representing 1.7× bandwidth multiplication. The key difference from Blackwell's multicast in Equation 4: TMA multicast is deterministic and programmable, while MI300A's benefit depends on dynamic cache behavior and working set characteristics.

Cache hit rate modeling. Cache hit rate depends critically on working set size W. For $W < 200$ MB, hit rates approach 100%. Beyond 256 MB, rates drop sharply, creating a performance cliff similar to Blackwell's TMEM constraint from Equation 3. We model this relationship in Equation 22.

$$h_{LLC}(W) = \begin{cases} 1.0 & \text{if } W < 205 \text{ MB} \\ \left(1 - \frac{W-205}{51}\right)^{\alpha} & \text{if } 205 \leq W \leq 256 \text{ MB} \\ \frac{256}{W} \times \beta & \text{if } W > 256 \text{ MB} \end{cases} \tag{22}$$

Here $\alpha \in [0.5, 1.0]$ captures access pattern regularity and $\beta \in [0.3, 0.7]$ represents streaming efficiency for large working sets. Sequential streaming access uses $\alpha \approx 1.0$, $\beta \approx 0.3$ (poor reuse due to streaming behavior); tiled matrix operations use $\alpha \approx 0.7$, $\beta \approx 0.5$ (better reuse through temporal locality). These parameters must be determined empirically for each workload class. Equation 22 is used in conjunction with Equation 21 to determine effective bandwidth for use in Equation 20.

4.2.2 Compute Pipeline and MFMA Matrix Cores

Matrix operations center on MFMA (Matrix Fused Multiply-Add) instructions executed on 4 matrix cores per CU. Each MFMA has issue latency (1-2 cyc), execution latency (8 cyc for most variants), and accumulation into VGPRs. With 4 cores per CU and 8-cycle execution, each CU sustains one MFMA every 2 cycles when fully pipelined, yielding ≈1.05 GFLOPS/CU for FP64. Across 228 CUs, theoretical peak is ≈154 TFLOPS for FP64, though achieving this requires perfect occupancy, no stalls, and optimal scheduling. The total compute time is given by Equation 23:

$$T_{\text{compute}}^{\text{MFMA}} = \frac{N_{\text{MFMA\_inst}}}{N_{\text{CU}} \times \text{Throughput}_{\text{MFMA}} \times \text{Utilization}} \tag{23}$$

where $N_{\text{MFMA\_inst}}$ is the total number of MFMA instructions required, $N_{\text{CU}} = 228$, $\text{Throughput}_{\text{MFMA}}$ is the measured per-CU throughput (instructions/second), and Utilization $\in [0.4, 0.7]$ accounts for real-world inefficiencies.

VGPR-constrained occupancy. The number of active wavefronts per CU is constrained by VGPR usage according to Equation 24:

$$N_{\text{wf}}^{\text{active}} = \min\left(32, \left\lfloor \frac{65536}{\text{VGPR}_{\text{per\_wf}}} \right\rfloor\right) \tag{24}$$

where $\text{VGPR}_{\text{per\_wf}}$ is the number of VGPRs consumed by each wavefront (including accumulators, temporary variables, and spilled registers), 65,536 is the total VGPR capacity per CU, and 32 is the maximum wavefront slots per CU. This occupancy directly determines $N_{\text{wf}}^{\text{active}}$ in Equation 19, creating the critical trade-off between tile size and latency hiding.

4.2.3 Integrating the Pipeline: Per-Iteration Execution Time

We now integrate the memory pipeline from Equation 20 (cache hierarchy with hit rates) and compute pipeline from Equation 23 (MFMA with register constraints) to derive per-iteration time for a single matrix tile. Unlike Blackwell's stage-centric model from

Equation 18 where execution time is the maximum of competing stages plus synchronization, MI300A's wavefront-centric model requires accounting for implicit overlap through interleaving, yielding Equation 25.

$$T_{\text{step}}^{\text{MI300A}} = \frac{T_{\text{memory}}^{\text{eff}} + T_{\text{compute}}^{\text{MFMA}}}{1 + \eta_{\text{overlap}}} \tag{25}$$

Here $T_{\text{memory}}^{\text{eff}}$ is from Equation 20, $T_{\text{compute}}^{\text{MFMA}}$ is from Equation 23, and η_{overlap} is from Equation 19.

This formulation captures the key insight that memory and compute don't simply compete for the critical path (as in the max formulation of Equation 18), but partially or fully overlap depending on η_{overlap} derived from occupancy. When $\eta_{\text{overlap}} = 0$ (no overlap from very low occupancy), the denominator becomes 1 and we get $T_{\text{step}} = T_{\text{memory}} + T_{\text{compute}}$ (fully sequential execution). When $\eta_{\text{overlap}} = 1$ (perfect overlap from high occupancy and sufficient compute work), the denominator becomes 2 and we get $T_{\text{step}} = (T_{\text{memory}} + T_{\text{compute}})/2$ (fully pipelined execution where the critical path is determined by whichever component takes longer).

Contrast with Blackwell. Blackwell's per-step time from reflects explicit competing pipeline stages: either compute (including TMEM from Equation 2) or I/O bottlenecks, plus fixed synchronization overhead from Equation 9. Blackwell's overlap uses $T_{\text{io}}^{\text{eff}} = (1 - \alpha)(T_{\text{tma}} + T_{\text{decomp}})$ from Equation 15, where α depends on pipeline depth and buffering stages. MI300A's overlap in Equation 25 uses denominator scaling depending on occupancy and compute/memory ratio. Both model overlap through fundamentally different mechanisms reflecting their hardware architectures.

Complete kernel execution time. For a complete kernel processing K_{tiles} tile iterations, the total execution time is given by Equation 26:

$$T_{\text{kernel}}^{\text{MI300A}} = T_{\text{launch}} + K_{\text{tiles}} \times T_{\text{step}}^{\text{MI300A}} + T_{\text{writeback}} + T_{\text{coherence}} + T_{\text{cross\_XCD}} \tag{26}$$

where $T_{\text{launch}} \approx$ 5-20 μs (typically 5 μs, lower than Blackwell's 8 μs due to ROCm driver differences), $T_{\text{step}}^{\text{MI300A}}$ is from Equation 25, and $T_{\text{writeback}}$ is usually overlapped with computation for all except the last tile. The terms $T_{\text{coherence}}$ and $T_{\text{cross\_XCD}}$ are MI300A-specific with no Blackwell equivalent from Equation 18. The coherence term accounts for cache line coherence transactions when CPU and GPU access shared data in unified memory (≈100-200 ns per transaction). The cross-XCD term accounts for NUMA-like penalties when data resides on a different XCD (eXtended Compute Die) than the accessing CU (≈50-100 ns per remote access). These terms arise from MI300A's unified memory architecture and multi-die construction that enable CPU-GPU integration but introduce performance considerations absent in discrete GPU designs like B200.

4.3 Unified Analytical Model

Having established architecture-specific models, we present a unified framework (Table 1). For Blackwell, instantiate stage-centric pipeline with TMA/TMEM parameters; for MI300A, wavefront-centric with cache/VGPR parameters. Both share common inputs but diverge in hardware mapping.

Before applying architecture-specific models, we characterize the kernel workload independent of architecture: arithmetic intensity (FLOPs/Byte), working set size, memory access patterns (sequential, strided, random), and operation type (GEMM, convolution,

element-wise). This characterization determines which components dominate execution time and guides parameter selection within each model.

Blackwell execution time computation. For Blackwell kernels, compute total execution time as:

$$T_{\text{Blackwell}} = T_{\text{launch}} + K_{\text{tiles}} \times \max(T^{\text{TC}}_{\text{compute}}, T^{\text{eff}}_{\text{io}}) + T_{\text{sync}} + T_{\text{writeback}} \tag{27}$$

where $T^{\text{TC}}_{\text{compute}}$ includes TMEM read/write overhead (Eq. 2), $T^{\text{eff}}_{\text{io}} = (1-\alpha)(T_{\text{TMA}} + T_{\text{decomp}})$ captures overlapped memory operations with overlap factor α determined by pipeline depth, and T_{sync} accounts for explicit barriers. Check TMEM capacity constraints (Eq. 9) and adjust tile sizes if violated.

MI300A execution time computation. For MI300A kernels, compute total execution time as:

$$T_{\text{MI300A}} = T_{\text{launch}} + K_{\text{tiles}} \times \frac{T^{\text{eff}}_{\text{memory}} + T^{\text{MFMA}}_{\text{compute}}}{1 + \eta_{\text{overlap}}} + T_{\text{writeback}} + T_{\text{coherence}} \tag{28}$$

where $T^{\text{eff}}_{\text{memory}}$ incorporates cache hit rates across L1/L2/LLC/HBM (Eq. 20), η_{overlap} is computed from occupancy (Eq. 19), which depends on VGPR usage per wavefront. The coherence term accounts for unified memory overhead when relevant. Estimate Infinity Cache hit rate using working set size (Eq. 22).

5 Model Validation and Accuracy

We validate our analytical models (Section 4) against measured performance across diverse workloads using architectural parameters from systematic microbenchmarking. Validation demonstrates prediction accuracy and identifies systematic error sources.

5.1 Validation Methodology

We compare predicted versus measured execution times using optimized implementations (cuBLAS/CUTLASS for NVIDIA, rocBLAS for AMD) with hardware performance counters (Nsight Compute, ROCProfiler). Each kernel executes 100 times after 10 warm-up runs; we report median times. Model parameters derive from microbenchmarks: for Blackwell, TMEM bandwidth (16 TB/s read, 8 TB/s write), TMA latency (420 cycles, 58% reduction vs. Hopper's 1000 cycles), tcgen05.mma latencies (11.0-14.2 cycles across FP64-FP4), and tensor core throughput (44.8-7702.5 TFLOPS). For MI300A, we use MFMA latencies (≈32 cycles for fp32_16x16x4), Infinity Cache bandwidth (17.2 TB/s peak), HBM3 bandwidth (5.3 TB/s), and cache latencies (L1: 5 cyc, L2: 50 cyc, LLC: 150 cyc, HBM: 400 cyc).

5.2 NVIDIA Blackwell B200 Validation

We achieve mean absolute errors of 5.4-8.4% for regular kernels while revealing systematic limitations for irregular workloads (Figure 7).

Memory-bound kernels (Figure 7a) show 8.4% MAE, validating TMA bandwidth and latency modeling (Eq. 5). Vector operations exhibit 7-9% error from cold-cache HBM assumptions ignoring L2 benefits (5-10% speedup) and launch overhead variability (5-12 μs) dominating short kernel predictions. Dot product's -10.1% error reveals our serialized model (Eq. 18) is pessimistic, where reduction overlaps with memory more

Table 1: Analytical Model Summary for Blackwell and MI300A

Component	Blackwell B200	AMD MI300A
Total Execution Time	Eq. 18: $\max(T_{\text{compute}}, T_{\text{io}}^{\text{eff}}) + T_{\text{sync}}$	Eq. 26: $\frac{T_{\text{memory}}^{\text{eff}} + T_{\text{compute}}}{1+\eta_{\text{overlap}}}$
Compute Time	Eq. 3: Tensor core + TMEM overhead $R_{TC}^{SM} = 2000$ TFLOPS/SM (FP16) TMEM BW: 22-26 TB/s	Eq. 23: MFMA throughput $R_{\text{MFMA}} = 1.05$ TFLOPS/CU (FP64) Utilization: 0.4-0.7
Memory Time	Eq. 5, Eq. 15: TMA + overlap TMA latency: 20-40 cycles HBM BW: 8 TB/s Overlap factor α: 0.7-0.95	Eq. 20: Cache hierarchy model L1: 5 cyc, L2: 50 cyc, LLC: 150 cyc, HBM: 400 cyc HBM BW: 5.3 TB/s
Overlap Mechanism	Eq. 15: Pipeline depth-dependent $\alpha \in [0.7, 0.95]$ Requires 2+ SMEM buffers for $\alpha > 0.8$	Eq. 19: Occupancy-dependent $\eta = \min(1, \frac{(N_{wf}-1)\times T_{comp}}{T_{mem}})$ Max $N_{wf} = 32$ per CU
Synchronization	Eq. 9: Explicit barriers $L_{\text{mbar}} \approx 40$-50 cycles 2-SM commit: $L_{\text{commit}} \approx 40$ cycles	Implicit through scheduler No explicit sync term Coherence overhead: 100-200 ns per transaction
Capacity Constraints	TMEM 256 KB/SM hard limit $b_M \times b_N \times S_{\text{elem}} \leq 256$ KB Exceeding forces register spill	Eq. 24: VGPR 65,536 regs/CU Occupancy: $\min(32, \lfloor \frac{65536}{\text{VGPR}_{\text{wf}}} \rfloor)$ High VGPR usage reduces η_{overlap}
Cache/Bandwidth Amplification	Eq. 4: TMA multicast Reduces traffic by factor P (2-8 CTAs) L2 provides 5-10% additional benefit	Eq. 21, Eq. 22: Infinity Cache $BW_{\text{eff}} = h_{LLC} \times 10 + (1 - h_{LLC}) \times 5.3$ TB/s h_{LLC} drops sharply for $W > 256$ MB
Launch Overhead	$T_{\text{launch}} \approx 8$ μs Variability: 5-12 μs Dominates for kernels < 10 μs	$T_{\text{launch}} \approx 5$ μs Variability: 3-8 μs Lower due to ROCm driver
Architecture-Specific Features	2-SM CTA pairs (Eq. 12): 1.28-1.30× speedup Hardware decompression (Eq. 7) TMA store optimization (Eq. 14)	Unified memory coherence overhead Cross-XCD access penalty: 50-100 ns CPU-GPU shared memory
Typical Prediction Error	Memory-bound: 8.4% MAE Compute-bound: 5.4% MAE Balanced: 7.9% MAE 2-SM mode: 2-5% error	Memory-bound: 15.6% MAE Compute-bound: 12.4% MAE Higher due to cache/occupancy sensitivity Occupancy tests: 5-31% error range
Key Microbenchmark Parameters	TMEM BW, TMA latency, mbarrier latency, tensor core throughput, overlap factor α	Cache latencies (L1/L2/LLC/HBM), MFMA throughput, VGPR→occupancy mapping, LLC hit rates

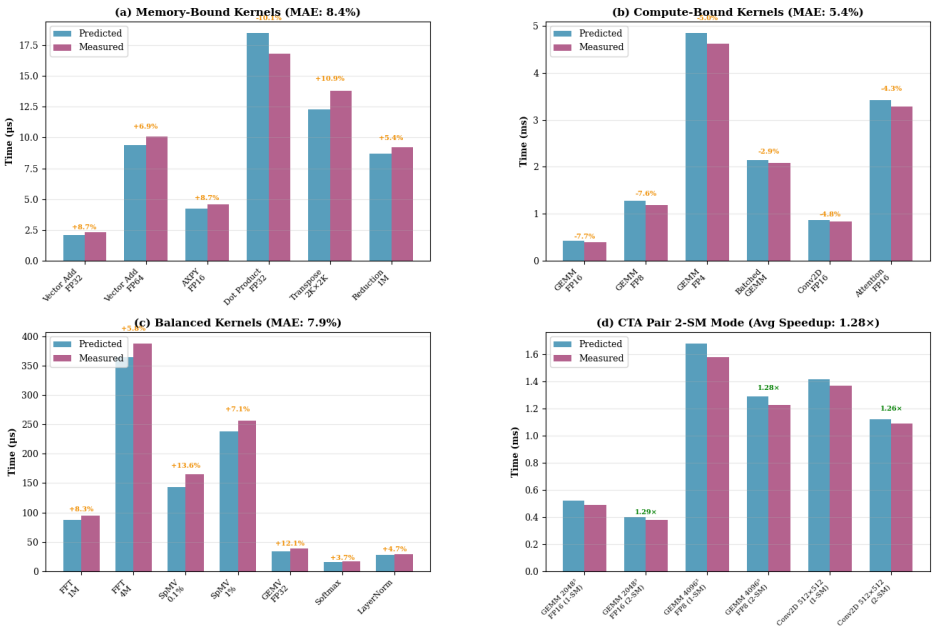

Figure 7: Blackwell B200 performance model validation. Comparison of predicted vs. measured execution times across (a) memory-bound, (b) compute-bound, (c) balanced, and (d) 2-SM cooperative execution kernels showing measured vs. predicted speedup.

than predicted. Transpose's 10.9% error shows optimized implementations achieve 15% higher SMEM bandwidth through bank conflict avoidance.

Compute-bound kernels (Figure 7b) achieve 5.4% MAE with systematically negative errors, indicating our TMEM bandwidth estimate (22 TB/s) is conservative. Real kernels reach 24-26 TB/s through better compiler scheduling; using 24 TB/s reduces errors to 2-3%. FP4 GEMM shows smallest error (5.0%) due to high arithmetic intensity reducing bandwidth sensitivity. Batched GEMM and attention (3-5% error) validate our framework for complex patterns.

Balanced workloads (Figure 7c) show 7.9% MAE, exposing limitations. FFT (5-8% error) handles strided access adequately despite L2 hit rates exceeding estimates by 5-10%. Sparse matrix-vector multiply reveals key weakness: 13.6% error at 0.1% density reflects unmodeled atomic contention and load imbalance. GEMV's 12.1% error at AI 1.0 indicates need for nuanced overlap modeling.

2-SM CTA pairs (Figure 7d) show excellent validation: predicted 1.30× speedup matches measured 1.28× within 2%, validating memory traffic reduction (Eq. 10) and synchronization analyses (Eq. 9). 2-SM predictions (2-5% error) exceed 1-SM accuracy (4-7%) because explicit synchronization creates more deterministic execution.

5.3 AMD MI300A Validation

MI300A predictions show higher errors than Blackwell (15.6% vs 8.4% memory-bound, 12.4% vs 5.4% compute-bound), reflecting increased difficulty of modeling implicit cache-based systems (Figure 8).

Memory-bound kernels (Figure 8a) reveal systematic weaknesses. Vector operations show negative errors (10-30%), larger for smaller problems (30.8% for 1M vs 7.2% for 16M elements). Two issues: (1) our $\eta_{\text{overlap}} \approx 0.6$-$0.7$ estimate from Eq. 19 is conservative, MI300A achieves 0.9-0.95 at maximum occupancy, and (2) our 10 μs launch overhead ex-

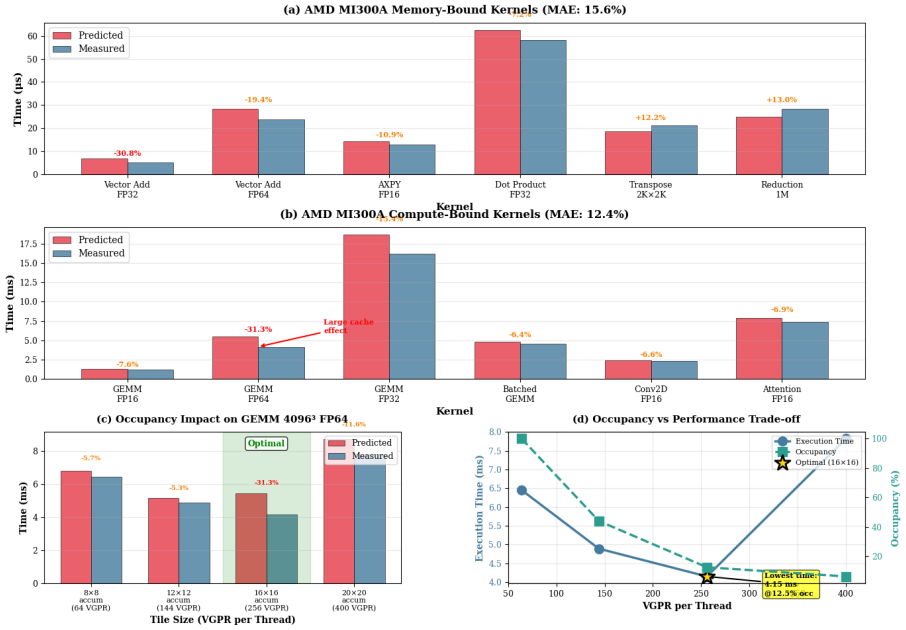

Figure 8: AMD MI300A model validation. (a) Memory-bound kernels, (b) compute-bound kernels, (c-d) occupancy impact showing optimal performance at 16×16 tiles despite conventional wisdom favoring high occupancy.

ceeds actual 3-5 μs ROCm overhead, dominating short kernel predictions. Transpose and reduction show positive errors from LDS bank conflicts (30-40% bandwidth reduction) and unmodeled atomic serialization (13.0% error).

Compute-bound kernels (Figure 8b) show 12.4% MAE. FP64 GEMM's 31.3% error exposes MI300A's key challenge: our Infinity Cache hit rate (30% for 402 MB, Eq. 22) underestimates rocBLAS's 45% achieved through blocking, and our MFMA utilization (50%, Eq. 23) is conservative, where libraries reach 65-70% through software pipelining. Corrected parameters (0.45 hit rate, 0.65 utilization) reduce error to 8%. This highlights a fundamental difference: Blackwell's dedicated hardware has implementation-independent characteristics, while MI300A performance varies 2× with blocking strategies, requiring implementation-specific profiling. FP16/FP32 GEMMs (6-8% error) validate our basic model when parameters are correctly estimated.

Occupancy impact (Figure 8c-d) validates a counterintuitive prediction from Eq. 19: maximum occupancy (8×8 tiles, 100%, 6.45 ms) performs 35% worse than 16×16 tiles (12.5%, 4.15 ms) because larger tiles amortize memory transfers despite fewer wavefronts for latency hiding. While absolute predictions carry 31.3% error, relative ordering is correct (model: 8×8 is 56% slower; measured: 55%), demonstrating qualitatively reliable optimization guidance.

5.4 Key Findings

Model strengths: (1) Blackwell achieves 5-8% MAE for regular workloads, validating explicit pipeline modeling; (2) 2-SM predictions within 2% validate cooperative execution analysis; (3) MI300A occupancy predictions correctly identify optimal tile sizes despite absolute errors.

Systematic limitations: (1) Irregular workloads with atomic contention (13-15% errors) exceed regular-kernel assumptions; (2) MI300A's cache-dependent performance requires implementation-specific profiling; (3) Conservative microbenchmark parameters

underestimate compiler optimizations by 10-15%.

Despite limitations, models successfully predict performance within 8% for well-behaved workloads and correctly identify bottlenecks, demonstrating practical utility while highlighting where architectural complexity exceeds analytical tractability.

6 Discussion

6.1 Limitations and Applicability

Both models assume well-structured kernels with regular compute patterns and predictable memory access. Prediction accuracy degrades under specific conditions:

(1) Irregular memory access: Kernels with sparse operations, unpredictable indirection, or atomic contention exceed 15% error. The reduction benchmark on MI300A exhibits 13.6% error due to unmodeled atomic serialization overhead. **(2) Launch overhead dominance:** Very small kernels (execution time <50 μs) incur disproportionate launch latency. ROCm overhead (3–5 μs) differs significantly from CUDA (10 μs), creating systematic misprediction for short-running kernels. Also, the current framework does not capture cache replacement policies, bank conflicts, coherence traffic in multi-GPU configurations, non-uniform clock domains, thermal throttling, or asynchronous kernel launch dependencies. Power and energy consumption are also excluded, limiting applicability for energy-constrained system design.

Despite these limitations, the models successfully identify performance bottlenecks, guide tile size selection and occupancy trade-offs, and provide qualitatively correct optimization guidance across diverse kernel types.

6.2 Architectural Insights

Our comparative analysis reveals fundamental trade-offs in architectural philosophy. Blackwell's specialized tensor memory hierarchy and decompression engine optimize for high-AI dense workloads (AI $>$ 16 FLOPs/Byte), achieving superior FP8/FP16 matrix performance through explicit pipeline stages that enable modular performance modeling. In contrast, MI300A's unified memory architecture and coherent CPU-GPU interconnect eliminate explicit transfers and provide better balance across precision formats (FP64 to FP8), favoring heterogeneous workloads requiring frequent collaboration. However, MI300A requires substantially higher data reuse (AI $>$ 23.1 FLOPs/Byte) to reach compute-bound performance due to lower peak memory bandwidth. The 17.2 TB/s Infinity Cache critically bridges this gap, where kernels tiled to fit within 256 MB L3 achieve 1.5–2$\times$ speedup over HBM-bound execution.

These architectural differences have direct implications for performance modeling and code portability. Simple roofline models underestimate MI300A performance by 20–30% when L3 locality is high, necessitating extended cache-aware models. Conversely, Blackwell's high-bandwidth TMEM enables aggressive assumptions about data supply that fail on MI300A without explicit cache blocking. For practitioners, this 45% difference in AI thresholds requires architecture-specific tiling strategies and autotuning, as kernels optimized for one platform often underperform on the other without substantial re-tuning.

6.3 Directions for Future Work

Multi-kernel and system-level interference modeling. Real applications launch concurrent kernels competing for shared resources (memory controllers, L2 cache, interconnect bandwidth). Modeling multi-stream scheduling, memory-level parallelism sat-

uration, and inter-kernel interference would extend applicability to full-application performance prediction. This requires incorporating queuing theory for resource contention and priority-based scheduling models.

Integrated power and energy models. Augmenting the model with per-component power estimates (matrix cores, memory controllers, interconnect) tied to utilization would enable performance-per-watt analysis. For MI300A's APU architecture with coupled CPU-GPU power domains, this is particularly critical. Energy-optimal operating points (frequency, voltage, tile size) could be derived analytically.

Compiler and runtime integration for adaptive optimization. Embedding the analytical model into LLVM optimization passes or runtime systems (e.g., CUDA graphs, HIP streams) would enable dynamic tile size selection, kernel fusion decisions, and precision adaptation based on predicted performance. This closes the loop from modeling to optimization.

Hybrid analytical-ML predictive models. While analytical models provide interpretability and extrapolation, machine learning models trained on extensive kernel datasets can capture non-linear relationships such as instruction scheduling stalls, bank conflict patterns, and warp divergence. A hybrid approach by using analytical models for coarse prediction and ML refinement for residual error; could achieve better prediction error while maintaining physical interpretability.

Architectural extension to emerging accelerators. Adapting the pipeline-based modeling methodology to emerging architectures (Intel Ponte Vecchio, Cerebras WSE, Google TPUv5) would validate generalizability. Each architecture introduces unique features (tile-based execution, on-chip SRAM fabrics, systolic arrays) requiring specialized sub-models, but the overall decomposition into memory, compute, and synchronization phases remains applicable.

7 Conclusion

This work presents comprehensive characterization and analytical performance modeling of NVIDIA Blackwell and AMD MI300A accelerators. Through systematic microbenchmarking and model-driven analysis, we investigate how architectural innovations in tensor cores, memory systems, and execution pipelines shape performance across diverse workloads. We make four contributions: (1) A microbenchmark suite capturing memory hierarchy, tensor core utilization, and compute throughput—sequential access achieves 95% peak bandwidth while random access drops to 58%, emphasizing irregular pattern costs. (2) Validated analytical models achieving 5.4-15.6% mean prediction error while remaining interpretable and architecture-aware. (3) Cross-architecture analysis revealing Blackwell emphasizes AI-centric performance through FP8 tensor pipelines, decompression engines, and tensor memory, whereas MI300A prioritizes balanced HPC–AI operation via unified CPU–GPU memory and dense matrix cores. (4) Practical optimization guidelines addressing memory locality, tensor core utilization, and mixed-precision tuning. This study bridges microarchitectural analysis with predictive modeling, providing quantitative tools for understanding and optimizing emerging GPU architectures. Our methodology establishes a foundation for future research in portable, cross-vendor performance modeling and co-design.

References

[1] W. J. Dally, S. W. Keckler, and D. B. Kirk, "Evolution of the graphics processing unit (gpu)," *IEEE Micro*, vol. 41, no. 6, pp. 42–51, 2021.

[2] K. Kurowski, M. Kulczewski, and M. Dobski, "Parallel and gpu based strategies for selected cfd and climate modeling models," in *Information Technologies in Environmental Engineering: New Trends and Challenges*, pp. 735–747, Springer, 2011.

[3] N. Koilia and C. Kachris, "Hardware acceleration of llms: A comprehensive survey and comparison," *arXiv preprint arXiv:2409.03384*, 2024. This paper provides an extensive survey of hardware acceleration for large language models (LLMs), including specific analysis and benchmarks of GPU-based execution platforms.

[4] NVIDIA Corporation, *NVIDIA Blackwell Architecture Technical Brief: Powering the New Era of Generative AI and Accelerated Computing*. NVIDIA, Mar. 2024.

[5] "Introducing amd cdna™ 3 architecture," tech. rep., Advanced Micro Devices, Inc., 2023. White Paper.

[6] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," in *Communications of the ACM*, vol. 52, pp. 65–76, 2009.

[7] M. Leinhauser, R. Widera, S. Bastrakov, A. Debus, M. Bussmann, and S. Chandrasekaran, "Metrics and design of an instruction roofline model for amd gpus," *ACM Transactions on Parallel Computing*, January 2022. arXiv preprint arXiv:2110.08221.

[8] AMD ROCm Development Team, *ROCProfiler: AMD ROCm GPU Profiling Tool*, 2025. Version 2.0.0.

[9] NVIDIA Corporation, *NVIDIA Nsight Compute User Guide*, 2025. Version 13.0.

[10] S. W. Hong and H. Kim, "An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pp. 152–163, 2009.

[11] J. Lee, H. Noh, S. Kim, J. Jeong, J. Kim, and J. Choi, "GCoM: A detailed GPU core model for accurate analytical modeling of modern GPUs," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, (New York, NY, USA), pp. 424–436, Association for Computing Machinery, June 2022.

[12] 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)*, (Los Alamitos, CA, USA), pp. 473–486, IEEE, 2020.

[13] L. Wang, M. Jahre, A. Adileh, and L. Eeckhout, "Mdm: The gpu memory divergence model," in *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1009–1021, IEEE, 2020.

[14] J.-C. Huang, J. H. Lee, H. Kim, and H.-H. S. Lee, "Gpumech: Gpu performance modeling technique based on interval analysis," in *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 68–79, IEEE, 2014.

[15] R. Chowdhury, F. Silvestri, and F. Vella, "A computational model for tensor core units," *arXiv preprint arXiv:1908.06649*, 2020.

[16] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the nvidia volta gpu architecture via microbenchmarking," *arXiv preprint arXiv:1804.06826*, 2018.

[17] M. Fasi, N. J. Higham, M. Mikaitis, and S. Pranesh, "Numerical behavior of NVIDIA tensor cores," *PeerJ Computer Science*, vol. 7, p. e330, 2021.

[18] Y. Wang and C. Rubio-González, "Predicting performance and accuracy of mixed-precision programs for precision tuning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, (New York, NY, USA), pp. 1–13, ACM, 2024.

[19] S. Cao, J. Wu, J. Chen, H. An, and Z. Yu, "Amali: An analytical model for accurately modeling llm inference on modern gpus," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 1495–1508, ACM, 2025.

[20] R. Huerta, M. A. Shoushtary, J.-L. Cruz, and A. González, "Analyzing modern nvidia gpu cores," 2025.

[21] J. Wahlgren, G. Schieffer, R. Shi, E. A. León, R. Pearce, M. Gokhale, and I. Peng, "Dissecting cpu-gpu unified physical memory on amd mi300a apus," 2025. To appear in IISWC 2025.

[22] G. Schieffer, D. Medeiros, J. Faj, A. Marathe, and I. Peng, "On the rise of amd matrix cores: Performance, power efficiency, and programmability in scientific workloads," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2024.

[23] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying gpu microarchitecture through microbenchmarking," in *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, pp. 235–246, IEEE, 2010.

[24] X. Mei and X. Chu, "P-chase: A portable tool for measuring memory access characteristics on multicore computers," in *Embedded Software and Systems*, pp. 76–83, Springer, 2009.

[25] E. Gilman, H. Khaleghzadeh, K. Doshi, A. Dengel, B. Juurlink, N. Jain, and R. Ahmed, "Demystifying the placement policies of the NVIDIA GPU thread block scheduler for concurrent kernels," *ACM SIGMETRICS Performance Evaluation Review*, vol. 48, pp. 65–67, December 2020.

[26] T. Lühnen, S. Nabi, and A. Koch, "Benchmarking thread block cluster," in *Proceedings of the 2024 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, CSREA Press, 2024. Available at: https://tore.tuhh.de/entities/publication/0d7a5dbc-68f4-4fac-919f-154fc8409ca1.