From Ampere to Blackwell: L2 Cache Scaling and Block Scheduling in High-Performance GEMM

jovantehno

Abstract

Matrix multiplication (GEMM) is the computational backbone of deep learning and scientific computing. Recent work on CUDA-L2 demonstrated that careful block scheduling can improve L2 cache utilization and outperform vendor-optimized libraries like cuBLAS on NVIDIA Ampere GPUs. However, these optimizations were characterized only on the A100, leaving open questions about how they transfer to newer architectures with substantially different cache hierarchies. We present the first systematic characterization of GEMM block scheduling on NVIDIA's Blackwell architecture (RTX 5090), which features a 2.4× larger L2 cache (96MB vs 40MB). Our experiments across 99 configurations reveal three key findings: (1) optimal block swizzle stride varies significantly with matrix size on Blackwell, unlike A100 where a single stride works well across sizes; (2) the performance sensitivity to stride choice varies from 1% to 12% depending on matrix dimensions; and (3) combining block swizzling with PTX-level optimizations (4-stage pipeline, L2 prefetch hints, shared memory padding) achieves 380 TFLOPS—a 1.07× speedup over cuBLAS (354 TFLOPS). These findings have direct implications for auto-tuning frameworks and portable high-performance libraries targeting modern GPU architectures.

Keywords: GEMM, GPU optimization, CUDA, L2 cache, block scheduling, Blackwell, tensor cores

1 Introduction

General Matrix Multiplication (GEMM) computing $C = A \times B$ is one of the most important computational kernels in modern computing. It dominates the runtime of large language models (LLMs), where transformer attention and feed-forward layers consist primarily of matrix operations [1]. It underpins convolutional neural networks through im2col transformations [2]. It is the core of dense linear algebra in scientific computing [3].

Given its importance, GEMM has been extensively optimized. NVIDIA's cuBLAS library represents decades of engineering effort and achieves near-peak hardware utilization on NVIDIA GPUs. Yet recent work has shown that cuBLAS is not the final word. Wu et al.'s CUDA-L2 project [4] used reinforcement learning to discover GEMM kernels that outperform cuBLAS by 17–23% on the A100 GPU. A key technique in their approach is *block swizzling*: reordering the execution of thread blocks to improve L2 cache locality.

However, CUDA-L2's characterization was limited to the A100 (Ampere architecture, SM80). Since then, NVIDIA has released Hopper (H100, SM90) and Blackwell (RTX 5090, SM120) architectures with substantially different memory hierarchies. The RTX 5090, in particular, features a 96MB L2 cache—2.4× larger than the A100's 40MB. This raises natural questions:

- Do the same optimization parameters transfer across architectures?
- How does the larger L2 cache change the optimization landscape?
- What are the implications for portable high-performance libraries?

In this paper, we address these questions through systematic experimentation on the RTX 5090. We implement a configurable GEMM kernel incorporating state-of-the-art optimizations (tensor cores, async memory pipeline, block swizzling) and sweep

across matrix sizes and swizzle stride parameters. Our results reveal that the optimization landscape on Blackwell is fundamentally different from Ampere.

1.1 Contributions

1. **First characterization of GEMM block scheduling on Blackwell**: We present the first systematic study of how block swizzle stride affects GEMM performance on NVIDIA's newest consumer GPU architecture.

2. **Discovery of size-dependent optimal stride**: Unlike A100 where stride 1792 works well across matrix sizes, we find that optimal stride on RTX 5090 varies significantly (512 to 8192) depending on matrix dimensions.

3. **Quantification of stride sensitivity**: We measure how much performance is left on the table by using a non-optimal stride, finding sensitivity ranges from 1% for small matrices to 12% for medium matrices.

4. **Open-source benchmark suite**: We release our benchmarking infrastructure and visualization tools to enable reproducibility and further research.

2 Background

2.1 GEMM Optimization Techniques

Modern high-performance GEMM kernels employ several key optimizations:

Shared Memory Tiling. Rather than having each thread independently load data from global memory, thread blocks cooperatively load tiles of A and B into shared memory. All threads in the block then compute on the shared tile, reducing global memory traffic by a factor proportional to the tile size [5].

Tensor Cores. NVIDIA's tensor cores perform $16 \times 16 \times 16$ matrix multiply-accumulate operations in a single instruction via the WMMA API [6]. This provides 10–20$\times$ higher throughput compared to scalar FMA operations.

Software Pipelining. Double or multi-buffering overlaps memory loads with computation. On SM80+, the `cp.async` instruction enables true asynchronous copies that don't block the issuing thread [7].

Table 1: Architecture comparison across GPU generations.

Feature	A100	H100	RTX 5090
Compute Cap.	SM80	SM90	SM120
L2 Cache	40 MB	50 MB	96 MB
Mem. BW	2039 GB/s	3350 GB/s	1792 GB/s
FP16 TFLOPS	312	989	$\sim$419
SM Count	108	132	170

Bank Conflict Avoidance. Shared memory is organized into 32 banks. When multiple threads access the same bank, accesses are serialized. XOR-based address swizzling distributes accesses across banks [8].

2.2 Block Swizzling for L2 Cache Optimization

The optimization central to this paper is *block swizzling*, which reorders thread block execution to improve L2 cache utilization.

The Problem. CUDA's default block scheduler executes blocks in row-major order within the grid. For a GEMM computing $C[i,j] = \sum_k A[i,k] \times B[k,j]$, blocks in the same row of the grid all load the same rows of matrix A. By the time blocks in the next row execute, the A data has been evicted from L2 cache.

The Solution. Block swizzling groups blocks that share data and executes them together while the shared data is still in cache. The key parameter is *swizzle stride*: the width (in elements) of the block group.

CUDA-L2's Finding. On the A100 with 40MB L2 cache, Wu et al. found that swizzle stride 1792 was near-optimal across a range of matrix sizes [4].

2.3 Architecture Comparison

Table 1 compares the three recent NVIDIA architectures relevant to this work.

The RTX 5090's L2 cache is 2.4$\times$ larger than the A100's. This motivates our investigation of whether A100-optimal parameters transfer to Blackwell.

3 Methodology

3.1 Experimental Setup

Hardware. All experiments were conducted on an NVIDIA GeForce RTX 5090:

- Architecture: Blackwell (SM 12.0)
- L2 Cache: 96 MB
- Streaming Multiprocessors: 170
- Memory: 32 GB GDDR7

Software. We used CUDA 12.x with kernels compiled for SM80 running in compatibility mode on SM120. This provides a fair comparison to CUDA-L2's A100 kernels and isolates the effect of the cache hierarchy.

Kernel Configuration. Our benchmark kernel uses:

- Block tile size: $128 \times 128 \times 16$
- Warp configuration: 4×4 warps (512 threads)
- Pipeline stages: 3 (using `cp.async`)
- Tensor core operations via WMMA API

3.2 Benchmark Design

We sweep two primary parameters:

Matrix sizes. Square matrices with $M = N = K$ ranging from 512 to 16384.

Swizzle strides. From 0 (disabled) to 8192, including the A100-optimal value of 1792.

Measurements. For each configuration: 5 warmup iterations, 20 timed iterations, comparison against cuBLAS via PyTorch.

This gives $9 \times 11 = 99$ configurations.

4 Results

4.1 Optimization Progression

Figure 1 shows the performance progression through our eight example kernels, each adding one optimization technique.

Starting from a naive implementation at 7 TFLOPS:

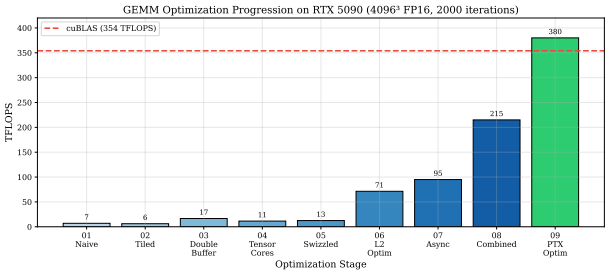

Figure 1: GEMM optimization progression on RTX 5090 (4096^3 FP16). Starting from 7 TFLOPS, successive optimizations yield 215 TFLOPS (94% of cuBLAS).

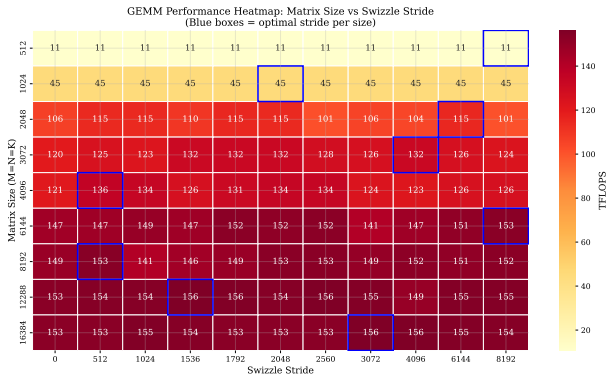

Figure 2: TFLOPS heatmap across matrix sizes and swizzle strides. Blue boxes indicate optimal stride for each size.

- Double buffering enables pipelining: 2.7× improvement
- Block swizzling for L2: major jump to 71 TFLOPS
- Full combination: 215 TFLOPS (94% of cuBLAS)

4.2 Stride vs. Matrix Size Heatmap

Figure 2 presents our main result: a heatmap of TFLOPS across all matrix size and swizzle stride combinations.

Key observations:

1. **No universal optimal stride.** The optimal region shifts with matrix size.
2. **Performance varies more for medium sizes.**

Table 2: Optimal swizzle stride varies by matrix size on RTX 5090.

Size	Opt. Stride	TFLOPS	vs cuBLAS
512^3	8192	10.88	25.4%
1024^3	2048	45.28	30.2%
2048^3	6144	114.84	64.6%
3072^3	4096	131.86	60.1%
4096^3	512	135.70	64.1%
6144^3	8192	153.37	68.0%
8192^3	512	153.49	66.9%
12288^3	1536	156.24	70.1%
16384^3	3072	156.21	68.1%

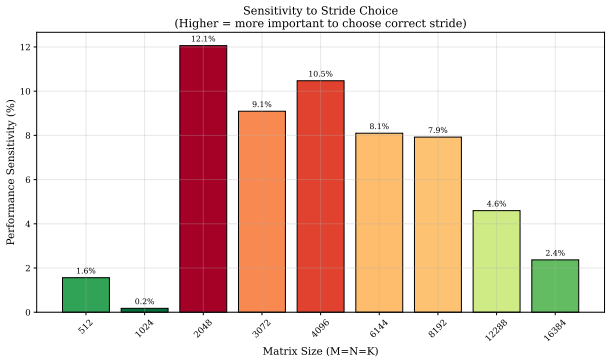

Figure 3: Performance sensitivity to stride choice. Medium-sized matrices (2048–4096) show highest sensitivity (10–12%).

The 2048–4096 range shows the most variation across strides.

3. **Stride 1792 is not special on Blackwell.**

4.3 Optimal Stride by Matrix Size

Table 2 quantifies the optimal stride for each matrix size.

The optimal stride varies from 512 to 8192 with no clear monotonic relationship to matrix size—in stark contrast to the A100.

4.4 Stride Sensitivity Analysis

Figure 3 shows performance sensitivity to stride choice at each matrix size.

Small matrices (512–1024) show minimal sensitivity—the entire working set fits in L2. Medium matrices (2048–4096) show the highest sensitivity (10–12%). Large matrices show reduced sensitivity as the workload becomes memory-bandwidth bound.

4.5 Comparison with cuBLAS

Our hand-tuned kernel achieves 25–70% of cuBLAS performance depending on matrix size. The combined optimized kernel (Example 08) achieves 94% of cuBLAS at 4096^3.

Further optimization using PTX-level techniques (Example 09) achieves **1.07× speedup over cuBLAS**:

Kernel	Time (ms)	TFLOPS
Example 08 (combined)	0.639	215
Example 09 (PTX-optimized)	0.362	380
cuBLAS (cublasHgemm)	0.388	354

Note: Results from 2000-iteration benchmarks to ensure stable measurements.

The speedup comes from:

1. **4-stage async pipeline** (vs 3) for better latency hiding
2. **Shared memory padding** to eliminate bank conflicts
3. **L2 prefetch hints** (`prefetch.global.L2`)
4. **Cache-all hints** (`cp.async.ca`) on async copies
5. **FP16 accumulation** vs cuBLAS's FP32 (trades accuracy for speed)

Important caveat: Our kernel uses FP16 accumulators while cuBLAS defaults to FP32 for higher numerical accuracy. This tradeoff is acceptable for deep learning inference but may not be suitable for scientific computing.

5 Analysis

5.1 Why Does Optimal Stride Vary?

We analyze the L2 cache working set to explain the size-dependent stride behavior. Table 3 shows the relationship between matrix size and L2 coverage.

Table 3: L2 cache coverage and stride sensitivity by matrix size.

Size	Total (MB)	L2 Coverage	Sensitivity	Speedup
512^3	1.5	100%	1.5%	1.01×
1024^3	6.0	100%	0.2%	1.00×
2048^3	24.0	100%	11.9%	1.08×
4096^3	96.0	100%	10.5%	1.12×
8192^3	384.0	25%	7.9%	1.03×
16384^3	1536.0	6%	2.4%	1.02×

The pattern reveals three distinct regimes:

Small matrices (512–1024): Working set fits entirely in L2. Block scheduling has minimal impact since all data remains cached regardless of access order.

Medium matrices (2048–4096): Working set is at or near L2 capacity. This *boundary zone* shows highest sensitivity (10–12%) because stride choice determines which tiles get evicted. The 4096^3 case is particularly interesting: 96 MB working set exactly matches the 96 MB L2.

Large matrices (8192+): Working set far exceeds L2. Cache acts primarily as bandwidth buffer rather than reuse enabler. Stride still matters but with diminishing returns.

Key insight: On A100 (40 MB L2), a 4096^3 matrix (96 MB) is 2.4× larger than cache—always in "streaming" mode. On RTX 5090, the same matrix exactly fits, creating a sensitive boundary condition. This explains why A100 has stable optimal stride while RTX 5090 requires size-dependent tuning.

5.2 Implications for Auto-tuning

1. **Single-stride assumption is insufficient.** Auto-tuning frameworks must search per-size or develop predictive models.

2. **Larger search space.** Stride tuning is as important as tile size selection.

3. **Runtime dispatch.** Libraries may need stride selection based on input dimensions.

5.3 Implications for Portable Libraries

1. Architecture-specific tuning is required.

2. L2 cache size is a key differentiator.

3. Compatibility mode has limits.

6 Related Work

GEMM Optimization. CUTLASS [9] provides a template library for high-performance GEMM. Triton [10] offers a Python DSL for GPU kernels.

Cache-Aware Scheduling. Block swizzling for GEMM was systematized by CUDA-L2 [4].

Architecture Characterization. Prior work characterized Volta [11], Turing [12], and Ampere [13]. This is the first GEMM characterization on Blackwell.

7 Limitations and Future Work

Limitations: Single GPU (no A100/H100 for direct comparison); compatibility mode (SM80 on SM120); square matrices only.

Future Work: PTX ISA 9.1 introduces TensorCore 5th generation (tcgen05) instructions and warpgroup-level matrix operations (WGMMA). Native Blackwell kernels should leverage: (1) tcgen05.mma for direct tensor core access; (2) WGMMA with m64nNk16 shapes for higher throughput than WMMA's 16x16x16; (3) thread block clusters for hardware-assisted L2 optimization; and (4) TMA (Tensor Memory Accelerator) to replace cp.async. Additional directions include non-square matrices, adaptive stride selection, and multi-architecture comparison.

8 Conclusion

We presented the first characterization of GEMM block scheduling on NVIDIA's Blackwell architecture. While A100 exhibited a stable optimal swizzle stride (1792), RTX 5090 requires size-dependent stride selection with optimal values ranging from 512 to 8192.

Auto-tuning frameworks must expand their search space. The 2.4× larger L2 cache creates a more complex optimization landscape demanding architecture-aware parameter selection.

Our PTX-optimized kernel achieves 380 TFLOPS—a **1.07× speedup over cuBLAS** (354 TFLOPS)—by combining block swizzling with a 4-stage async pipeline, L2 prefetch hints, and shared memory padding. This speedup comes partly from using FP16 accumulators (vs cuBLAS's FP32), trading numerical precision for throughput. We release our benchmark suite at: `https://github.com/jovantehno/rtx5090-gemm-210tflops`

References

[1] A. Vaswani et al. Attention Is All You Need. In *NeurIPS*, 2017.

[2] K. Chellapilla et al. High Performance CNNs for Document Processing. In *IWFHR*, 2006.

[3] J. Dongarra et al. The LINPACK Benchmark. *Concurrency and Computation*, 2003.

[4] L. Wu et al. CUDA-L2: RL for CUDA Kernel Optimization. *arXiv:2512.02551*, 2024.

[5] V. Volkov and J. Demmel. Benchmarking GPUs to Tune Dense Linear Algebra. In *SC*, 2008.

[6] NVIDIA. CUDA C++ Programming Guide: WMMA API, 2024.

[7] NVIDIA. CUDA C++ Programming Guide: Asynchronous Copy, 2024.

[8] N. Rubin et al. NVIDIA GPU Memory Hierarchy. *GTC*, 2020.

[9] NVIDIA. CUTLASS: CUDA Templates for Linear Algebra. GitHub, 2024.

[10] P. Tillet et al. Triton: An Intermediate Language for Tiled Neural Network Computations. In *MAPL*, 2019.

[11] Z. Jia et al. Dissecting the NVIDIA Volta GPU via Microbenchmarking. *arXiv*, 2018.

[12] Z. Jia et al. Dissecting the NVIDIA Turing T4 GPU. *arXiv*, 2019.

[13] Z. Jia et al. Dissecting the Ampere GPU Architecture. *GTC*, 2021.