Alejandro Lucena
2/6/2026

Experiment 1: GEMM Kernels

Introduction and Methodology

This report evaluates the performance of three CUDA-based General Matrix Multiplication (GEMM) implementations—Naive, Tiled, and cuBLAS—across varying matrix dimensions and thread block size configurations. By comparing custom kernels against the built-in cuBLAS library, this report analyzes the impact of memory hierarchy on GPU throughput.

First, the platform details. All kernels were executed on an NVIDIA V100 GPU using CUDA version 12.4.1. Because GPU performance is highly dependent on hardware specifications, the GLOPS/S subsequently reported are specific to this GPU environment.

Next, a brief implementation overview of the GEMM kernels. In order, these kernels represent a progression in memory optimization. In the Naive kernel, threads perform direct global memory access. In this approach, every multiplication requires two computationally expense loads from global memory. Because threads within a block require overlapping data at different times, this O(N^3) algorithm creates a significant redundant memory traffic bottleneck, particularly as N increases. The tiled GEMM kernel is the bridge that permits the exploitation of the L1 Shared Memory cache. By loading TILE_SIZE x TILE_SIZE chunks into shared memory, every thread in the block can reuse the data TILE_SIZE times. This tiling effectively reduces global memory access by a factor of TILE_SIZE. Finally, we have cuBLAS, which is a built-in GEMM implementation. As a highly-optimized “black-box” library, cuBLAS serves as the performance baseline. It makes use of low-level hardware optimizations that establish the theoretical performance ceiling for the hardware.

Finally, the performance metric. To qualify and compare throughput, this report utilizes GFLOPS as its sole performance metric. For a standard GEMM operation, the operation count is 2*M*N*K. Performance is thus calculated as: GFLOPS= 2*M*N*K / Time (s) * 10^9.

Performance Comparison: Custom GEMM Kernels vs. cuBLAS

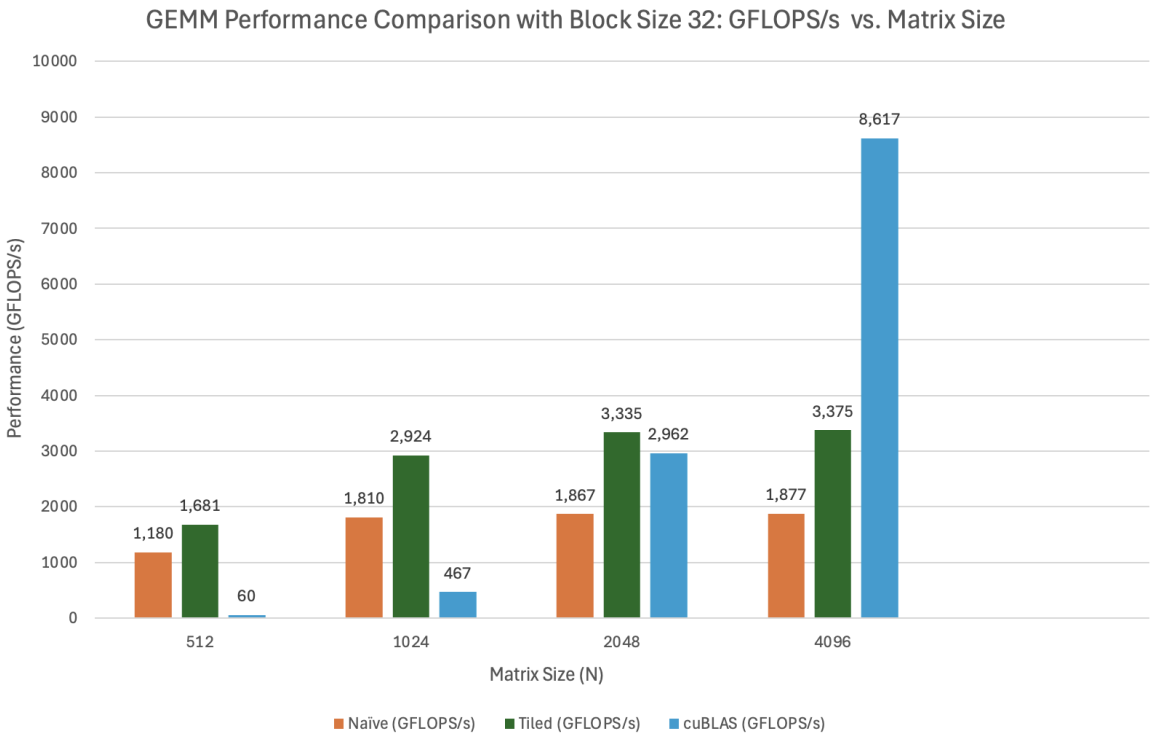

Figure 1

As illustrated in Figure 1, the Naive kernel consistently demonstrated the lowest throughout across all tested matrix sizes. While its performance scaled initially, it reached a plateau at approximately 1877 GFLOPS/s for N=4096. This plateauing effect, which becomes evident at N=1024 (1810 GFLOPS/s), indicates that the kernel is latency constrained; the lack of data reuse forces the execution units to remain idle while waiting for high-latency global memory transactions.

The Tiled kernel demonstrated a superior scaling trajectory, though it also encountered a performance ceiling near N=2048. Notably, at the N=2048 mark, the Tiled implementation outperformed the cuBLAS baseline, achieving 3335 GFLOPS/s compared to 2962 GFLOPS/s—a 12.6% performance advantage. This advantage over the optimized cuBLAS library can be attributed to the specific overhead associated with cuBLAS; for small-to-medium matrices, the latency of library initialization and kernel selection heuristics often outweighs the computational gains.

Not surprisingly, cuBLAS dominated at N=4096, reaching a peak performance of 8617 GFLOPS/s. This is roughly 2.5× higher than the custom Tiled kernel. This divergence likely stems from cuBLAS's ability to leverage specialized hardware

features—such as Tensor Cores or assembly-level optimizations, both of which provide significantly higher arithmetic throughput than standard C++ CUDA kernels can achieve for large matrices.

Impact of Tiling Sizes and Compute Limits

To evaluate the impact of tiling on shared memory efficiency, the custom kernel was tested with N=2048 and Block Sizes (BS) of 8, 16, and 32. The results indicate a direct correlation between tile size and throughput: as the block dimension doubles, performance increases accordingly. This scaling is driven by the block size itself; a 32 x 32 tile allows each element loaded into shared memory to be reused 32 times, drastically reducing the pressure on global memory access compared to a 16 x 16 or 8 x 8 configuration.

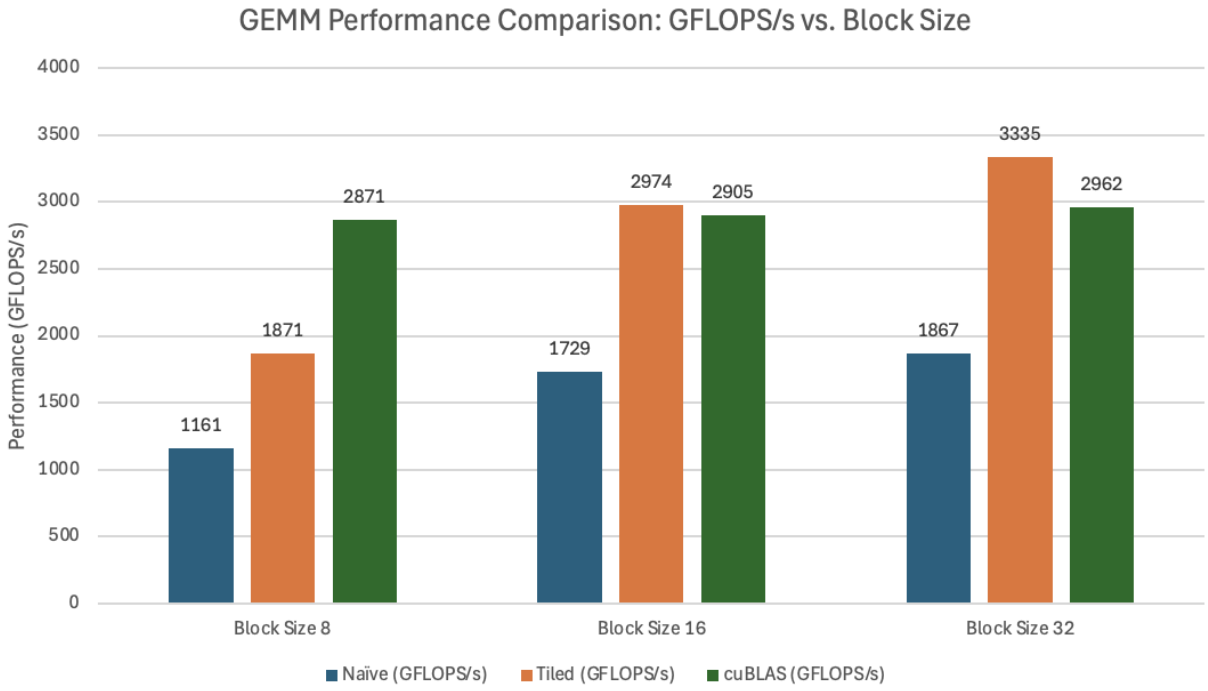

Figure 2

The performance peak was observed at BS=32, which provided the most significant speedup and enabled the custom Tiled kernel to surpass cuBLAS at the 2048 matrix size. This specific configuration represents an optimal balance between high data reuse and GPU occupancy (the number of active blocks at a given time), as it utilizes the maximum number of threads (1024) on this NVIDIA V100 GPU.

While increasing tile size improves data reuse, it is strictly governed by the physical constraints of the GPU architecture. During testing, a Block Size of 64 was attempted. However, this resulted in immediate kernel launch failures because a 64 x 64 block requires 4096 threads, far exceeding the hardware limit of 1024 threads per block in this GPU architecture.

Conclusion and SpeedUp Summary

The results of this report demonstrate the important role of the memory hierarchy in accelerating GEMM kernel workloads. By implementing a shared memory tiling strategy, the global memory bottleneck seen in the Naive kernel implementation can be mitiageted, resulting in substantial throughput gains across all measured matrix dimensions.

The Tiled implementation (using the optimal BS=32) consistently outperformed the Naive kernel, with the performance gap widening as the matrix size increased. At the N=2048 mark, the Tiled kernel achieved a peak speedup of 1.79x over the Naive kernel. Furthermore, the custom Tiled kernel demonstrated a significant advantage over the built-in cuBLAS library for medium-sized matrices, yielding a 1.12x speedup at N=2048.

The experimental data confirms that the Naive kernel suffers from high-latency global memory access costs, with the Tiled kernel then shifting the bottleneck toward the compute limits of the GPU, thereby showing that optimization is strictly bounded by the physical resources of the GPU architecture. The failure of the BS=64 configuration highlights the fact that kernel design must respect hardware limits regarding threads per block and shared memory capacity to maintain high occupancy.

Nevertheless, for large-scale workloads (N >= 4096), cuBLAS remains the implementation of choice due to due its specialized hardware optimizations. However, for specific matrix dimensions where library overhead is non-trivial, a custom Tiled kernel can provide a superior alternative.

Raw Timing Logs

Block Size 8:

```
[ajl18@bc11u23n2 scripts]$ ./measure.sh
Running cublas N=512
Impl=cublas M=512 N=512 K=512 Time(ms)=4.58 GFLOP/s=58.66
Running tiled N=512
Impl=tiled M=512 N=512 K=512 Time(ms)=0.20 GFLOP/s=1318.34
Running naive N=512
Impl=naive M=512 N=512 K=512 Time(ms)=0.29 GFLOP/s=937.80
Running cublas N=1024
Impl=cublas M=1024 N=1024 K=1024 Time(ms)=4.72 GFLOP/s=454.96
Running tiled N=1024
Impl=tiled M=1024 N=1024 K=1024 Time(ms)=1.20 GFLOP/s=1791.72
Running naive N=1024
Impl=naive M=1024 N=1024 K=1024 Time(ms)=1.78 GFLOP/s=1207.62
Running cublas N=2048
Impl=cublas M=2048 N=2048 K=2048 Time(ms)=5.98 GFLOP/s=2870.71
Running tiled N=2048
Impl=tiled M=2048 N=2048 K=2048 Time(ms)=9.18 GFLOP/s=1870.59
Running naive N=2048
Impl=naive M=2048 N=2048 K=2048 Time(ms)=14.80 GFLOP/s=1161.10
Running cublas N=4096
Impl=cublas M=4096 N=4096 K=4096 Time(ms)=16.01 GFLOP/s=8586.26
Running tiled N=4096
Impl=tiled M=4096 N=4096 K=4096 Time(ms)=82.14 GFLOP/s=1673.23
Running naive N=4096
Impl=naive M=4096 N=4096 K=4096 Time(ms)=145.41 GFLOP/s=945.16
Results stored in ../data/20260205_140803_sweep.csv
[ajl18@bc11u23n2 scripts]$
```

Block Size 16:

```
[ajl18@bc11u21n2 scripts]$ ./measure.sh
Running cublas N=512
Impl=cublas M=512 N=512 K=512 Time(ms)=4.58 GFLOP/s=58.64
Running tiled N=512
Impl=tiled M=512 N=512 K=512 Time(ms)=0.14 GFLOP/s=1872.04
Running naive N=512
Impl=naive M=512 N=512 K=512 Time(ms)=0.22 GFLOP/s=1201.98
Running cublas N=1024
Impl=cublas M=1024 N=1024 K=1024 Time(ms)=4.68 GFLOP/s=458.80
Running tiled N=1024
Impl=tiled M=1024 N=1024 K=1024 Time(ms)=0.79 GFLOP/s=2734.90
Running naive N=1024
Impl=naive M=1024 N=1024 K=1024 Time(ms)=1.27 GFLOP/s=1691.98
Running cublas N=2048
Impl=cublas M=2048 N=2048 K=2048 Time(ms)=5.91 GFLOP/s=2905.08
Running tiled N=2048
Impl=tiled M=2048 N=2048 K=2048 Time(ms)=5.78 GFLOP/s=2973.98
Running naive N=2048
Impl=naive M=2048 N=2048 K=2048 Time(ms)=9.94 GFLOP/s=1728.82
Running cublas N=4096
Impl=cublas M=4096 N=4096 K=4096 Time(ms)=15.96 GFLOP/s=8610.01
Running tiled N=4096
Impl=tiled M=4096 N=4096 K=4096 Time(ms)=44.97 GFLOP/s=3056.27
Running naive N=4096
Impl=naive M=4096 N=4096 K=4096 Time(ms)=79.85 GFLOP/s=1721.27
Results stored in ../data/20260205_000619_sweep.csv
[ajl18@bc11u21n2 scripts]$
```

Block Size 32:

```
[ajl18@bc11u21n2 scripts]$ ./measure.sh
Running cublas N=512
Impl=cublas M=512 N=512 K=512 Time(ms)=4.45 GFLOP/s=60.38
Running tiled N=512
Impl=tiled M=512 N=512 K=512 Time(ms)=0.16 GFLOP/s=1681.42
Running naive N=512
Impl=naive M=512 N=512 K=512 Time(ms)=0.23 GFLOP/s=1179.50
Running cublas N=1024
Impl=cublas M=1024 N=1024 K=1024 Time(ms)=4.60 GFLOP/s=467.04
Running tiled N=1024
Impl=tiled M=1024 N=1024 K=1024 Time(ms)=0.73 GFLOP/s=2924.26
Running naive N=1024
Impl=naive M=1024 N=1024 K=1024 Time(ms)=1.19 GFLOP/s=1810.38
Running cublas N=2048
Impl=cublas M=2048 N=2048 K=2048 Time(ms)=5.80 GFLOP/s=2962.19
Running tiled N=2048
Impl=tiled M=2048 N=2048 K=2048 Time(ms)=5.15 GFLOP/s=3335.14
Running naive N=2048
Impl=naive M=2048 N=2048 K=2048 Time(ms)=9.20 GFLOP/s=1867.49
Running cublas N=4096
Impl=cublas M=4096 N=4096 K=4096 Time(ms)=15.95 GFLOP/s=8616.89
Running tiled N=4096
Impl=tiled M=4096 N=4096 K=4096 Time(ms)=40.73 GFLOP/s=3374.61
Running naive N=4096
Impl=naive M=4096 N=4096 K=4096 Time(ms)=73.21 GFLOP/s=1877.35
Results stored in ../data/20260205_001002_sweep.csv
[ajl18@bc11u21n2 scripts]$
```

Block Size 64:

```
[ajl18@bc11u21n2 scripts]$ ./measure.sh
Running cublas N=512
Impl=cublas M=512 N=512 K=512 Time(ms)=4.47 GFLOP/s=60.03
Running tiled N=512
Impl=tiled M=512 N=512 K=512 Time(ms)=0.04 GFLOP/s=6255.49
Running naive N=512
Impl=naive M=512 N=512 K=512 Time(ms)=0.04 GFLOP/s=6437.92
Running cublas N=1024
Impl=cublas M=1024 N=1024 K=1024 Time(ms)=4.77 GFLOP/s=450.48
Running tiled N=1024
Impl=tiled M=1024 N=1024 K=1024 Time(ms)=0.03 GFLOP/s=62543.21
Running naive N=1024
Impl=naive M=1024 N=1024 K=1024 Time(ms)=0.04 GFLOP/s=51861.57
Running cublas N=2048
Impl=cublas M=2048 N=2048 K=2048 Time(ms)=5.90 GFLOP/s=2911.08
Running tiled N=2048
Impl=tiled M=2048 N=2048 K=2048 Time(ms)=0.04 GFLOP/s=457300.62
Running naive N=2048
Impl=naive M=2048 N=2048 K=2048 Time(ms)=0.04 GFLOP/s=445536.03
Running cublas N=4096
Impl=cublas M=4096 N=4096 K=4096 Time(ms)=15.84 GFLOP/s=8675.21
Running tiled N=4096
Impl=tiled M=4096 N=4096 K=4096 Time(ms)=0.05 GFLOP/s=2628498.98
Running naive N=4096
Impl=naive M=4096 N=4096 K=4096 Time(ms)=0.05 GFLOP/s=2499980.90
Results stored in ../data/20260205_000811_sweep.csv
[ajl18@bc11u21n2 scripts]$
```