System-Level Optimization of Large-Scale Matrix Multiplication on GPUs

Nguyen Duong

January 19, 2026

1 Introduction

Matrix multiplication is a core operation in large-scale data analysis and machine learning workloads. When matrix sizes exceed device memory capacity, overall performance is often dominated not by compute throughput but by host–device data movement and memory management.

This project focuses on engineering efficient end-to-end matrix multiplication pipelines on NVIDIA GPU systems using cuBLAS and cuBLASLt as the compute backend. Rather than implementing custom GPU kernels, the work explores system-level optimizations including pinned host memory allocation, NUMA-aware memory placement, multi-threaded data staging, asynchronous transfers, double buffering, and multi-GPU execution. The goal is to evaluate how these techniques affect throughput and scalability for very large matrices under different numerical precision settings.

2 System Design and Implementation

2.1 System Architecture Overview

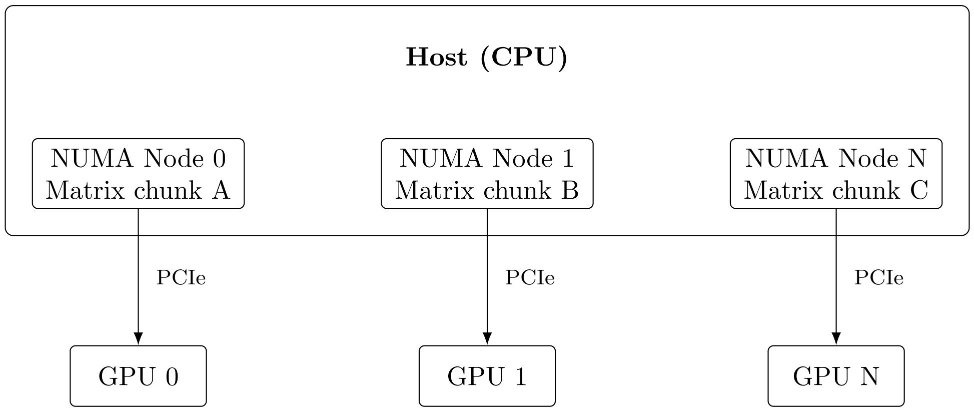

Host (CPU)
NUMA Node 0 Matrix chunk A
NUMA Node 1 Matrix chunk B
NUMA Node N Matrix chunk C
PCIe
PCIe
PCIe
GPU 0
GPU 1
GPU N

Figure 1: High-level system architecture showing NUMA-aware host memory and multi-GPU execution.

2.2 Memory management

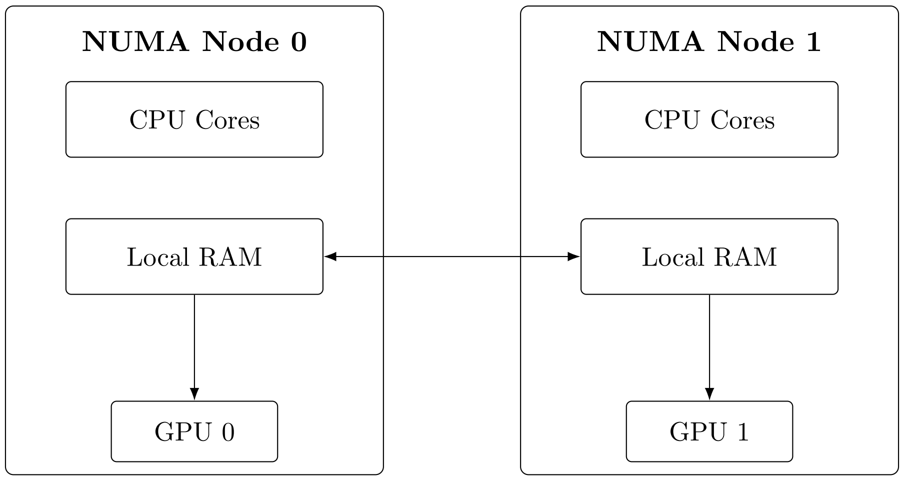


Figure 2: NUMA topology with per-node CPU cores, local RAM, and attached GPUs. Bidirectional link indicates cross-NUMA memory access/interconnect.

On NUMA systems, memory access latency and bandwidth depend on the physical location of memory relative to CPU cores and attached GPUs. Each NUMA node contains a disjoint set of CPU cores and node-local system memory, as shown in Figure 2.

To reduce cross-NUMA traffic and improve host-to-device transfer efficiency, large input matrices are allocated in pinned host memory on the NUMA node closest to the target GPU. Memory allocation and initialization are performed by CPU threads bound to the corresponding NUMA node to ensure correct page placement.

2.3 Double buffering

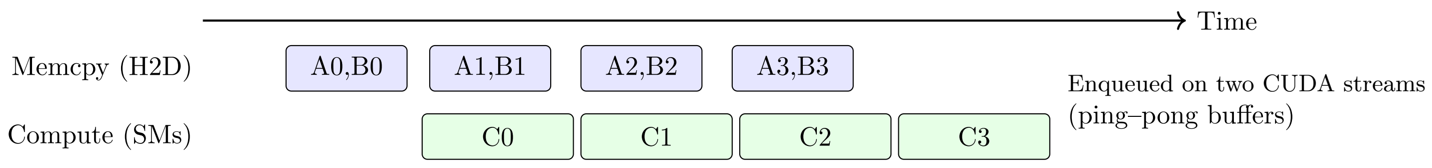
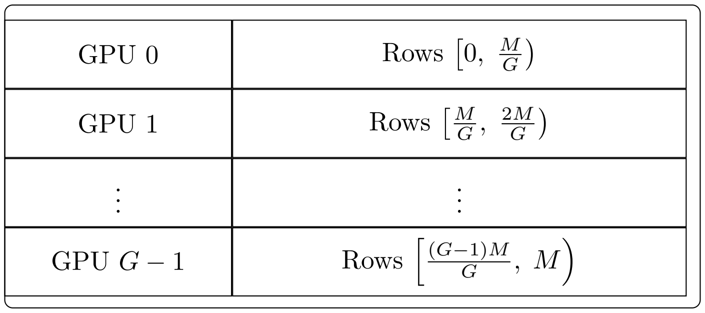


Figure 3: Double-buffered execution showing overlapped host-to-device transfers and compute.

Figure 3 illustrates the double-buffered execution scheme used to overlap host-to-device (H2D) transfers with GPU computation. Input data is streamed in chunks using two CUDA streams and two device buffers (ping–pong). While cuBLAS/cuBLASLt computes on chunk i in one buffer, the next chunk $i+1$ is transferred asynchronously into the other buffer. Overlap begins from the second iteration because the first chunk must be transferred before any compute can start.

2.4 Multi-GPU strategy

The output matrix C is partitioned by contiguous row blocks across G GPUs, where GPU i computes rows $\left[\frac{iM}{G}, \frac{(i+1)M}{G}\right)$. Each GPU operates independently without inter-GPU communication, yielding balanced workload distribution and natural NUMA alignment.



GPU 0	Rows $\left[0,\ \frac{M}{G}\right)$
GPU 1	Rows $\left[\frac{M}{G},\ \frac{2M}{G}\right)$
$\vdots$	$\vdots$
GPU $G-1$	Rows $\left[\frac{(G-1)M}{G},\ M\right)$

Figure 4: Row-wise work distribution of the output matrix C across G GPUs.

3 Experimental Setup

3.1 Hardware Specifications

Table 1: Theoretical peak dense throughput (manufacturer specifications).

GPU	VRAM (GB)	BW (GB/s)	Boost (MHz)	FP32	BF16	FP64	TF32	Use Case
RTX 4080 Super	16	736	2550	52.0	104.4	0.81	52.2	Consumer
RTX A6000	48	768	1800	38.7	154.8	0.60	77.4	Workstation
NVIDIA L40S	48	864	2520	91.6	362.0	1.43	183.0	Data Center
NVIDIA L40	48	864	1845	90.5	181.0	N/A	90.5	Data Center

Dense theoretical peaks only (no sparsity). FP32, TF32, and BF16 assume Tensor Core execution where applicable. Boost clocks are vendor-reported maxima; sustained clocks under load may be lower due to power and thermal limits.

Table 2: Hardware configurations used in experiments.

Config	CPU Model	Sockets	RAM (GB)	NUMA Nodes	GPUs
Server A	Intel(R) Xeon(R) Gold 5318Y	2	256	2	2× NVIDIA L40S
Server B	Intel(R) Xeon(R) Silver 4516Y+	2	256	2	2× NVIDIA L40
Server C	Intel(R) Xeon(R) w5-3423	1	120	1	1× RTX A6000
Desktop	AMD Ryzen 7 7800X3D	1	16	1	1× RTX 4080 Super

All systems use PCIe Gen4 GPUs. NUMA affinity and pinned memory are explicitly controlled in all multi-socket experiments.

We evaluate performance across three representative platforms: a multi-GPU data-center server, a single-GPU workstation, and a consumer desktop, covering common deployment scenarios.

3.2 Software Environment

All experiments are conducted on Linux systems using NVIDIA GPUs. The software stack consists of CUDA Toolkit 12.x with cuBLAS, and NVIDIA driver version 550.xx. Code is compiled using `gcc` (version 13.x) with `nvcc`.

3.3 Experiment Setup

3.3.1 Chunked Matrix Processing

To accommodate inputs that exceed device memory, we process the input matrix $X \in \mathbb{R}^{M \times N}$ in row-wise chunks. In all experiments, we choose the number of rows per chunk to equal the matrix width, i.e., $K_i = N$, such that each chunk has shape $X_i \in \mathbb{R}^{N \times N}$.

For each chunk X_i, we transfer X_i to the GPU and compute its local contribution $X_i^\top X_i \in \mathbb{R}^{N \times N}$, which is then accumulated into the final result. This blockwise computation is mathematically equivalent to evaluating $X^\top X$, up to floating-point rounding effects.

3.3.2 Numerical precision and compute mode

All reported performance results in Section 4 use BF16 Tensor Core GEMM for computation. Input matrices are stored in FP32 in host memory and are transferred to the GPU in FP32. Before GEMM, each chunk is converted to BF16 on the device. The product is accumulated into the output matrix in FP32 to reduce rounding error during accumulation.

Concretely, for each chunk $X_i \in \mathbb{R}^{N \times N}$, we execute:

$$X_i^{(\mathrm{bf16})} \leftarrow \mathrm{cast}_{\mathrm{fp32} \to \mathrm{bf16}}(X_i), \qquad C \leftarrow C + \left(X_i^{(\mathrm{bf16})}\right)^\top X_i^{(\mathrm{bf16})},$$

where the GEMM uses BF16 inputs with FP32 accumulation.

3.3.3 Experimental Variables

Table 3: Experiment coverage across hardware platforms and execution methods.

Hardware	Chunk Sizes ($N \times N$)	Buffering Modes	Multi-GPU
RTX 4080 Super	8K–16K	Single, Double	–
RTX A6000	8K–20K	Single, Double	–
NVIDIA L40S (1×)	8K–20K	Single, Double	–
NVIDIA L40S (2×)	16K	Double	Enabled
NVIDIA L40 (2×)	16K	Double	Enabled

Each GPU platform is evaluated across multiple chunk sizes and buffering strategies. Unless otherwise stated, results shown correspond to representative configurations that achieve stable overlap between transfer and compute.

Chunk size corresponds to the matrix width N. For each configuration, we choose (M, N) pairs (e.g., 8K × 2M, 12K × 1.5M, 16K × 1M, 20K × 0.9M) such that the total input size $M \cdot N$ remains approximately comparable across chunk widths. Each chunk computes a partial contribution $X_i^\top X_i \in \mathbb{R}^{N \times N}$.

Scaling definitions. In strong-scaling experiments, the total problem size (M, N) is held fixed while the number of GPUs G increases; each GPU computes approximately M/G rows of the output. In weak-scaling experiments, the per-GPU workload is held constant by increasing the total number of rows proportionally with G, i.e., $M \propto G$ with fixed N.

3.4 What We Test

Table 4: Experimental design organized by research question.

#	Research Question	Variables	Fixed
1	How does chunk size affect performance?	Chunk size: 8K $\rightarrow$ 20K	GPU: L40S, A6000; Method: Double buffering
2	What is the speedup from double buffering?	Method: Single vs. Double ; Chunk size: 8K $\rightarrow$ 20K	GPU: L40S, A6000
3	How does performance scale across multiple GPUs?	GPUs: 1 $\rightarrow$ 2 (L40S), 1 $\rightarrow$ 2 (L40)	Chunk size: 16K; Method: Double buffering
4	How do consumer and server GPUs compare?	GPU: all platforms	Chunk size: 8K; Method: Double buffering

3.5 Measurement Metrics

We report three timing metrics throughout this work:

- **Kernel time**: execution time of the GEMM kernel alone, measured using CUDA events.
- **End-to-end wall time (H2D+Kernel)**: elapsed wall-clock time measured from immediately before the compute function is invoked until it returns, capturing the critical path of host-to-device transfer and kernel execution, including any overlap.
- **End-to-end wall time (H2D+Kernel+D2H)**: elapsed wall-clock time measured from function entry until all results are copied back to host memory.

Wall times are not computed as the sum of individual components; they reflect the measured critical path under asynchronous execution and double buffering.

Timing methodology. While CUDA events are used to measure kernel execution time in isolation, they are insufficient for accurately capturing end-to-end performance under asynchronous execution and double buffering. In such cases, host-side wall-clock timers are used to measure elapsed time along the critical path.

3.6 Profiling and Validation

We use NVIDIA Nsight Systems (`nsys`) to profile selected executions and validate overlap between host–device transfers and kernel execution under asynchronous and double-buffered configurations. Profiling is used for validation only; all reported performance metrics are obtained from wall-clock timers and CUDA events as described in Section 3.5.

4 Results

4.1 Effect of Chunk Size

Table 5: Chunk-size effect on L40S without D2H. Efficiency is normalized to the sustained kernel reference of 192.9 TFLOP/s.

Chunk Size (rows)	Wall TFLOP/s	Efficiency (%)	Gap (%)
8,192	98.1	50.9	49.1
12,288	138.4	71.7	28.3
16,384	170.9	88.6	11.4
20,480	174.0	90.2	9.8

Table 5 shows the effect of chunk size on wall-clock throughput for the L40S GPU without device-to-host transfers. Performance increases monotonically with chunk size, improving from 98.1 TFLOP/s at 8K rows to 174.0 TFLOP/s at 20K rows. The corresponding efficiency rises from 50.9% to 90.2%, with diminishing gains beyond 16K rows.

Table 6: Chunk-size effect on A6000 without D2H. Efficiency is normalized to the sustained kernel reference of 116.652 TFLOP/s.

Chunk Size (rows)	Wall TFLOP/s	Efficiency (%)	Gap (%)
8,192	101.8	87.30	12.70
12,288	108.8	93.25	6.75
16,384	111.8	95.84	4.16
20,480	112.9	96.79	3.21

Table 7: Chunk-size effect on A6000 including D2H. Efficiency is normalized to the common no-D2H sustained reference of 116.652 TFLOP/s.

Chunk Size (rows)	Wall TFLOP/s	Efficiency (%)	Gap (%)
8,192	101.6	87.13	12.87
12,288	107.4	92.08	7.92
16,384	110.0	94.31	5.69
20,480	111.6	95.63	4.37

Tables 6 and 7 report the same experiment on the A6000 GPU without and with device-to-host transfers, respectively. In both cases, performance improves steadily with larger chunk sizes, reaching over 95% efficiency at 20K rows. Including device-to-host transfers results in only a minor reduction in throughput, indicating that chunk size effects remain consistent under end-to-end execution.

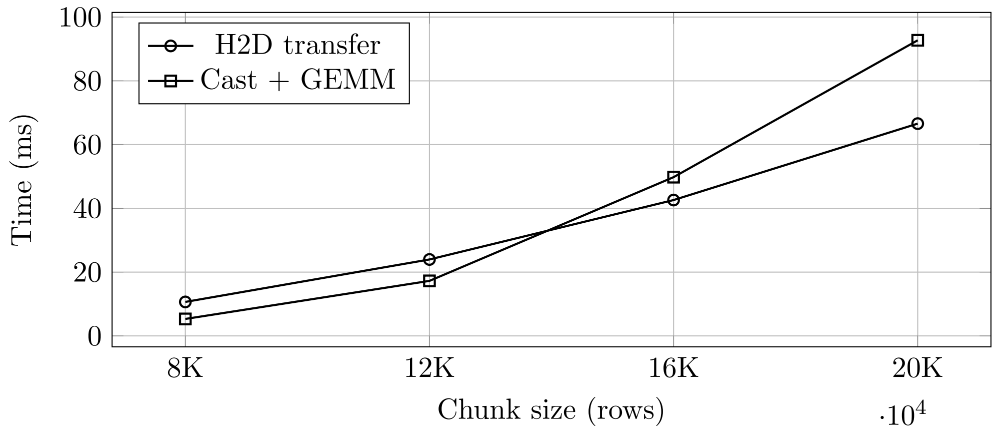


Figure 5: Measured host-to-device transfer time and compute time (cast + GEMM) as a function of chunk size on the L40S GPU.

The measured effective host-to-device bandwidth is approximately 25.2 GB/s, explaining the transfer-dominated regime observed at smaller chunk sizes.

4.2 Impact of double buffering

Table 8: Impact of double buffering on L40S end-to-end throughput without D2H.

Chunk Size (rows)	SB TFLOP/s	DB TFLOP/s	Speedup	Gain (%)
8,192	68.2	98.1	1.44×	43.8
12,288	73.8	138.4	1.88×	87.5
16,384	105.6	170.9	1.62×	61.8
20,480	108.0	174.0	1.61×	61.1

Table 9: Impact of double buffering on A6000 end-to-end throughput without D2H.

Chunk Size (rows)	SB TFLOP/s	DB TFLOP/s	Speedup	Gain (%)
8,192	55.8	101.8	1.82×	82.4
12,288	68.3	108.8	1.59×	59.3
16,384	75.0	111.8	1.49×	49.1
20,480	79.9	112.9	1.41×	41.3

Table 10: Impact of double buffering on A6000 end-to-end throughput including D2H.

Chunk Size (rows)	SB TFLOP/s	DB TFLOP/s	Speedup	Gain (%)
8,192	55.6	101.6	1.83×	82.7
12,288	68.0	107.4	1.58×	57.9
16,384	74.5	110.0	1.48×	47.7
20,480	79.3	111.6	1.41×	40.7

Tables 8–10 report the impact of double buffering on end-to-end throughput across different chunk sizes. Double buffering improves wall-clock performance for all configurations, yielding speedups between 1.4× and 1.9× depending on chunk size and GPU.

The relative speedup is largest at smaller chunk sizes, where single buffering suffers from lower utilization. However, absolute throughput under both buffering strategies increases with chunk size and saturates at larger chunks.

4.3 Multi-GPU Scaling

Table 11: Strong scaling across identical L40S GPUs under a fixed total workload. Speedup S_N and parallel efficiency E_N are computed from wall-clock time.

GPUs	Wall TFLOP/s	Speedup S_N	Efficiency E_N	Slowest-GPU kernel TFLOP/s
1	177.3	1.00	1.00	202.7
2	348.7	1.97	0.983	201.5

Table 12: Weak scaling across identical L40 GPUs with proportional workload per GPU. Speedup S_N and parallel efficiency E_N are computed from wall-clock time. Kernel throughput reports the slowest GPU.

GPUs	Wall TFLOP/s	Speedup S_N	Efficiency E_N	Slowest-GPU kernel TFLOP/s
1	89.4	1.00	1.00	93.3
2	176.0	1.97	0.984	93.1

Slowest-GPU kernel TFLOP/s is computed as slowest GPUs after taking the median over repeated runs.

4.4 Consumer vs. Server GPU Comparison

This section compares consumer- and server-class GPUs under the same matrix multiplication methodology. Server-class GPUs (L40S and A6000) consistently achieve higher and more stable end-to-end throughput and higher effective performance per watt under sustained workloads, reflecting their power delivery, cooling, and clock stability characteristics.

A consumer GPU reference (RTX 4080 Super) is additionally reported for context. On a smaller dense workload constrained by host memory, the RTX 4080 Super achieved approximately

108 TFLOP/s in BF16 and 54 TFLOP/s in TF32 when measured at the kernel level. These values slightly exceed nominal published dense Tensor-core specification figures, which is attributed to sustained boost clock behavior under favorable thermal conditions. Because this result was obtained on a reduced problem size and is not directly comparable to server GPU measurements, it is reported for qualitative context only and is not used for quantitative comparison.

5 Discussion

This section discusses the observed performance trends reported in Section 4 and provides qualitative explanations ...

5.1 Chunk size and overlap behavior

The chunk-size experiments show a transition from a transfer-bound to a compute-bound regime as chunk size increases. For small chunks, host-to-device transfer dominates execution time and limits throughput. As chunk size grows, computation increasingly overlaps with data transfer under double buffering, enabling higher sustained performance.

Beyond this transition, increasing chunk size yields diminishing returns. Although larger chunks increase arithmetic intensity, they also increase memory traffic and kernel execution time without proportional gains in effective FLOP throughput. While slightly higher peak performance is occasionally observed at larger chunk sizes (e.g., 20K rows), these gains are not consistently reproducible. In practice, a chunk size of approximately 16K rows provides a stable sweet spot, achieving near-peak performance while avoiding sensitivity to system-level variability.

Kernel throughput and sustained performance. Kernel-only measurements show that smaller chunk sizes can achieve higher instantaneous FLOP rates than larger chunks. This behavior does not arise from reduced utilization at large chunk sizes. Instead, larger chunks drive sustained SM and Tensor Core utilization and push the GPU toward its power and thermal limits. Under these conditions, power and thermal constraints limit the achievable operating frequency, resulting in slightly lower kernel-only FLOP rates despite higher utilization.

In contrast, smaller chunk sizes operate under less restrictive power conditions, allowing higher instantaneous operating frequencies and higher kernel-only FLOP rates. These measurements reflect transient boost behavior rather than sustained compute performance. For this reason, the single-buffered kernel throughput measured at the largest chunk size (20K rows) is used as a conservative reference for sustained performance, aligning with the regime that maximizes end-to-end throughput under double buffering.

5.2 Effectiveness of double buffering

Double buffering consistently improves wall-clock throughput across all tested configurations by overlapping host-to-device transfers with kernel execution. The relative benefit of double buffering is largest at smaller chunk sizes, where single-buffered execution suffers from transfer-induced idle periods. As chunk size increases and execution becomes compute-dominated, the incremental benefit of overlap diminishes, leading to reduced relative speedup while maintaining higher absolute throughput.

5.3 Multi-GPU Scaling Behavior

Strong-scaling experiments across identical L40S GPUs demonstrate that partitioning a fixed workload across multiple devices significantly reduces wall-clock execution time, yielding near-linear speedup. For the evaluated configuration, scaling from one to two GPUs achieves a 1.97× speedup with approximately 98% parallel efficiency. The measured runtime exceeds the ideal $T_1/2$ baseline by only a small margin, indicating modest system-level overhead arising from host-side orchestration, synchronization, and transfer contention.

Per-GPU kernel throughput remains comparable between the single- and multi-GPU cases, confirming that the observed scaling behavior is primarily driven by reduced wall-clock execution time rather than changes in kernel efficiency. While slight run-to-run variability is observed, no systematic degradation in per-GPU compute performance occurs when increasing the number of GPUs.

Weak-scaling experiments on L40 GPUs further show that proportional increases in workload and GPU count lead to near-linear growth in aggregate throughput, with wall-clock time remaining approximately constant. Minor deviations from ideal weak scaling are attributed to fixed host-side overheads and transfer costs that do not scale with problem size. Overall, these results indicate that both L40S and L40 platforms exhibit robust multi-GPU scaling behavior, with performance primarily limited by system-level effects rather than device-side compute efficiency.

5.4 Consumer versus server GPU characteristics

The comparison between consumer- and server-class GPUs highlights qualitative differences in performance stability and scaling behavior. Server GPUs maintain more consistent sustained throughput under large, memory-intensive workloads, benefiting from higher memory bandwidth, robust power delivery, and thermal stability. In contrast, consumer GPUs may achieve high peak kernel throughput under favorable conditions, particularly on smaller workloads, but exhibit greater sensitivity to problem size and execution environment.

6 Conclusion

This work presents a systematic performance study of large-scale $X^\top X$ computation on modern GPUs using chunked processing and overlapped data transfer. By partitioning the input into square chunks and accumulating partial results, the proposed approach enables efficient execution for problem sizes that exceed device memory while preserving mathematical equivalence to a single-pass computation.

Our results demonstrate that chunk size plays a critical role in end-to-end performance. Increasing chunk width transitions execution from a transfer-bound regime to a compute-dominated regime, after which performance gains diminish. Double buffering is shown to be effective in hiding data transfer latency, particularly at smaller chunk sizes, and consistently improves wall-clock throughput.

Strong-scaling experiments across multiple GPUs indicate near-linear speedup for fixed workloads, with observed efficiency close to ideal. Finally, a qualitative comparison between consumer- and server-class GPUs highlights that server GPUs deliver more stable sustained performance under large, memory-intensive workloads, while consumer GPUs may achieve high peak kernel throughput under favorable conditions.

Overall, these findings provide practical guidance for selecting chunk sizes, buffering strategies, and hardware platforms when implementing large-scale matrix multiplication on GPU systems.

Code Availability

The implementation used for all experiments is available at: `https://github.com/duongtrongnguyen123/gpu-out-of-core-xtx`.

References

Numerical Accuracy Validation

We validate numerical accuracy for the computation $C = X^\top X$, where $X \in \mathbb{R}^{M\times N}$ with i.i.d. $\mathcal{N}(0,1)$ entries, using a representative large-scale configuration with $M = 2^{22}$. All inputs are stored in FP32. An FP64 reference is obtained using FP64 compute with FP64 accumulation. All other configurations use FP32 inputs, perform computation in the specified precision (FP32, TF32, BF16, or FP16), and accumulate results in FP32.

Table 13: Numerical accuracy of $X^\top X$ relative to FP64 reference. All inputs are FP32. FP64 uses FP64 compute and accumulation; other modes use the specified compute precision with FP32 accumulation.

Compute dtype	Relative Frobenius error	Smallest eig. rel. (%)	Largest eig. rel. (%)
FP32	4.5×10^{-7}	3.6×10^{-5}	2.9×10^{-5}
TF32	9.4×10^{-5}	9.96×10^{-3}	8.73×10^{-3}
BF16	1.17×10^{-4}	5.81×10^{-3}	4.92×10^{-3}
FP16	8.38×10^{-5}	8.94×10^{-3}	7.70×10^{-3}

Table 13 reports numerical accuracy relative to the FP64 reference. FP32 exhibits near-exact agreement with FP64, while reduced-precision compute modes (TF32, BF16, FP16) introduce relative Frobenius errors on the order of 10^{-4}.

Elementwise accuracy. Elementwise error metrics relative to the FP64 reference are reported using the relative Frobenius norm, RMSE, and mean absolute error. FP32 exhibits near-exact agreement with FP64, while reduced-precision compute modes (TF32, BF16, FP16) introduce relative Frobenius errors on the order of 10^{-4}. Maximum relative errors are dominated by near-zero off-diagonal entries and are therefore not representative of global numerical accuracy.

Distributional sanity check. For $X_{ti} \sim \mathcal{N}(0,1)$, diagonal entries satisfy $\mathbb{E}[C_{ii}] = M$, while off-diagonal entries satisfy $\mathbb{E}[C_{ij}] = 0$ with standard deviation $\sqrt{M}$. Observed diagonal and off-diagonal statistics closely match these theoretical expectations, confirming correct accumulation across all rows and chunks and providing a strong sanity check at scale.

Spectral validation. The smallest and largest eigenvalues of C are compared against the FP64 reference. Across all reduced-precision compute modes, relative deviations remain below 0.01%, indicating that the global spectral structure of $X^\top X$ is well preserved despite reduced-precision compute with FP32 accumulation.