Can Tensor Cores Benefit Memory-Bound Kernels? (No!)

Lingqi Zhang[1], Jiajun Huang[2], Sheng Di[3], Satoshi Matsuoka[1], Mohamed Wahib[1]
[1] RIKEN Center for Computational Science, Japan, {lingqi.zhang@riken.jp, matsu@acm.org, mohamed.attia@riken.jp}
[2] University of California, Riverside, USA, {jhuan380@ucr.edu}
[3] Argonne National Laboratory, USA, {sdi1@anl.gov}

Abstract

Tensor cores are specialized processing units within GPUs that have demonstrated significant efficiency gains in compute-bound applications such as Deep Learning Training by accelerating dense matrix operations. Given their success, researchers have attempted to extend tensor core capabilities beyond dense matrix computations to other computational patterns, including memory-bound kernels. Recent studies have reported that tensor cores can outperform traditional CUDA cores even on memory-bound kernels, where the primary performance bottleneck is not computation. In this research, we challenge these findings through both theoretical and empirical analysis. Our theoretical analysis reveals that tensor cores can achieve a maximum speedup of only 1.33× over CUDA cores for memory-bound kernels in double precision (for V100, A100, and H100 GPUs). We validate this theoretical limit through empirical analysis of three representative memory-bound kernels-STREAM Scale, SpMV, and stencil. We demonstrate that optimizing memory-bound kernels using tensor cores does not yield sound performance improvements over CUDA cores.

1 Introduction

Since Nvidia introduced tensor cores in their Volta architecture in 2017 [6], researchers have extensively explored their applications across various domains, including dense linear algebra [4], sparse linear algebra [10, 24], and spectral methods [9, 27]. The low-precision computation capabilities of tensor cores, which offer substantially higher performance, have spawned innovative approaches such as mixed-precision algorithms [12, 16] and precision recovery [25, 26, 31]. However, despite this broad adoption, fundamental analysis of tensor core performance remains limited, with only a few studies focusing on microbenchmarking [1, 17, 29].

While tensor cores represent a powerful tool for program acceleration, their effective utilization requires a thorough understanding of their performance characteristics. This research addresses this knowledge gap by focusing specifically on memory-bound kernels, which constitute a significant portion of HPC workloads [3]. Our study seeks to answer:

- What is the theoretical performance ceiling for tensor cores when applied to memory-bound kernels?
- Do current tensor core implementation strategies provide real performance benefits for memory-bound kernels?

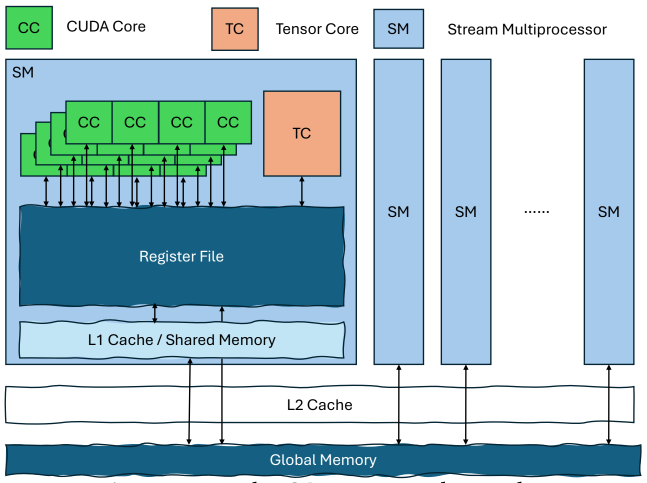

Figure 1. Nvidia GPU memory hierarchy.

To address these questions, we make the following contributions:

- A comprehensive theoretical analysis of tensor core performance for memory-bound kernels.
- An empirical evaluation comparing tensor core implementations against their CUDA core counterparts across representative memory-bound kernels.

The rest of the paper is organized as follows. Section 2 provides essential background concepts. Section 3 introduces our studied memory-bound workloads. Section 4 analyzes two extreme scenarios to establish tensor core performance bounds. Section 5 empirically substantiates our claims using representative memory-bound kernels. Finally, we present key takeaways and conclusions.

2 Background

2.1 Tensor Core

Tensor Core is a specialized systolic array matrix engine [8] integrated within the Stream Multiprocessor (SM) of modern Nvidia GPUs. It operates alongside traditional CUDA cores. The execution pathway for both units follows the GPU's memory hierarchy: data is first loaded from global memory into the register file, from which either CUDA cores or Tensor Cores can access it for computation. This memory access abstraction aligns with previous studies [17, 29]. Figure 1 illustrates this memory hierarchy and the relationship between different memory levels in Nvidia GPUs.

2.2 Machine Balance

We define machine balance ($\mathbb{B}$) [20] as the ratio between peak computational performance (P) and memory bandwidth (B):

$$\mathbb{B} = \frac{P}{B} \tag{1}$$

2.3 Roofline Model

The roofline model [23, 30] provides an upper-bound performance prediction framework based on operational intensity ($\mathbb{I}$), which is defined as the ratio of computational work ($\mathbb{W}$) to memory traffic ($\mathbb{Q}$):

$$\mathbb{I} = \frac{\mathbb{W}}{\mathbb{Q}} \tag{2}$$

The model calculates attainable performance ($\mathbb{P}$) as:

$$\mathbb{P} = \min(P, B \times \mathbb{I}) \tag{3}$$

This visualization framework helps identify system bottlenecks and performance limits.

2.4 Tensor Core in the Roofline Model

As discussed in Section 2.1, tensor cores can be represented as an additional performance ceiling above the CUDA core baseline in the roofline model, because tensor cores and CUDA cores share the same memory hierarchy and cannot operate simultaneously due to the Dark Silicon Effect. This roofline abstraction aligns with an existing study [32].

2.5 Relationship Between Machine Balance and Roofline Model

The roofline model provides a framework for understanding performance bottlenecks through the interplay of two critical metrics: operational intensity and machine balance. While operational intensity ($\mathbb{I}$) characterizes a kernel's computational density, machine balance ($\mathbb{B}$) represents the hardware's ratio of computational capability to memory bandwidth. The relationship between these metrics determines whether a kernel's performance is mainly limited by computation or memory access:

$$A\ kernel\ is = \begin{cases} compute-bound, & \text{if } \mathbb{I} > \mathbb{B} \\ memory-bound, & \text{if } \mathbb{I} < \mathbb{B} \end{cases} \tag{4}$$

As illustrated in Figure 2, machine balance manifests as the inflection point in the roofline curve for both GH200 and A100-80GB GPUs. This point marks the transition from memory-bound to compute-bound behavior.

3 Workloads: Memory-Bound Kernels

This section examines three representative memory-bound kernels: SCALE (Section 3.1), Sparse Matrix-Vector Multiplication (SpMV) (Section 3.2), and Stencil (Section 3.3). We analyze these kernels through the lens of operational intensity ($\mathbb{I}$), focusing on double-precision operations (data size $\mathbb{D} = 8$ bytes).

While our analysis centers on double precision, the methodology can be extended to lower-precision scenarios.

3.1 SCALE

SCALE, one of the STREAM benchmark [19], is defined as

$$a_i = qb_i, \quad \forall i \in 1, \dots, n, \quad a, b \in \mathbb{R}^n, \quad q \in \mathbb{R} \tag{5}$$

Each element operation requires one load, one store, and one computation, yielding: $\mathbb{W}(\text{SCALE}) = 1$, $\mathbb{Q}(\text{SCALE}) = 2 \times \mathbb{D}$, and consequently $\mathbb{I}(\text{SCALE}) = \frac{1}{16}$. STREAM benchmark is commonly used to measure sustainable memory bandwidth due to its low computational intensity.

3.2 Sparse Matrix–Vector Multiplication (SpMV)

SpMV, crucial for iterative solvers, has format-dependent operational intensity. In this section, we begin by analyzing dense matrix-vector multiplication (GEMV) as a baseline.
GEMV: For matrix $A \in \mathbb{R}^{m\times n}$ and vectors $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, GEMV is defined as:

$$y = Ax \tag{6}$$

With computing $\mathbb{W}(\text{GEMV}) = m \times n \times 2$ operations and memory traffic $\mathbb{Q}(\text{GEMV}) = (m \times n + m + n) \times \mathbb{D}$ yields:

$$\mathbb{I}(\text{GEMV}) = \frac{m \times n \times 2}{(m \times n + m + n) \times \mathbb{D}} \approx \frac{2}{\mathbb{D}} = \frac{1}{4} \tag{7}$$

SpMV: For sparse matrices with nnz non-zeros, SpMV is defined as:

$$y = Ax,\ A \in \mathbb{R}^{m\times n}, \quad \text{nnz}(A) \quad \ll mn, \quad x \in \mathbb{R}^n, \quad y \in \mathbb{R}^m \tag{8}$$

With computation $\mathbb{W}(\text{SpMV}) = 2 \times nnz$ and memory traffic including coordinate information $\alpha\mathbb{I}$ or packed values $\beta\mathbb{Z}$:

$$\mathbb{I}(\text{SpMV}) = \frac{nnz \times 2}{(nnz + m + n) \times \mathbb{D} + \alpha\mathbb{I} + \beta\mathbb{Z}} \tag{9}$$

Given $nnz \ll m \times n$, we have $\mathbb{I}(\text{SpMV}) < \mathbb{I}(\text{GEMV})$.
Compressed Sparse Row (CSR) format: CSR format, the most common sparse representation, requires storing column indices and row pointers. With memory traffic $\mathbb{Q}(\text{SpMV,CSR}) = (nnz + m + n) \times \mathbb{D} + (nnz + m + 1) \times \mathbb{I}$ and computation $\mathbb{W}(\text{SpMV,CSR}) = 2 \times nnz$:

$$\begin{aligned} \mathbb{I}(\text{SpMV,CSR}) &= \frac{2 \times nnz}{(nnz + m + n) \times \mathbb{D} + (nnz + m + 1) \times \mathbb{I}} \\ &\approx \frac{2}{\mathbb{D} + \mathbb{I}} = \frac{1}{6} < \mathbb{I}(\text{GEMV}) \end{aligned} \tag{10}$$

This analysis confirms SpMV's memory-bound nature, consistent with prior works [14, 33].

3.3 Iterative Stencils

Stencil computations is common in HPC [11]. For 2D stencil, we have:

$$v(i, j) = \sum_{(p,q)\in\mathbb{S}} w_{p,q} \cdot u(i + p, j + q) \tag{11}$$

where $v(i, j)$ and $u(i, j)$ are updated and original values at point (i, j), and $\mathbb{S}$ defines relative offsets (e.g., 5-point stencil:

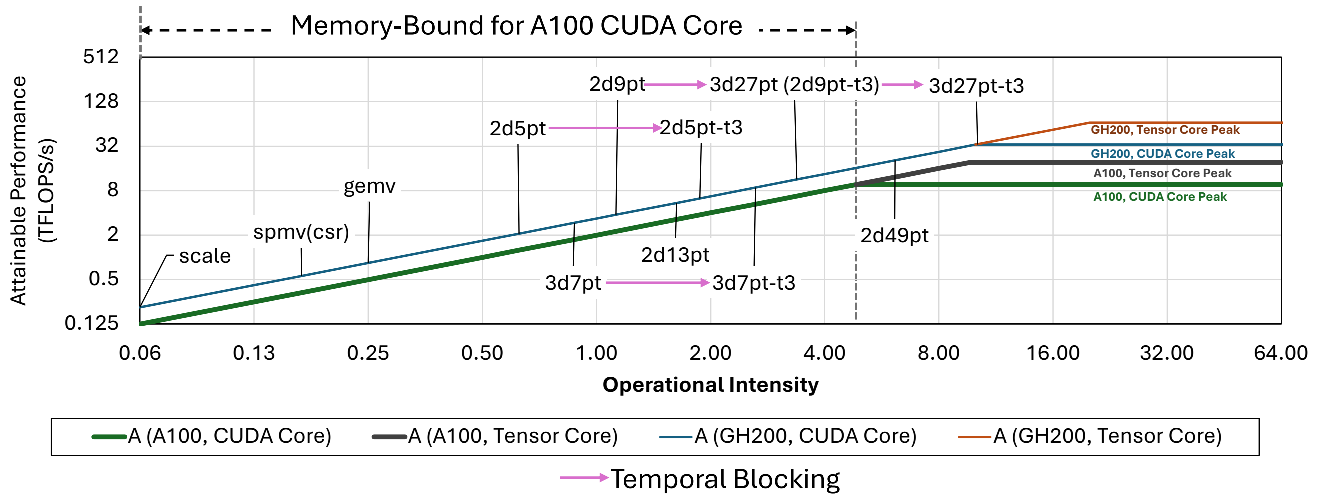

Figure 2. An example of the roofline model for both GH200 and A100-80GB GPU.

$(-1, 0)$, $(1, 0)$, $(0, 1)$, $(0, -1)$, $(0, 0)$). Ideally, only one load of u and one store of v are necessary:

$$\mathbb{Q} = 2 \times \mathbb{D}, \quad \mathbb{W} = 2 \times |\mathbb{S}|, \quad \mathbb{I} = \frac{|\mathbb{S}|}{\mathbb{D}} \tag{12}$$

For a 2d5pt stencil where $|\mathbb{S}(2d5pt)| = 5$, $\mathbb{I}(2d5pt) = \frac{5}{8}$. **Temporal blocking [18, 34]** combines t timesteps together:

$$\mathbb{W}_t = t \times 2 \times |\mathbb{S}|, \quad \mathbb{I}_t = t \times \frac{|\mathbb{S}|}{\mathbb{D}} \tag{13}$$

While temporal blocking can theoretically transform memory-bound stencils into compute-bound kernels by increasing operational intensity, practical limitations exist.

For a 2d5pt stencil on GH200 ($\mathbb{B}_{GH200} = 9.99$), compute-bound behavior requires:

$$t \times \mathbb{I}(2d5pt) > \mathbb{B}_{GH200} \implies t \times 0.625 > 9.99 \implies t > 15.98 \tag{14}$$

However, deep temporal blocking (e.g., $t > 16$) usually faces hardware limits from register pressure [18, 34].

Thus, shallow temporal blocking ($t < 16$) 2d5pt stencil remains memory-bound, while deep temporal blocking might make stencil kernel register-bound.

4 Theoretical Analysis

Based on the operational intensity analysis from Section 3, Figure 2 shows all studied kernels are memory-bound on GH200, and most memory-bound on A100. To simplify the following discussion, we assume that all kernels are throughput-bound.

when it is throughput bound, which is usually the case in High-Performance workloads, we have time for computation $T_{\text{cmp}} = \frac{\mathbb{W}}{P}$. and time for memory access $T_{mem} = \frac{\mathbb{Q}}{B}$. So we have:

$$\frac{T_{mem}}{T_{cmp}} = \frac{\frac{\mathbb{Q}}{B}}{\frac{\mathbb{W}}{P}} = \frac{\mathbb{B}}{\mathbb{I}} \tag{15}$$

For memory-bound kernel, $\mathbb{B} > \mathbb{I}$, we have:

$$T_{\text{mem}} > T_{\text{cmp}} \tag{16}$$

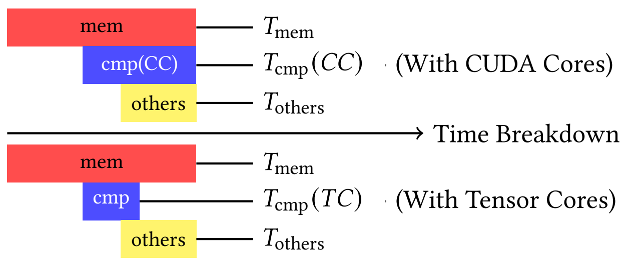

Figure 3. Fully overlapped kernel time breakdown

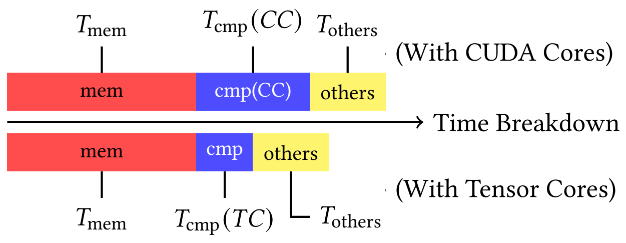

Figure 4. Fully un-overlapped kernel time breakdown

We analyze two extreme cases: fully overlapped and fully un-overlapped for memory access and computation.

4.1 Fully Overlapped

An example of an overlapped kernel time breakdown is shown in Figure 3. For memory-bound kernels:

$$T = \max(T_{\text{cmp}}, T_{\text{mem}}, T_{\text{others}}) = \max(T_{\text{mem}}, T_{\text{others}}) \tag{17}$$

where T_{mem}, T_{cmp}, T_{others} and T represent memory access, computation, other operation and total times, respectively. In this scenario, reducing computation time cannot influence total runtime.

4.2 Fully Un-overlapped

For fully un-overlapped kernels (Figure 4):

$$T = T_{\text{cmp}} + T_{\text{mem}} + T_{\text{others}} \tag{18}$$

Table 1. Specifications of the experimental platforms.

Metric		A100-80GB	GH200
CUDA Version		12.1	12.6
L2 Cache (MB)		40	50
Memory Bandwidth (TB/s)		1.94	4.00
FP64 Peak (TFLOPS)	**CUDA Core**	9.7	34.0
	Tensor Core	19.5	67.0

With tensor cores providing speedup α ($\alpha = \frac{P(TC)}{P(CC)}$,$\alpha > 1$): $T'_{\text{cmp}}(TC) = \frac{1}{\alpha}T_{\text{cmp}}(CC)$, yielding:

$$Speedup = \frac{T(CC)}{T(TC)} = \frac{T_{\text{cmp}(CC)} + T_{\text{mem}} + T_{\text{others}}}{\frac{1}{\alpha}T_{\text{cmp}(CC)} + T_{\text{mem}} + T_{\text{others}}} \tag{19}$$

$$= 1 + \frac{\alpha - 1}{1 + \alpha\frac{T_{\text{mem}} + T_{\text{others}}}{T_{\text{cmp}}(CC)}} \tag{20}$$

$$= 1 + \frac{\alpha - 1}{1 + \alpha\frac{T_{\text{cmp}}(CC) \times \frac{\mathbb{B}}{\mathbb{I}} + T_{\text{others}}}{T_{\text{cmp}}(CC)}} \tag{21}$$

$$< 1 + \frac{\alpha - 1}{1 + \alpha(\frac{\mathbb{B}}{\mathbb{I}})} \tag{22}$$

Tensor Core Upper Bound: For memory-bound kernel, we have $T_{\text{cmp}} \longrightarrow T_{\text{mem}}$:

$$Speedup < 1 + \frac{\alpha - 1}{1 + \alpha} = 2 - \frac{2}{1 + \alpha} \tag{23}$$

Example: FP64 Nvidia GPUs (with $\alpha = 2$): $Speedup < 1.33$.
Example: Assuming that $\alpha \longrightarrow \infty$, we have $Speedup < 2$.
Workload Upper Bound: we assume that $\alpha \longrightarrow \infty$:

$$Speedup < 1 + \frac{\mathbb{I}}{\mathbb{B}} \tag{24}$$

Example: $Speedup_{A100}$(GEMV) < 1.05.

4.3 Summary

Our analysis covers the two extremes of memory-computation overlap. Real-world kernels typically exhibit partial overlap, resulting in speedups between 1× and 1.33× for double precision. Performance differences beyond that would require memory access optimizations, which, we argue, function equally when applied to tensor and CUDA cores since both access data through the register file (As Figure 1 shows).

5 Empirical Analysis

In this section, we use empirical analysis to verify our theoretical findings regarding tensor core performance on memory-bound kernels. We conduct experiments across multiple hardware platforms, detailed in Table 1, to systematically evaluate the performance relationship between tensor core and CUDA core implementations.

5.1 SCALE

Tensor Core Implementation: Inspiring by work [22], we implement tensor core SCALE as matrix multiplication $A = B(qI)$, where I is the identity matrix, as shown in Figure 5. However, this implementation utilizes only $\frac{1}{max(m,n)}$ of the tensor core's compute capacity (m, n = tensor core dimensions). For A100 and H100's 8×4 double precision tensor

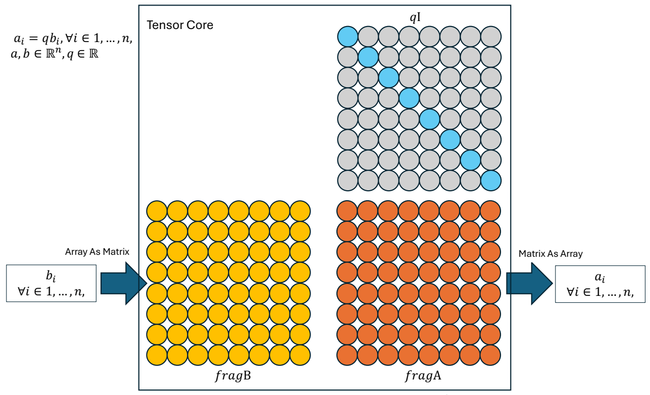
$a_i = qb_i, \forall i \in 1, \ldots, n,$
$a, b \in \mathbb{R}^n, q \in \mathbb{R}$
Tensor Core
qI
Array As Matrix
b_i
$\forall i \in 1, \ldots, n,$
Matrix As Array
a_i
$\forall i \in 1, \ldots, n,$
$frag$B
$frag$A

Figure 5. Tensor core SCALE implementation.

cores, only 1/8 of compute power is used: $\mathbb{P}_{A100}(TC, SCALE) =$ 2.4, TFLOPS/s $\mathbb{P}_{GH200}(TC, SCALE) = 8.37$ TFLOPS/s. This is lower than CUDA core performance. However, this should not significantly impact SCALE kernel performance, with or without overlap, given its extremely low operational intensity $\mathbb{I}$ (According to Section 4).
CUDA Core Implementation: The CUDA core baseline uses a ChatGPT-generated STREAM implementation[1], modified only to include warmup iterations.

5.2 SpMV

We evaluate performance using the same 21 representative sparse matrices from the DASP study [15] (Table 2).
DASP [15] (Tensor Core): DASP, the SoTA tensor core SpMV implementation, employs a hybrid approach, categorizing matrix rows as long, middle, or small, and applies specialized processing strategies for each category, including row sorting for middle-length rows.
cuSPARSE CSR [21] (CUDA Core): While formats with reordering (e.g., SELL-C-σ[13]) might provide a more direct comparison to DASP, sorting can alter matrix characteristics and complicate performance analysis[2]. Therefore, we use the widely adopted cuSPARSE CSR format [21] as our baseline.

5.3 Iterative Stencils

Stencil implementations were evaluated using ConvStencil [5] benchmark suite (Table 3). Due to bugs in both ConvStencil's and LoRAStencil's Artifact Description/Artifact Evaluation (AD/AE) on the GH200 platform, our experiments are restricted to the A100 platform.
ConvStencil [5] (Tensor Core): ConvStencil leverages tensor cores by transforming stencil computation into matrix-matrix multiplication, incorporating temporal blocking through kernel fusion. We evaluate using their default configuration and domain sizes.
LoRAStencil [35] (Tensor Core): LoRAStencil applies Low-Rank adaptation to reduce stencil computational redundancy.

[1]https://chatgpt.com/share/67570ae8-4554-8007-9f91-48f01722af85

Table 2. Datasets for the SpMV benchmark (from DASP [15]). The dataset is ranked by the non-zero value size

Code	Name [7]	Rows	NNZ	Code	Name [7]	Rows	NNZ	Code	Name [7]	Rows	NNZ
D1	dc2	116,835	766,396	**D8**	webbase-1M	1,000,005	3,105,536	**D15**	pwtk	217,918	11,634,424
D2	scircuit	170,998	958,936	**D9**	ASIC_680k	682,862	3,871,773	**D16**	Si41Ge41H72	185,639	15,011,265
D3	mac_econ_fwd500	206,500	1,273,389	**D10**	cant	62,451	4,007,383	**D17**	in-2004	1,382,908	16,917,053
D4	conf5_4-8x8-10	49,152	1,916,928	**D11**	pdb1HYS	36,417	4,344,765	**D18**	Ga41As41H72	268,096	18,488,476
D5	mc2depi	525,825	2,100,225	**D12**	consph	83,334	6,010,480	**D19**	eu-2005	862,664	19,235,140
D6	rma10	46,835	2,374,001	**D13**	shipsec1	140,874	7,813,404	**D20**	FullChip	2,987,012	26,621,990
D7	cop20k_A	121,192	2,624,331	**D14**	mip1	66,463	10,352,819	**D21**	circuit5M	5,558,326	59,524,291

Table 3. Stencil benchmarks and domain sizes we use. A detailed description of the stencil benchmarks can be found in [28, 36].

	domain(temporal blocking depth)		
	ConvStencil [5]	Brick [36]	EBISU [34]
2d5pt	$10240^2(3)$	-	$9000^2(3)$
2d13pt	$10240^2(1)$	-	$9000^2(1)$
2d9pt	$10240^2(3)$	-	$9000^2(3)$
2d49pt	$10240^2(1)$	-	$9000^2(1)$
3d7pt	$1024^3(3)$	$512^3(1)$	$234 \times 312 \times 2560(3)$
3d27pt	$1024^3(3)$	$512^3(1)$	$234 \times 312 \times 2560(3)$

While innovative, their artifact evaluation relies on assumptions about the rank of stencil weights, which limits its practical applicability. Due to these constraints, we use their published performance results for comparison.

Brick [36] (CUDA Core): Baseline CUDA Core implementation without temporal blocking. We evaluated it using the default configuration.

EBISU [34] (CUDA Core): EBISU is a state-of-the-art CUDA Core implementation that incorporates temporal blocking. To ensure a fair comparison, we configured EBISU's temporal blocking parameters to match those of ConvStencil.

5.4 Evaluation

Figures 6, 7, and 8 present performance comparisons for SCALE, SpMV, and stencil kernels respectively.

SCALE: Figure 6 reveals consistent, though modest, performance degradation when using tensor cores compared to CUDA cores. Given that the computational time difference is negligible, this performance gap likely arises from suboptimal memory access patterns associated with tensor core usage on current GPU architectures.

SpMV: Figure 7 demonstrates that for datasets exceeding the L2 cache size, cuSPARSE (CUDA core) outperforms DASP (tensor core) on average.

Stencils: Figure 8 shows that equivalently optimized tensor core implementations generally underperform their CUDA core counterparts.

Summary: While our goal was to verify the theoretical bounds, empirical evaluation reveals that tensor core implementations usually underperform their CUDA core counterparts.

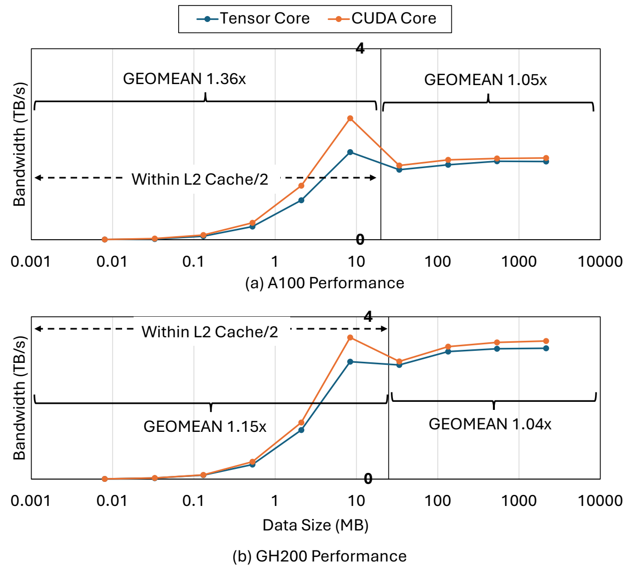

Figure 6. SCALE performance evaluation on A100 (top) and GH200 (bottom). We show the speedup geometric mean (GEOMEAN) of the CUDA cores over the tensor cores is reported for input domain sizes larger and smaller than half of the L2 cache, respectively.

5.5 Other Observations

L2 Cache Impact: L2 cache interactions exhibit nontrivial between implementations. For SCALE, tensor core performance degradation intensifies within L2 cache bounds. Conversely, DASP demonstrates improved performance for cache-resident data, suggesting crucial performance implications of L2 cache optimization.

Compute-Bound Cases: The 2d49pt stencil, which is compute-bound on A100, shows comparable performance between tensor and CUDA core implementations. However, on GH200, the same kernel becomes memory-bound, where CUDA cores theoretically maintain superior performance.

Resource-Constrained Cases: 3D stencils and high-order 2D stencils typically encounter bottlenecks beyond memory or compute limitations, such as register capacity, cache capacity, or cache bandwidth [34]. Since tensor core optimizations target only computational aspects, they provide no inherent advantage for these resource-constrained workloads. As expected, our evaluation shows no performance benefits from tensor core implementations in these stencil benchmarks.

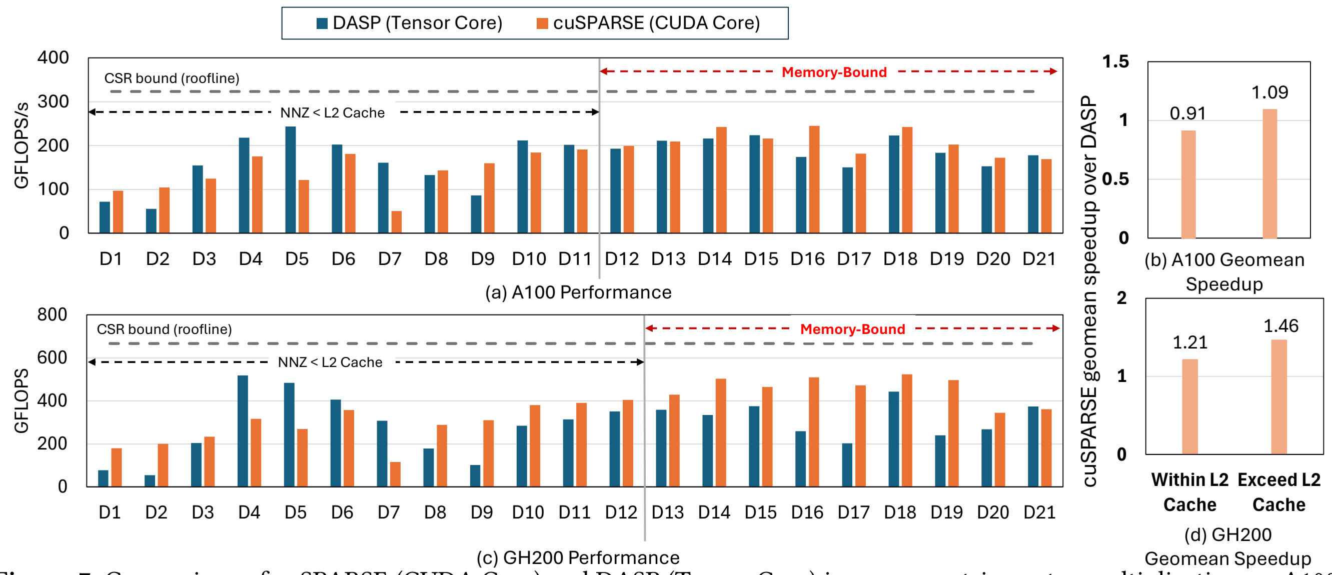

Figure 7. Comparison of cuSPARSE (CUDA Core) and DASP (Tensor Core) in sparse matrix-vector multiplication on A100 (top) and GH200 (bottom). (a) & (c) on the left report the performance in effective flops; (b) & (d) on the right report the geometric mean speedup of cuSPARSE over DASP.

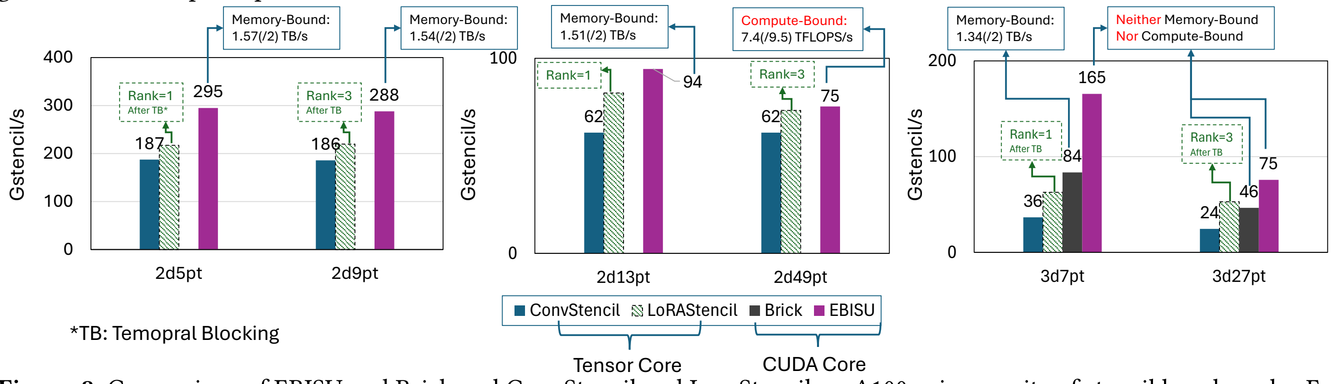

Figure 8. Comparison of EBISU and Brick and ConvStencil and LoraStencil on A100 using a suite of stencil benchmarks. For LoRAStencil [35], performance data and their assumed rank values from their artifact evaluation are included.

6 Key Takeaways

We highlight the key takeaways from a practical perspective:

- Identifying kernel characteristics (compute-bound/ memory-bound) (Section 3).
- Compute-bound
  - Tensor cores remain advantageous for compute-bound operations.
- Memory-bound
  - Prioritizing CUDA cores for memory-bound kernels due to their simplicity and effectiveness (Section 5).
  - Focusing on memory access optimizations, e.g. cache-aware algorithms and reducing memory traffic [33, 34].
  - Prioritizing pipeline and overlap optimizations before considering tensor core adoption (Section 4.1).
  - Considering theoretical limits: tensor cores provide at most 1.33× speedup in double precision, with a 2× ceiling assuming infinite tensor core computational speedup (Section 4.2).

7 Conclusion

Through systematic theoretical and empirical analysis, we demonstrate that leveraging the tensor cores for computation in memory-bound kernels fails to deliver sound performance benefits. Our theoretical analysis establishes an upper bound of 1.33× speedup for double-precision memory-bound kernels, while empirical results across SCALE, SpMV, and stencil show that tensor core implementations usually underperform their CUDA core counterparts. While these findings may temper expectations for using tensor cores in memory-bound kernels, efforts leveraging tensor cores still provide valuable insights for the design and utilization of matrix processing units in broader contexts.

Acknowledgments

This material was supported by the U.S. Dept. of Energy, Office of Science, Advanced Scientific Computing Research (ASCR), under contracts DE-AC02-06CH11357 and DE-SC0024207.

References

[1] Hamdy Abdelkhalik, Yehia Arafa, Nandakishore Santhi, and Abdel-Hameed A. Badawy. 2022. Demystifying the Nvidia Ampere Architecture through Microbenchmarking and Instruction-level Analysis. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–8. https://doi.org/10.1109/HPEC55821.2022.9926299

[2] Hartwig Anzt, Stanimire Tomov, and Jack Dongarra. 2014. Implementing a Sparse Matrix Vector Product for the SELL-C/SELL-C-σ formats on NVIDIA GPUs. *University of Tennessee, Tech. Rep. ut-eecs-14-727* (2014).

[3] Brian Austin, Dhruva Kulkarni, Brandon Cook, Samuel Williams, and Nicholas J Wright. 2024. System-Wide Roofline Profiling-a Case Study on NERSC's Perlmutter Supercomputer. In *PMBS24: The 15th International Workshop on Performance Modeling, Benchmarking, and Simulation of High-Performance Computer Systems*.

[4] Somashekaracharya G. Bhaskaracharya, Julien Demouth, and Vinod Grover. 2020. Automatic Kernel Generation for Volta Tensor Cores. arXiv:2006.12645 [cs.PL] https://arxiv.org/abs/2006.12645

[5] Yuetao Chen, Kun Li, Yuhao Wang, Donglin Bai, Lei Wang, Lingxiao Ma, Liang Yuan, Yunquan Zhang, Ting Cao, and Mao Yang. 2024. ConvStencil: Transform Stencil Computation to Matrix Multiplication on Tensor Cores. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming* (Edinburgh, United Kingdom) *(PPoPP '24)*. Association for Computing Machinery, New York, NY, USA, 333–347. https://doi.org/10.1145/3627535.3638476

[6] Jack Choquette, Olivier Giroux, and Denis Foley. 2018. Volta: Performance and Programmability. *IEEE Micro* 38, 2 (2018), 42–52. https://doi.org/10.1109/MM.2018.022071134

[7] Timothy A Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 1–25.

[8] Jens Domke, Emil Vatai, Aleksandr Drozd, Peng ChenT, Yosuke Oyama, Lingqi Zhang, Shweta Salaria, Daichi Mukunoki, Artur Podobas, Mohamed WahibT, and Satoshi Matsuoka. 2021. Matrix Engines for High Performance Computing: A Paragon of Performance or Grasping at Straws?. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1056–1065. https://doi.org/10.1109/IPDPS49936.2021.00114

[9] Sultan Durrani, Muhammad Saad Chughtai, Mert Hidayetoglu, Rashid Tahir, Abdul Dakkak, Lawrence Rauchwerger, Fareed Zaffar, and Wen-mei Hwu. 2021. Accelerating Fourier and Number Theoretic Transforms using Tensor Cores and Warp Shuffles. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 345–355. https://doi.org/10.1109/PACT52795.2021.00032

[10] Ruibo Fan, Wei Wang, and Xiaowen Chu. 2024. DTC-SpMM: Bridging the Gap in Accelerating General Sparse Matrix Multiplication with Tensor Cores. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 253–267. https://doi.org/10.1145/3620666.3651378

[11] Bastian Hagedorn, Larisa Stoltzfus, Michel Steuwer, Sergei Gorlatch, and Christophe Dubach. 2018. High performance stencil code generation with lift. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. 100–112.

[12] Azzam Haidar, Stanimire Tomov, Jack Dongarra, and Nicholas J Higham. 2018. Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 603–613.

[13] Moritz Kreutzer, Georg Hager, Gerhard Wellein, Holger Fehske, and Alan R. Bishop. 2014. A Unified Sparse Matrix Data Format for Efficient General Sparse Matrix-Vector Multiplication on Modern Processors with Wide SIMD Units. *SIAM Journal on Scientific Computing* 36, 5 (2014), C401–C423. https://doi.org/10.1137/130930352 arXiv:https://doi.org/10.1137/130930352

[14] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. 2010. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. *SIGARCH Comput. Archit. News* 38, 3 (June 2010), 451–460. https://doi.org/10.1145/1816038.1816021

[15] Yuechen Lu and Weifeng Liu. 2023. DASP: Specific Dense Matrix Multiply-Accumulate Units Accelerated General Sparse Matrix-Vector Multiplication. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23)*. Association for Computing Machinery, New York, NY, USA, Article 73, 14 pages. https://doi.org/10.1145/3581784.3607051

[16] Yuechen Lu, Lijie Zeng, Tengcheng Wang, Xu Fu, Wenxuan Li, Helin Cheng, Dechuang Yang, Zhou Jin, Marc Casas, and Weifeng Liu. 2024. Amgt: Algebraic multigrid solver on tensor cores. In *2024 SC24: International Conference for High Performance Computing, Networking, Storage and Analysis SC*. IEEE Computer Society, 823–838.

[17] Weile Luo, Ruibo Fan, Zeyu Li, Dayou Du, Qiang Wang, and Xiaowen Chu. 2024. Benchmarking and Dissecting the Nvidia Hopper GPU Architecture . In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, Los Alamitos, CA, USA, 656–667. https://doi.org/10.1109/IPDPS57955.2024.00064

[18] Kazuaki Matsumura, Hamid Reza Zohouri, Mohamed Wahib, Toshio Endo, and Satoshi Matsuoka. 2020. AN5D: automated stencil framework for high-degree temporal blocking on GPUs. In *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization* (San Diego, CA, USA) *(CGO '20)*. Association for Computing Machinery, New York, NY, USA, 199–211. https://doi.org/10.1145/3368826.3377904

[19] John D. McCalpin. 1991-2007. *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. Technical Report. University of Virginia, Charlottesville, Virginia. http://www.cs.virginia.edu/stream/ A continually updated technical report. http://www.cs.virginia.edu/stream/.

[20] John D McCalpin et al. 1995. Memory bandwidth and machine balance in current high performance computers. *IEEE computer society technical committee on computer architecture (TCCA) newsletter* 2, 19-25 (1995).

[21] Maxim Naumov, L Chien, Philippe Vandermersch, and Ujval Kapasi. 2010. Cusparse library. In *GPU Technology Conference*, Vol. 12.

[22] Cristóbal A Navarro, Roberto Carrasco, Ricardo J Barrientos, Javier A Riquelme, and Raimundo Vega. 2020. GPU tensor cores for fast arithmetic reductions. *IEEE Transactions on Parallel and Distributed Systems* 32, 1 (2020), 72–84.

[23] Georg Ofenbeck, Ruedi Steinmann, Victoria Caparros, Daniele G Spampinato, and Markus Püschel. 2014. Applying the roofline model. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 76–85.

[24] Patrik Okanovic, Grzegorz Kwasniewski, Paolo Sylos Labini, Maciej Besta, Flavio Vella, and Torsten Hoefler. 2024. High Performance Unstructured SpMM Computation Using Tensor Cores. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'24)* (Atlanta, GA, USA). IEEE Press, 154:1–154:14. https://doi.org/10.1109/SC41406.2024.00060

[25] Hiroyuki Ootomo, Katsuhisa Ozaki, and Rio Yokota. 2024. DGEMM on integer matrix multiplication unit. *The International Journal of High Performance Computing Applications* (2024), 10943420241239588.

[26] Hiroyuki Ootomo and Rio Yokota. 2022. Recovering single precision accuracy from Tensor Cores while surpassing the FP32 theoretical peak performance. *The International Journal of High Performance Computing Applications* 36, 4 (2022), 475–491.

[27] Louis Pisha and Łukasz Ligowski. 2021. Accelerating non-power-of-2 size Fourier transforms with GPU Tensor Cores. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 507–516. https://doi.org/10.1109/IPDPS49936.2021.00059

[28] Prashant Singh Rawat, Changwan Hong, Mahesh Ravishankar, Vinod Grover, Louis-Noël Pouchet, and P Sadayappan. 2016. Effective resource management for enhancing performance of 2D and 3D stencils on GPUs. In *Proceedings of the 9th Annual Workshop on General Purpose Processing using Graphics Processing Unit*. 92–102.

[29] Wei Sun, Ang Li, Tong Geng, Sander Stuijk, and Henk Corporaal. 2023. Dissecting Tensor Cores via Microbenchmarks: Latency, Throughput and Numeric Behaviors. *IEEE Transactions on Parallel and Distributed Systems* 34, 1 (2023), 246–261. https://doi.org/10.1109/TPDS.2022.3217824

[30] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76.

[31] Du Wu, Peng Chen, Xiao Wang, Issac Lyngaas, Takaaki Miyajima, Toshio Endo, Satoshi Matsuoka, and Mohamed Wahib. 2024. Real-time High-resolution X-Ray Computed Tomography. In *Proceedings of the 38th ACM International Conference on Supercomputing* (Kyoto, Japan) *(ICS '24)*. Association for Computing Machinery, New York, NY, USA, 110–123. https://doi.org/10.1145/3650200.3656634

[32] Charlene Yang, Thorsten Kurth, and Samuel Williams. 2020. Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system. *Concurrency and Computation: Practice and Experience* 32, 20 (2020), e5547.

[33] Lingqi Zhang, Mohamed Wahib, Peng Chen, Jintao Meng, Xiao Wang, Toshio Endo, and Satoshi Matsuoka. 2023. PERKS: a Locality-Optimized Execution Model for Iterative Memory-bound GPU Applications. In *Proceedings of the 37th ACM International Conference on Supercomputing* (Orlando, FL, USA) *(ICS '23)*. Association for Computing Machinery, New York, NY, USA, 167–179. https://doi.org/10.1145/3577193.3593705

[34] Lingqi Zhang, Mohamed Wahib, Peng Chen, Jintao Meng, Xiao Wang, Toshio Endo, and Satoshi Matsuoka. 2023. Revisiting Temporal Blocking Stencil Optimizations. In *Proceedings of the 37th ACM International Conference on Supercomputing* (Orlando, FL, USA) *(ICS '23)*. Association for Computing Machinery, New York, NY, USA, 251–263. https://doi.org/10.1145/3577193.3593716

[35] Yiwei Zhang, Kun Li, Liang Yuan, Jiawen Cheng, Yunquan Zhang, Ting Cao, and Mao Yang. 2024. LoRAStencil: Low-Rank Adaptation of Stencil Computation on Tensor Cores . In *2024 SC24: International Conference for High Performance Computing, Networking, Storage and Analysis SC*. IEEE Computer Society, Los Alamitos, CA, USA, 839–855. https://doi.org/10.1109/SC41406.2024.00059

[36] Tuowen Zhao, Protonu Basu, Samuel Williams, Mary Hall, and Hans Johansen. 2019. Exploiting reuse and vectorization in blocked stencil computations on CPUs and GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–44.