



Accurate Models of NVIDIA Tensor Cores

Mantas Mikaitis

Joint work with Faizan Khattak

School of Computer Science, University of Leeds, Leeds, UK

FPTalks monthly talk (virtual)

7 May, 2026



Mixed-precision matrix multipliers

Floating-point **matrix multiply-accumulate (MMA)** is a hardware operation introduced in GPUs relatively recently.

$$D = C + A \times B,$$

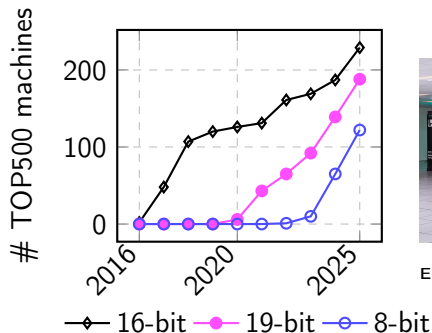
Diagram illustrating the MMA operation with mixed precision:

- D (4x4 matrix) is the result, labeled "binary16 or binary32".
- C (4x4 matrix) is the first operand, labeled "binary16 or binary32".
- A (4x4 matrix) is the second operand, labeled "8/16/19-bit FP".
- B (4x4 matrix) is the third operand, labeled "8/16/19-bit FP".

Hardware matrix multipliers in mixed precision

- MMA is ubiquitous and drives algorithm research (mixed precision).
- Example is 4×4 , but dimensions differ across architectures.

Matrix multiplier hardware on the TOP500 (Nov. 2025)



El Capitan, Top 1, asc.llnl.gov/

Devices counted: V100, A100, H100, MI210, MI250X, MI300X, Intel Data Center GPU, from <https://www.top500.org>.

NVIDIA Rubin GPU throughputs (FLOPS)
fp8 (1.7×10^{16}) fp16 (4×10^{15}) fp64 (3.3×10^{13}).

<https://www.nvidia.com/en-us/data-center/vera-rubin-nvl72/>

Iter. refinement, high accuracy GEMM.

Acta Numerica (2022), pp. 347–414
doi:10.1017/S0962492922000022

Printed in the United Kingdom

Mixed precision algorithms in numerical linear algebra

Nicholas J. Higham

*Department of Mathematics, University of Manchester,
Manchester, M13 9PL, UK*

E-mail: nick.higham@manchester.ac.uk

Theo Mary

*Sorbonne Université, CNRS, LIP6,
Paris, F-75005, France*

E-mail: theo.mary@lip6.fr

Today's floating-point arithmetic landscape is broader than ever. While scientific computing has traditionally used single precision and double precision floating-point arithmetics, half precision is increasingly available in hardware and quadruple preci-

Quantum chemistry, genome-wide association studies, FFT.

The Journal of Supercomputing (2026) 82:287
<https://doi.org/10.1007/s11227-026-08264-4>



Mixed-precision numerics in scientific applications: survey and perspectives

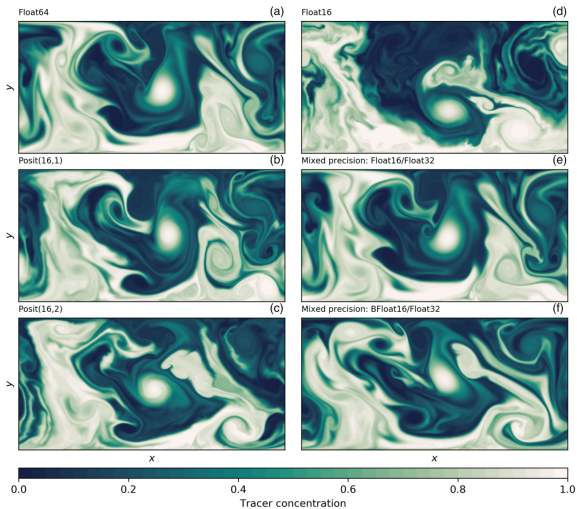
Aditya Kashi¹ · Hao Lu¹ · Wesley Brewer¹ · David Rogers¹ · Michael Matheson¹ · Mallikarjun Shankar¹ · Feiyi Wang¹

Received: 8 September 2025 / Accepted: 19 January 2026

This is a U.S. Government work and not under copyright protection in the US; foreign copyright protection may apply 2026

Abstract

The explosive demand for artificial intelligence (AI) workloads has led to a significant increase in silicon area dedicated to lower-precision computations on recent high-performance computing hardware designs. However, mixed-precision capabilities, which can achieve performance improvements of up to 8× compared to double-precision in extreme compute-intensive workloads, remain largely untapped in most



Shallow water model simulation ([Klower et al. 2020]).

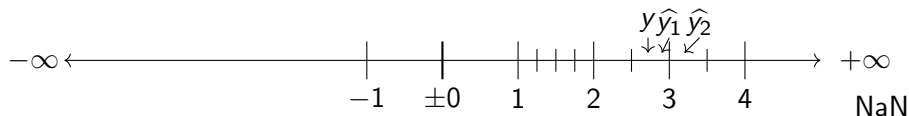
The many floating-point formats

Format	precision	min norm. pos.	max pos.
binary64 (double)	53	2^{-1022}	$\sim 1.798 \times 10^{308}$
binary32 (single)	24	2^{-126}	$\sim 3.403 \times 10^{38}$
tf19 (19-bit)	11	2^{-126}	$\sim 3.401 \times 10^{38}$
bfloat16	8	2^{-126}	$\sim 3.389 \times 10^{38}$
binary16 (half)	11	2^{-14}	65504
fp8-E4M3	4	2^{-6}	448
fp8-E5M2	3	2^{-14}	57344
fp6-E2M3 (block FP)	4	2^0	7.5
fp6-E3M2 (block FP)	3	2^{-2}	28
fp4-E2M1 (block FP)	2	2^0	6

IEEE 754 floating-point arithmetic: what is standardised?

- Rounding modes: result lies between two FP numbers—return one of them consistently.
- *Correct rounding* (directed): exact result y and approximation \hat{y} are surrounded by the same two FP numbers, consistently.
- Correct rounding (nearest): nearest FP number to y is also a nearest FP number to \hat{y} .

\mathbb{F}



Accuracy is therefore not a consideration in IEEE 754—compliant operations are as accurate as precision allows. And consistent across implementations.

IEEE 754 floating-point arithmetic: What is standardised?

IEEE 754 requires *correctly rounded*

- $+$, $-$, \times , \div
- Square root
- FMA: $a \times b + c$

However, it does not require x^y and mathematical functions \exp , \sin , \cos , \log , and others.

Nevertheless, it recommends providing them, and they *shall be* correctly rounded—most mathematical function libraries do not assure this today.

Some of the known properties of floating point:

- ✓ Commutativity: $a \times b = b \times a$.
- × Associativity: $(a + (b + c)) \neq (a + b) + c$.
- × Distributivity: $(a \times (b + c)) \neq (a \times b) + (a \times c)$.

Even though $+$ is correctly rounded, IEEE 754 does not mandate the order in $\sum_{i=0}^n x_i$ —results are implementation dependent.

But that is OK: ordering is in programmer's control, and two systems can match their implementations if needed.

(except when race conditions and thread scheduling change ordering)

IEEE 754: What about matrix multiplication?

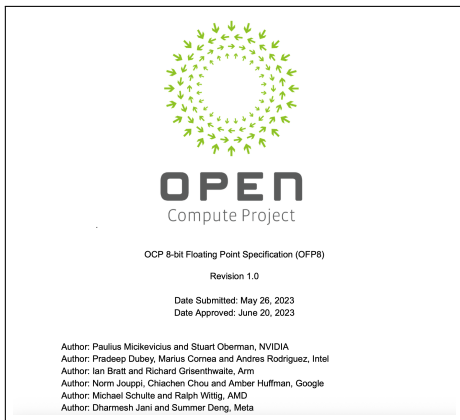
- Key properties of *reduction operations* in IEEE 754 are not specified: (see Clause 9.4 in 754-2019).
- “*Implementations may associate in any order or evaluate in any wider format*”. Commercial impl. usually not documented and change over years. Different optimisation targets lead to different designs.
- **No reproducibility, portability, and errors cannot be explained. Ordering of additions is enforced.**

$$\begin{array}{ccccccc} D & = & C & + & A & \times & B, \\ \left[\begin{array}{cccc} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{array} \right] & = & \left[\begin{array}{cccc} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{array} \right] & + & \left[\begin{array}{cccc} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{array} \right] & \times & \left[\begin{array}{cccc} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{array} \right] \\ \underbrace{\hspace{1.5cm}} & & \underbrace{\hspace{1.5cm}} & & \underbrace{\hspace{1.5cm}} & & \underbrace{\hspace{1.5cm}} \\ \text{binary16 or} & & \text{binary16 or} & & \text{8/16/19-bit FP} & & \text{8/16/19-bit FP} \\ \text{binary32} & & \text{binary32} & & & & \end{array}$$

Two standards:

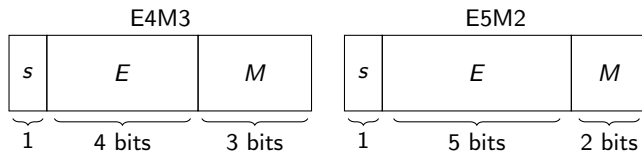
- Standard 1: OCP 8-bit Floating Point Specification
- Standard 2: OCP Microscaling Formats (MX) Specification

Version 1.0 released June and September 2023.



OCP floating-point standard 1

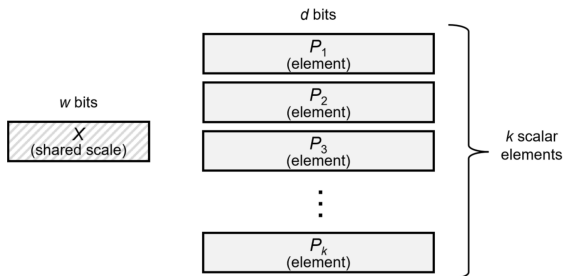
Key aspects in the Standard 1



- Defines two formats: OFP8. E4M3 no $\pm\infty$, one NaN.
- Defines conversion from higher precision formats (binary32, binary16, bfloat16) to OFP8.
- Conversion includes
 - Round to nearest (no other rounding modes required)
 - Saturation mode: after rounding, if $> f_{max}$, return f_{max} instead of ∞ .

Arithmetic operations are not in scope of standard 1.

OCP standard 2 (screenshot from the MX standard)



Dot product of two MX vectors must be provided

$$C = \text{Dot}(A, B) = X^A X^B \sum_{i=1}^{32} (P_i^A \times P_i^B)$$

("internal precision of the dot product and order of operations is implementation-defined")

New IEEE standard: P3109

- Defines formats with one NaN, no neg. zero.
- Generalised, but focus on 8-bit formats.
- Defines correctly rounded dot product and a κ -approximate one.
- “*A κ -approximate implementation may similarly choose any accumulation order. The way partial sums and products are developed can reduce internal overflow and underflow.*”
- No strong requirement at present.

Interim report available:

<https://github.com/P3109/Public>.

IEEE Working Group P3109 Interim Report
on Binary Floating-point Formats for Machine Learning

Initial release: 18 September 2023
Version 2.0: 29 October 2024
Version 3.0: 21 July 2025
Version 3.2: 5 January 2026
Version 3.2.1: 12 January 2026
(compiled 2026-01-12)

DRAFT DOCUMENT

To cite this report please see the CITATION.cff file
found at <https://github.com/P3109/Public>.

Our project goals

Goal 1: Describe matrix multipliers of five generations of NVIDIA GPUs.

GPU Model	Architecture	SM Compute Capability
V100	Volta	70
A100	Ampere	80
A30	Ampere	80
A40	Ampere	86
A2	Ampere	86
L40S	Ada Lovelace	89
RTX 1000	Ada Lovelace	89
H100	Hopper	90
GH200	Hopper	90
B200	Blackwell	100

Goal 2: Develop MATLAB models for mixed-precision experimentation.

Example MATLAB Tensor Core v0.4.1 run: small MMA

```
>> inopts.format = 'fp8-e4m3';
>> outopts.format = 'binary32';
>> A = cpfloat(randn(4, 32), inopts);
>> B = cpfloat(randn(32, 4), inopts);
>> C = cpfloat(randn(32, 32), outopts);
>> alpha = 1; beta = 1;
>> B200TC(alpha, A, B, beta, C, inopts.format, outopts.format)
ans =
   -0.7092   -0.0027    7.1206    6.5745
   -4.8832  -10.9299    7.6650    2.0503
   -4.1610   -6.7480   -0.7719   -0.7542
    2.4509    9.8056    3.1729   -8.7325
>> H100TC(alpha, A, B, beta, C, inopts.format, outopts.format)
ans =
   -0.7092   -0.0027    7.1206    6.5742
   -4.8833  -10.9297    7.6646    2.0500
   -4.1611   -6.7476   -0.7720   -0.7544
    2.4512    9.8047    3.1729   -8.7324
```

Example application: high-precision matrix multiply

- Use low-prec matrix mult. to emulate high-prec. [Mary and Mikaitis, 2025]
- Split input matrices into p -word representation.
- Scaling can be applied to avoid under/overflows, but not shown here for simplicity.

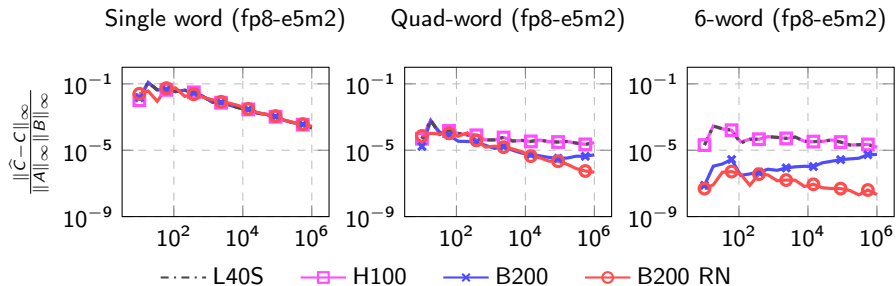
$$A_i = \text{fl}_{\text{low}} \left(A - \sum_{k=1}^{i-1} A_k \right), \quad B_j = \text{fl}_{\text{low}} \left(B - \sum_{k=1}^{j-1} B_k \right).$$

Then the product $C = AB$ is given by

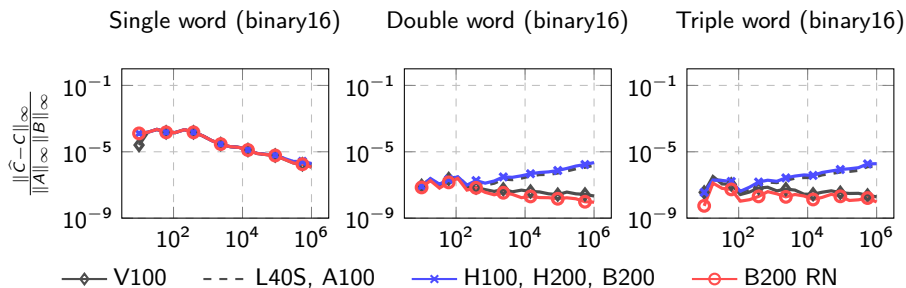
$$C \approx \sum_{i=1}^p \sum_{j=1}^p A_i B_j.$$

Let's run it with the MATLAB Tensor Core.

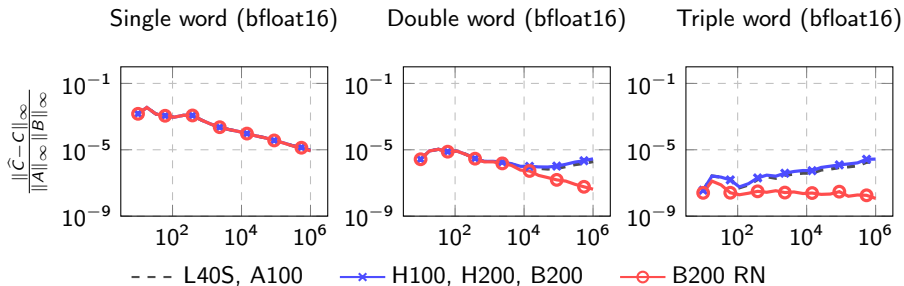
Example application: high-precision matrix multiply



Example application: high-precision matrix multiply



Example application: high-precision matrix multiply



How do we derive the models? Formal problem statement

Take two matrices $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$. The matrix multiply-accumulate (MMA) operation produces

$$D = AB + C \in \mathbb{R}^{m \times n}.$$

Denote with d_{ij} the element at i th row and j th column of D . We can express it as the inner product between the i th row of A and j th column of B as

$$d_{ij} = \sum_{\ell=1}^k a_{i\ell} b_{\ell j} + c_{ij}.$$

To focus on the inner product as an underlying operation, rather than on particular elements of D , we drop the subscripts for simplicity, and we have

$$d = \sum_{\ell=1}^k a_{\ell} b_{\ell} + c = \sum_{\ell=1}^k p_{\ell} + c.$$

Formal problem statement

$$d = \sum_{\ell=1}^k a_{\ell} b_{\ell} + c = \sum_{\ell=1}^k p_{\ell} + c.$$

- Take p_{in} : precision of the format of a and b .
- Take p_{out} : precision of the output format.
- Find N_{FMA} : number of p_{ℓ} added together without intermediate normalisation.
- Find n_{eab} : number of extra bits, relative to p_{out} .
- Determine the rounding mode of alignments/additions.
- Determine the rounding mode from internal accumulator to p_{out} .
- Check if p_{ℓ} can under/overflow.
- Determine precision and normalisation of p_{ℓ} .

Input space size?

- For 8-bit FP formats, each member of a and b has roughly 256 possible values.
- $N_{\text{FMA}} = 32$, each a and b have $\sim 10^{77}$ possible inputs.
- For 16-bit formats, this goes up to $\sim 10^{154}$.
- Implications (challenges):
 - We cannot exhaustively cover the input space to check how the matrix multiplier works in all cases—targeted vectors are needed.
 - Once we build a model, we cannot exhaustively compare it to a GPU.

Refinement algorithm

- 1: GNFT: apply generalised test vectors ▷ *Initial approximation of the model is determined.*
- 2: **while** True **do** ▷ *Refine the approximation of the model*
- 3: ISSM: generate random test vectors
- 4: For each test vector, compute matrix multiply on a GPU and on its approximate model
- 5: **if** GPU and the model results mismatch **then**
- 6: Inspect failure cases and modify the model
- 7: **else**
- 8: Break ▷ *All test vectors pass. The model is sufficiently accurate.*
- 9: **end if**
- 10: **end while**

Concept of an idea to test a numerical feature:

- Create an input test vector $X = \{a, b, c\}$.
- Create a set of potential outcomes, covering the design space of MMA: Y_1, Y_2, \dots
- The test $\{X, Y_1, Y_2, \dots\}$ is expressed via $p_{\text{in}}, p_{\text{out}}$, and other parameters, not constants.
- Applicable across any current and future input FP formats.
- Apply X to a matrix multiplier.
- Output $d = Y_1$ - feature is present, output Y_2 - feature is not present, etc.

Provides an initial approximation of the model. Step 1 in the algorithm.

- Input space of dot products is very large.
- How to find inputs that produce mismatches between GPU and the GNFT-approximated model?
- **Normal and uniform distributions (all GPUs):**
 - 10^5 input samples, norm, distr.
 - 10^7 input samples, unif. distribution in $\pm 2^{15}$.
- **Targeted tests (all GPUs):**
 - Test vectors to evaluate behaviour for $\pm\text{Inf}$ and NaN.
 - 10^7 input samples drawn from a uniform distribution with $A, B \in [-2^{-65}, 2^{-65}]$ and $C \in [-2^{-126}, 2^{-126}]$ for bf16 and tf19
 - A smaller ensemble of 5×10^5 samples targeting the fp16 subnormal region, with $A, B \in [-2^{-15}, 2^{-15}]$ and $C \in [-2^{-126}, 2^{-126}]$, on V100 GPU.
- **Extended ensemble test (Hopper and Blackwell in bf16):**
 - 10^8 samples, unif. distr., where $A, B \in [-2^{63}, 2^{63}]$ and $C \in [-2^{122}, 2^{122}]$, for bf16. **~40 hours.**

Why this improves on the past research?

Our work builds on a set of excellent previous research:

- Paranoia, 1982: tests elementary double-precision operations. Generalised.
- Paper [Tan et al. 2012], elementary ops on FPGAs.
- Paper [Hickam and Bradford, 2019] characterised V100. Used constants.
- Our paper, [Fasi et al. 2021], V100, T4, A100. Used constants.
- Paper [Li et al. 2024] H100, AMD MI100, MI250X. Used constants.
- Paper [Valpey et al. 2025] uses SMT to find test vectors: long run time; no guarantee of completion. Requires rerunning SMT search for new input formats.

We introduce generalised vectors and iterative refinement of a model.

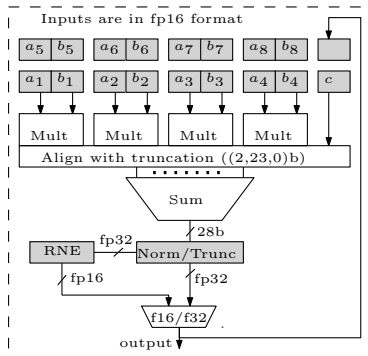
Results: GPUs under test

GPU Model	Architecture	SM Compute Capability
V100	Volta	70
A100	Ampere	80
A30	Ampere	80
A40	Ampere	86
A2	Ampere	86
L40S	Ada Lovelace	89
RTX 1000	Ada Lovelace	89
H100	Hopper	90
GH200	Hopper	90
B200	Blackwell	100

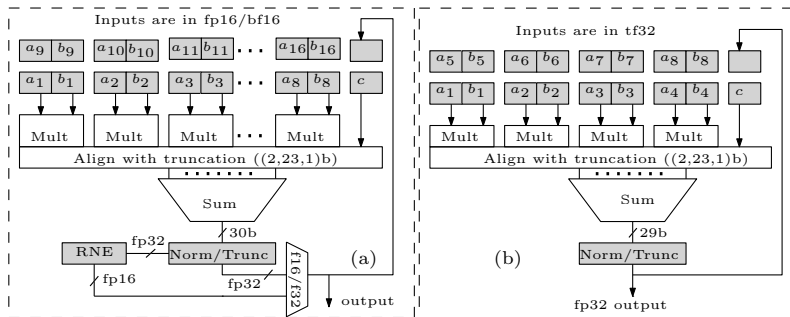
Results: Array of PTX instructions

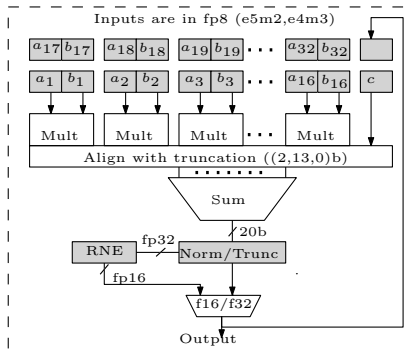
Input	CUDA/PTX Instr.	GPUs	SASS Instr.
fp8	mma.sync.aligned.m16n8k32	H100, H200, B200 L40S, Ada RTX 1000	HMMA.16816 QMMA.16816
	wgmma.mma_async.sync.m64nNk32	H100, H200 L40S, Ada RTX 1000	QGMMA.16832 NA
	tcgen05.mma.cta_group	B200	UTCQMMA
fp16, bf16	WMMA API	H100, H200, B200 L40S, Ada RTX 1000 A100, A2, A30 V100	HMMA.16816 HMMA.1688 HMMA.1688 HMMA.844
	tcgen05.mma.cta_group.kind=f16	B200	UTCHMMA
	tf19	WMMA API	H100, H200, B200 L40S, Ada RTX 1000 A100, A2, A30
tf19	mma.sync.aligned.m16n8k4/8	H100, H200, B200 L40S, Ada RTX 1000 A100, A2, A30	HMMA.1684/8 HMMA.1684 HMMA.1684
	tcgen05.mma.cta_group.kind=tf32	B200	UTCHMMA
	fp64	WMMA API	H100, H200, B200 A100

WMMA API for tf19 maps to HMMA.1684 for GPUs that support tf19 as input, but H100, H200, and B200 also support HMMA.1688 via `mma.sync.aligned.m16n8k8`

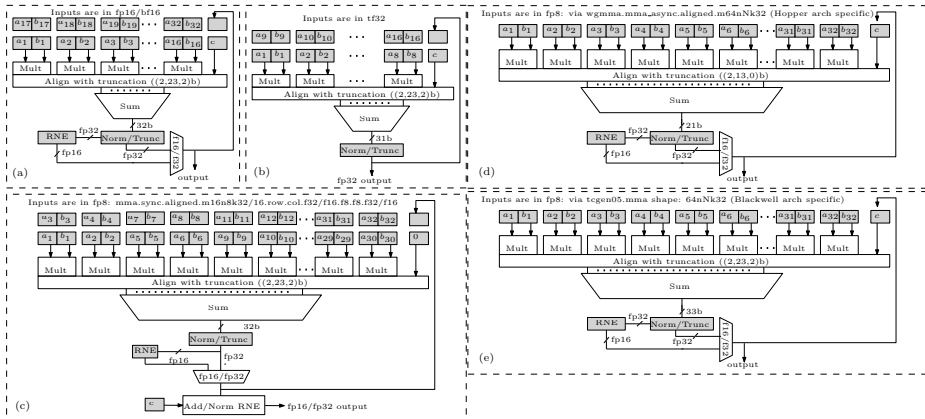


Results: A100





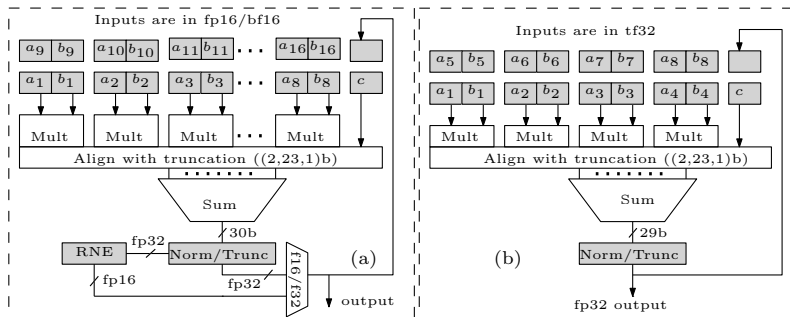
Results: H100, H200 and B200



Results: Summary

Input	Output	GPUs	Prd. Align.	Acc.	N_{FMA}	Final rounding
fp8	fp32	B200	(2,25)	33	32	Trunc.
		H100, H200	(2,13)	21	32	Trunc
		L40S, Ada RTX 1000	(2,13)	20	16	Trunc
fp8	fp16	B200	(2,25)	33	32	RNE
		H100, H200	(2,13)	21	32	RNE
		L40S, Ada RTX 1000	(2,13)	20	16	RNE
fp16, bf16	fp32	H100, H200, B200	(2,25)	32	16	Trunc
		L40S, Ada RTX 1000	(2,24)	30	8	Trunc
		A100, A2, A30	(2,24)	30	8	Trunc
		V100	(2,23)	28	4	Trunc
tf19	fp32	H100, H200, B200	(2,25)	31	8	Trunc
		L40S, Ada RTX 1000	(2,24)	29	4	Trunc
		A100, A2, A30	(2,24)	29	4	Trunc
fp64	fp64	H100, H200, B200	-	-	1	all 4
		A100	-	-	1	all 4

Extra results



- Products not normalised. Bit to left of implicit bit 1 may be set.
- Exponent, on alignment, is capped from below to -133, for bf16 and tf19.





Summary




- Matrix multiplication instruction is not standardised.
- We have future-proof methods to determine undocumented features.
- Bit-wise models of many GPUs are available in MATLAB at <https://github.com/north-numerical-computing/MATLAB-tensor-core>.
- Ozaki-style algorithms ([Mukunoki et al., 2020]) are not affected - rounding errors, under/overflows are avoided via intelligent slicing.
- We are now working on AMD CDNA 1-3 architectures.

Preprint

F. A. Khattak and M. Mikaitis. *Accurate Models of NVIDIA Tensor Cores*.
Preprint, arXiv:2512.07004 [cs.MS]. Apr. 2026.

Slides at <http://mmikaitis.github.io/talks>

-  IEEE P3109 Working Group
Interim Report on Binary Floating-point Formats for Machine Learning
<https://github.com/P3109/Public>
-  K. Hillesland and A. Lastra
GPU Floating-Point Paranoia
Preprint.
-  X. Tan, D. Boland, and G. A. Constantinides
FPGA Paranoia: Testing Numerical Properties of FPGA Floating Point IP-Cores
LNCS 7199. 2012.
-  B. Hickmann and D. Bradford
Experimental Analysis of Matrix Multiplication Functional Units
IEEE 26th Symposium on Computer Arithmetic. 2019.

-  M. Fasi, N. J. Higham, M. Mikaitis, and S. Pranesh
Numerical Behavior of NVIDIA Tensor Cores
PeerJ Comp. Sci. Feb. 2021
-  X. Li, A. Li, B. Fang, K. Swirydowicz, I. Laguna, G. Gopalakrishnan
FTTN: Feature-Targeted Testing for Numerical Properties of NVIDIA
& AMD Matrix Accelerators newblock
Preprint. arXiv:2403.00232. 2024.
-  D. Mukunoki, K. Ozaki, T. Ogita, T. Imamura
DGEMM Using Tensor Cores, and Its Accurate and Reproducible
Versions
LNCS 12151. 2020.



T. Mary and M. Mikaitis

Error analysis of matrix multiplication with narrow range floating-point arithmetic

SIAM J. Sci. Comput., 47. 2025.



M. Klöwer , P. D. Düben , and T. N. Palmer

Number formats, error mitigation, and scope for 16-bit arithmetics in weather and climate modeling analyzed with a shallow water model

J. Adv. Model. Earth Systems, 12. 2020.