Multiword ~~and multimodular~~ algorithms for emulating high accuracy with low precision

Theo Mary
Sorbonne Université, CNRS, LIP6

GAMM ANLA Workshop @ Università di Bologna
23-24 October 2025

The accelerator market

NVIDIA stock price history

Speed vs precision on NVIDIA GPUs

Peak performance (TFLOPS)

	Pascal 2016	Volta 2018	Ampere 2020	Hopper 2022	Blackwell 2025
fp64	5	8	20	67	40
fp32	10	16	20	67	80
tfloat32	–	–	160	495	1,100
fp16/bfloat16	20	125	320	990	2,250
fp8/int8	–	–	–	2,000	4,500
fp4	–	–	–	–	9,000

NVIDIA Hopper (H100) GPU

fp64/fp8 speed ratio:

- Hopper (2022): 30×
- Blackwell (2025): 112×

The limitations of lower precisions

	Signif. bits	Exp. bits	Range ($f_{\max}/f_{\min}$)	Unit roundoff u
fp128	113	15	$2^{32766} \approx 10^{9863}$	$2^{-114} \approx 1 \times 10^{-34}$
fp64	52	11	$2^{2046} \approx 10^{616}$	$2^{-53} \approx 1 \times 10^{-16}$
fp32	23	8	$2^{254} \approx 10^{76}$	$2^{-24} \approx 6 \times 10^{-8}$
tfloat32	10	8	$2^{254} \approx 10^{76}$	$2^{-11} \approx 5 \times 10^{-4}$
fp16	10	5	$2^{30} \approx 10^{9}$	$2^{-11} \approx 5 \times 10^{-4}$
bfloat16	7	8	$2^{254} \approx 10^{76}$	$2^{-8} \approx 4 \times 10^{-3}$
fp8 (E4M3)	3	4	$2^{15} \approx 3 \times 10^{4}$	$2^{-4} \approx 6 \times 10^{-2}$
fp8 (E5M2)	2	5	$2^{30} \approx 10^{9}$	$2^{-3} \approx 1 \times 10^{-1}$
fp6 (E2M3)	3	2	$2^{3} \approx 8$	$2^{-4} \approx 6 \times 10^{-2}$
fp6 (E3M2)	2	3	$2^{7} \approx 128$	$2^{-3} \approx 0.125$
fp4 (E2M1)	1	2	$2^{3} \approx 8$	$2^{-2} \approx 0.25$

Lower precisions:

- ☺ Faster, consume less memory and energy
- ☹ Lower accuracy and narrower range
- ⇒ Mixed precision algorithms

Standard model of FPA:

For any x such that $|x| \in [f_{\min}, f_{\max}]$,
$\mathrm{fl}(x) = x(1+\delta), \quad |\delta| \le u$

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

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

https://bit.ly/mixed-survey

CONTENTS

Habilitation manuscript

Approximate computing in numerical linear algebra: algorithms, analysis, and applications

Théo Mary
Chargé de recherche, CNRS

Mémoire d'habilitation à diriger des recherches

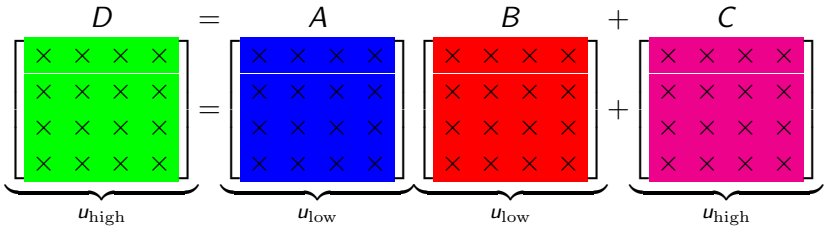
présenté et soutenu publiquement le 7 Octobre 2025

https://bit.ly/HDR_manuscript

Contents ix

NVIDIA GPU tensor cores

Tensor cores units available on NVIDIA GPUs carry out a mixed precision matrix multiply–accumulate ($u_{\text{high}} \equiv$ fp32 and $u_{\text{low}} \equiv$ fp16/fp8/fp4)



Element-wise multiplication of matrix A and B is performed with at least single precision. When .ctype or .dtype is .f32, accumulation of the intermediate values is performed with at least single precision. When both .ctype and .dtype are specified as .f16, the accumulation is performed with at least half precision.
The accumulation order, rounding and handling of subnormal inputs is unspecified.

Mixed precision MMA: model and error analysis

- We consider an MMA (matrix multiply–accumulate) unit that computes $C = AB$, $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times q}$, $C \in \mathbb{R}^{m\times q}$, as follows:
- First, we convert A and B to low precision:

$$\widetilde{A} = \mathrm{fl}_{\mathrm{low}}(A) = A + \Delta A, \quad |\Delta A| \le u_{\mathrm{low}}|A|,$$
$$\widetilde{B} = \mathrm{fl}_{\mathrm{low}}(B) = B + \Delta B, \quad |\Delta B| \le u_{\mathrm{low}}|B|.$$

- Second, we compute the product:

$$\begin{aligned}\widehat{C} &= \widetilde{A}\widetilde{B} + \Delta C, \qquad |\Delta C| \lesssim n u_{\mathrm{high}}|\widetilde{A}||\widetilde{B}|,\\ &= AB + \Delta AB + A\Delta B + \Delta A\Delta B + \Delta C\\ &= AB + E, \qquad |E| \le \big(\underbrace{2u_{\mathrm{low}} + u_{\mathrm{low}}^2}_{\text{Conversion}} + \underbrace{n u_{\mathrm{high}}}_{\text{Accumulation}}\big)|A||B|\end{aligned}$$

Mixed precision MMA: model and error analysis

- We consider an MMA (matrix multiply–accumulate) unit that computes $C = AB$, $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times q}$, $C \in \mathbb{R}^{m\times q}$, as follows:
- First, we convert A and B to low precision:

$$\widetilde{A} = \mathrm{fl}_{\mathrm{low}}(A) = A + \Delta A, \quad |\Delta A| \le u_{\mathrm{low}}|A|,$$
$$\widetilde{B} = \mathrm{fl}_{\mathrm{low}}(B) = B + \Delta B, \quad |\Delta B| \le u_{\mathrm{low}}|B|.$$

- Second, we compute the product:

$$\begin{aligned}\widehat{C} &= \widetilde{A}\widetilde{B} + \Delta C, \qquad |\Delta C| \lesssim n u_{\mathrm{high}}|\widetilde{A}||\widetilde{B}|,\\ &= AB + \Delta AB + A\Delta B + \Delta A\Delta B + \Delta C\\ &= AB + E, \qquad |E| \le (\underbrace{2u_{\mathrm{low}} + u_{\mathrm{low}}^2}_{\text{Conversion}} + \underbrace{n u_{\mathrm{high}}}_{\text{Accumulation}})|A||B|\end{aligned}$$

Evaluation method	Bound
Standard in precision u_{low}	$n u_{\mathrm{low}}$
Standard in precision u_{high}	$n u_{\mathrm{high}}$
Tensor cores	$2u_{\mathrm{low}} + n u_{\mathrm{high}}$

$\Rightarrow$ reduction by a factor $\min\big(n/2, u_{\mathrm{low}}/u_{\mathrm{high}}\big)$

[Blanchard, Higham, Lopez, M., Pranesh, SISC 2020]

Part I

fp32 emulation

Multiword arithmetic on mixed precision MMA units

- Step 1: compute the multiword decompositions

$$A \approx \sum_{i=0}^{s-1} A_i \quad \text{and} \quad B \approx \sum_{j=0}^{s-1} B_j$$

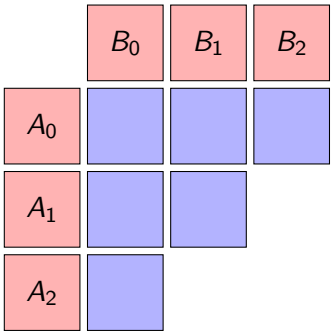
with A_i and B_j stored in precision u_{low}

- Step 2: compute the $s(s+1)/2$ leading products

$$C = \sum_{i+j<s} A_i B_j$$

with a mixed precision MMA with accumulation precision u_{high}

Example: **fp32 emulation** with bfloat16 tensor cores (28× speed ratio on Blackwell)
$u_{\text{low}} \equiv$ fp16, $u_{\text{high}} \equiv$ fp32, $s = 3$



Error analysis

Step 1: build

$$A_i = \mathrm{fl}_{\mathrm{low}}\left(A - \sum_{k=0}^{i-1} A_k\right), \quad B_j = \mathrm{fl}_{\mathrm{low}}\left(B - \sum_{k=0}^{j-1} B_k\right).$$

to obtain

$$A = \sum_{i=0}^{s-1} A_i + \Delta A, \qquad |\Delta A| \leq u_{\mathrm{low}}^s |A|,$$

$$B = \sum_{j=0}^{s-1} B_j + \Delta B, \qquad |\Delta B| \leq u_{\mathrm{low}}^s |B|.$$

Step 2: compute the s^2 products $A_i B_j$ by chaining calls to the MMA:

$$\widehat{C} = \sum_{i=0}^{s-1}\sum_{j=0}^{s-1} A_i B_j + \Delta C, \quad |\Delta C| \lesssim (n+s^2) u_{\mathrm{high}} |A||B|.$$

Overall

$$\widehat{C} = AB + E, \quad |E| \lesssim \left(2u_{\mathrm{low}}^s + u_{\mathrm{low}}^{2s} + (n+s^2)u_{\mathrm{high}}\right)|A||B|.$$

Dropping terms

$A_i = \mathrm{fl}_{\mathrm{low}}\left(A - \sum_{k=0}^{i-1} A_k\right)$ is the approximation residual from the first $i-1$ words

$$|A_i| \leq u_{\mathrm{low}}^{i}(1+u_{\mathrm{low}})|A|$$
$$|B_j| \leq u_{\mathrm{low}}^{j}(1+u_{\mathrm{low}})|B|$$
$$|A_i||B_j| \leq u_{\mathrm{low}}^{i+j}(1+u_{\mathrm{low}})^2|A||B|$$

$\Rightarrow$ Not all s^2 products A_iB_j need be computed! Skipping any product A_iB_j such that $i+j \geq s$ only adds $O(u_{\mathrm{low}}^s)$ error terms: $\widehat{C} = AB + E$,

$$|E| \leq \left(2u_{\mathrm{low}}^s + u_{\mathrm{low}}^{2s} + (n+s^2)u_{\mathrm{high}} + \sum_{i=0}^{s-1}(s-i-1)u_{\mathrm{low}}^{s+i}(1+u_{\mathrm{low}})^2\right)|A||B|.$$

- error to order u_{low}^s: constant $2 \to s+1$
- number of products: $s^2 \to s(s+1)/2$

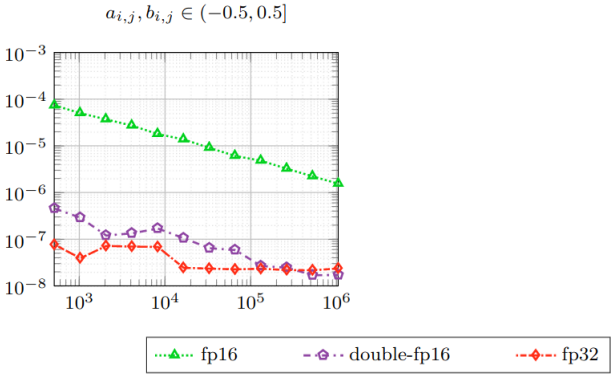
Summary of theory

Multiword MMA error bound (Fasi, Higham, Lopez, M., Mikaitis, SISC 2023)

The computed $\widehat{C}$ satisfies $|\widehat{C} - AB| \lesssim \big((s+1)u_{\text{low}}^{s} + nu_{\text{high}}\big)|A||B|$.

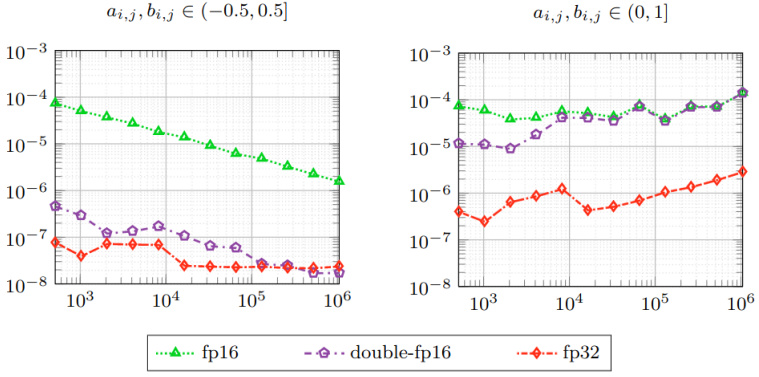
u_{high}	u_{low}		Error bound
2^{-24} (fp32)	2^{-11} (fp16)	$s=1$	$2\times 2^{-11} + n\times 2^{-24}$
		$s=2$	$3\times 2^{-22} + n\times 2^{-24}$
	2^{-8} (bfloat16)	$s=1$	$2\times 2^{-8} + n\times 2^{-24}$
		$s=2$	$3\times 2^{-16} + n\times 2^{-24}$
		$s=3$	$n\times 2^{-24}$

From theory to practice: a word of warning



- Approach works OK for matrices with random $[-1, 1]$ uniform entries...

From theory to practice: a word of warning



- Approach works OK for matrices with random $[-1, 1]$ uniform entries...
- ... but not $[0, 1]$ entries!!

From theory to practice: a word of warning

The explanation: the culprit is round to zero (RZ)

- fp32 uses the standard RTN, but tensor cores only support RZ [Fasi, Higham, Mikaitis, Pranesh, PeerJ CS 2021]
- With data of nonzero mean and RZ, most rounding errors happen in the same direction

$\Rightarrow$ Worst-case bound nu_{32} is attained with RZ, whereas with RTN we can usually replace it by $\sqrt{n}u_{32}$ [Higham and M., SISC 2019, 2020]

- Same error *bound* $\neq$ same error !

From theory to practice: a word of warning

The explanation: the culprit is round to zero (RZ)

- fp32 uses the standard RTN, but tensor cores only support RZ [Fasi, Higham, Mikaitis, Pranesh, PeerJ CS 2021]
- With data of nonzero mean and RZ, most rounding errors happen in the same direction

$\Rightarrow$ Worst-case bound nu_{32} is attained with RZ, whereas with RTN we can usually replace it by $\sqrt{n}u_{32}$ [Higham and M., SISC 2019, 2020]

- Same error *bound* $\neq$ same error !
- Possible cures:
  - Since the worst-case accumulation bound nu_{32} is attained $\Rightarrow$ reduce the bound ! e.g., with blocked summation/FABsum [Blanchard, Higham, M., SISC 2020]
  - Alternatively, avoid using tensor cores for accumulation [Ootomo and Yokota, IJHPCA 2022]

fp32 emulation with bfloat16-TC in cuBLAS

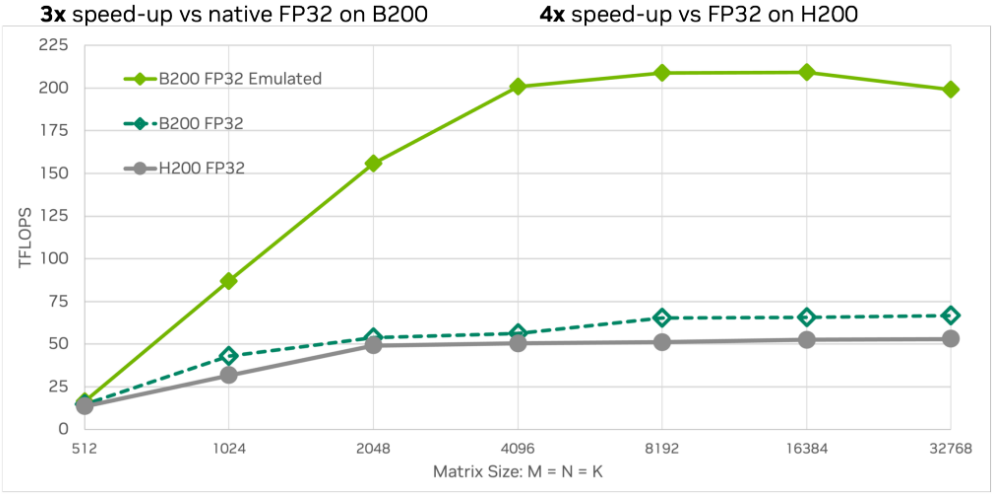


Figure courtesy of NVIDIA

Part II

fp64 emulation

Multiword integer decompositions

- Tensor cores do not provide fp64 accumulators. Can we nevertheless emulate fp64 accuracy?
- Ozaki scheme: decompose A and B such that $A_i B_j$ can be computed *exactly* [Ozaki, Ogita, Oishi, Rump, NumAlgs 2012]
- A particularly simple yet efficient approach is obtained by pushing this logic to the extreme: decompose A and B into *integers*! [Ootomo, Ozaki, Yokota, IJHPCA 2024]
  - Step 1: compute scaled integer approximations $A \approx D_A A'$ and $B \approx B' D_B$ (where A', B' have integer coefficients)
  - Step 2: compute $C' = A'B'$ with (multiword) integer arithmetic
  - Step 3: recover $C = D_A C' D_B$ (exact if scaling factors are powers of two)

Step 1: approximation error

Goal: find scalings D_A, D_B and integer A', B' such that $A \approx D_A A'$ and $B \approx B' D_B$

$\Rightarrow$ Block floating-point representations of A and B (rows of A and columns of B must share same exponent)

- Base-10 examples with 3-digit integers:

$$A = [1.234] = \underbrace{[10^{-2}]}_{D_A} \times \underbrace{[123]}_{A'} + \underbrace{[0.004]}_{\text{error}}$$

Step 1: approximation error

Goal: find scalings D_A, D_B and integer A', B' such that $A \approx D_A A'$ and $B \approx B' D_B$

$\Rightarrow$ Block floating-point representations of A and B (rows of A and columns of B must share same exponent)

- Base-10 examples with 3-digit integers:

$$A = \begin{bmatrix} 1.234 \\ 0.05678 \end{bmatrix} = \underbrace{\begin{bmatrix} 10^{-2} & \\ & 10^{-4} \end{bmatrix}}_{D_A} \times \underbrace{\begin{bmatrix} 123 \\ 568 \end{bmatrix}}_{A'} + \underbrace{\begin{bmatrix} 0.004 \\ -0.00002 \end{bmatrix}}_{\text{error}}$$

Step 1: approximation error

Goal: find scalings D_A, D_B and integer A', B' such that $A \approx D_A A'$ and $B \approx B' D_B$

$\Rightarrow$ Block floating-point representations of A and B (rows of A and columns of B must share same exponent)

- Base-10 examples with 3-digit integers:

$$A = \begin{bmatrix} 1.234 \\ 0.05678 \end{bmatrix} = \underbrace{\begin{bmatrix} 10^{-2} & \\ & 10^{-4} \end{bmatrix}}_{D_A} \times \underbrace{\begin{bmatrix} 123 \\ 568 \end{bmatrix}}_{A'} + \underbrace{\begin{bmatrix} 0.004 \\ -0.00002 \end{bmatrix}}_{\text{error}}$$

$$A = \begin{bmatrix} 1.234 & 0.05678 \end{bmatrix} = \underbrace{\begin{bmatrix} 10^{-2} \end{bmatrix}}_{D_A} \times \underbrace{\begin{bmatrix} 123 & 006 \end{bmatrix}}_{A'} + \underbrace{\begin{bmatrix} 0.004 & -0.00322 \end{bmatrix}}_{\text{error}}$$

Step 1: approximation error

Goal: find scalings D_A, D_B and integer A', B' such that $A \approx D_A A'$ and $B \approx B' D_B$

$\Rightarrow$ Block floating-point representations of A and B (rows of A and columns of B must share same exponent)

- Base-10 examples with 3-digit integers:

$$A = \begin{bmatrix} 1.234 \\ 0.05678 \end{bmatrix} = \underbrace{\begin{bmatrix} 10^{-2} & \\ & 10^{-4} \end{bmatrix}}_{D_A} \times \underbrace{\begin{bmatrix} 123 \\ 568 \end{bmatrix}}_{A'} + \underbrace{\begin{bmatrix} 0.004 \\ -0.00002 \end{bmatrix}}_{\text{error}}$$

$$A = \begin{bmatrix} 1.234 & 0.05678 \end{bmatrix} = \underbrace{\begin{bmatrix} 10^{-2} \end{bmatrix}}_{D_A} \times \underbrace{\begin{bmatrix} 123 & 006 \end{bmatrix}}_{A'} + \underbrace{\begin{bmatrix} 0.004 & -0.00322 \end{bmatrix}}_{\text{error}}$$

- In general, error $|e_{ij}|$ is less than $\text{precision} \times \max_j |a_{ij}|$ for A (conversely, $\max_i |b_{ij}|$ for B). A simpler but weaker normwise bound is

$$\|AB - D_A A' B' D_B\| \leq c(n) \times \text{precision} \times \|A\| \|B\|$$

Step 1: approximation error

Goal: find scalings D_A, D_B and integer A', B' such that $A \approx D_A A'$ and $B \approx B' D_B$

- In general, error $|e_{ij}|$ is less than $\text{precision} \times \max_j |a_{ij}|$ for A (conversely, $\max_i |b_{ij}|$ for B). A simpler but weaker normwise bound is

$$\|AB - D_A A' B' D_B\| \leq c(n) \times \text{precision} \times \|A\| \|B\|$$

$\Rightarrow$ Is this satisfactory?

- Argument for YES: several common algorithms can only guarantee normwise accuracy (underflow, Strassen, compression...)
- Argument for NO: standard algorithm guarantees componentwise accuracy (relative to $|A||B|$)

Step 1: approximation error

Goal: find scalings D_A, D_B and integer A', B' such that $A \approx D_A A'$ and $B \approx B' D_B$

- In general, error $|e_{ij}|$ is less than $\text{precision} \times \max_j |a_{ij}|$ for A (conversely, $\max_i |b_{ij}|$ for B). A simpler but weaker normwise bound is

$$\|AB - D_A A' B' D_B\| \leq c(n) \times \text{precision} \times \|A\| \|B\|$$

$\Rightarrow$ Is this satisfactory?

  - Argument for YES: several common algorithms can only guarantee normwise accuracy (underflow, Strassen, compression...)
  - Argument for NO: standard algorithm guarantees componentwise accuracy (relative to $|A||B|$)

- If NO: use extra digits! May need up to

$$\kappa_A = \max_i \frac{\max_j |a_{ij}|}{\min_j |a_{ij}|} \quad \text{for } A \quad \text{and} \quad \kappa_B = \max_j \frac{\max_i |b_{ij}|}{\min_i |b_{ij}|} \quad \text{for } B$$

[Abdelfattah, Dongarra, Fasi, Mikaitis, Tisseur, 2025]

Step 2: accumulation error

Goal: compute $C' = A'B'$, $A' \in \mathbb{Z}^{m\times n}$, $B' \in \mathbb{Z}^{n\times q}$, coefficients bounded by p

- Step 2a: compute the decompositions

  $$A' = \sum_{i=0}^{s-1} \gamma^{s-i-1} A_i \quad \text{and} \quad B' = \sum_{j=0}^{s-1} \gamma^{s-j-1} B_j$$

  with coefficients A_i and B_j bounded by $\gamma = \lceil p^{1/s} \rceil$

  $\Rightarrow$ exact if $\gamma \leq 2^{\sigma} - 1$, where σ is the storage bitsize (e.g., $\sigma = 7$ with int8-TC)

- Step 2b: compute each individual product:

  $$C_{ij} = A_i B_j$$

  in integer arithmetic by blocks of size b

  $\Rightarrow$ exact if $b\gamma^2 \leq 2^{\tau} - 1$, where τ is the accumulator bitsize (e.g., $\tau = 31$ with int8-TC)

- Step 2c: accumulate all the products in fp64:

  $$C = \sum_{i,j} \gamma^{2s-i-j-2} C_{ij}$$

  $\Rightarrow$ fp64 accumulation errors + dropping errors if we skip computing C_{ij} when $i + j \geq s$

fp64 emulation with int8-TC in cuBLAS

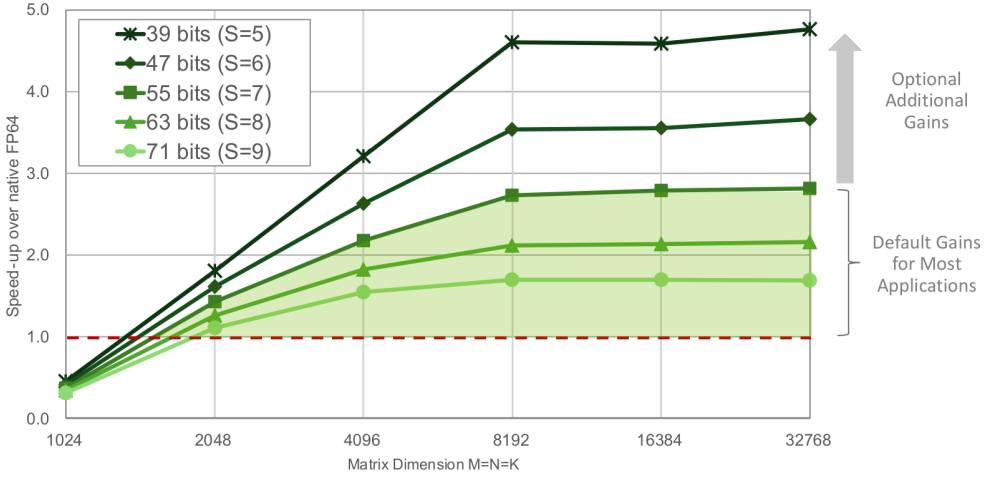


Figure courtesy of NVIDIA

Conclusion

- AI accelerators are headed towards lower and lower precisions, at the expense of the higher precisions. Can we still make use of them for scientific computing?
- Yes, thanks to emulation.
  - fp32 emulation with multiword + mixed precision MMA
  - fp64 emulation with multiword + integer arithmetic
  - (fp64 emulation with multimodular + integer arithmetic) [Ozaki, Uchino, Imamura, 2025]
- Many open questions and ongoing work
  - Adaptive precision choice in Step 1
  - Unbalanced integer decompositions in Step 2
  - Normwise vs componentwise accuracy: when is the latter really needed?
  - Microscaling (MX) formats
  - Emulation in higher level NLA algorithms

Thanks! Questions?