Multiword and multimodular algorithms for emulating high accuracy with low precision

Theo Mary
Sorbonne Université, CNRS, LIP6

RAIM 2025 @ ENS Lyon
3-7 November 2025

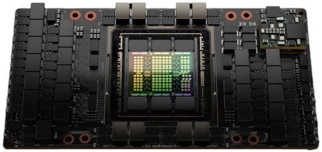
The accelerator market

NVIDIA stock price history

Speed vs precision on NVIDIA GPUs

Peak performance (TFLOPS)

	Pascal P100 2016	Volta V100 2018	Ampere A100 2020	Hopper GH200 SXM 2022	Blackwell GB200 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): $30\times$
- Blackwell (2025): $112\times$

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
- $\Rightarrow$ Mixed precision algorithms

Standard model of FPA:

For any x such that $|x| \in [f_{\min}, f_{\max}]$,

$$\mathrm{fl}(x) = x(1+\delta), \quad |\delta| \le u$$

Survey

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

SORBONNE UNIVERSITÉ

cnrs

LIP6

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

Outline

1. fp32 emulation — multiword approach
   [Blanchard, Higham, Lopez, M., Pranesh, SISC 2020]
   [Ootomo and Yokota, IJHPCA 2022]
   [Fasi, Higham, Lopez, M., Mikaitis, SISC 2023]
2. fp64 emulation — multiword approach
   [Ootomo, Ozaki, Yokota, IJHPCA 2024]
   [Uchino, Ozaki, Imamura, IJHPCA 2025]
   [Abdelfattah, Dongarra, Fasi, Mikaitis, Tisseur, 2025]
3. fp64 emulation — multimodular approach
   [Ozaki, Uchino, Imamura, 2025]
4. Ongoing work (with M. Fasi and M. Mikaitis)

Part I

fp32 emulation — multiword approach

NVIDIA GPU tensor cores

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

$$\begin{array}{ccccccc} D & = & A & & B & + & C \\ \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{u_{\mathrm{high}}} & = & \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{u_{\mathrm{low}}} & & \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{u_{\mathrm{low}}} & + & \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{u_{\mathrm{high}}} \end{array}$$

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:

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

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:

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

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

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

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

Multiword arithmetic on mixed precision MMA units

- Step a) compute the multiword decompositions

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

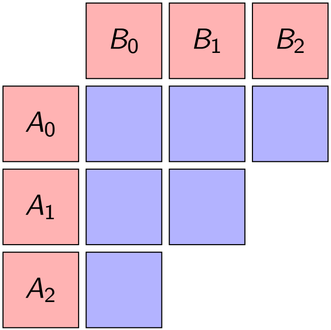
with A_i and B_j stored in precision u_{low}

- Step b) 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 ($28\times$ speed ratio on Blackwell)
$u_{\mathrm{low}} \equiv$ fp16, $u_{\mathrm{high}} \equiv$ fp32, $s = 3$



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

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

fp32 emulation with bfloat16-TC in cuBLAS

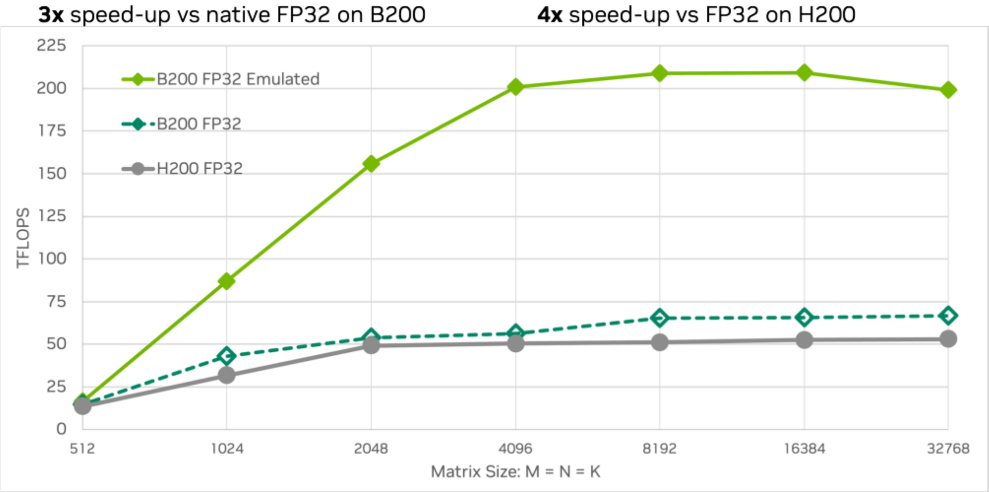


Figure courtesy of NVIDIA

Part II

fp64 emulation — multiword approach

Ozaki-I scheme

- 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

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

$\Rightarrow$ 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:

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

Step 1: approximation error

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

$\Rightarrow$ 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:

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

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

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:

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

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

- For integers bounded by p, the error $|e_{ij}|$ is less than $p^{-1} \times \max_j |a_{ij}|$ for A (conversely, $p^{-1} \times \max_i |b_{ij}|$ for B), A simpler but weaker normwise bound is

$$\|AB - D_A A' B' D_B\| \leq c(n) \times p^{-1} \times \|A\| \|B\|$$

Step 1: approximation error

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

- For integers bounded by p, the error $|e_{ij}|$ is less than $p^{-1} \times \max_j |a_{ij}|$ for A (conversely, $p^{-1} \times \max_i |b_{ij}|$ for B), A simpler but weaker normwise bound is

$$\|AB - D_A A' B' D_B\| \leq c(n) \times p^{-1} \times \|A\| \|B\|$$

$\Rightarrow$ 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: extra precision and adaptive choice of p

- Since error $|e_{ij}|$ is less than $p^{-1} \times \max_j |a_{ij}|$ for A and $p^{-1} \times \max_i |b_{ij}|$ for B, use extra digits! May need up to $p \leftarrow \max(\kappa_A, \kappa_B) \times p$, where

$$\kappa_A = \max_i \frac{\max_j |a_{ij}|}{\min_j |a_{ij}|}, \qquad \kappa_B = \max_j \frac{\max_i |b_{ij}|}{\min_i |b_{ij}|}$$

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

- This is a very pessimistic bound. Can be sharpened in several ways:
- Use different integer sizes:
  - for A and B, $p_A = \kappa_A \times p$ and $p_B = \kappa_B \times p$
  - for different rows of A (columns of B), e.g., $p_A^{(i)} = \frac{\max_j |a_{ij}|}{\min_j |a_{ij}|} \times p$
  - for block-columns of A (block-rows of B)
- More importantly, a large relative error on $|a_{ij}|$ is only problematic if there exists a large $|b_{jk}|$ in front of it $\Rightarrow$ given a row x^T of A and a column y of B, compute $z = x \circ y$, $\kappa = \frac{\max |x| \max |y|}{\max |z|}$, and set $p \leftarrow p \times \kappa$ [NVIDIA, 2025]

Step 2: accumulation error

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

- 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

- Step 2b) compute each individual product:

$$C_{ij} = A_i B_j$$

in integer arithmetic

$\Rightarrow$ exact if $n\gamma^2 \leq 2^\tau - 1$, where τ is the accumulator bitsize

- 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 $i + j \geq s$

Step 2: further details and optimizations

- Which of $\gamma \leq 2^\sigma - 1$ and $n\gamma^2 \leq 2^\tau - 1$ is the limiting condition?

$\Rightarrow$ Depends!

uniform (e.g., fp32)	$\sigma = \tau$		$\Rightarrow \tau$ is limiting
fp16-TC	$\sigma = 11$	$\tau = 24$	$\Rightarrow \tau$ is limiting for $n > 4$
bfloat16-TC	$\sigma = 8$	$\tau = 24$	$\Rightarrow \tau$ is limiting for $n > 2^8$
int8-TC	$\sigma = 7$	$\tau = 31$	$\Rightarrow \sigma$ is limiting for $n < 2^{17}$

- When τ is limiting, use blocking to replace n with block size: $b\gamma^2 < 2^\tau - 1$

$$A_i B_j = \begin{bmatrix} A_i^{(1)} & \dots & A_i^{(n/b)} \end{bmatrix} \begin{bmatrix} B_j^{(1)} \\ \vdots \\ B_j^{(n/b)} \end{bmatrix} \quad \Rightarrow \quad C = \sum_{i,j} \gamma^{2s-i-j-2} \sum_{k=1}^{n/b} A_i^{(k)} B_j^{(k)}$$

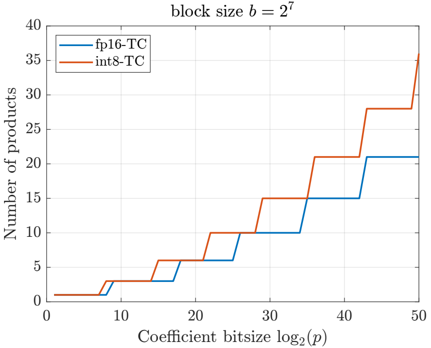
- When σ is limiting, the number of fp64 additions can be reduced by summing C_{ij} for fixed $i + j$ together in integer arithmetic [Uchino, Ozaki, Imamura, IJHPCA 2025]

Number of products vs precision — multiword

- Precision determined by integer bitsize of A' and B':

$$\log_2(p) \approx \log_2(\gamma^s) \approx s \times \min\left(\sigma, \frac{\tau - \log_2 b}{2}\right)$$

- Number of products equal to $\frac{s(s+1)}{2}$ (with dropping) $\Rightarrow$ quadratic in s



fp64 emulation with int8-TC in cuBLAS

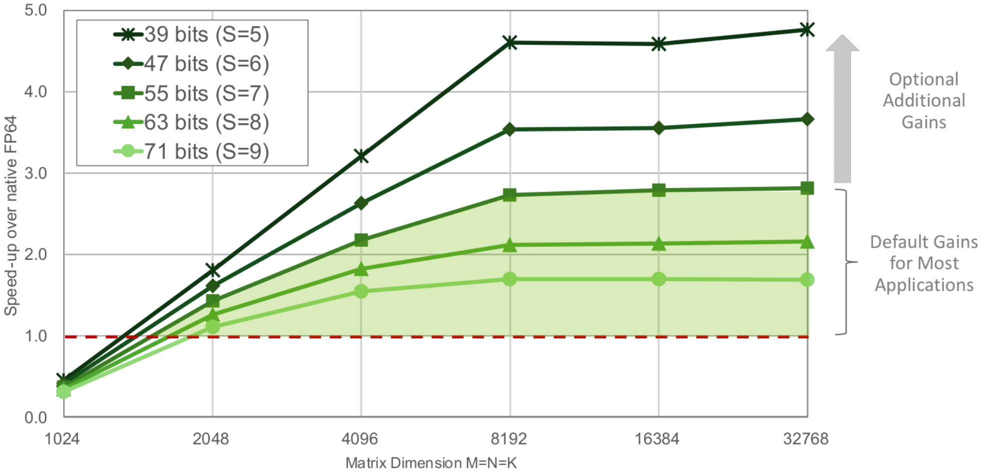


Figure courtesy of NVIDIA

Part III

fp64 emulation — multimodular approach

Ozaki-II scheme

Ozaki-II scheme is based on multimodular matrix multiplication [Ozaki, Uchino, Imamura, 2025]

- 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 multimodular integer arithmetic
- Step 3: recover $C = D_A C' D_B$ (exact if scaling factors are powers of two)

Chinese remainder theorem

- Let $m_1, \dots, m_s$ be pairwise coprime integers (called moduli) and let $M = \prod_{i=1}^{s} m_i$. For given remainders $a_1 < m_1, \dots, a_s < m_s$, there exists a unique $x < M$ such that $x \equiv a_i \bmod m_i$ for $i = 1, \dots, s$. Moreover x is given by

$$x = \sum_{i=1}^{s} a_i M_i N_i \bmod M, \quad \text{with } M_i = M/m_i \text{ and } M_i N_i \equiv 1 \bmod m_i.$$

- In other words, if $x \bmod m_i$ is known for sufficiently many m_i (for sufficiently large $M = \prod_{i=1}^{s} m_i$), then x can be recovered.

- Key observation: $C = AB \bmod m_i = (A \bmod m_i)(B \bmod m_i) \bmod m_i$, and the coefficients of $A \bmod m_i$ and $B \bmod m_i$ are less than m_i! $\Rightarrow$ use modular reductions to reduce the coefficient sizes

Step 2 with multimodal approach

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

- Step 2a) compute the modular reductions

  $A_i = A' \bmod m_i$ and $B_i = B' \bmod m_i$

  with coefficients A_i and B_i bounded by $\gamma = \max_i m_i$

  $\Rightarrow$ exact if $\gamma \leq 2^\sigma - 1$, where σ is the storage bitsize

- Step 2b) compute each individual product:

  $$C_i = A_i B_i$$

  in integer arithmetic

  $\Rightarrow$ exact if $n\gamma^2 \leq 2^\tau - 1$, where τ is the accumulator bitsize

- Step 2c) recover the true product in fp64:

  $$C = \sum_{i=1}^{s} C_i M_i N_i \bmod M$$

  $\Rightarrow$ fp64 accumulation errors
  $+$ CRT condition:
  $M > np^2 \Rightarrow \gamma^s \gtrsim np^2$
  ($n \leftarrow b$ with blocking)

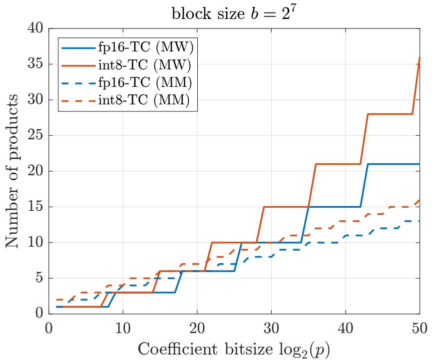
Number of products vs precision — multimodular

- Precision determined by integer bitsize of A' and B':

$$\log_2(p) \approx \frac{\log_2(\gamma^s) - \log_2(b)}{2} \approx \frac{s \times \min\left(\sigma, \frac{\tau - \log_2 b}{2}\right) - \log_2(b)}{2}$$

- Number of products equal to s $\Rightarrow$ linear in s

$\Rightarrow$ Compared to multiword for fixed s: less precision (by about a factor of 2) but less products (by about a factor $s/2$) $\Rightarrow$ cutoff around $s_{\mathrm{MW}} \approx 4$ (10 products)



Part IV

Ongoing work

(with M. Fasi and M. Mikaitis)

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
- Many open questions and ongoing work!

Unbalanced multiword decompositions in modular matrix products

Related problem from computer algebra:

$$\boxed{\text{Compute } C = AB \bmod p,\ A \in \mathbb{Z}^{m\times n},\ B \in \mathbb{Z}^{n\times q}, \text{ coefficients bounded by } p}$$

- Step 2a) compute the **unbalanced** decompositions

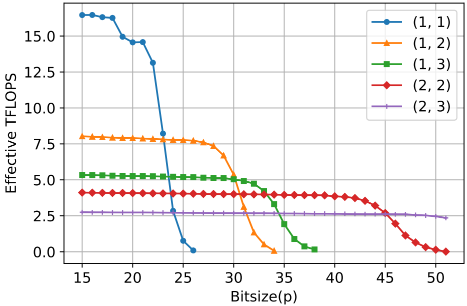
$$A = \sum_{i=0}^{s_A-1} \alpha^{s_A-i-1} A_i \quad \text{and} \quad B = \sum_{j=0}^{s_B-1} \beta^{s_B-j-1} B_j$$

  with coefficients A_i and B_j bounded by $\alpha = \lceil p^{1/s_A} \rceil$ and $\beta = \lceil p^{1/s_B} \rceil$

- Step 2b) compute the $s_A s_B$ products

$$C = \sum_{i,j} \alpha^i \beta^j A_i B_j \bmod p$$

  by blocks of size b



Bound on p (Berthomieu, Graillat, Lesnoff, M., 2025)

The product is computed exactly in τ-bit arithmetic if $\log_2(p) \le \frac{s_A s_B(\tau - \log_2(b))}{s_A + s_B}$.

Unbalanced multiword decompositions in Step 2

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

$$A' \approx \sum_{i=0}^{s_A-1} \alpha^{s_A-i-1} A_i \quad \text{and} \quad B' \approx \sum_{j=0}^{s_B-1} \beta^{s_B-j-1} B_j$$

with coefficients A_i and B_j bounded by $\alpha = \lceil p^{1/s_A} \rceil$ and $\beta = \lceil p^{1/s_B} \rceil$

$\Rightarrow$ exact if $\max(\alpha, \beta) \leq 2^\sigma - 1$, where σ is the storage bitsize

- Step 2b) compute each individual product:

$$C_{ij} = A_i B_j$$

in integer arithmetic

$\Rightarrow$ exact if $n\alpha\beta \leq 2^\tau - 1$, where τ is the accumulator bitsize

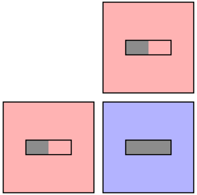
- Step 2c) accumulate all the products in fp64:

$$C = \sum_{i,j} \alpha^{s_A-i-1} \beta^{s_B-j-1} C_{ij}$$

$\Rightarrow$ fp64 accumulation errors + **dropping errors** if we skip computing some of the C_{ij}

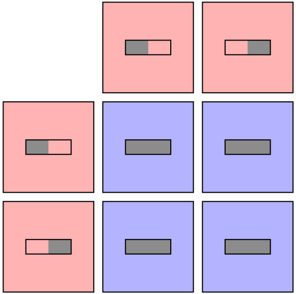
- Assuming τ is limiting, precision determined by:

$$\log_2(p) \approx \frac{s_A s_B(\tau - \log_2(b))}{s_A + s_B}$$



- Assuming τ is limiting, precision determined by:

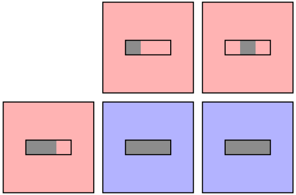
$$\log_2(p) \approx \frac{s_A s_B(\tau - \log_2(b))}{s_A + s_B}$$



Number of products vs precision — unbalanced multiword

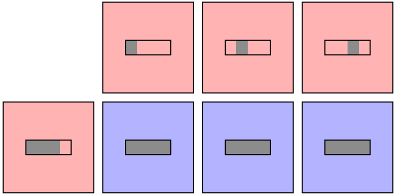
- Assuming τ is limiting, precision determined by:

$$\log_2(p) \approx \frac{s_A s_B(\tau - \log_2(b))}{s_A + s_B}$$



- Assuming τ is limiting, precision determined by:

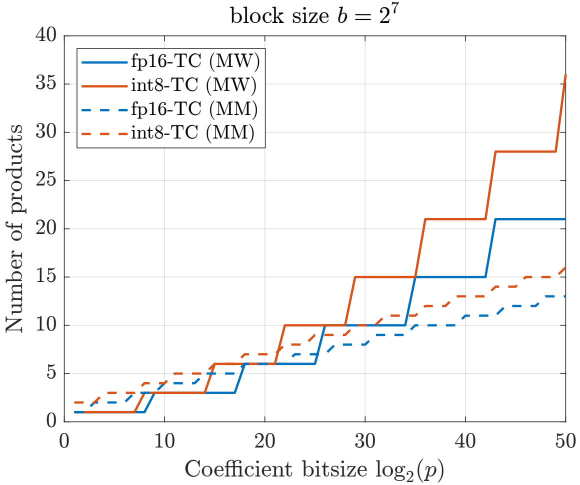
$$\log_2(p) \approx \frac{s_A s_B(\tau - \log_2(b))}{s_A + s_B}$$



Number of products vs precision — unbalanced multiword

- Assuming τ is limiting, precision determined by:

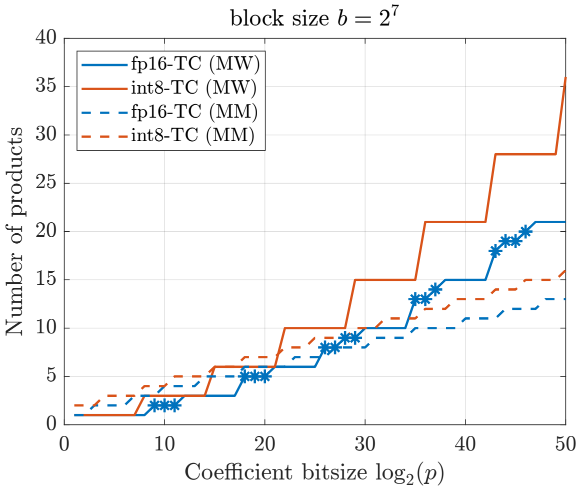
$$\log_2(p) \approx \frac{s_A s_B(\tau - \log_2(b))}{s_A + s_B}$$



Number of products vs precision — unbalanced multiword

- Assuming τ is limiting, precision determined by:

$$\log_2(p) \approx \frac{s_A s_B(\tau - \log_2(b))}{s_A + s_B}$$

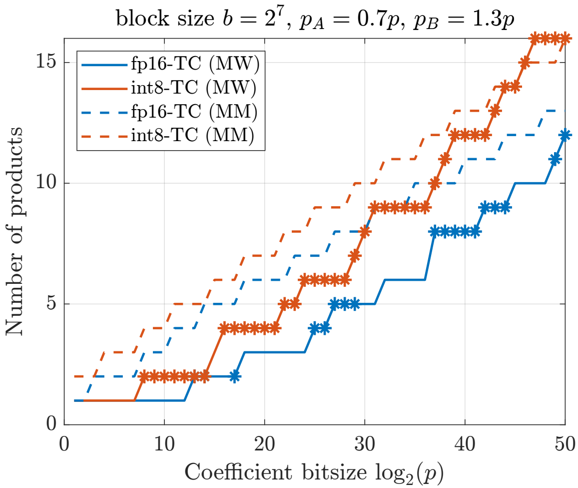


$*$ = unbalanced decomposition

Number of products vs precision — unbalanced multiword

- Assuming τ is limiting, precision determined by:

$$\log_2(p) \approx \frac{s_A s_B(\tau - \log_2(b))}{s_A + s_B}$$

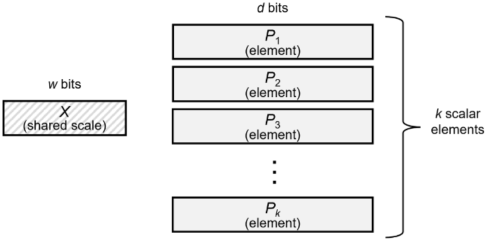


*** = unbalanced decomposition**

OCP microscaling (MX) formats

Format Name	Element Data Type	Element Bits (d)	Scaling Block Size (k)	Scale Data Type	Scale Bits (w)
MXFP8	FP8 (E5M2)	8	32	E8M0	8
	FP8 (E4M3)				
MXFP6	FP6 (E3M2)	6	32	E8M0	8
	FP6 (E2M3)				
MXFP4	FP4 (E2M1)	4	32	E8M0	8
MXINT8	INT8	8	32	E8M0	8

Table 1. Format names and parameters of concrete MX-compliant formats.



- Clearly of interest to reduce approximation error in Step 1
- Permuting rows of A / columns of B will become important!
- However, minimizing the error involves a complex combinatorial problem

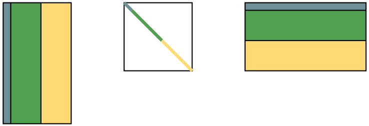
Applications of emulation in NLA

- Emulation provides a somewhat continuous level of precision (tunable via s)

$\Rightarrow$ many NLA computations can leverage this !

- Iterative refinement/preconditioned iterative solvers:
  [Amestoy, Buttari, Higham, L'Excellent, M., Vieublé, SIMAX 2025]
  [Buttari, Liu, M., Vieublé, 2025]
  emulate factorization/preconditioner with s chosen depending on $\kappa(A)$

- Adaptive precision low-rank approximations:
  [Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M., IMAJNA 2022]
  [Buttari, M., Pacteau, SISC 2025]
  decrease s as factorization progresses



- Adaptive precision tile factorizations:
  [Abdulah et al., IPDPS 2022]
  use different s for different tiles