SIAM LA 2021

May 20, 2021

Multiple Word Arithmetic with GPU Tensor Cores: Theory and Practice

Theo Mary

Sorbonne Université, CNRS, LIP6

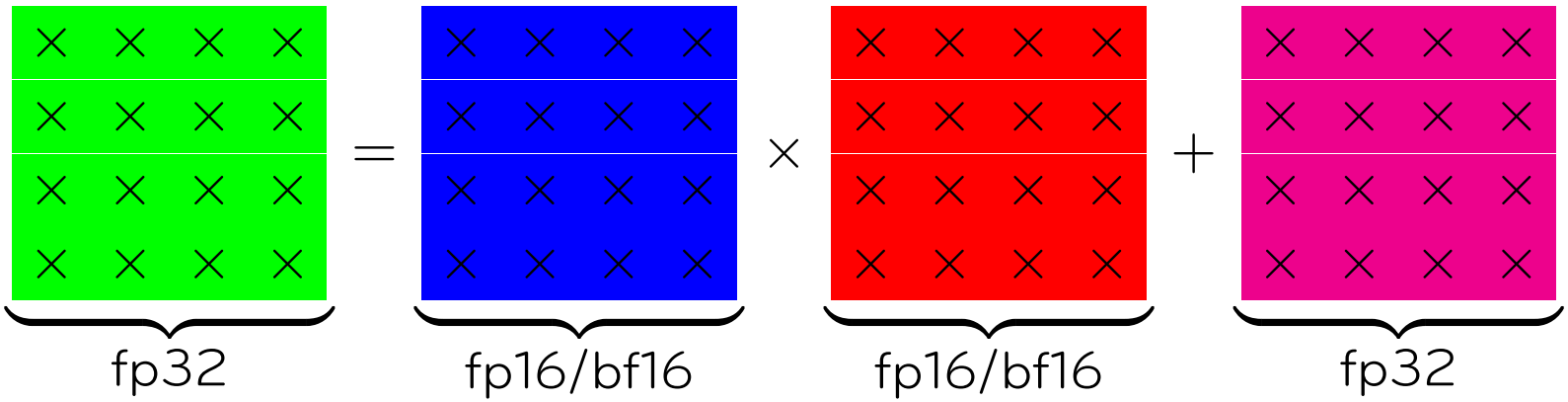
https://www-pequan.lip6.fr/~tmary/

Slides available at https://bit.ly/la21multiword

Joint work with Massimiliano Fasi (Örebro Univ.), Nicholas J. Higham, Mantas Mikaitis, Srikara Pranesh (Univ. Manchester), Florent Lopez (LSTC, Ansys).

GPU tensor cores

Tensor cores compute $D = C + AB$:



fp32 $\rightarrow$ fp16/bf16 speedup evolution:

P100: 2× V100: 8× A100: 16×

Matrix multiplication with tensor cores

Let $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ and compute $C = AB$.
The computed $\widehat{C}$ satisfies

$$|\widehat{C} - C| \leq nu_{16}|A||B| \qquad \text{in standard fp16/bf16 arithmetic}$$
$$|\widehat{C} - C| \leq (2u_{16} + nu_{32})|A||B| \qquad \text{with tensor cores}$$
$$|\widehat{C} - C| \leq nu_{32}|A||B| \qquad \text{in standard fp32 arithmetic}$$

Blanchard, Higham, Lopez, M., Pranesh (2020) .

- Tensor cores greatly reduce the impact of error accumulation
- But error still depends on u_{16} because of the conversion of A and B

$\Rightarrow$ Can we get rid of it to achieve an **accuracy equivalent to fp32 ?**

Multiword arithmetic

- Represent high precision number as the unevaluated sum of lower precision numbers
- Double–double arithmetic:

$$x = \underbrace{x_1}_{\text{fp64}} + \underbrace{x_2}_{\text{fp64}}$$

$\Rightarrow$ x has up to $2 \times 53 = 106$ significand bits $\approx 10^{-32}$ precision

- Less than fp128 (113 significand bits), but much faster, because computations rely on fp64 arithmetic
- Need for error-free transformations makes it much slower than fp64 $\Rightarrow$ double–single arithmetic not meaningful on most processors

Double–half and triple–half arithmetics

	Signif. bits	Exp. bits	Range	Unit roundoff u
fp32	24	8	$10^{\pm 38}$	6×10^{-8}
fp16	11	5	$10^{\pm 5}$	5×10^{-4}
bfloat16	8	8	$10^{\pm 38}$	4×10^{-3}

Let $x \in \mathbb{R}$ and $u_s = 2^{-24}$

$$x = \underbrace{x_1}_{\text{fp16}} + \underbrace{x_2}_{\text{fp16}} + \epsilon \qquad |\epsilon| \leq 4u_s$$

$$x = \underbrace{x_1}_{\text{bfloat16}} + \underbrace{x_2}_{\text{bfloat16}} + \underbrace{x_3}_{\text{bfloat16}} + \epsilon \qquad |\epsilon| \leq u_s$$

Double–half arithmetic with tensor cores

Apply this elementwise to $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$:

$$A = A_1 + A_2, \qquad B = B_1 + B_2$$

and compute $C = AB$ as

$$C \approx \sum_{i,j} A_i B_j \quad \text{using tensor cores}$$

GPU tensor cores provide a new perspective:

- Intermediate computations are done in fp32 $\Rightarrow$ no need for error-free transformations!
- Double–fp16 $\Rightarrow 4\times$ more flops (can be reduced to $3\times$)
- Triple–bfloat16 $\Rightarrow 9\times$ more flops (can be reduced to $6\times$)
- Tensor cores $8\times$–$16\times$ faster than fp32

$\Rightarrow$ **Multiword half arithmetic potentially faster at same accuracy!**

Related work

Several recent papers around this idea:

- S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng and J. S. Vetter, NVIDIA Tensor Core Programmability, Performance & Precision, IPDPSW 2018. Double–fp16 arithmetic for GEMM with Tensor Cores.
- A. Sorna, X. Cheng, E. D'Azevedo, K. Won and S. Tomov, Optimizing the Fast Fourier Transform Using Mixed Precision on Tensor Core Hardware, HiPCW 2018. Double–fp16 arithmetic for FFT with Tensor Cores.
- G. Henry, P. T. P. Tang and A. Heinecke, Leveraging the bfloat16 Artificial Intelligence Datatype For Higher-Precision Computations, ARITH 2019. Triple–bfloat16 arithmetic.
- D. Mukunoki, K. Ozaki, T. Ogita, T. Imamura, DGEMM Using Tensor Cores, and Its Accurate and Reproducible Versions, ISC 2020. Double–fp16 arithmetic with Tensor and with fp64 target.
- Several others

Block FMA framework

We use the Block FMA framework from Blanchard et al. (2020)

- $A \in \mathbb{R}^{b_1 \times b}$, $B \in \mathbb{R}^{b \times b_2}$, and $C \in \mathbb{R}^{b_1 \times b_2}$,

$$\underbrace{D}_{u_{\text{high}}} = \underbrace{C}_{u_{\text{high}}} + \underbrace{A}_{u_{\text{low}}} \underbrace{B}_{u_{\text{low}}}$$

- Internal computation $C + AB$ is done in precision u_{high}

	b_1	b	b_2	u_{low}	u_{high}
Google TPU v1	256	256	256	bfloat16	fp32
Google TPU v2	128	128	128	bfloat16	fp32
NVIDIA Volta	4	4	4	fp16	fp32
NVIDIA Ampere	8	8	4	fp16	fp32
NVIDIA Ampere	8	8	4	bfloat16	fp32
NVIDIA Ampere	4	8	4	tfloat32	fp32
Intel NNP-T	32	32	32	bfloat16	fp32
Armv8-A	2	4	2	bfloat16	fp32

General error analysis

For any $x \in \mathbb{R}$ and $p > 0$, let

$$\begin{aligned} x_1 &= \mathrm{fl}_{\mathrm{low}}(x) \\ x_2 &= \mathrm{fl}_{\mathrm{low}}(x - x_1) \\ &\vdots \\ x_p &= \mathrm{fl}_{\mathrm{low}}\left(x - \sum_{i=1}^{p-1} x_i\right) \end{aligned}$$

We obtain

$$x = \sum_{i=1}^{p} x_i + \Delta x, \quad |\Delta x| \leq u_{\mathrm{low}}^p |x|.$$

Using this representation elementwise on A and B:

$$A = \sum_{i=1}^{p} A_i + \Delta A, \quad |\Delta A| \leq u_{\mathrm{low}}^p |A|,$$

$$B = \sum_{j=1}^{p} B_j + \Delta B, \quad |\Delta B| \leq u_{\mathrm{low}}^p |B|.$$

Then the product $C = AB$ is given by

$$C = \sum_{i=1}^{p} \sum_{j=1}^{p} A_i B_j + A\,\Delta B + \Delta A\,B - \Delta A\,\Delta B.$$

Compute the p^2 products $A_i B_j$ by chaining calls to the block FMA:

$$\widehat{C} = C + \Delta C, \quad |\Delta C| \leq (n + p^2) u_{\text{high}} |A||B|.$$

Overall

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

$x_k = \mathrm{fl}_{\mathrm{low}}\left(x - \sum_{i=1}^{k-1} x_i\right)$ is the approximation residual from the first $k-1$ words

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

$\Rightarrow$ Not all p^2 products A_iB_j need be computed! Skipping any product A_iB_j such that $i+j > p+1$ yields $\widehat{C} = AB + E$,

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

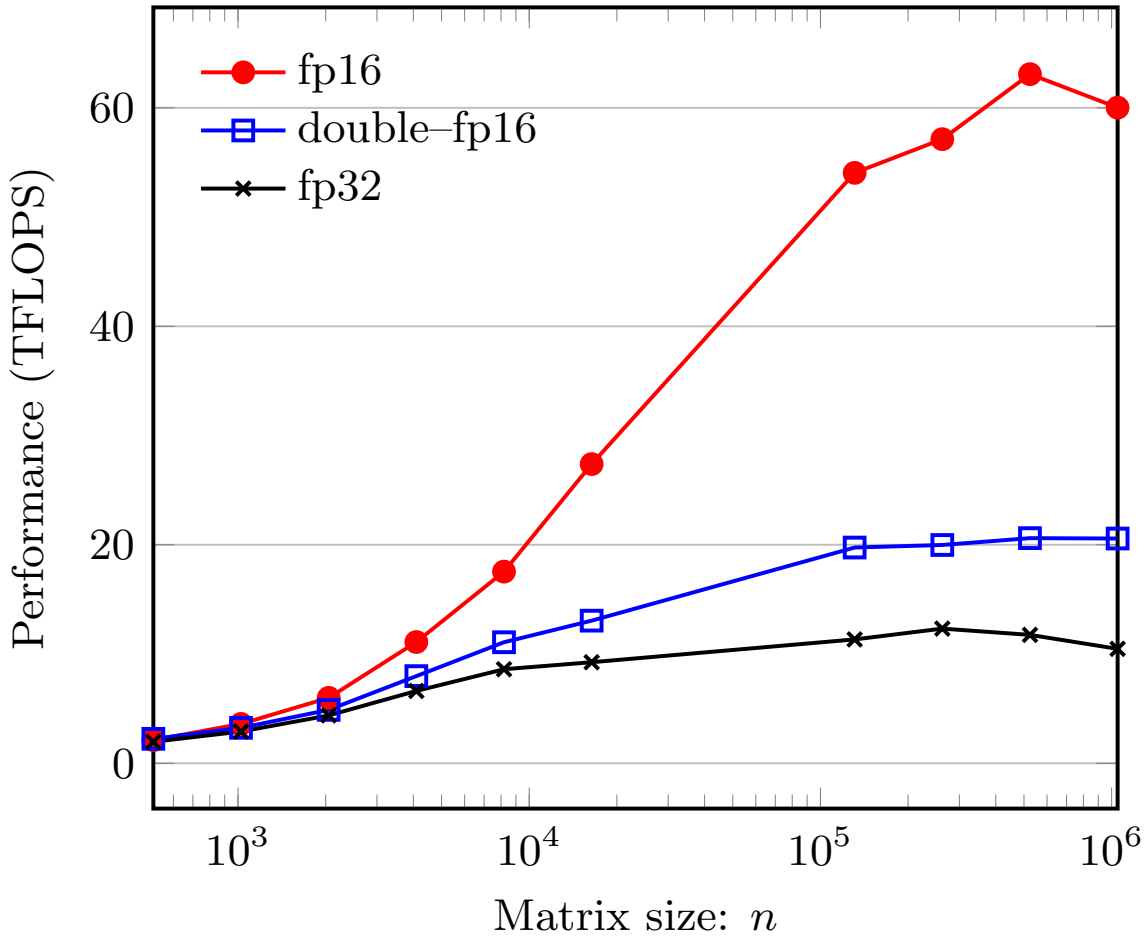
- number of products: $p^2 \to p(p+1)/2$
- error to order u_{low}^{p}: constant $2 \to p+1$

Summary of theory

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

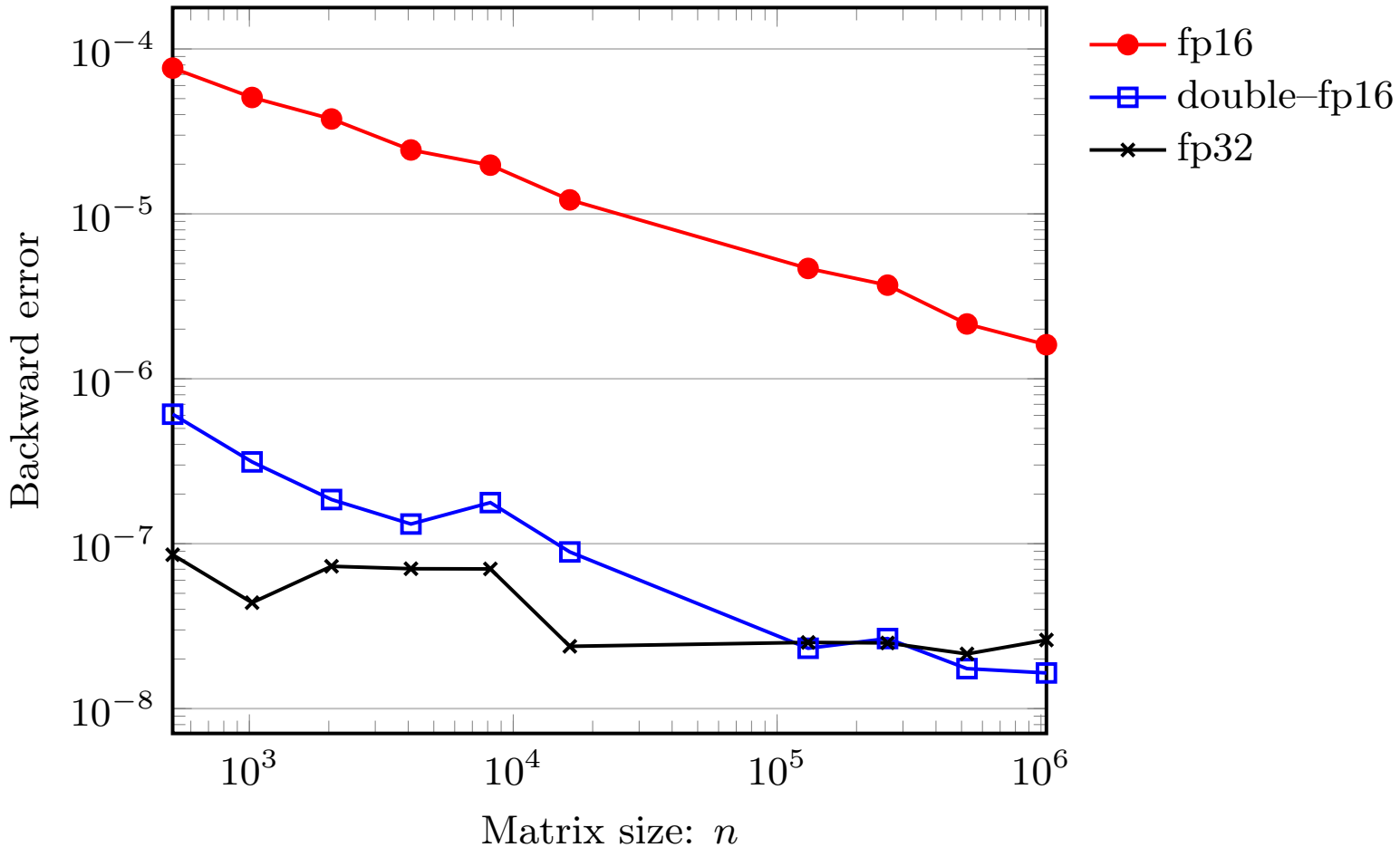
Encompasses existing approaches and some new ones

From theory to practice



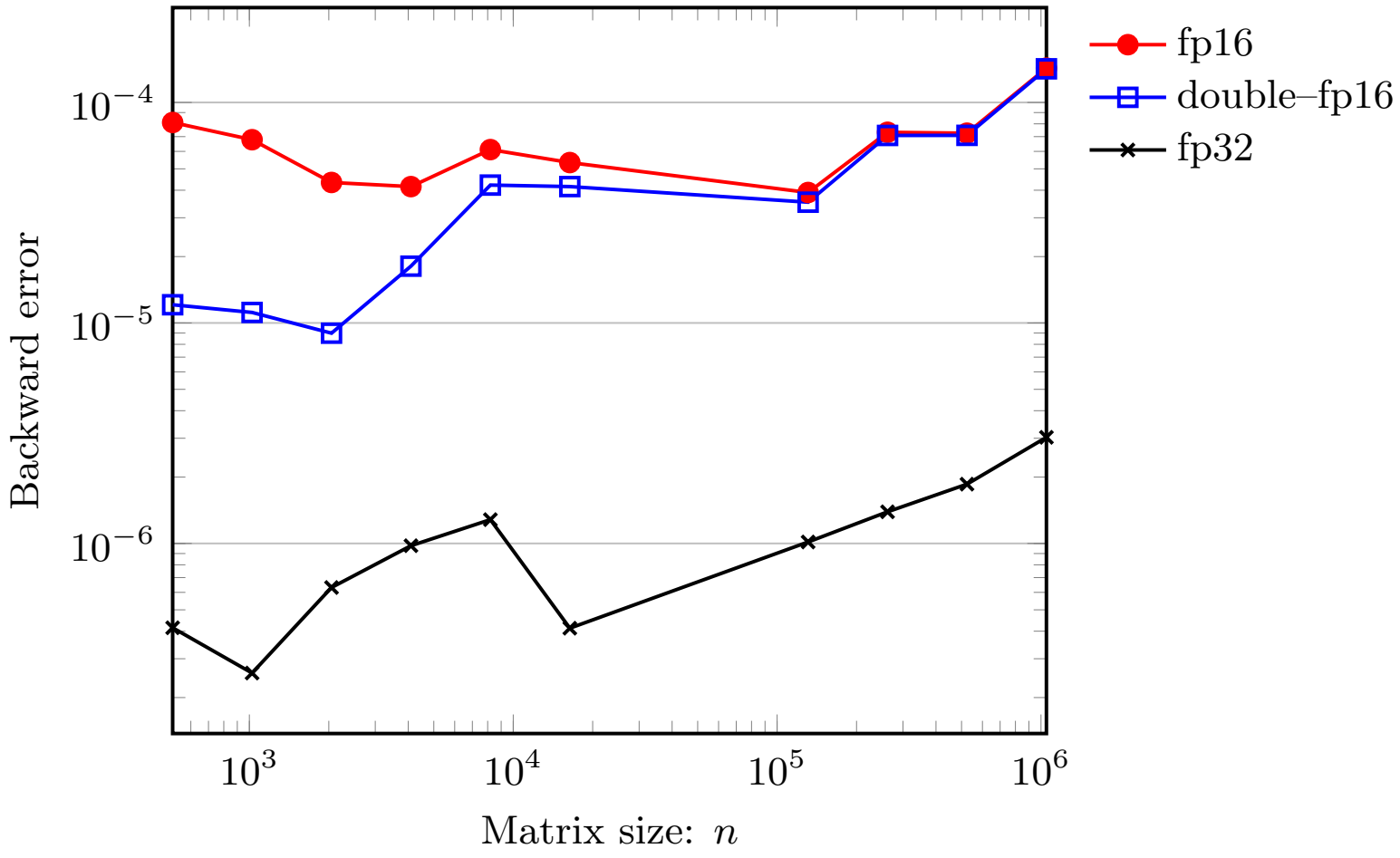
- Double–fp16 up to 2× faster than fp32

From theory to practice



- Double–fp16 up to $2\times$ faster than fp32
- Similar backward error for matrices with random $[-1, 1]$ uniform entries (decreasing error is expected Higham and M. (2020))

From theory to practice



- Double–fp16 up to $2\times$ faster than fp32
- Similar backward error for matrices with random $[-1, 1]$ uniform entries (decreasing error is expected Higham and M. (2020))
- $[0, 1]$ uniform entries!!

The issue

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

- fp32 uses the standard RTN, but tensor cores only support RZ
  Fasi, Higham, Mikaitis, Pranesh (2020)
- 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. (2019)

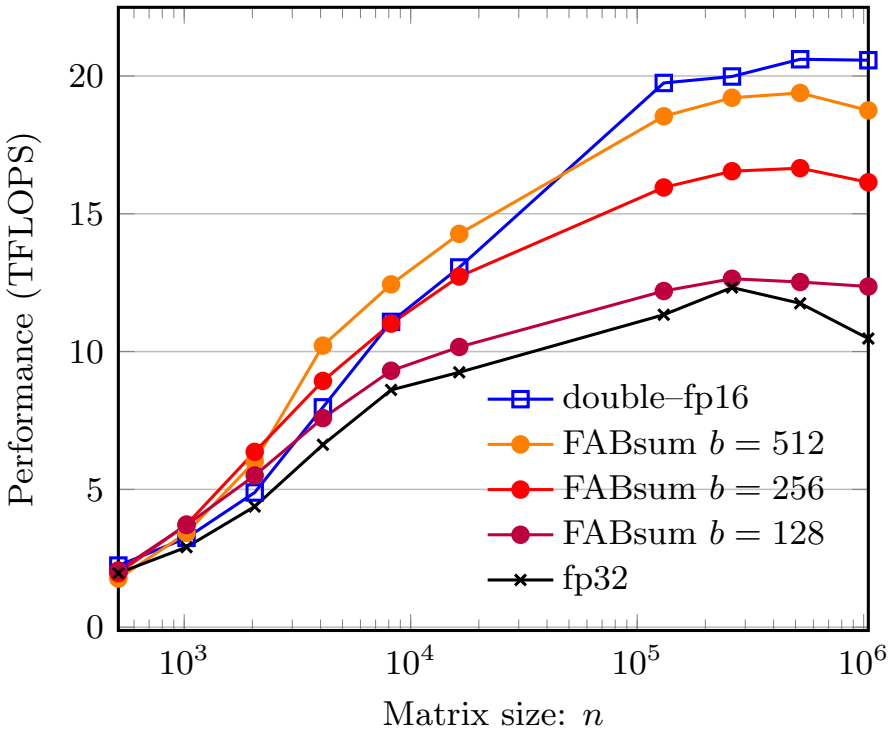
- Same error bound $\neq$ same error !

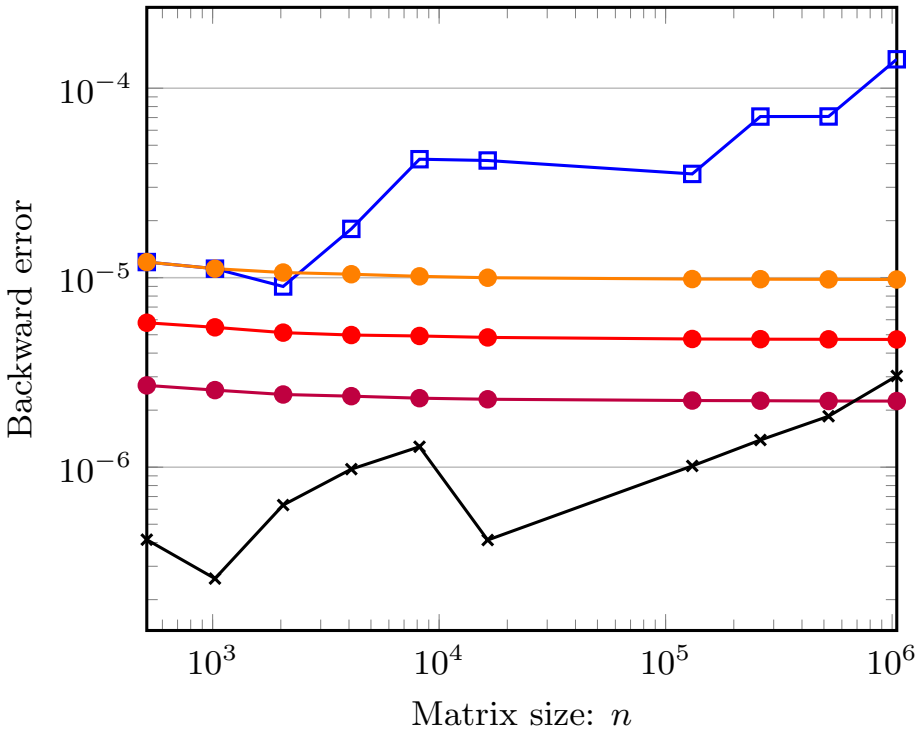
A proposed cure

- The worst-case accumulation bound nu_{32} is attained $\Rightarrow$ need to reduce the bound
- FABsum Blanchard, Higham, M. (2020)
  - Sum blocks of size b in precision u_{32}
  - Combine n/b blocks in precision u_{64}

  $\Rightarrow$ Reduced error bound $bu_{32} + nu_{64}/b$
- Parameter b controls tradeoff between accuracy and performance
- FABsum with Tensor Cores: based on CUTLASS library, which implements uniform precision blocked summation ("splitK")

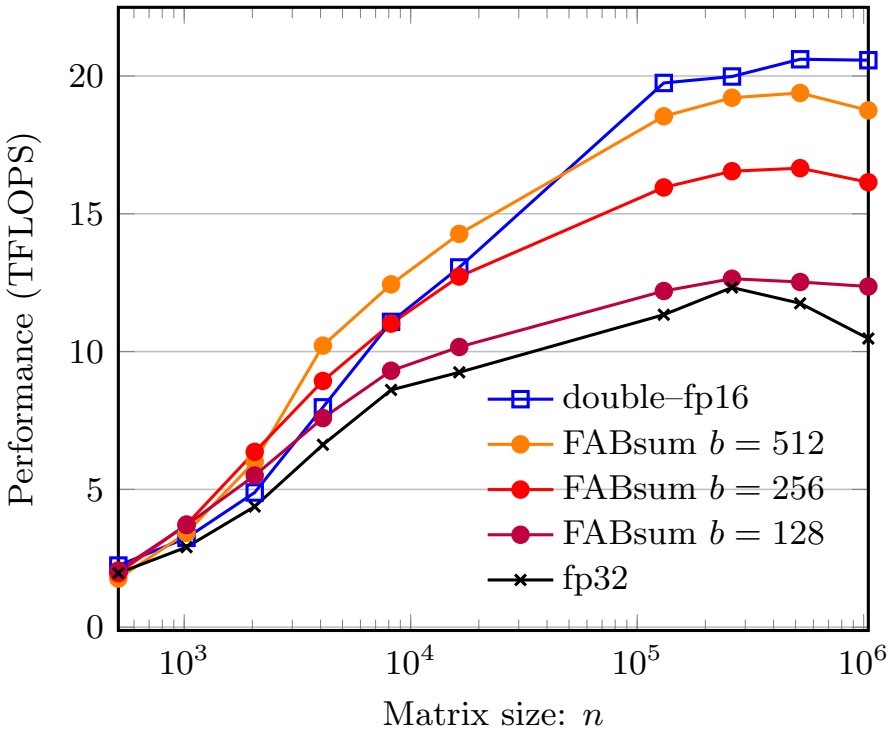
A proposed cure (results)

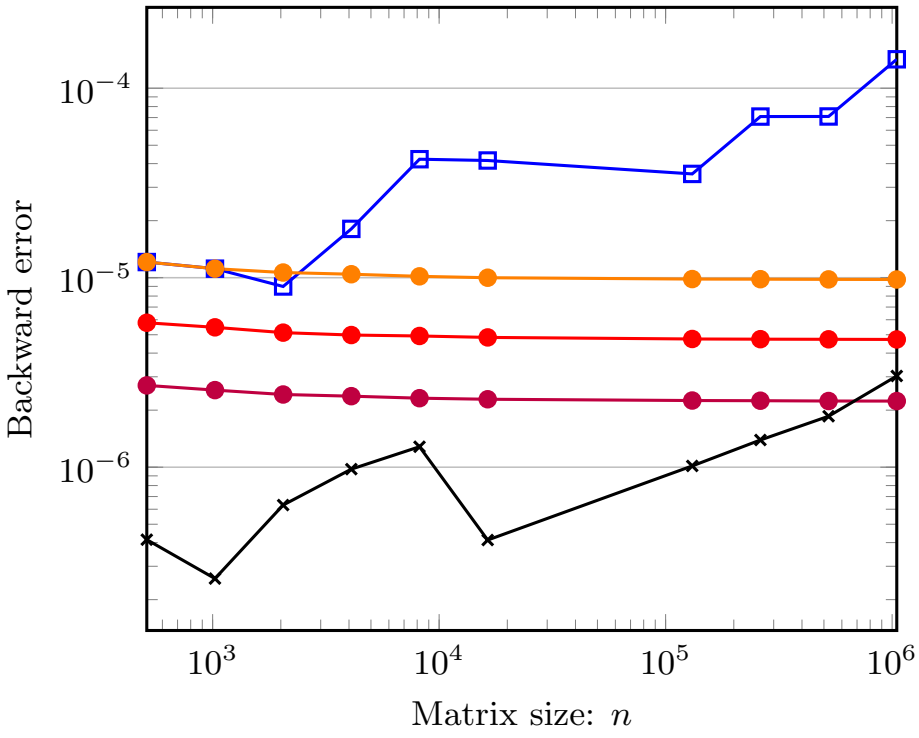




- As fast as cuBLAS but an order of magnitude more accurate
- Almost as accurate as fp32 and slightly faster
- And anything in between: flexible tradeoff

A proposed cure (results)





- As fast as cuBLAS but an order of magnitude more accurate
- Almost as accurate as fp32 and slightly faster
- And anything in between: flexible tradeoff

Thank you! Questions?