Performance and Accuracy of Mixed-Precision Matrix Factorizations with GPU Tensor Cores

Theo Mary, CNRS
Joint work with P. Blanchard, N. J. Higham, F. Lopez, and S. Pranesh

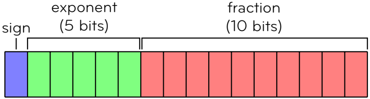
SIAM PP, 2020

Today's floating-point precision arithmetics

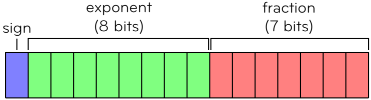
Type		Bits	Range	$u = 2^{-t}$
fp128	quad	128	$10^{\pm 4932}$	$2^{-113} \approx 1 \times 10^{-34}$
fp64	double	64	$10^{\pm 308}$	$2^{-53} \approx 1 \times 10^{-16}$
fp32	single	32	$10^{\pm 38}$	$2^{-24} \approx 6 \times 10^{-8}$
fp16	half	16	$10^{\pm 5}$	$2^{-11} \approx 5 \times 10^{-4}$
bfloat16	half	16	$10^{\pm 38}$	$2^{-8} \approx 4 \times 10^{-3}$

Half precision increasingly **supported by hardware**:

- Present: **NVIDIA** Pascal & Volta GPUs, **AMD** Radeon Instinct MI25 GPU, **Google** TPU, **ARM** NEON
- Near future: Fujitsu A64FX ARM, **IBM** AI chips, **Intel** Xeon Cooper Lake and Intel Nervana Neural Network



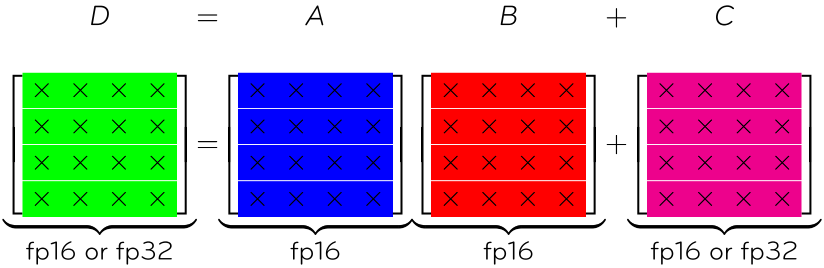
fp16



bfloat16

NVIDIA GPU tensor cores units

4×4 matrix multiplication **in 1 clock cycle**:



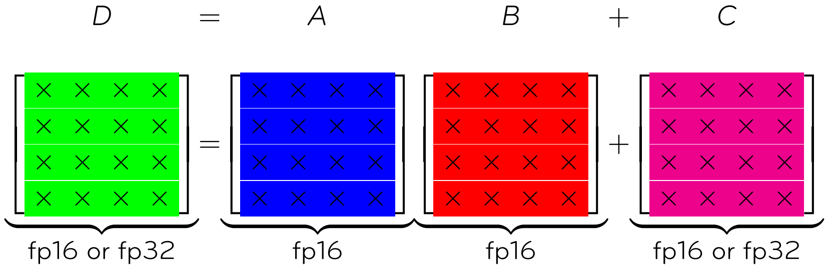
- This is a **block fused multiply-add** (FMA) in terms of speed (in terms of accuracy, depends on the implementation)

$\Rightarrow$ Performance peak 125 TFlops/s ($8\times$ speedup vs fp32!)

- Algorithms now become intrinsically **mixed precision**—and more complicated to analyze

NVIDIA GPU tensor cores units

4×4 matrix multiplication **in 1 clock cycle**:



- This is a **block fused multiply-add** (FMA) in terms of speed (in terms of accuracy, depends on the implementation)

⇒ Performance peak 125 TFlops/s ($8\times$ speedup vs fp32!)

- Algorithms now become intrinsically **mixed precision**—and more complicated to analyze

⇒ Need for **new analysis** to understand how to **best use these new units**

Block FMA: a general framework

We consider the following framework

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

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

- AB is computed with multiplications in precision u_{mul} and additions in precision u_{add}, and then rounded to precision $u_{\text{FMA}} = u_{\text{high}}$ or u_{low}

$$|\widehat{D} - D| \lesssim u_{\text{FMA}}(|C| + |A||B|) + \big((b-1)u_{\text{add}} + u_{\text{mul}}\big)|A||B|$$

- What choice of u_{add} and u_{mul}?
  - $u_{\text{add}} = u_{\text{mul}} = 0$: true FMA (only 1 rounding error per element of D)
  - $u_{\text{add}} = u_{\text{mul}} = u_{\text{low}}$: not an FMA in terms of accuracy, just speed
  - $u_{\text{add}} = u_{\text{low}}$, $u_{\text{mul}} = u_{\text{high}}$: not really an FMA either
  - $u_{\text{add}} = u_{\text{mul}} = u_{\text{high}}$: almost an FMA (FMA to first order)

Examples of block FMA units (present and future)

	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
Intel NNP-T	32	32	32	bfloat16	fp32
Armv8-A	2	4	2	bfloat16	fp32

- What are u_{add} and u_{mul}? $\Rightarrow$ not entirely clear. For tensor cores:

  > *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.*

$\Rightarrow$ In the following we distinguish two variants:

- TC16 ($u_{\text{FMA}} = u_{\text{add}} = u_{\text{low}} = u_{16}$, $u_{\text{mul}} = u_{\text{high}} = u_{32}$)
- TC32 ($u_{\text{FMA}} = u_{\text{add}} = u_{\text{mul}} = u_{\text{high}} = u_{32}$)
- Intermediate variant $u_{\text{FMA}} = u_{\text{mul}} = u_{32}$ and $u_{\text{add}} = u_{16}$ not discussed here

Matrix multiplication with block FMA

This algorithm computes $C = AB$ using a block FMA, where $A, B, C \in \mathbb{R}^{n\times n}$, and returns C in precision u_{FMA}

$\widetilde{A} \leftarrow \text{fl}_{\text{low}}(A)$ and $\widetilde{B} \leftarrow \text{fl}_{\text{low}}(B)$ (if necessary)
for $i = 1\colon n/b_1$ **do**
  for $j = 1\colon n/b_2$ **do**
    $C_{ij} = 0$
    for $k = 1\colon n/b$ **do**
      Compute $C_{ij} = C_{ij} + \widetilde{A}_{ik}\widetilde{B}_{kj}$ using a block FMA
    end for
  end for
end for

Matrix multiplication: error analysis

Let A and B already be given in precision u_{low}. For any row x of A and any column y of B, computing $s = c + x^T y$ classically produces

$$\widehat{s} = c(1+\theta_n) + x_1 y_1 (1+\theta_{n+1}) + x_2 y_2 (1+\theta'_n) + x_3 y_3 (1+\theta_{n-1}) + \cdots + x_n y_n (1+\theta_2),$$

where $z_k = x_k y_k$ and $|\theta_k| \lesssim ku$.

Matrix multiplication: error analysis

Let A and B already be given in precision u_{low}. For any row x of A and any column y of B, computing $s = c + x^T y$ classically produces

$$\begin{aligned}\widehat{s} = c(1+\theta_n) &+ x_1 y_1(1+\theta_{n+1}) + x_2 y_2(1+\theta'_n) \\ &+ x_3 y_3 (1+\theta_{n-1}) + \cdots + x_n y_n (1+\theta_2),\end{aligned}$$

where $z_k = x_k y_k$ and $|\theta_k| \lesssim ku$. With a block FMA, we have instead

$$\widehat{s} = \Big(z_1(1+\epsilon_1)\big(1+\theta^{(1)}_{b-1}\big) + \cdots + z_b(1+\epsilon_b)\big(1+\theta^{(1)}_1\big)\Big) \prod_{i=1}^{n/b}(1+\delta_i)$$

$$+ \cdots +$$

$$\Big(z_{n-b+1}(1+\epsilon_{n-b+1})\big(1+\theta^{(n/b)}_{b-1}\big) + \cdots + z_n(1+\epsilon_n)\big(1+\theta^{(n/b)}_1\big)\Big)(1+\delta_{n/b})$$

where $|\epsilon_k| \le u_{\text{mul}}$, $|\theta_k| \lesssim k u_{\text{add}}$, and $|\delta_k| \le u_{\text{FMA}}$

Overall: $|s - \widehat{s}| \lesssim \big(\frac{n}{b} u_{\text{FMA}} + (b-1) u_{\text{add}} + u_{\text{mul}}\big)|x|^T|y|$

Matrix multiplication: error analysis (cont'd)

If A and B are already given in precision u_{low}:

$$\widehat{C} = AB + \Delta C, \quad |\Delta C| \lesssim \Big(\frac{n}{b} u_{\text{FMA}} + (b-1)u_{\text{add}} + u_{\text{mul}}\Big)|A||B|$$

If not, we must account for the initial conversion:

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

$$\begin{aligned}\widehat{C} &= \widetilde{A}\widetilde{B} + \Delta C, \qquad |\Delta C| \lesssim \Big(\frac{n}{b} u_{\text{FMA}} + (b-1)u_{\text{add}} + u_{\text{mul}}\Big)|\widetilde{A}||\widetilde{B}|,\\ &= AB + \Delta A B + A \Delta B + \Delta A \Delta B + \Delta C\\ &= AB + E, \qquad |E| \lesssim \Big(2u_{\text{low}} + \frac{n}{b} u_{\text{FMA}} + (b-1)u_{\text{add}} + u_{\text{mul}}\Big)|A||B|\end{aligned}$$

Matrix multiplication: error analysis (cont'd)

If A and B are already given in precision u_{low}:

$$\widehat{C} = AB + \Delta C, \quad |\Delta C| \lesssim \big(\frac{n}{b} u_{\text{FMA}} + (b-1)u_{\text{add}} + u_{\text{mul}}\big)|A||B|$$

If not, we must account for the initial conversion:

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

$$\begin{aligned}\widehat{C} &= \widetilde{A}\widetilde{B} + \Delta C, \qquad |\Delta C| \lesssim \big(\frac{n}{b} u_{\text{FMA}} + (b-1)u_{\text{add}} + u_{\text{mul}}\big)|\widetilde{A}||\widetilde{B}|,\\ &= AB + \Delta A B + A \Delta B + \Delta A \Delta B + \Delta C\\ &= AB + E, \qquad |E| \lesssim \big(\underbrace{2u_{\text{low}}}_{\text{Conversion}} + \underbrace{\frac{n}{b} u_{\text{FMA}} + (b-1)u_{\text{add}} + u_{\text{mul}}}_{\text{Accumulation}}\big)|A||B|\end{aligned}$$

Matrix multiplication: error bounds interpretation

Evaluation method ($u_{mul} = u_{high}$)			Bound
Standard in precision u_{low}			nu_{low}
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = u_{low}$	$(n/b + b)u_{low}$
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = u_{high}$	$(n/b)u_{low} + bu_{high}$
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = 0$	$(n/b)u_{low}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = u_{low}$	$(b + 2)u_{low} + (n/b)u_{high}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = u_{high}$	$2u_{low} + (n/b + b)u_{high}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = 0$	$2u_{low} + (n/b)u_{high}$
Standard in precision u_{high}			nu_{high}

Matrix multiplication: error bounds interpretation

Evaluation method ($u_{mul} = u_{high}$)			Bound
Standard in precision u_{low}			nu_{low}
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = u_{low}$	$(n/b + b)u_{low}$
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = u_{high}$	$(n/b)u_{low} + bu_{high}$
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = 0$	$(n/b)u_{low}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = u_{low}$	$(b + 2)u_{low} + (n/b)u_{high}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = u_{high}$	$2u_{low} + (n/b + b)u_{high}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = 0$	$2u_{low} + (n/b)u_{high}$
Standard in precision u_{high}			nu_{high}

- $u_{FMA} = u_{add} = u_{low} \Rightarrow$ reduction by factor b from blocked sum

Matrix multiplication: error bounds interpretation

Evaluation method ($u_{mul} = u_{high}$)			Bound
Standard in precision u_{low}			nu_{low}
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = u_{low}$	$(n/b + b)u_{low}$
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = u_{high}$	$(n/b)u_{low} + bu_{high}$
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = 0$	$(n/b)u_{low}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = u_{low}$	$(b + 2)u_{low} + (n/b)u_{high}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = u_{high}$	$2u_{low} + (n/b + b)u_{high}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = 0$	$2u_{low} + (n/b)u_{high}$
Standard in precision u_{high}			nu_{high}

- $u_{FMA} = u_{add} = u_{low}$ ⇒ reduction by factor b from blocked sum
- $u_{FMA} = u_{low}$, $u_{add} \ll u_{low}$ ⇒ smaller u_{add} not very useful

Matrix multiplication: error bounds interpretation

Evaluation method ($u_{mul} = u_{high}$)			Bound
Standard in precision u_{low}			nu_{low}
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = u_{low}$	$(n/b + b)u_{low}$
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = u_{high}$	$(n/b)u_{low} + bu_{high}$
Block FMA	$u_{FMA} = u_{low}$	$u_{add} = 0$	$(n/b)u_{low}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = u_{low}$	$(b+2)u_{low} + (n/b)u_{high}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = u_{high}$	$2u_{low} + (n/b + b)u_{high}$
Block FMA	$u_{FMA} = u_{high}$	$u_{add} = 0$	$2u_{low} + (n/b)u_{high}$
Standard in precision u_{high}			nu_{high}

- $u_{FMA} = u_{add} = u_{low} \Rightarrow$ reduction by factor b from blocked sum
- $u_{FMA} = u_{low}$, $u_{add} \ll u_{low} \Rightarrow$ smaller u_{add} not very useful
- $u_{FMA} = u_{high} \Rightarrow$ reduction by factor $\min\big(n/2, u_{low}/u_{high}\big)$, $u_{add} \ll u_{low}$ useful, $u_{add} = 0$ not useful

Matrix multiplication: error bounds interpretation

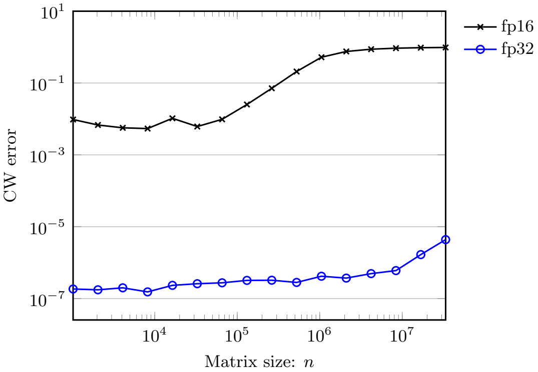
Evaluation method ($u_{\text{mul}} = u_{\text{high}}$)			Bound
Standard in precision u_{low}			nu_{low}
Block FMA	$u_{\text{FMA}} = u_{\text{low}}$	$u_{\text{add}} = u_{\text{low}}$	$(n/b + b)u_{\text{low}}$
Block FMA	$u_{\text{FMA}} = u_{\text{low}}$	$u_{\text{add}} = u_{\text{high}}$	$(n/b)u_{\text{low}} + bu_{\text{high}}$
Block FMA	$u_{\text{FMA}} = u_{\text{low}}$	$u_{\text{add}} = 0$	$(n/b)u_{\text{low}}$
Block FMA	$u_{\text{FMA}} = u_{\text{high}}$	$u_{\text{add}} = u_{\text{low}}$	$(b+2)u_{\text{low}} + (n/b)u_{\text{high}}$
Block FMA	$u_{\text{FMA}} = u_{\text{high}}$	$u_{\text{add}} = u_{\text{high}}$	$2u_{\text{low}} + (n/b + b)u_{\text{high}}$
Block FMA	$u_{\text{FMA}} = u_{\text{high}}$	$u_{\text{add}} = 0$	$2u_{\text{low}} + (n/b)u_{\text{high}}$
Standard in precision u_{high}			nu_{high}

- $u_{\text{FMA}} = u_{\text{add}} = u_{\text{low}} \Rightarrow$ reduction by factor b from blocked sum
- $u_{\text{FMA}} = u_{\text{low}}$, $u_{\text{add}} \ll u_{\text{low}} \Rightarrow$ smaller u_{add} not very useful
- $u_{\text{FMA}} = u_{\text{high}} \Rightarrow$ reduction by factor $\min(n/2, u_{\text{low}}/u_{\text{high}})$, $u_{\text{add}} \ll u_{\text{low}}$ useful, $u_{\text{add}} = 0$ not useful

Conclusion: choice of u_{FMA} critical, u_{add} less so

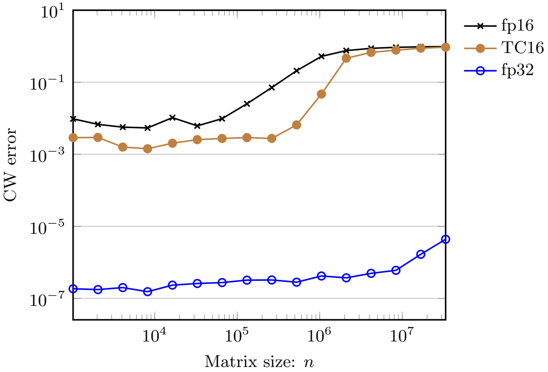
Matrix multiplication with tensor cores

Standard fp16	Tensor core TC16	Tensor core TC32	Standard fp32
nu_{16}	$(n/4)u_{16}$	$2u_{16} + (n/4)u_{32}$	nu_{32}



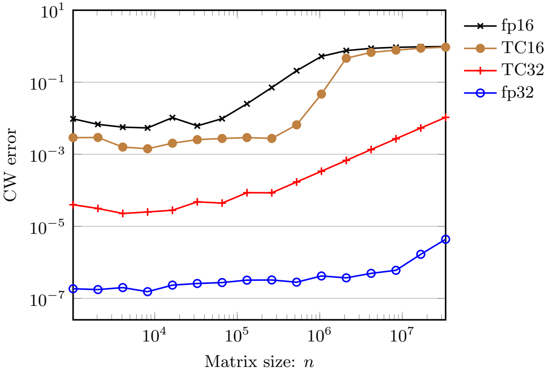
Matrix multiplication with tensor cores

Standard fp16	Tensor core TC16	Tensor core TC32	Standard fp32
nu_{16}	$(n/4)u_{16}$	$2u_{16} + (n/4)u_{32}$	nu_{32}



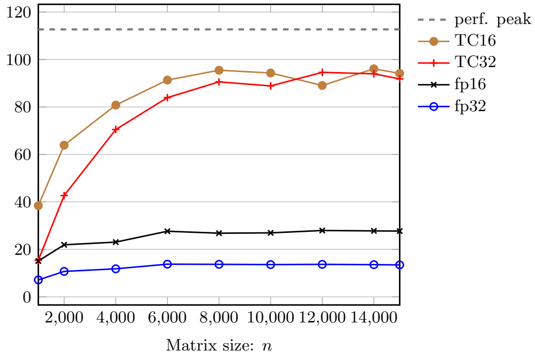
Matrix multiplication with tensor cores

Standard fp16	Tensor core TC16	Tensor core TC32	Standard fp32
nu_{16}	$(n/4)u_{16}$	$2u_{16} + (n/4)u_{32}$	nu_{32}



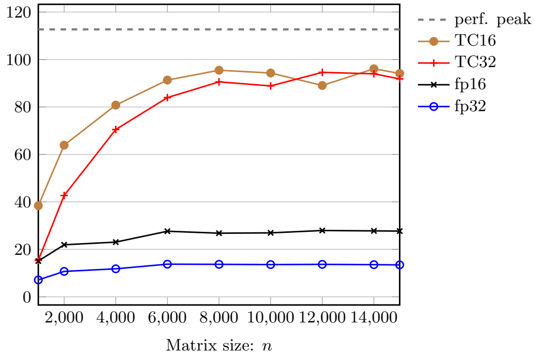
Matrix multiplication with tensor cores

Standard fp16	Tensor core TC16	Tensor core TC32	Standard fp32
nu_{16}	$(n/4)u_{16}$	$2u_{16} + (n/4)u_{32}$	nu_{32}



Matrix multiplication with tensor cores

Standard fp16	Tensor core TC16	Tensor core TC32	Standard fp32
nu_{16}	$(n/4)u_{16}$	$2u_{16} + (n/4)u_{32}$	nu_{32}



Conclusion: TC32 significantly more accurate than TC16, with almost no performance loss

LU factorization with block FMA

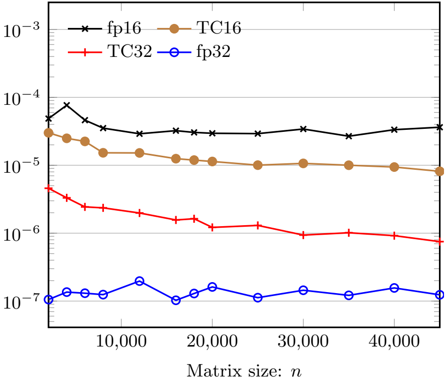
This algorithm computes $A = LU$ using a block FMA, where $A \in \mathbb{R}^{n \times n}$ is given in precision u_{high}, and L and U are returned in precision u_{FMA}

```
for k = 1: n/b do
    Factorize L_kk U_kk = A_kk
    for i = k + 1: n/b do
        Solve L_ik U_kk = A_ik and L_kk U_ki = A_ki for L_ik and U_ki
    end for
    for i = k + 1: n/b do
        for j = k + 1: n/b do
            L~_ik ← fl_low(L_ik) and U~_ki ← fl_low(U_ki)
            A_ij ← A_ij − L~_ik U~_kj using a block FMA
        end for
    end for
end for
```

LU factorization with tensor cores

Error analysis for LU follows from matrix multiplication analysis and gives same bounds to first order (minor changes in the constants)

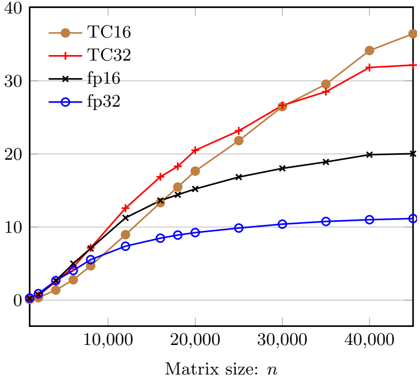
Standard fp16	Tensor core TC16	Tensor core TC32	Standard fp32
nu_{16}	$n/4u_{16}$	$2u_{16} + (n/4)u_{32}$	nu_{32}



LU factorization with tensor cores

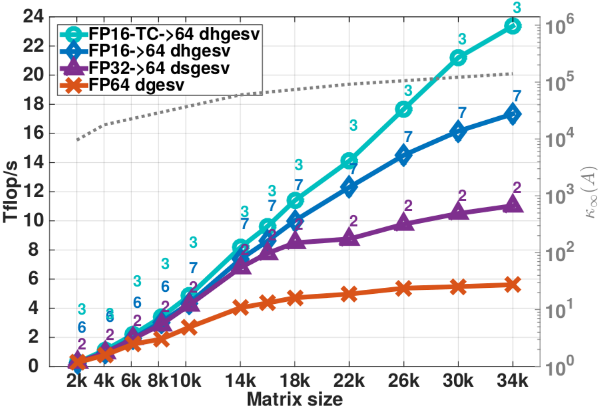
Error analysis for LU follows from matrix multiplication analysis and gives same bounds to first order (minor changes in the constants)

Standard fp16	Tensor core TC16	Tensor core TC32	Standard fp32
nu_{16}	$n/4u_{16}$	$2u_{16} + (n/4)u_{32}$	nu_{32}



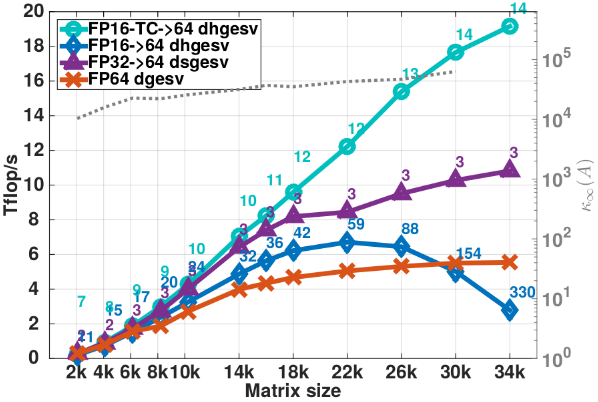
Iterative refinement

Results from Haidar et al, *Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers* (SC'2018):



- Performance TC boost not fully translated in their implementation

Results from Haidar et al, *Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers* (SC'2018):



- Performance TC boost not fully translated in their implementation
- But accuracy boost sometimes critical!

Conclusions

- Mixed-precision units increasingly available in hardware
- Proposed a general mixed-precision block FMA framework, should be applicable to existing and future units
- Performed error analysis for matrix mult. and LU factorization
- Application to NVIDIA GPU tensor cores: compared two variants, TC16 and TC32 (different computeType parameter)

⇒ TC32 is significantly more accurate than fp16 and TC16: reduction from $O(nu_{16})$ to $2u_{16} + O(nu_{32})$, while being almost as fast as TC16 (and much faster than fp16)

Take-home message: we recommend using TC32 over TC16

Preprint and slides

See our preprint: *Mixed Precision Block Fused Multiply-Add: Error Analysis and Application to GPU Tensor Cores*
Slides available on my website: `bit.ly/tmaryLIP6`

Ongoing work on LU factorization with tensor cores

Partitioned LU factorization with tensor cores

The LU analysis assumes a panel of size $b = 4 \Rightarrow$ not realistic for performance, where $b = O(100)$. Possible solutions:

- Do the panel factorization in fp32 (Haidar et al, 2018) $\Rightarrow$ suboptimal performance
- Do the panel factorization in fp16 $\Rightarrow$ suboptimal accuracy
- **Our proposed solution:** use a double panel hierarchy to use mixed precision TC32 in the panel factorization

Storing the LU factors in fp16

The LU analysis assumes the LU factors to be stored in fp32 precision $\Rightarrow$ no storage gain! Possible solutions:

- Store them in fp16 $\Rightarrow$ repeated rounding to fp16 (after each update) causes great loss of accuracy (TC16 even with computeType=fp32)
- **Our proposed solution:** use a left-looking factorization with a temporary fp32 buffer to accumulate updates $\Rightarrow$ no accuracy loss!