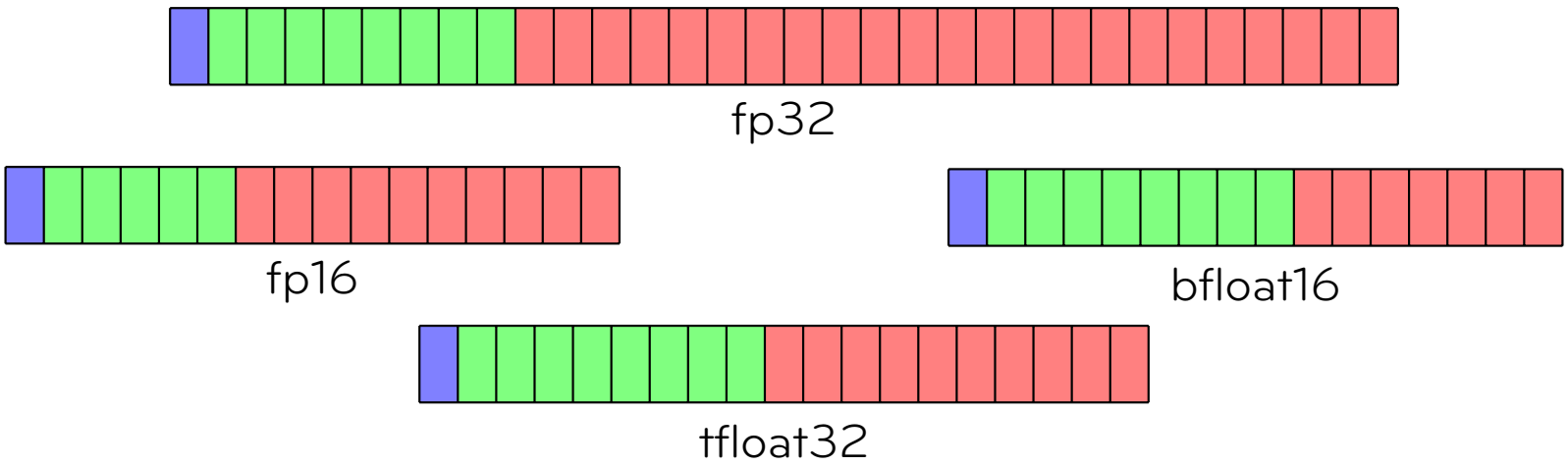
Mixed precision arithmetic for HPC

Theo Mary (CNRS)

HPCA course, Sorbonne Université (2020 version)

Slides available at
https://www-pequan.lip6.fr/~tmary/doc/HPCA.pdf



Intro

Mixed precision hardware

Mixed precision algorithms

- FABsum
- Iterative refinement
  - Mixed precision LU factorization
  - Mixed precision GMRES
- Half–half arithmetic

Challenges in HPC

- Increasingly large problems (10^7–10^9 unknowns)
- Increasingly parallel computers
- Heterogeneity in the computing units: CPUs, GPUs, other accelerators
- Increasing gap between speed of computations and communications
- Increasing power consumption

$\Rightarrow$ A promising idea to tackle these challenges: work with **lower precision arithmetics**

Basics of floating-point arithmetic

Topic of this lecture: mixed precision arithmetic, with a focus on **floating-point** arithmetic

A floating-point number is represented by

$$x = \pm m \times \beta^{e-t}, \quad m \in [0, \beta^t - 1]$$

A floating-point number system is thus charaterized by

- Base β (usually 2)
- Precision t
- Exponent range: $e \in [e_{\min}, e_{\max}]$

which are encoded with a finite number of bits assigned to the mantissa and exponent

Basics of floating-point arithmetic (cont'd)

The unit roundoff $u = \beta^{1-t}/2$ ($= 2^{-t}$ in base 2) determines the relative accuracy any number in the representable range can be approximated with:

If $x \in \mathbb{R}$ belongs to $[e_{\min}, e_{\max}]$, then $\mathrm{fl}(x) = x(1+\delta), \quad |\delta| \leq u$

Moreover the **standard model of arithmetic** is

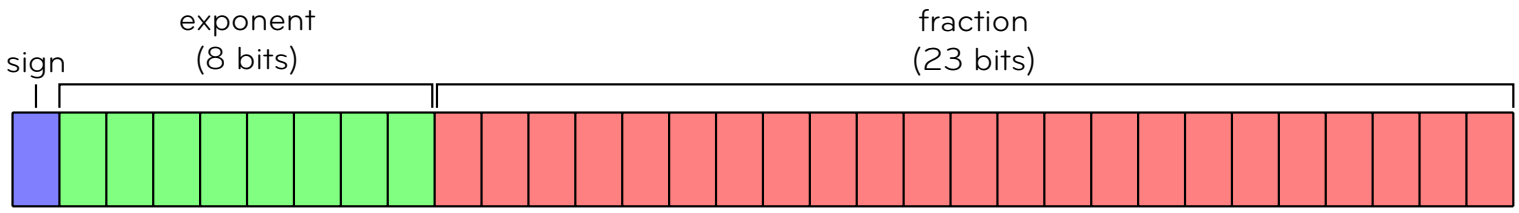
$\mathrm{fl}(x \mathrm{\ op\ } y) = (x \mathrm{\ op\ } y)(1+\delta), \quad |\delta| \leq u$, for $\mathrm{op} \in \{+, -, \times, \div\}$

Some common arithmetics in HPC

	Number of bits		Range	Unit roundoff u
	Mantissa	Exponent		
fp128	113	15	$10^{\pm 4932}$	1×10^{-34}
fp64	53	11	$10^{\pm 308}$	1×10^{-16}
fp32	24	8	$10^{\pm 38}$	6×10^{-8}
tfloat32	11	8	$10^{\pm 38}$	5×10^{-4}
fp16	11	5	$10^{\pm 5}$	5×10^{-4}
bfloat16	8	8	$10^{\pm 38}$	4×10^{-3}

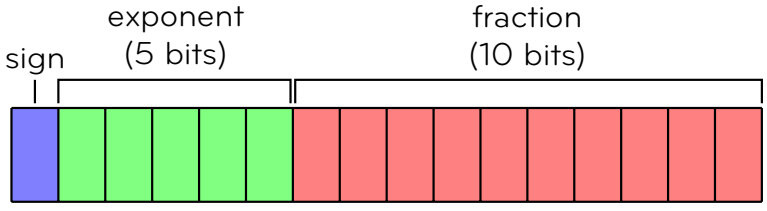
- Originally fp64 and fp32
- fp128 usually not available in hardware, only in software
- Rise of half precision on modern hardware, driven by AI

Lower precisions



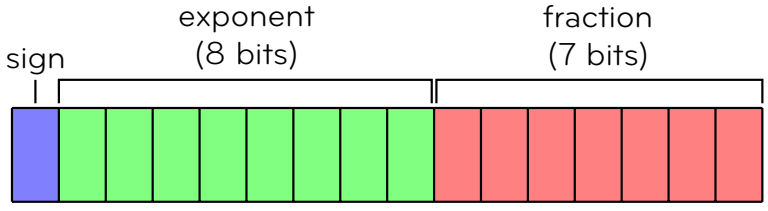
fp32

Range $10^{\pm 38}$, $u = 6 \times 10^{-8}$



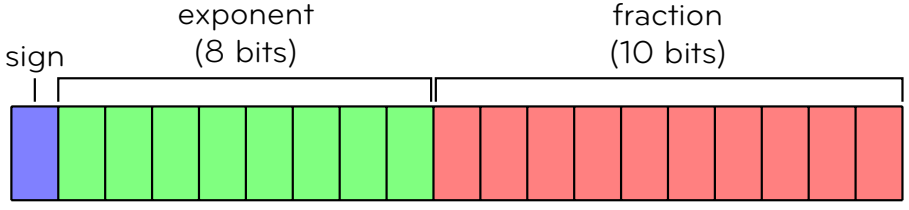
fp16

Range $10^{\pm 5}$, $u = 5 \times 10^{-4}$



bfloat16

Range $10^{\pm 38}$, $u = 4 \times 10^{-3}$



tfloat32

Range $10^{\pm 38}$, $u = 5 \times 10^{-4}$

Benefits of lower precisions

- Storage, data movement and communications are all proportional to total number of bits (mantissa + exponent)
  $\Rightarrow$ lower precisions reduce them

- Until now, speed of computations was also generally proportional
  $\Rightarrow$ on most architectures, fp32 is twice faster than fp64

CELL processor

A notable exception: CELL processor (2006–2008)

1 CELL = 1 PPE + 8× SPE

PPE peak (GFLOPS): 6.4 (fp64) → 25.6 (fp32) 4× speedup!

SPE peak (GFLOPS): 1.8 (fp64) → 25.6 (fp32) 14× speedup!!

CELL processor

A notable exception: CELL processor (2006–2008)

1 CELL = 1 PPE + 8× SPE

PPE peak (GFLOPS): 6.4 (fp64) → 25.6 (fp32) 4× speedup!

SPE peak (GFLOPS): 1.8 (fp64) → 25.6 (fp32) 14× speedup!!

Paper from 2008: Kurzak et al (2008)

The PlayStation 3 for High-Performance Scientific Computing

Publisher: IEEE Cite This PDF

Jakub Kurzak; Alfredo Buttari; Piotr Luszczek; Jack Dongarra **All Authors**

Paper from 2008: Kurzak et al (2008)

The PlayStation 3 for High-Performance Scientific Computing

Publisher: IEEE Cite This PDF

Jakub Kurzak; Alfredo Buttari; Piotr Luszczek; Jack Dongarra **All Authors**

Condor Cluster (peak: 500 TFLOPS)
Made of 1760 PS3s !

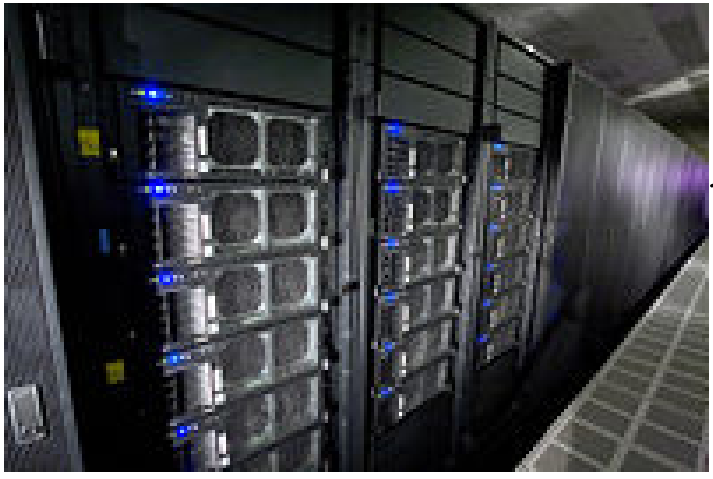
Paper from 2008: Kurzak et al (2008)

The PlayStation 3 for High-Performance Scientific Computing

Publisher: IEEE Cite This PDF

Jakub Kurzak; Alfredo Buttari; Piotr Luszczek; Jack Dongarra **All Authors**

Condor Cluster (peak: 500 TFLOPS)
Made of 1760 PS3s !



IBM Roadrunner (peak: 1.7 PFLOPS)
1st on TOP500 ranking in 2008
First computer to surpass 1 PFLOP on LINPACK benchmark!

The exception is becoming the rule for half precision on modern hardware
NVIDIA Tesla GPUs

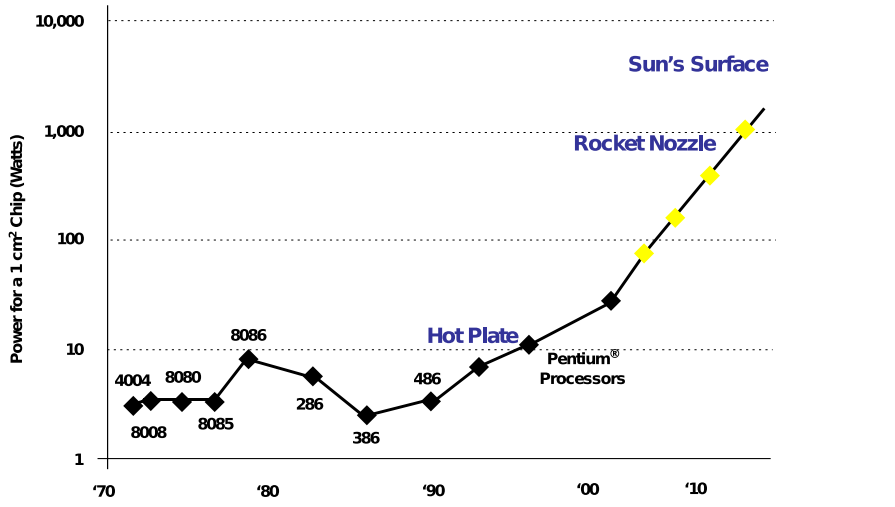
Peak performance (TFLOPS)

		fp64	fp32	tfloat32	fp16	bfloat16
P100	2016	5	9	–	19	–
V100	2017–2019	8	16	–	125*	–
A100	2020	19*	19	156*	312*	312*

* with tensor cores

Power consumption

- Power consumption of supercomputers has exploded with time and is today a primary concern



Source: Pat Gelsinger, Intel, ISSCC 2001

- Power consumption is proportional to the square of the number of mantissa bits. Thus:
- fp16 and tfloat32 (11 bits) consume $5\times$ less energy than fp32 (24 bits)
- bfloat16 (8 bits) consumes $2\times$ less energy than fp16/tfloat32 and $9\times$ less than fp32!

Risks of lower precisions (1/4): underflow/overflow

- fp16 has a very narrow range: $[6.1 \times 10^{-5}, 6.5 \times 10^{4}]$

$\Rightarrow$ high risk of underflow/overflow. Some examples:

Risks of lower precisions (1/4): underflow/overflow

- fp16 has a very narrow range: $[6.1 \times 10^{-5}, 6.5 \times 10^{4}]$

$\Rightarrow$ high risk of underflow/overflow. Some examples:

```
>> A = gallery('randsvd', 5, 1e12, 3);
>> s1 = svd(A); s2 = svd(fp16(A));
>> err = abs(s1-s2)./s1;
>> [s1 s2 err]
```

s1	s2	err
1.0000e+00	9.9999e-01	9.0588e-06
1.0000e-03	9.8917e-04	1.0832e-02
1.0000e-06	4.2918e-05	4.1918e+01
1.0000e-09	9.5870e-06	9.5860e+03
9.9999e-13	8.0750e-06	8.0751e+06

Risks of lower precisions (1/4): underflow/overflow

- fp16 has a very narrow range: $[6.1 \times 10^{-5}, 6.5 \times 10^{4}]$

$\Rightarrow$ high risk of underflow/overflow. Some examples:

"Large" dot products

```
>> x = rand(n,1); y = rand(n,1); fp16(x'*y)
```

$\Rightarrow$ returns `Inf` for $n \gtrsim 300,000$

Chain matrix–vector products (example: neural network backpropagation)

```
>> A = rand(n); x = rand(n);
for i=1:p
x = fp16(A*x);
end
```

$\Rightarrow$ overflows even for small n when $p \geq 7$

Risks of lower precisions (1/4): underflow/overflow

- fp16 has a very narrow range: $[6.1 \times 10^{-5}, 6.5 \times 10^{4}]$

$\Rightarrow$ high risk of underflow/overflow. Some examples:

LU factorization of mode-2 randsvd matrices

```
>> A = gallery('randsvd', n, kappa, 2);
>> [L,U] = lu(A);
```

$\Rightarrow$ `Inf` entries appear in U for $n \gtrsim 60,000$ (don't try this at home!)

$\Rightarrow$ caused the HPL-AI benchmark to fail

Higham, Higham, and Pranesh (2020)

Risks of lower precisions (2/4): absorption

- Even in absence of underflow, small elements can often be absorbed. Example: the harmonic series

$$\sum_{k=1}^{\infty} 1/k = \infty$$

  is well-known to diverge ...

Risks of lower precisions (2/4): absorption

- Even in absence of underflow, small elements can often be absorbed. Example: the harmonic series

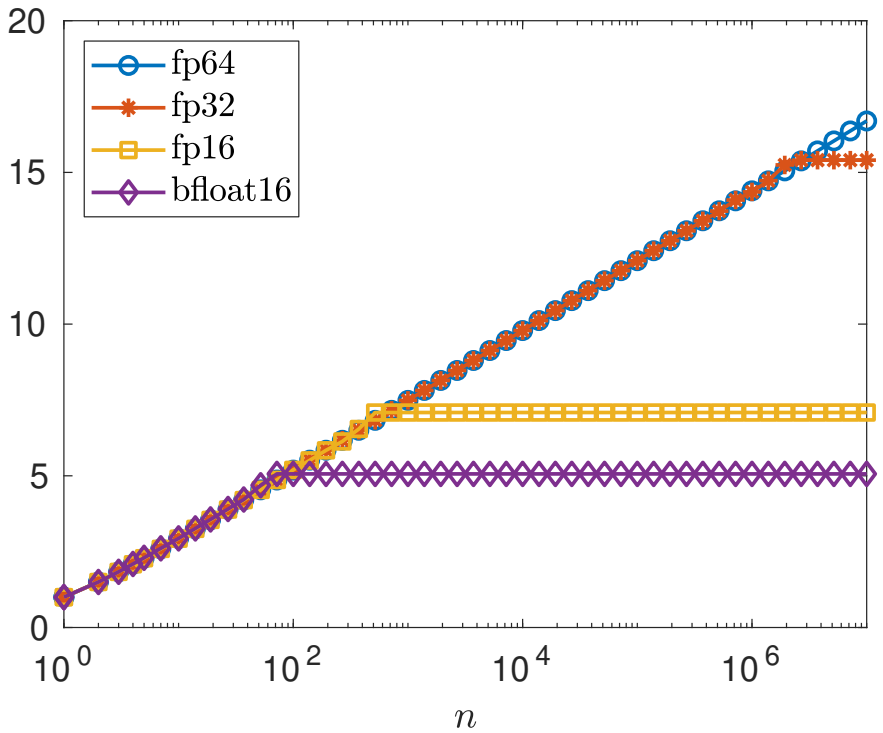
$$\sum_{k=1}^{\infty} 1/k = \infty$$

is well-known to diverge ...

Value of $\sum_{k=1}^{n} 1/k$
in floating-point arithmetic:

```
>> s(1) = 1;
for i=2:n
s(i) = s(i-1) + 1/i;
end
```

Malone (2013)



Risks of lower precisions (3/4): accumulation

Consider the computation of $s = \sum_{i=1}^{n} x_i$ by recursive summation:

$$
\begin{array}{lllll}
s_2 & = x_1 + x_2 & \Rightarrow & \widehat{s}_2 & = (x_1 + x_2)(1 + \delta_1) \\
s_3 & = \widehat{s}_2 + x_3 & \Rightarrow & \widehat{s}_3 & = (\widehat{s}_2 + x_3)(1 + \delta_2) \\
 & & & & = (x_1 + x_2) \underbrace{(1 + \delta_1)(1 + \delta_2)}_{\delta_1 \text{ and } \delta_2 \text{ accumulate!}} + x_3(1 + \delta_2)
\end{array}
$$

$s_4 = \ldots$ etc.

In total $n - 1$ errors δ_i are accumulated, with $|\delta_i| \leq u$

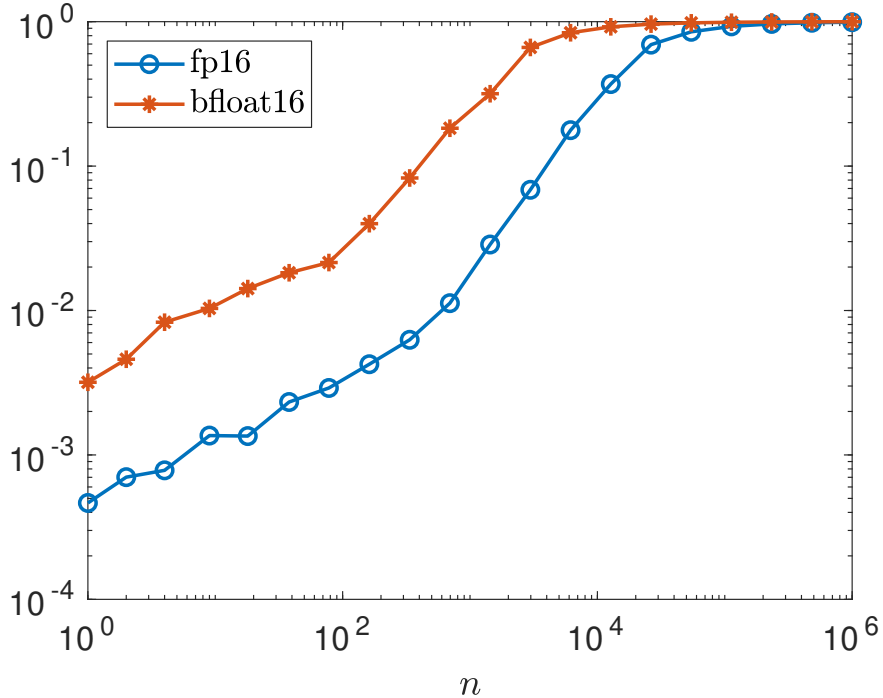
$$
\Rightarrow \quad |\widehat{s} - s| \leq (n - 1)u \sum_{i=1}^{n} |x_i|
$$

- For the same reason, most numerical algorithms involving n-vectors, $n \times n$ matrices, etc. satisfy error bounds $\propto nu$
- In fp16, $u = 5 \times 10^{-4} \Rightarrow nu > 1$ for $n > 2048$!!
- Even worse for bfloat16: $u = 4 \times 10^{-3} \Rightarrow nu > 1$ for $n > 256$
- nu is a **worst-case bound**: in practice, can expect $\sqrt{n}u$ instead
  Higham and Mary (2019)
  Still, with half precision, $\sqrt{n}u > 1$ for typical target sizes

- For the same reason, most numerical algorithms involving n-vectors, $n \times n$ matrices, etc. satisfy error bounds $\propto nu$
- In fp16, $u = 5 \times 10^{-4} \Rightarrow nu > 1$ for $n > 2048$!!
- Even worse for bfloat16: $u = 4 \times 10^{-3} \Rightarrow nu > 1$ for $n > 256$
- nu is a **worst-case bound**: in practice, can expect $\sqrt{n}u$ instead

  Higham and Mary (2019)

  Still, with half precision, $\sqrt{n}u > 1$ for typical target sizes



Error in computing a $10 \times n \times 10$ product $X^T Y$

Risks of lower precisions (4/4): ill conditioning

$$\text{We have shown} \qquad \Rightarrow \qquad |\widehat{s} - s| \leq (n-1)u \sum_{i=1}^{n} |x_i|$$

But what if $|s| = \left|\sum_{i=1}^{n} x_i\right| \ll \sum_{i=1}^{n} |x_i|$??
This happens in case of **cancellation**. We also say the sum is **ill conditioned**, the condition number being defined as

$$\kappa = \frac{\sum_{i=1}^{n} |x_i|}{\left|\sum_{i=1}^{n} x_i\right|}$$

Example: $s = 10.8 + 0.4 - 11.1 = 0.1$
In bfloat16, $\widehat{s} \approx 0.0874 \Rightarrow |\widehat{s} - s| > 0.1|s|$
Even though we have only done 3 operations!

More generally, ill-conditioned data is ubiquitous in scientific computing
Example: condition number of a matrix A is

$$\kappa(A) = \|A\|\|A^{-1}\| \quad \left(= \sigma_1/\sigma_n \text{ with } \|\cdot\|_2 \right)$$

Let $Ax = b$ be solved by Gaussian elimination (LU factorization) in precision u. Then the computed solution $\widehat{x}$ satisfies

$$\frac{\|\widehat{x} - x\|}{\|x\|} \leq f(n)\kappa(A)u$$

Highly ill-conditioned matrices ($\kappa(A)$ of several orders of magnitude) routinely arise in many applications!

Mixed precision arithmetic

Recap so far: low precisions present...

- great opportunities to tackle today's HPC challenges (speed, scalability, energy, ...)
- significant risks to the numerical stability and accuracy of the computation

Mixed precision arithmetic: an approach to preserve the benefits while avoiding risks?
Main idea: combine low and high precisions together in a strategic way.

Two main approaches to employ mixed precision arithmetic:

- **at the hardware level**
- **at the algorithm level**

Intro

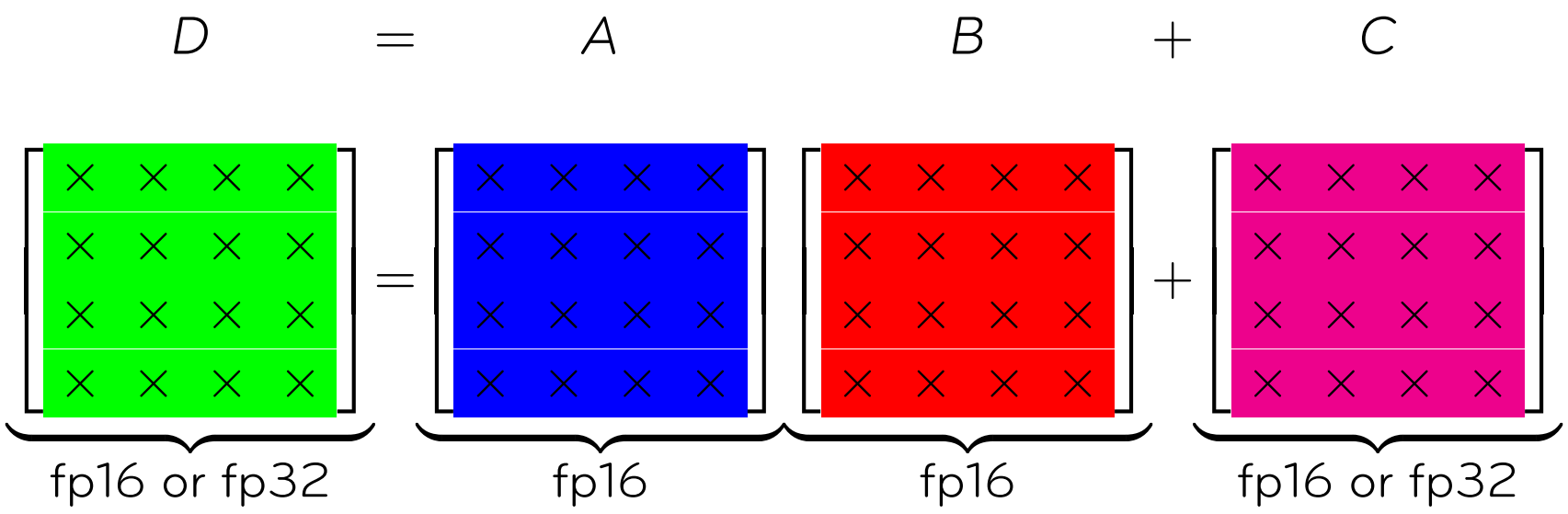
Mixed precision hardware

Mixed precision algorithms

- FABsum
- Iterative refinement
  - Mixed precision LU factorization
  - Mixed precision GMRES
- Half–half arithmetic

NVIDIA GPU tensor cores

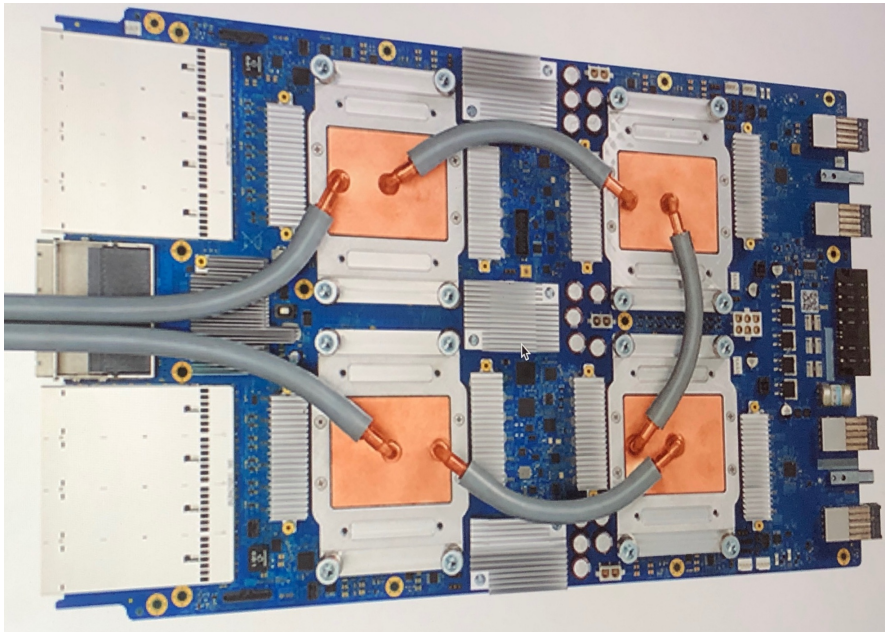
Tensor cores units available on NVIDIA GPUs V100 carry out a 4×4 matrix multiplication **in 1 clock cycle**:



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.

On A100, support for bfloat16 and tfloat32 was added

Google TPUs



- MXUs (matrix units) from Google's TPUs (Tensor Processing Units) carry out a MAC (Multiply and Accumulate) on 256×256 or 128×128 matrices
- Intel Cooper Lake CPUs will also have similar capabilities for dot products

Block FMA: a general framework

We consider the following framework Blanchard et al. (2020)

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

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

- AB is computed in precision $\overline{u}$, and then rounded to precision $u_{\text{FMA}} = u_{\text{high}}$ or u_{low}

$$|\widehat{D} - D| \lesssim u_{\text{FMA}}(|C| + |A||B|) + b\overline{u}|A||B|$$

- What choice of $\overline{u}$?
  - $\overline{u} = 0$: true FMA (only 1 rounding error per element of D)
  - $\overline{u} = u_{\text{low}}$: not an FMA in terms of accuracy, just speed
  - $\overline{u} = u_{\text{high}}$: not a true FMA, but close to one (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
NVIDIA Ampere	4	4	4	fp16	fp32
NVIDIA Ampere	4	4	4	bfloat16	fp32
NVIDIA Ampere	4	4	4	tfloat32	fp32
Intel NNP-T	32	32	32	bfloat16	fp32
Armv8-A	2	4	2	bfloat16	fp32

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 $i = 1 : n/b_1$ **do**
    for $j = 1 : n/b_2$ **do**
        $C_{ij} = 0$
        for $k = 1 : n/b$ **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

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

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

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

$$\begin{aligned}\widehat{s} = &\left(z_1\left(1+\theta_b^{(1)}\right) + \cdots + z_b\left(1+\theta_2^{(1)}\right)\right)\prod_{i=1}^{n/b}(1+\delta_i) \\ &+ \cdots + \\ &\left(z_{n-b+1}\left(1+\theta_b^{(n/b)}\right) + \cdots + z_n\left(1+\theta_2^{(n/b)}\right)\right)(1+\delta_{n/b})\end{aligned}$$

where $|\theta_k| \lesssim k\bar{u}$, and $|\delta_k| \le u_{\text{FMA}}$

Overall: $|s - \widehat{s}| \lesssim \left(\frac{n}{b}u_{\text{FMA}} + b\bar{u}\right)|x|^T|y|$

Matrix multiplication: error analysis (cont'd)

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

$$\widehat{C} = AB + \Delta C, \quad |\Delta C| \lesssim \left(\frac{n}{b} u_{\text{FMA}} + b\overline{u}\right)|A||B|$$

If not, we must account for the initial conversion:

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

$$\begin{aligned}
\widehat{C} &= \widetilde{A}\widetilde{B} + \Delta C, \qquad |\Delta C| \lesssim \left(\frac{n}{b} u_{\text{FMA}} + b\overline{u}\right)|\widetilde{A}||\widetilde{B}|, \\
&= AB + \Delta AB + A\Delta B + \Delta A\Delta B + \Delta C \\
&= AB + E, \qquad |E| \lesssim \left(2u_{\text{low}} + \frac{n}{b} u_{\text{FMA}} + b\overline{u}\right)|A||B|
\end{aligned}$$

Matrix multiplication: error analysis (cont'd)

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

$$\widehat{C} = AB + \Delta C, \quad |\Delta C| \lesssim \left(\frac{n}{b} u_{\text{FMA}} + b\overline{u}\right)|A||B|$$

If not, we must account for the initial conversion:

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

$$\begin{aligned}
\widehat{C} &= \widetilde{A}\widetilde{B} + \Delta C, \qquad |\Delta C| \lesssim \left(\frac{n}{b} u_{\text{FMA}} + b\overline{u}\right)|\widetilde{A}||\widetilde{B}|, \\
&= AB + \Delta AB + A\Delta B + \Delta A\Delta B + \Delta C \\
&= AB + E, \qquad |E| \lesssim \Big(\underbrace{2u_{\text{low}}}_{\text{Conversion}} + \underbrace{\frac{n}{b} u_{\text{FMA}} + b\overline{u}}_{\text{Accumulation}} \Big)|A||B|
\end{aligned}$$

Matrix multiplication: error bounds interpretation

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

Matrix multiplication: error bounds interpretation

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

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

Matrix multiplication: error bounds interpretation

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

- $u_{\text{FMA}} = \overline{u} = u_{\text{low}} \Rightarrow$ reduction by factor b from blocked sum
- $u_{\text{FMA}} = u_{\text{low}}$, $\overline{u} \ll u_{\text{low}} \Rightarrow$ smaller $\overline{u}$ not very useful

Matrix multiplication: error bounds interpretation

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

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

Matrix multiplication: error bounds interpretation

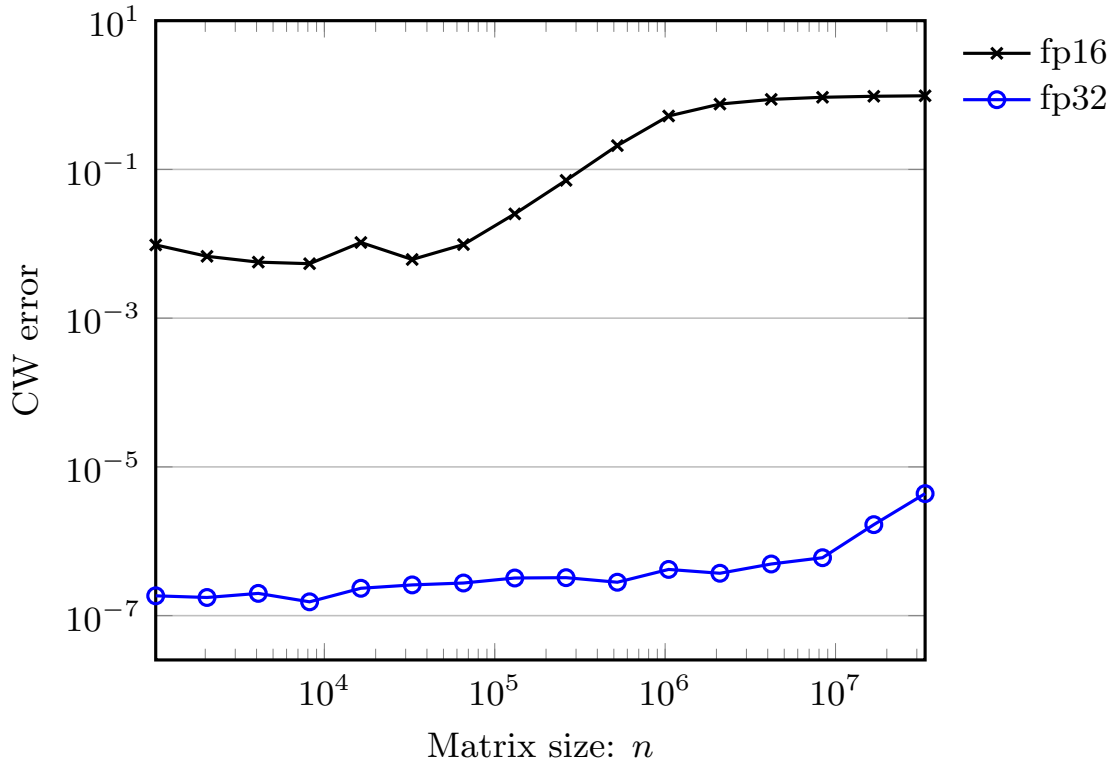
Evaluation method			Bound
Standard in precision u_{low}			nu_{low}
Block FMA	$u_{\text{FMA}} = u_{\text{low}}$	$\overline{u} = u_{\text{low}}$	$(n/b + b)u_{\text{low}}$
Block FMA	$u_{\text{FMA}} = u_{\text{low}}$	$\overline{u} = u_{\text{high}}$	$(n/b)u_{\text{low}} + bu_{\text{high}}$
Block FMA	$u_{\text{FMA}}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

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

Conclusion: choice of u_{FMA} critical, $\overline{u}$ less so

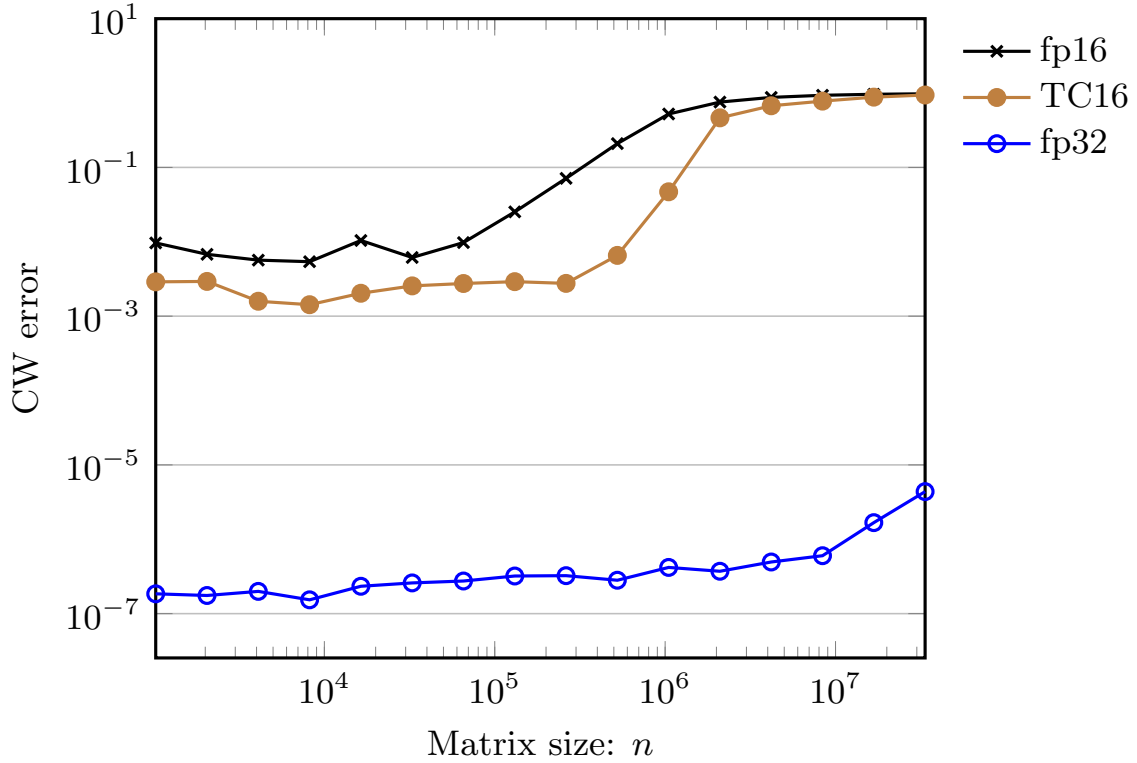
Matrix multiplication with tensor cores

Standard fp16	Tensor core TC16 ($u_{\text{FMA}} = \overline{u} = u_{16}$)	Tensor core TC32 ($u_{\text{FMA}} = \overline{u} = u_{32}$)	Standard fp32
nu_{16}	$(n/4)u_{16}$	$2u_{16} + (n/4)u_{32}$	nu_{32}



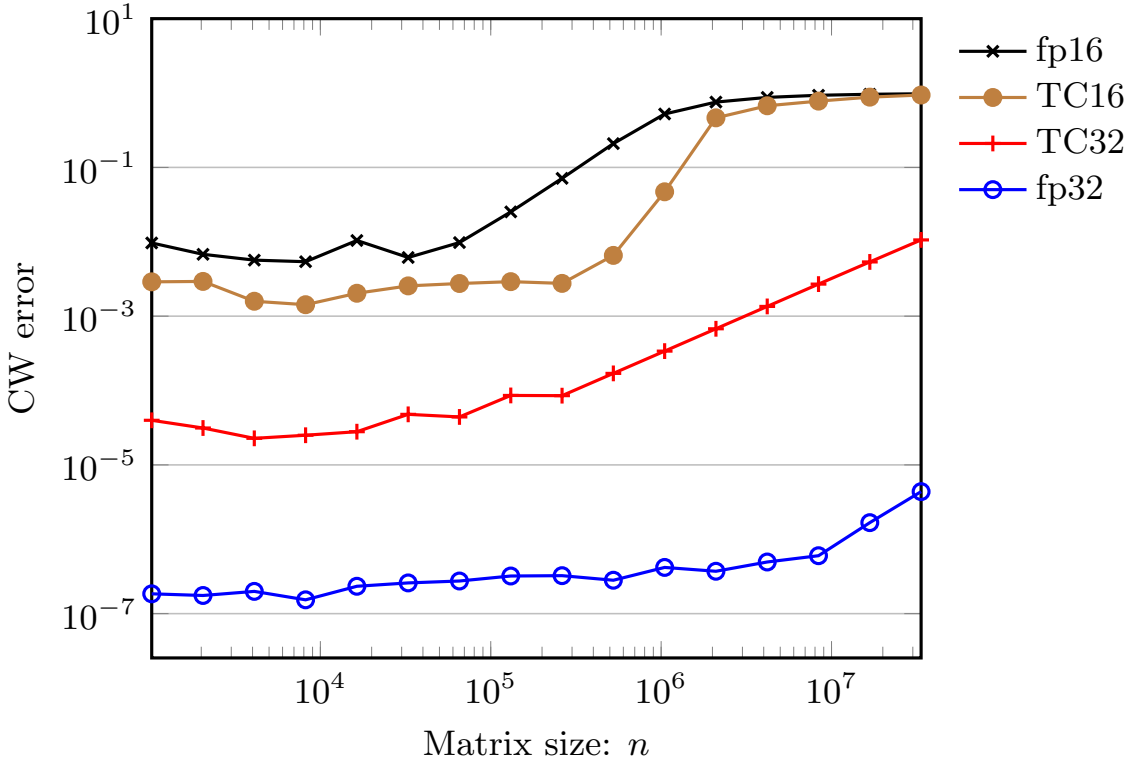
Matrix multiplication with tensor cores

Standard fp16	Tensor core TC16 ($u_{\text{FMA}} = \overline{u} = u_{16}$)	Tensor core TC32 ($u_{\text{FMA}} = \overline{u} = u_{32}$)	Standard fp32
nu_{16}	$(n/4)u_{16}$	$2u_{16} + (n/4)u_{32}$	nu_{32}



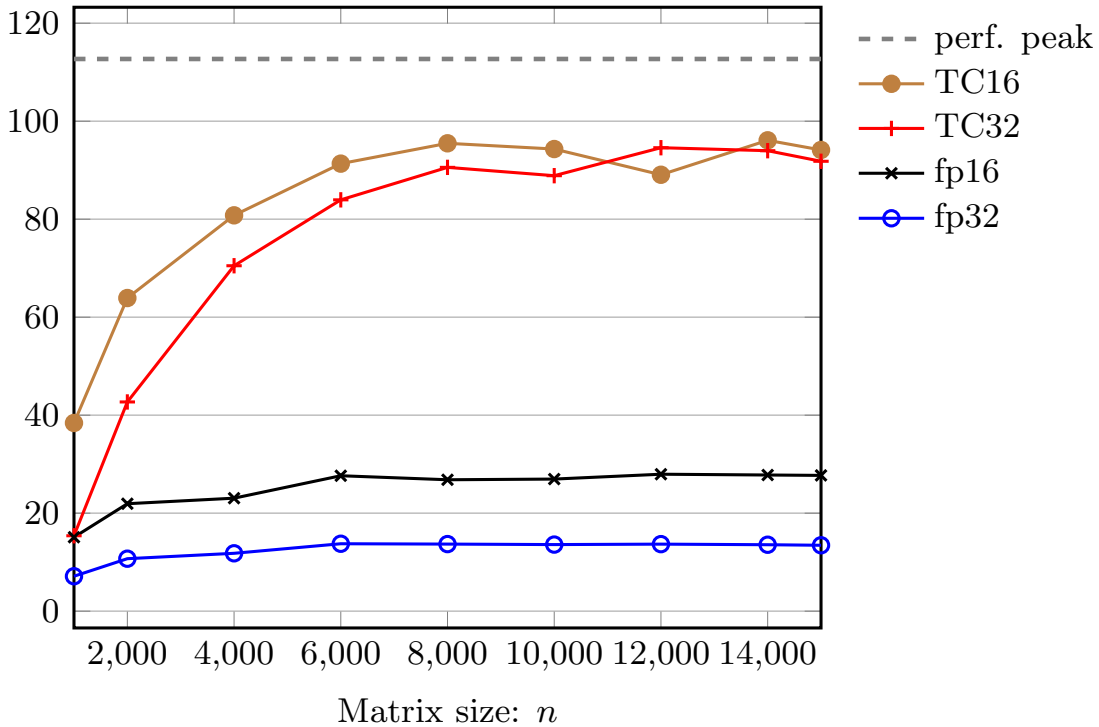
Matrix multiplication with tensor cores

Standard fp16	Tensor core TC16 ($u_{\text{FMA}} = \overline{u} = u_{16}$)	Tensor core TC32 ($u_{\text{FMA}} = \overline{u} = u_{32}$)	Standard fp32
nu_{16}	$(n/4)u_{16}$	$2u_{16} + (n/4)u_{32}$	nu_{32}



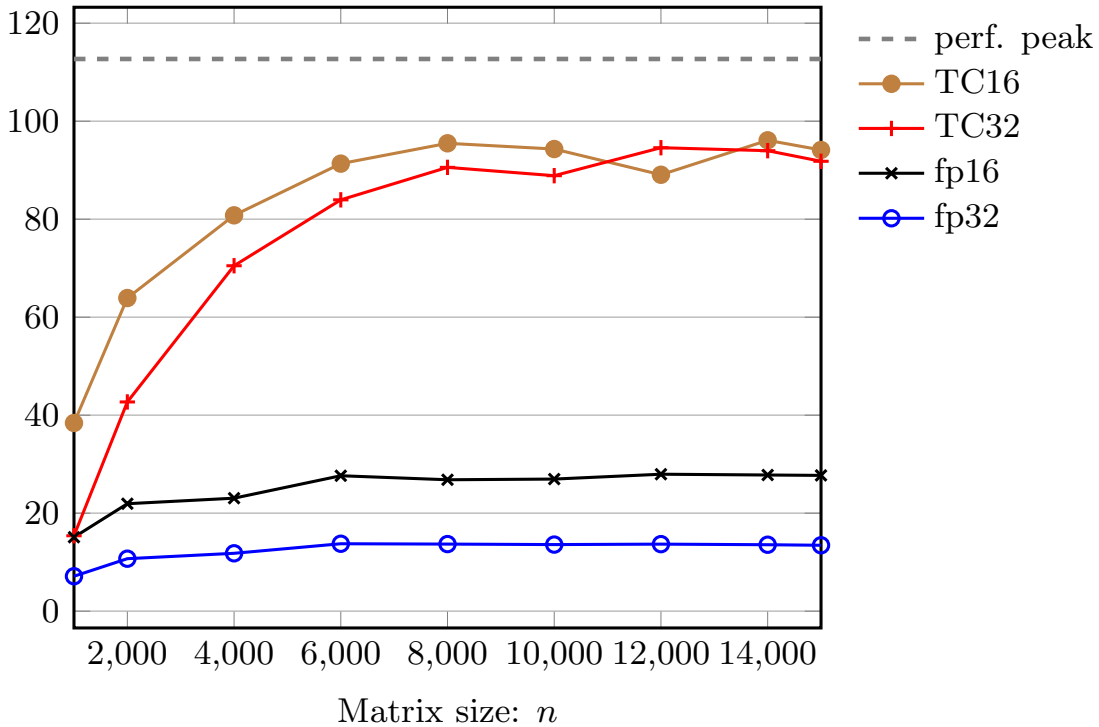
Matrix multiplication with tensor cores

Standard fp16	Tensor core TC16 ($u_{\text{FMA}} = \overline{u} = u_{16}$)	Tensor core TC32 ($u_{\text{FMA}} = \overline{u} = u_{32}$)	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 ($u_{\text{FMA}} = \overline{u} = u_{16}$)	Tensor core TC32 ($u_{\text{FMA}} = \overline{u} = u_{32}$)	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

Reducing accumulation without tensor cores?

Existing algorithms to avoid error accumulation are expensive.
For example, **compensated summation** Kahan (1965)

$s = 0;\ e = 0;$
for $i = 1 : n$ **do**
  $y = x_i + e;$
  $t = s; \quad s = t + y;$
  $e = (t - s) + y;$
end for

yields an **error bound** $2u$ but is $4\times$ **more expensive**

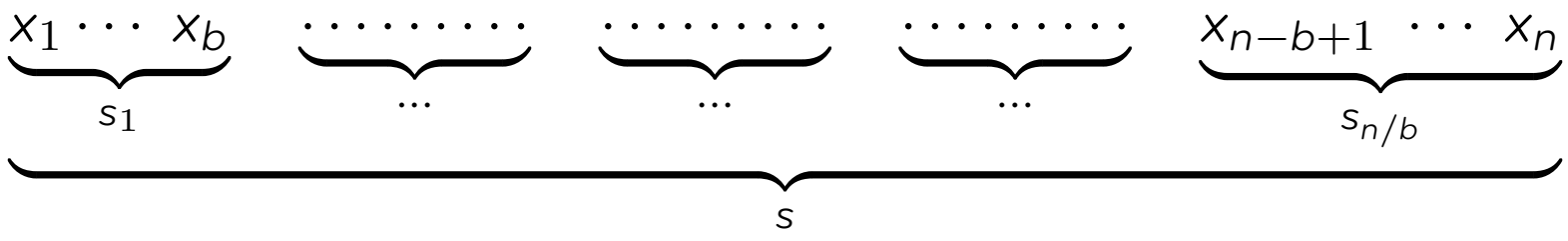
$\Rightarrow$ Not suited for low precisions: simply using higher precision would be cheaper!

Can we design more accurate algorithms while preserving high performance?

Blocked summation

Classical Blocked summation algorithm:

for $i = 1 : n/b$ **do**
  Compute $s_i = \sum_{j=(i-1)b+1}^{ib} x_j$.
end for
Compute $s = \sum_{i=1}^{n/b} s_i$.

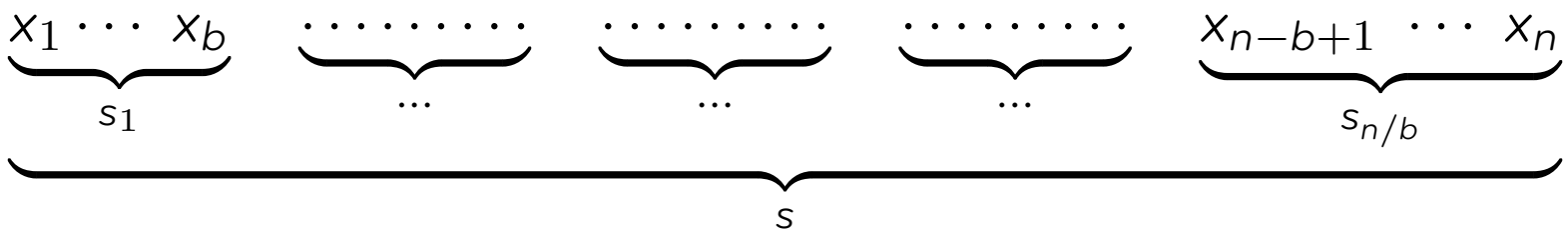


- Widely used in NLA libraries (BLAS, LAPACK, ...)
- Error bound $nu \to (b + n/b)u$

Blocked summation

Fast Accurate Blocked summation algorithm (FABsum):

for $i = 1 : n/b$ **do**
  Compute $s_i = \sum_{j=(i-1)b+1}^{ib} x_j$ with `FastSum`.
end for
Compute $s = \sum_{i=1}^{n/b} s_i$ with `AccurateSum`.

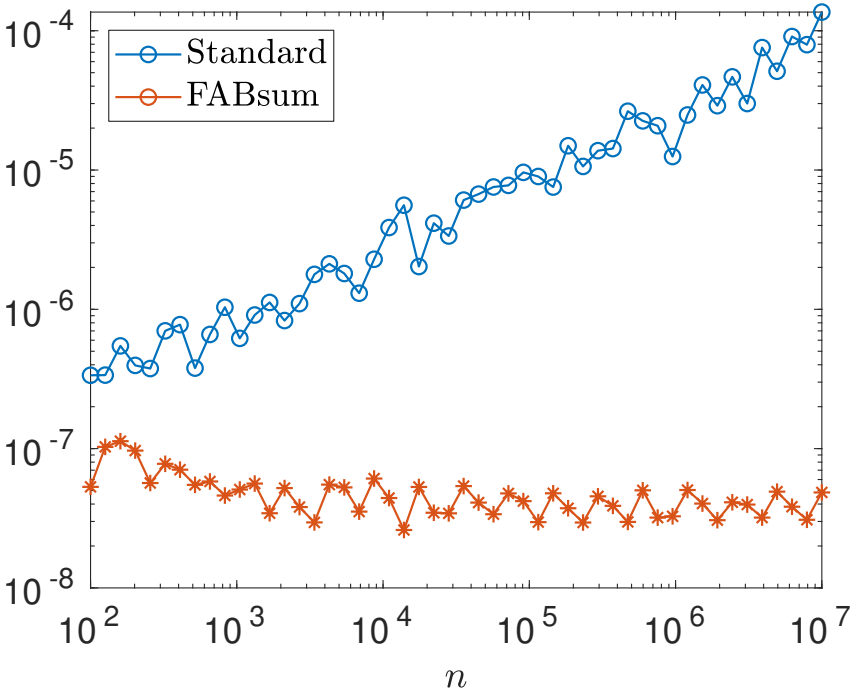


- Widely used in NLA libraries (BLAS, LAPACK, ...)
- Error bound $nu \to (b + n/b)u \to bu$ with FABsum
- Only n/b **additions with `AccurateSum`**
- Blanchard, Higham, and Mary (2020)

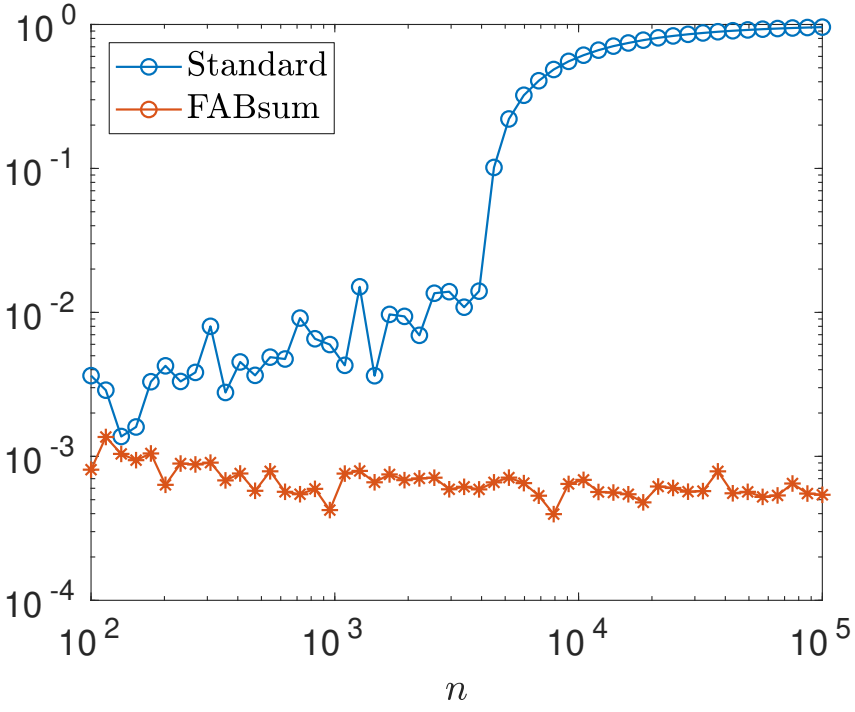
FABsum: numerical results

Backward error for summing random uniform $[0, 1]$ data

Single precision

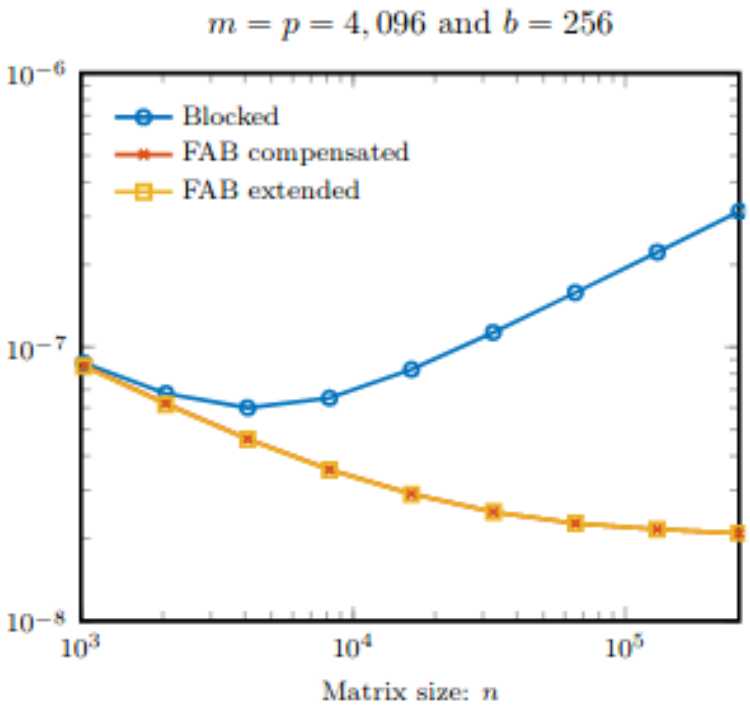


Half precision

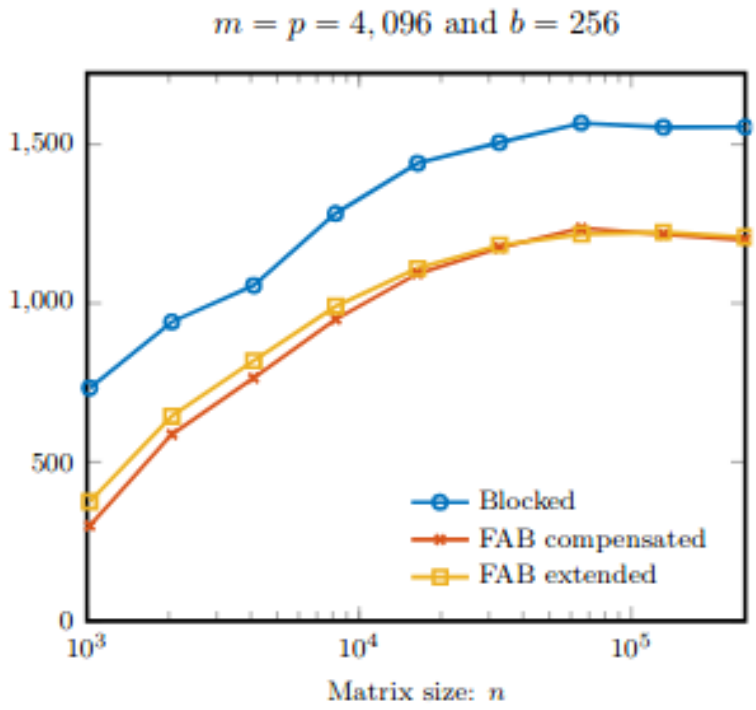


Implementation of FABsum in PLASMA library (tiled matrix NLA operations)

- First approach: set b equal to tile size t



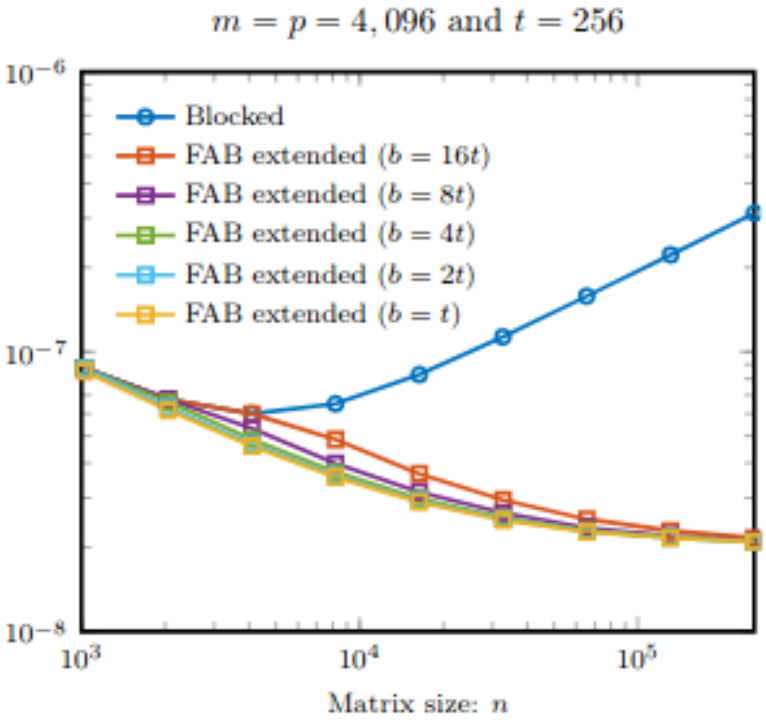
(a) Error (5.1).



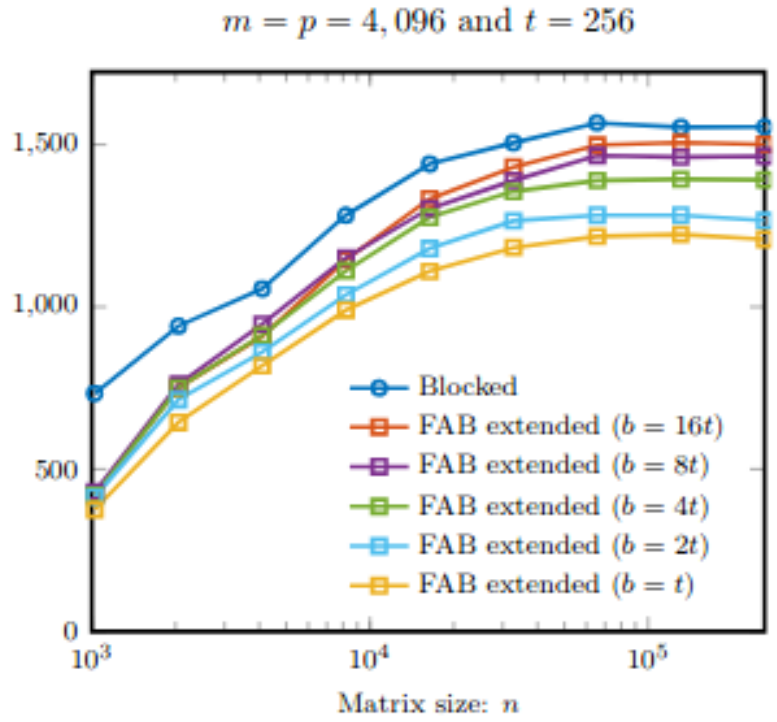
(b) Performance in Gflops (using 28 cores).

Implementation of FABsum in PLASMA library (tiled matrix NLA operations)

- First approach: set b equal to tile size t
- Improved approach: allow $b \neq t$ ($b \gg t$ interesting here)



(a) Error (5.1).



(b) Performance in Gflops (using 28 cores).

Solving $Ax = b$

Standard method to solve $Ax = b$:

1. Factorize $A = LU$, where L and U are lower and upper triangular
2. Solve $Ly = b$ and $Ux = y$

Solving $Ax = b$

Standard method to solve $Ax = b$:

1. Factorize $A = LU$, where L and U are lower and upper triangular
2. Solve $Ly = b$ and $Ux = y$

An algorithm to refine the solution: **iterative refinement** (IR)

Solve $Ax_1 = b$ as above
for $i = 1 : nsteps$ **do**
    $r_i = b - Ax_i$
    Solve $Ad_i = r_i$ via $d_i = U^{-1}(L^{-1}r_i)$
    $x_{i+1} = x_i + d_i$
end for

- Newton's method for $f(x) = Ax - b$
- Many variants over the years, depending on choice of precisions and solver for $Ad_i = r_i$

Error analysis of general IR

Carson and Higham (2018) analyze the most general version of IR to date:

Solve $Ax_1 = b$ by LU factorization at precision u_f
for $i = 1 : nsteps$ **do**
  $r_i = b - Ax_i$ at precision u_r
  Solve $Ad_i = r_i$ such that $\|\widehat{d}_i - d_i\| \leq \phi_i \|d_i\|$
  $x_{i+1} = x_i + d_i$ at precision u
end for

Theorem (simplified from Carson and Higham, 2018)

Under the condition $\phi_i < 1$, the forward error is reduced at each step by a factor ϕ_i until it reaches its limiting value

$$\frac{\|\widehat{x} - x\|}{\|x\|} \leq u_r \kappa(A) + u$$

Choice of solver determines ϕ_i:

- **LU solver:** $d_i = U^{-1}L^{-1}r_i \quad \Rightarrow \quad \|\widehat{d}_i - d_i\| \leq \kappa(A)u_f\|d_i\|$

$\Rightarrow \ \phi_i < 1 \Leftrightarrow \kappa(A)u_f < 1$

Choice of solver determines ϕ_i:

- **LU solver:** $d_i = U^{-1}L^{-1}r_i \quad \Rightarrow \quad \|\widehat{d}_i - d_i\| \leq \kappa(A)u_f\|d_i\|$

$\Rightarrow \ \phi_i < 1 \Leftrightarrow \kappa(A)u_f < 1$

- **GMRES solver:** solve $Ad_i = r_i$ via GMRES preconditioned by LU factors
- $\kappa(U^{-1}L^{-1}A) \leq (1 + \kappa(A)u_f)^2 \quad \Rightarrow \quad$ potentially much lower than $\kappa(A)$
- Precise value of ϕ_i depends on precisions used within GMRES
- Best possible condition: $\kappa(A)^2 u_f^2 u < 1$

Solve $Ax_1 = b$ by LU factorization — in precision u_f
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ — in precision u_r
  Solve $Ad_i = r_i$
  $x_{i+1} = x_i + d_i$ — in precision u
end for

u_f	u	u_r	max $\kappa(A)$	Forward error

Solve $Ax_1 = b$ by LU factorization — $u_f =$ **double**

for $i = 1 : nsteps$ **do**

$r_i = b - Ax_i$ — $u_r =$ **quadruple**

Solve $Ad_i = r_i$ via $d_i = U^{-1}(L^{-1}r_i)$

$x_{i+1} = x_i + d_i$ — $u =$ **double**

end for

Traditional

Wilkinson (1948) Moler (1967)

	u_f	u	u_r	max $\kappa(A)$	Forward error
Traditional	D	D	Q	10^{16}	10^{-16}

Solve $Ax_1 = b$ by LU factorization — $u_f =$ **double**

for $i = 1 : nsteps$ **do**

$r_i = b - Ax_i$ — $u_r =$ **double**

Solve $Ad_i = r_i$ via $d_i = U^{-1}(L^{-1}r_i)$

$x_{i+1} = x_i + d_i$ — $u =$ **double**

end for

Fixed-precision

Jankowski and Wozniakowski (1977) Skeel (1980)

	u_f	u	u_r	max $\kappa(A)$	Forward error
Traditional	D	D	Q	10^{16}	10^{-16}
Fixed	D	D	D	10^{16}	$\kappa(A) \cdot 10^{-16}$

Solve $Ax_1 = b$ by LU factorization — $u_f =$ **single**

for $i = 1 : nsteps$ **do**

$r_i = b - Ax_i$ — $u_r =$ **double**

Solve $Ad_i = r_i$ via $d_i = U^{-1}(L^{-1}r_i)$

$x_{i+1} = x_i + d_i$ — $u =$ **double**

end for

Low precision factorization

Langou et al (2006)

	u_f	u	u_r	max $\kappa(A)$	Forward error
Traditional	D	D	Q	10^{16}	10^{-16}
Fixed	D	D	D	10^{16}	$\kappa(A) \cdot 10^{-16}$
LP factorization	S	D	D	10^{8}	$\kappa(A) \cdot 10^{-16}$

Solve $Ax_1 = b$ by LU factorization — $u_f =$ **single**

for $i = 1 : nsteps$ **do**

$r_i = b - Ax_i$ — $u_r =$ **quadruple**

Solve $Ad_i = r_i$ via $d_i = U^{-1}(L^{-1}r_i)$

$x_{i+1} = x_i + d_i$ — $u =$ **double**

end for

Three precisions

Carson and Higham (2018)

	u_f	u	u_r	max $\kappa(A)$	Forward error
Traditional	D	D	Q	10^{16}	10^{-16}
Fixed	D	D	D	10^{16}	$\kappa(A) \cdot 10^{-16}$
LP factorization	S	D	D	10^{8}	$\kappa(A) \cdot 10^{-16}$
3 precisions	S	D	Q	10^{8}	10^{-16}

Solve $Ax_1 = b$ by LU factorization — $u_f =$ **single**

for $i = 1 : nsteps$ **do**

$r_i = b - Ax_i$ — $u_r =$ **quadruple**

Solve $Ad_i = r_i$ via preconditioned GMRES

$x_{i+1} = x_i + d_i$ — $u =$ **double**

end for

GMRES-based

Carson and Higham (2017)

	u_f	u	u_r	max $\kappa(A)$	Forward error
Traditional	D	D	Q	10^{16}	10^{-16}
Fixed	D	D	D	10^{16}	$\kappa(A) \cdot 10^{-16}$
LP factorization	S	D	D	10^{8}	$\kappa(A) \cdot 10^{-16}$
3 precisions	S	D	Q	10^{8}	10^{-16}
GMRES-based	S	D	Q	Up to 10^{16}	10^{-16}

Solve $Ax_1 = b$ by LU factorization — $u_f =$ **half**

for $i = 1 : nsteps$ **do**

$r_i = b - Ax_i$ — $u_r =$ **quadruple**

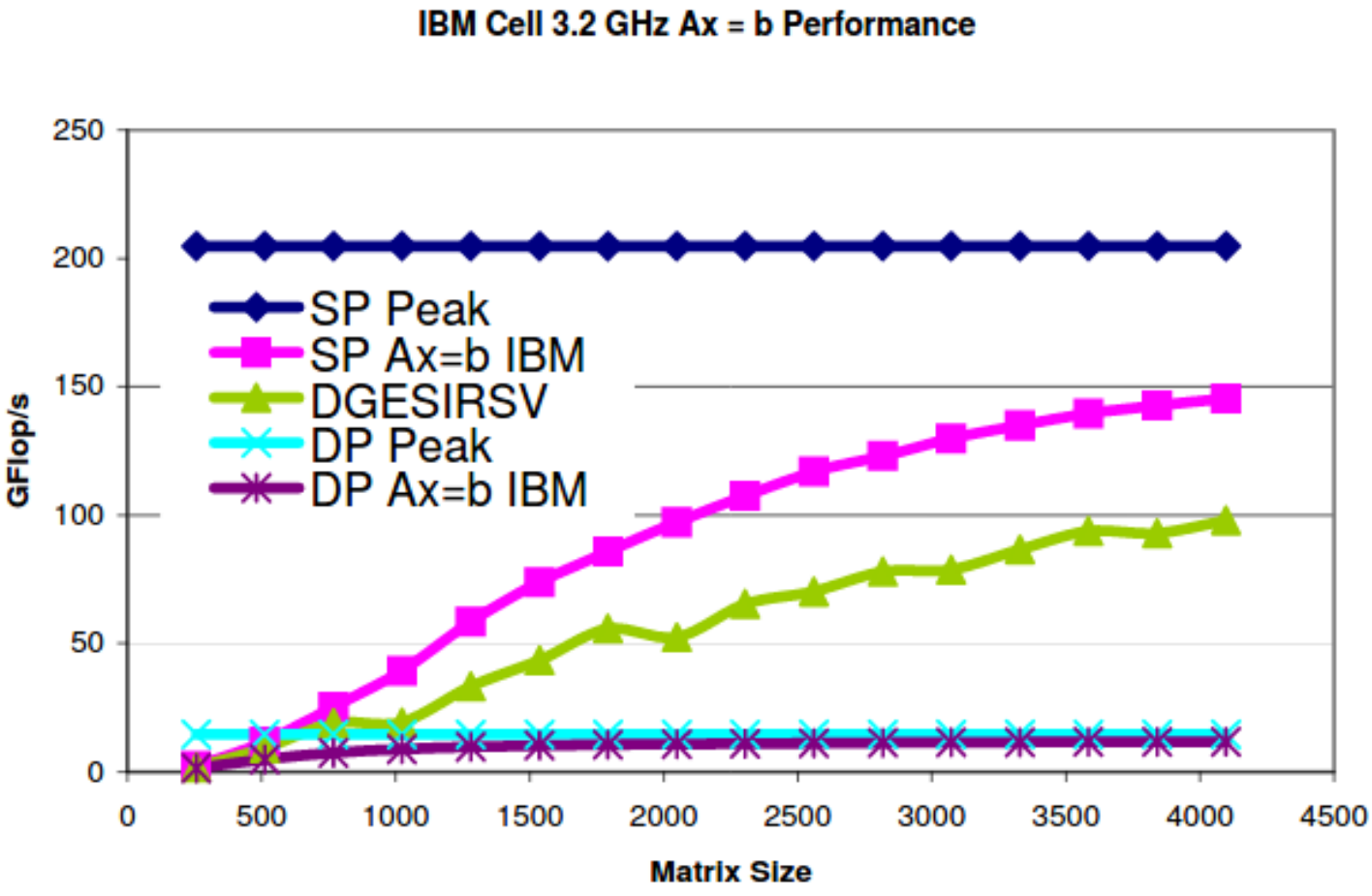
Solve $Ad_i = r_i$

$x_{i+1} = x_i + d_i$ — $u =$ **double**

end for

	u_f	u	u_r	max $\kappa(A)$	Forward error
Traditional	D	D	Q	10^{16}	10^{-16}
Fixed	D	D	D	10^{16}	$\kappa(A) \cdot 10^{-16}$
LP factorization	H	D	D	10^{4}	$\kappa(A) \cdot 10^{-16}$
3 precisions	H	D	Q	10^{4}	10^{-16}
GMRES-based	H	D	Q	Up to 10^{12}	10^{-16}

Many new variants possibe with half precision!



Langou et al (2006)

LU factorization (Gaussian elimination)

- Objective: given $A \in \mathbb{R}^{n \times n}$, compute lower and upper triangular matrices L and U such that $A = LU$
- $\forall i, j \quad a_{ij} = \sum_{k=1}^{\min(i,j)} \ell_{ik} u_{kj}$

for $k = 1 : n$ **do**
  $u_{kk} = a_{kk}$ $(\ell_{kk} = 1)$
  for $i = k + 1 : n$ **do**
    $\ell_{ik} = a_{ik}/u_{kk}$ and $u_{ki} = a_{ki}$
  end for
  for $i = k + 1 : n/b$ **do**
    for $j = k + 1 : n/b$ **do**
      $a_{ij} \leftarrow a_{ij} - \ell_{ik} u_{kj}$
    end for
  end for
end for

- $2n^3/3$ flops

Block LU factorization

- Block version to use matrix–matrix operations

for $k = 1 : n/b$ **do**
  Factorize $L_{kk}U_{kk} = A_{kk}$ (with unblocked alg.)
  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**

      $A_{ij} \leftarrow A_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj}$

    end for
  end for
end for

Block LU factorization with block FMA

- Block version to use matrix–matrix operations
- With a block FMA: $A \in \mathbb{R}^{n\times n}$ is given in precision u_{high}, and L and U are returned in precision u_{FMA}

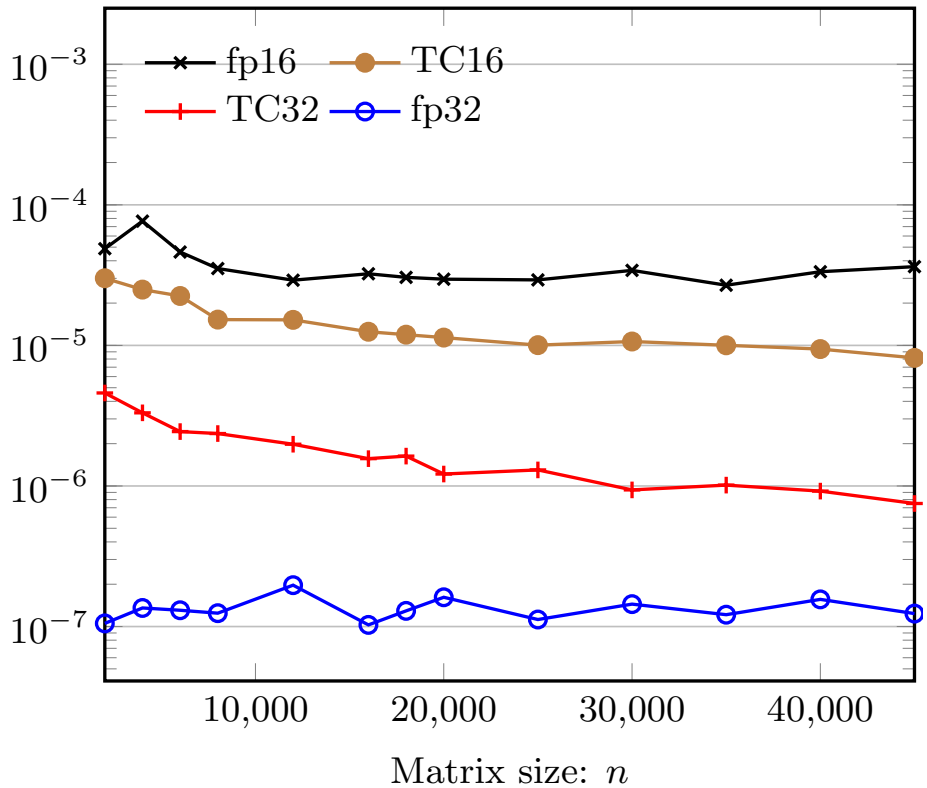
for $k = 1 : n/b$ **do**
  Factorize $L_{kk}U_{kk} = A_{kk}$ (with unblocked alg.)
  for $i = k+1 : n/b$ **do**
    Solve $L_{ik}U_{kk} = A_{ik}$ and $L_{kk}U_{ki} = A_{ki}$ for L_{ik} and U_{ki}
  end for
  for $i = k+1 : n/b$ **do**
    for $j = k+1 : n/b$ **do**
      $\widetilde{L}_{ik} \leftarrow \text{fl}_{\text{low}}(L_{ik})$ and $\widetilde{U}_{ki} \leftarrow \text{fl}_{\text{low}}(U_{ki})$
      $A_{ij} \leftarrow A_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj}$ using a block FMA
    end for
  end for
end for

- $O(n^3)$ part of the flops done with block FMA

LU factorization with tensor cores

Error analysis for LU follows from matrix multiplication analysis and gives same bounds to first order Blanchard et al. (2020)

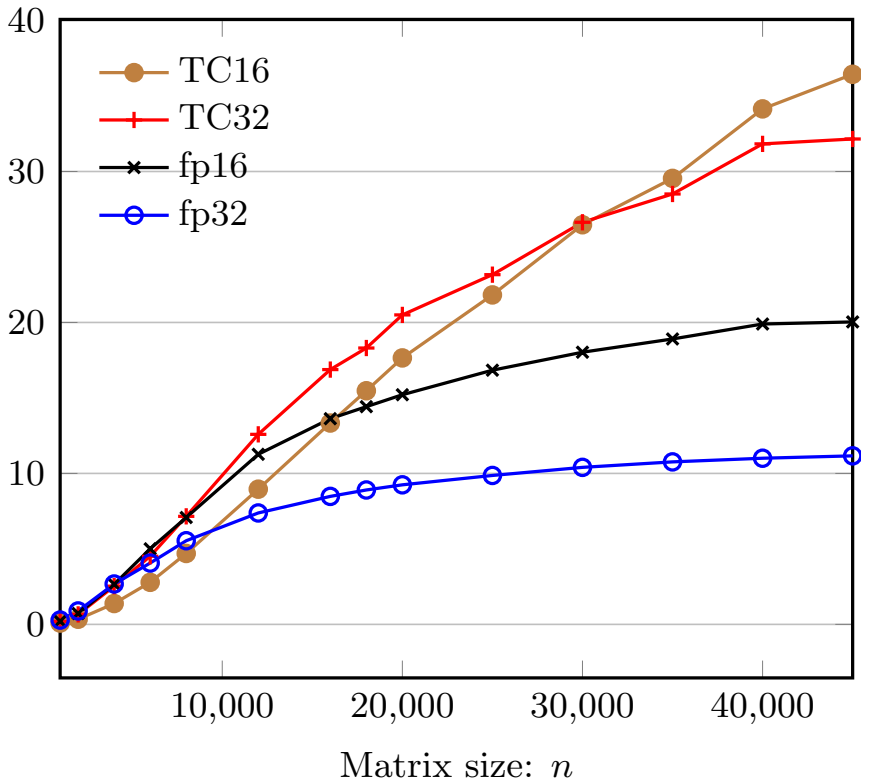
Standard fp16	Tensor core TC16	Tensor core TC32	Standard fp32
nu_{16}	$n/4u_{16}$	$2u_{16} + (n/4)u_{32}$	nu_{32}



LU factorization with tensor cores

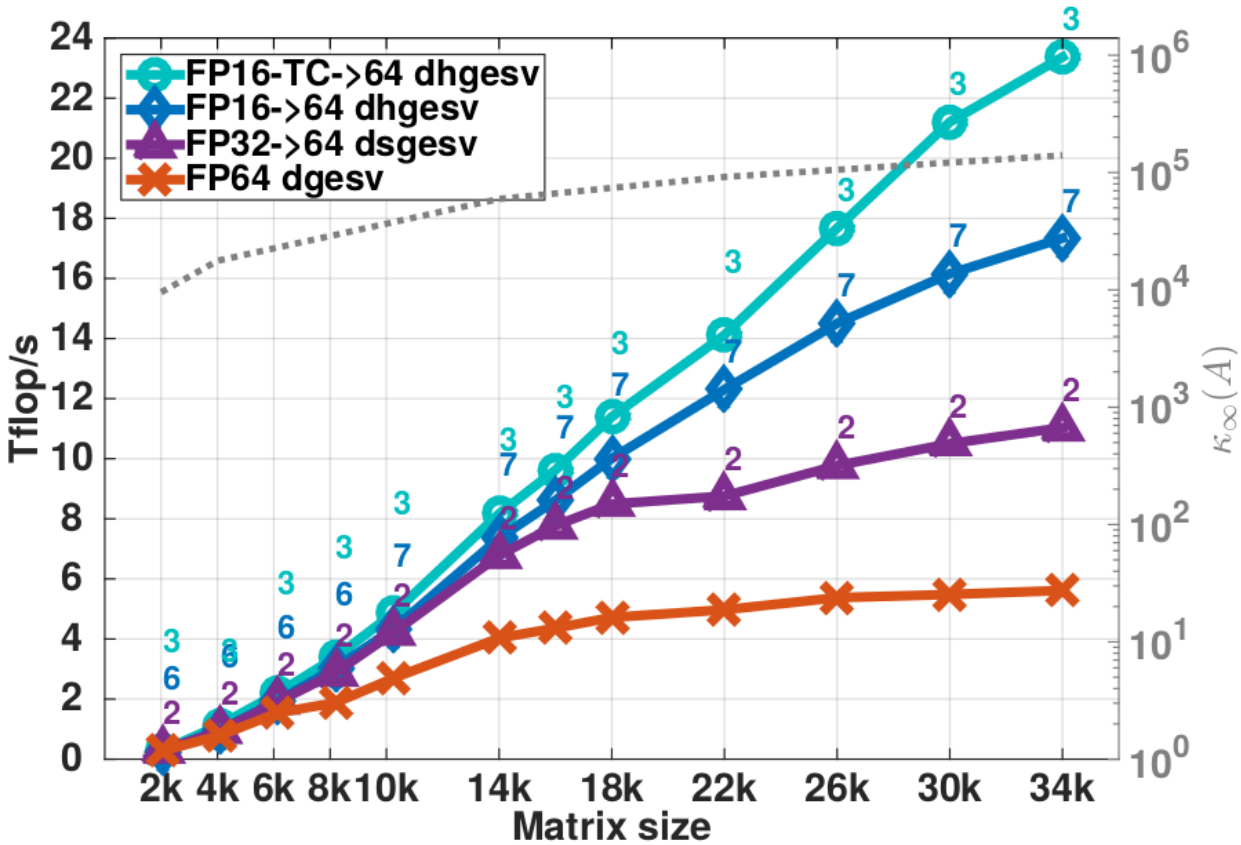
Error analysis for LU follows from matrix multiplication analysis and gives same bounds to first order Blanchard et al. (2020)

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



Impact on iterative refinement

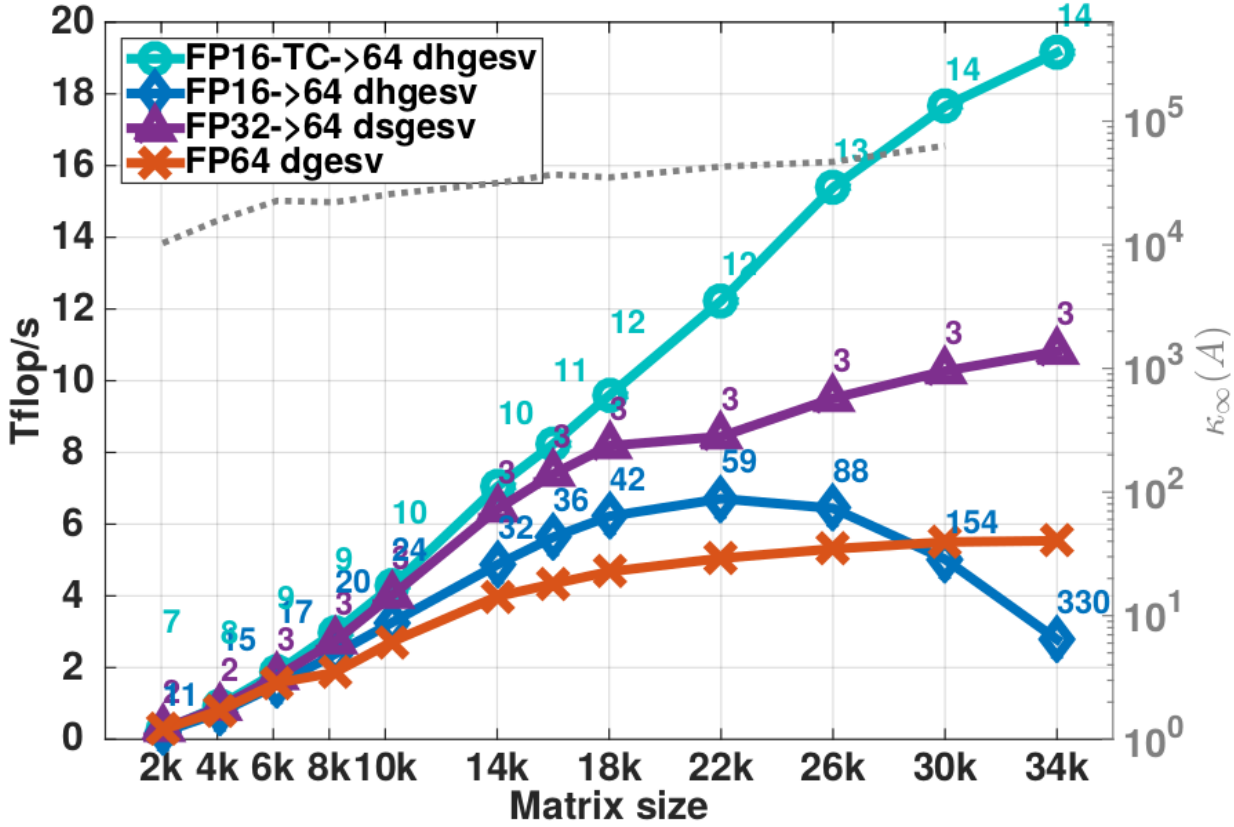
Results from Haidar et al. (2018)



- TC performance suboptimal here, improved version by Lopez and Mary (2020) surpasses 50 TFLOPS mark

Impact on iterative refinement

Results from Haidar et al. (2018)



- TC performance suboptimal here, improved version by Lopez and Mary (2020) surpasses 50 TFLOPS mark
- TC accuracy boost sometimes critical!

GMRES 101

- Krylov method to solve $Ax = b$: build orthogonal basis $B_k = \{r_0, Ar_0, A^2 r_0, \ldots, A^k r_0\}$
- At iteration k:
  - Add $q_k = A^{k-1} r_0 = A q_{k-1}$ to the basis $\Rightarrow$ matrix–vector product with A
  - Find $x_k \in B_k$ minimizing $r_k = Ax_k - b$
  - Repeat if $\|r_k\|$ is not small enough
- Convergence strongly depends on matrix $\Rightarrow$ preconditioning is needed
- Preconditioned GMRES: apply GMRES to $MAx = Mb$ where $M \approx A^{-1}$
- Low precision LU preconditioner: $M = U^{-1} L^{-1}$ $\Rightarrow$ requires triangular solves

GMRES solver in IR

- Goal: solve $Ad_i = r_i$ with GMRES and bound $\phi_i = \|\widehat{d}_i - d_i\|/\|d_i\|$
- Theorem from Paige, Rozloznik, Strakos (2006) : if unpreconditioned GMRES run in precision u converges to a solution $\widehat{x}$ of $Ax = b$, then $\widehat{x}$ satisfies $\|\widehat{x} - x\| \lesssim u\kappa(A)\|x\| \Rightarrow$ $\phi_i = \kappa(A)u$
- What about LU-preconditioned GMRES? Two key differences:
  - The system being solved is $\widetilde{A}x = b \Rightarrow$ bound becomes of order $\kappa(\widetilde{A})u$, where $\kappa(\widetilde{A}) \leq u_f^2\kappa(A)^2$
  - The matrix–vector products are performed with $\widetilde{A} = U^{-1}L^{-1}A$
    - $y = Ax \Rightarrow \|\widehat{y} - y\| \leq u\|A\|\|x\|$
    - $y = U^{-1}L^{-1}Ax \Rightarrow \|\widehat{y} - y\| \leq u\|A\|\|U^{-1}\|\|L^{-1}\|\|x\| \lesssim \kappa(A)u\|\widetilde{A}\|\|x\|$

    $\Rightarrow$ extra $\kappa(A)$ term appears, it is as if GMRES was run in "precision" $\kappa(A)u$
- Overall: $\phi_i = \kappa(\widetilde{A})\kappa(A)u$

Five precision GMRES-IR

- Matrix–vector products bring extra $\kappa(A)$ $\Rightarrow$ perform them in higher precision?
- Conversely, we only need $\phi_i < 1$, can use lower precision?

$\Rightarrow$ General mixed precision GMRES: matvec in precision $u_p \leq u^2$, rest of GMRES in precision $u_g \leq u$

$\Rightarrow$ $\phi_i = \kappa(\widetilde{A})\big(\kappa(A)u_p + u_g\big)u$

Solve $Ax_1 = b$ by LU factorization at precision u_f
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ at precision u_r
  Solve $Ad_i = r_i$ with preconditioned GMRES in precision u_g except matvecs in precision u_p
  $x_{i+1} = x_i + d_i$ at precision u
end for

FIVE precisions in total!

Maximal $\kappa(A)$ for which convergence is guaranteed

with $u_f = H$ and $u = D$

LU-IR condition: $\kappa(A) < 2 \times 10^3$

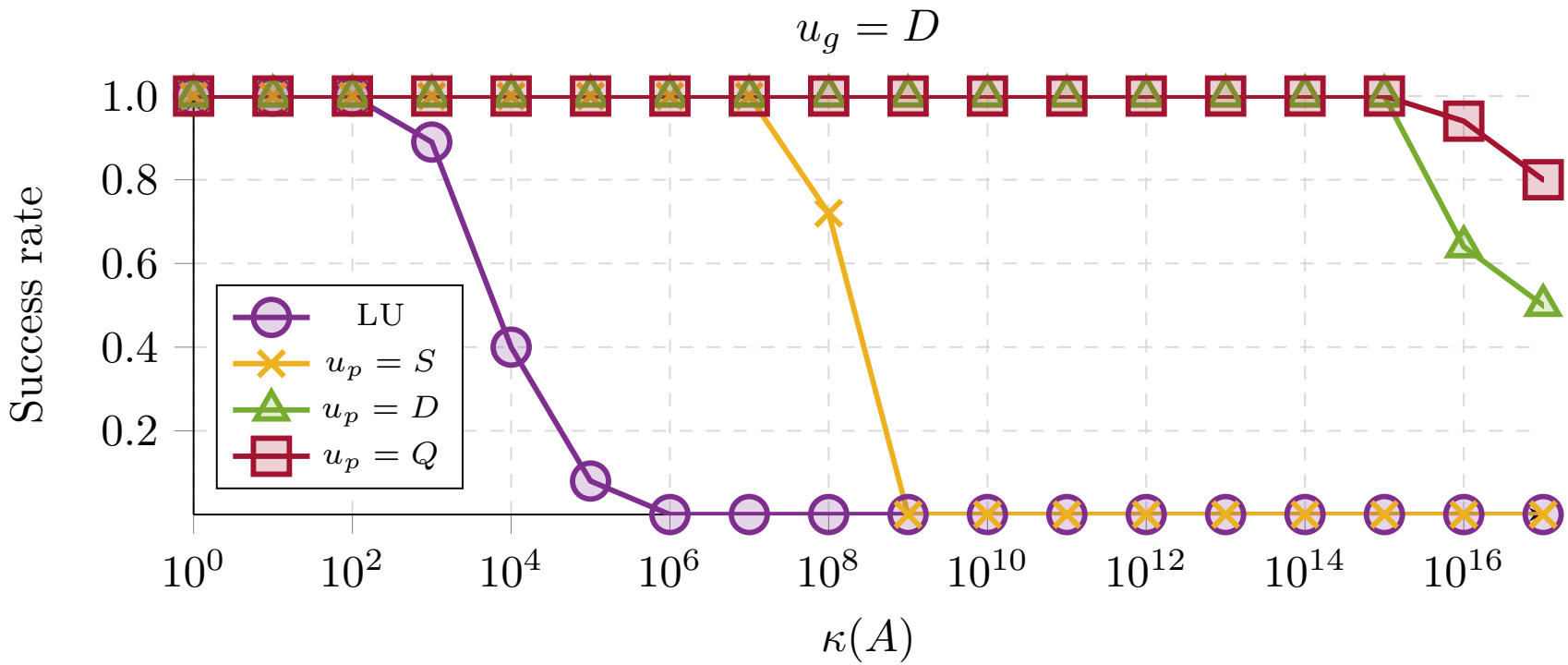
	$u_p = Q$	$u_p = D$	$u_p = S$
$u_g = D$	2×10^{11}	3×10^7	4×10^4
$u_g = S$	8×10^6	8×10^6	4×10^4
$u_g = H$	9×10^4	9×10^4	4×10^4

Maximal $\kappa(A)$ for which convergence is guaranteed

with $u_f = H$ and $u = D$

LU-IR condition: $\kappa(A) < 2 \times 10^3$

	$u_p = Q$	$u_p = D$	$u_p = S$
$u_g = D$	2×10^{11}	3×10^7	4×10^4
$u_g = S$	8×10^6	8×10^6	4×10^4
$u_g = H$	9×10^4	9×10^4	4×10^4

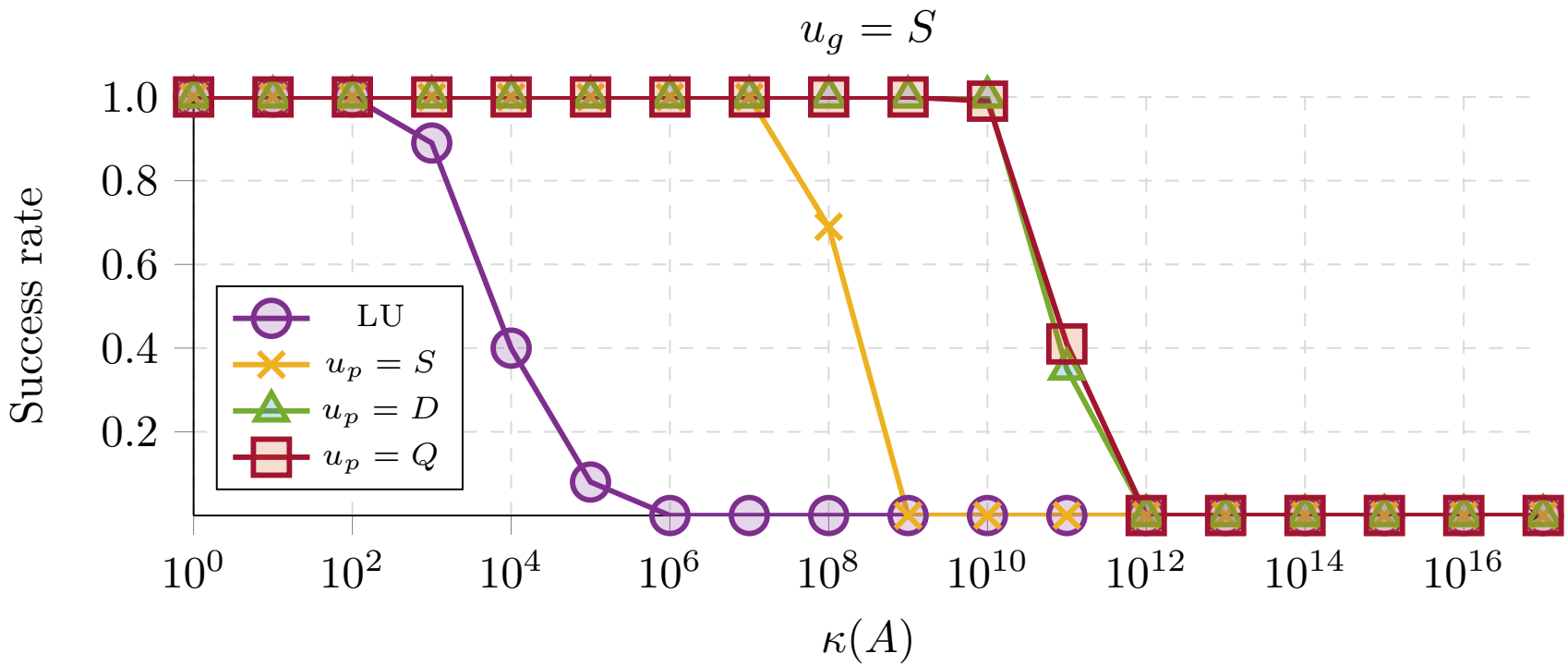


Maximal $\kappa(A)$ for which convergence is guaranteed

with $u_f = H$ and $u = D$

LU-IR condition: $\kappa(A) < 2 \times 10^3$

	$u_p = Q$	$u_p = D$	$u_p = S$
$u_g = D$	2×10^{11}	3×10^7	4×10^4
$u_g = S$	8×10^6	8×10^6	4×10^4
$u_g = H$	9×10^4	9×10^4	4×10^4

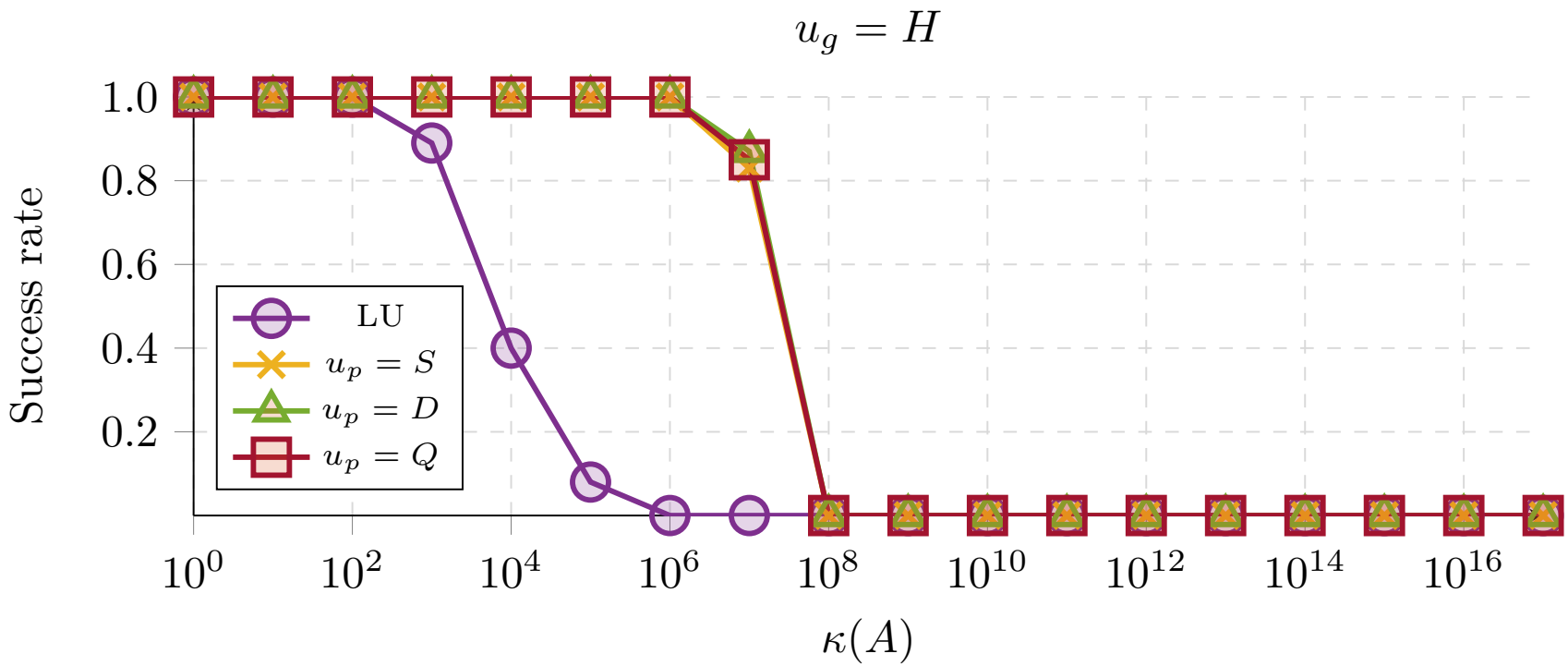


Maximal $\kappa(A)$ for which convergence is guaranteed

with $u_f = H$ and $u = D$

LU-IR condition: $\kappa(A) < 2 \times 10^3$

	$u_p = Q$	$u_p = D$	$u_p = S$
$u_g = D$	2×10^{11}	3×10^7	4×10^4
$u_g = S$	8×10^6	8×10^6	4×10^4
$u_g = H$	9×10^4	9×10^4	4×10^4



Half–half arithmetic

The idea: barring overflow/underflow, we can split A as

$A = A_1 + A_2 + E$	with A_i in fp16	Markidis et al. (2018)
$A = A_1 + A_2 + A_3 + E$	with A_i in bfloat16	Henry et al. (2019)

with $|E| \leq cu_{32}|A|$. Split B similarly and compute $C = AB$ as

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

Similar to double-double arithmetic, but different

- Typical use of double-double: input in quad, target accuracy is quad, computations in double with compensated techniques
- Here: input at least single, target is single, computations with tensor cores $\Rightarrow$ requires converting input to half

For any $x \in \mathbb{R}$, let

$$x_1 = \mathrm{fl}_{\mathrm{low}}(x) = x(1+\delta_1), \quad |\delta_1| \le u_{\mathrm{low}}$$
$$x_2 = \mathrm{fl}_{\mathrm{low}}(x - x_1) = -x\delta_1(1+\delta_2), \quad |\delta_2| \le u_{\mathrm{low}}$$

Thefore

$$x_1 + x_2 = x - x\delta_1\delta_2 = x(1+\delta), \quad |\delta| \le u_{\mathrm{low}}^2.$$

Applying this idea recursively:

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

Using this representation elementwise on A and B:

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

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:

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

The precise constant is very dependent on implementation: up to $p^2 n$ with recursive summation, $n + p^2$ is for blocked summation (each $A_i B_j$ is computed independently). Overall

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

Small p is enough for practical choices of u_{low} and u_{high}:

- $p = 2$ for fp16 and fp32 ($u_{\text{low}}^2 = 4u_{\text{high}}$). Minor improvement for $p \geq 3$ because $\gamma_{n+p^2}^{\text{high}}$ starts dominating
- $p = 2$ and $p = 3$ both of interest for bfloat16 and fp32

Not all p^2 products A_iB_j need be computed!

$$|A_i| \le u_{\text{low}}^{i-1}(1+u_{\text{low}})|A|, \quad i = 1\colon p$$
$$|B_j| \le u_{\text{low}}^{j-1}(1+u_{\text{low}})|B|, \quad j = 1\colon p$$

and thus

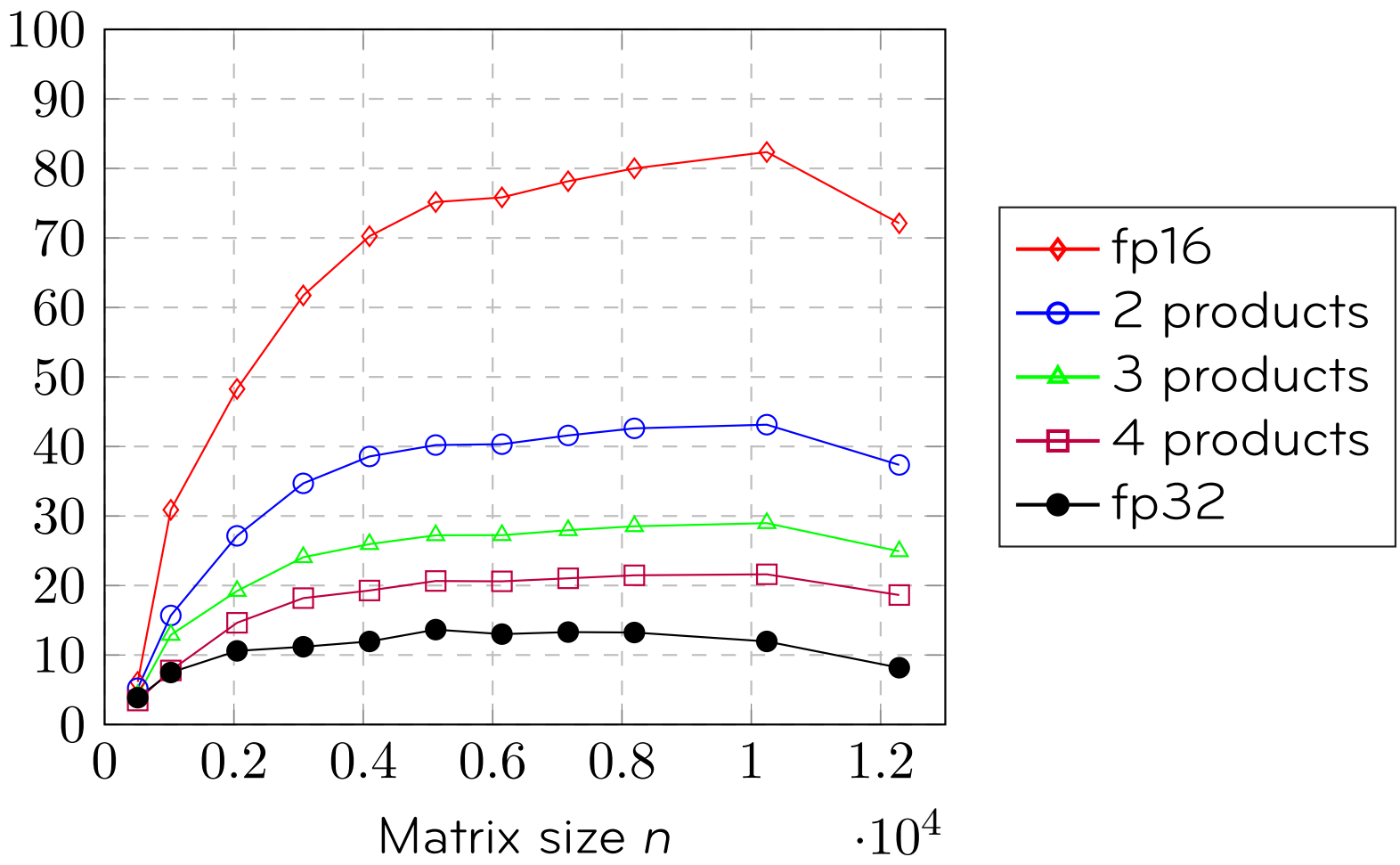
$$|A_i||B_j| \le u_{\text{low}}^{i+j-2}(1+u_{\text{low}})^2|A||B|.$$

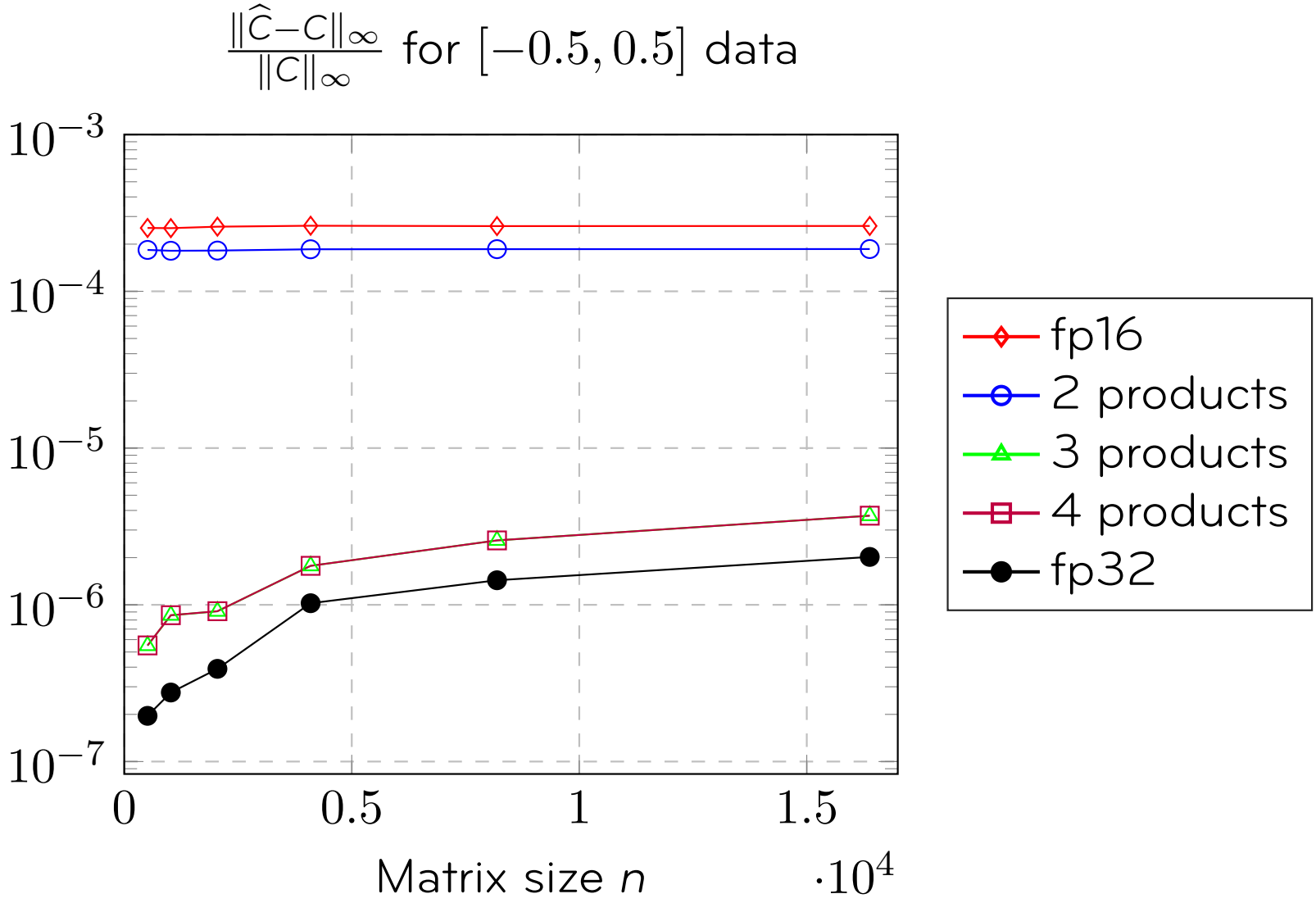
Therefore we can skip any product A_iB_j such that $i+j > p+1$ and obtain the modified bound $\widehat{C} = AB + E$,

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

- error to order u_{low}^{p}: constant $2 \to p+1$
- number of products: $p^2 \to p(p+1)/2$
- further reducing the number of products: not useful (similar error as $p-1$ splits)

Experiments with V100 (credits: Mantas Mikaitis)




$\frac{\|\widehat{C}-C\|_\infty}{\|C\|_\infty}$ for $[-0.5, 0.5]$ data
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
0
0.5
1
1.5
Matrix size n
$\cdot 10^4$
fp16
2 products
3 products
4 products
fp32

Conclusions

- **Low precision arithmetics** provide great opportunities to tackle today's HPC challenges, but also put at risk the stability and accuracy of computations

$\Rightarrow$ **Mixed precision arithmetic** has emerged as a highly successful compromise

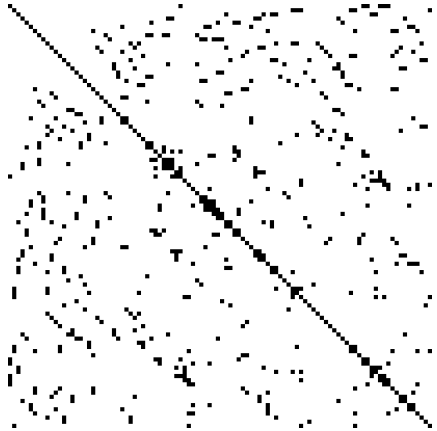
- More and more **hardware** possess intrinsic mixed precision capabilities $\Rightarrow$ crucial to understand them and exploit them effectively
- Mixed precision variants being developed for a wide range of numerical algorithms

Slides available at
https://www-pequan.lip6.fr/~tmary/doc/HPCA.pdf

Sujet de stage 1

Topic: mixed precision sparse matrix–vector product (SpMV)

SpMV $y = Ax$: core kernel of numerous applications: PDE solution, graph problems, machine learning, etc.
SpMVs are memory bound: goal is to reduce number of accesses to A



- Ahmad, Sundar, and Hall (2019) : split $A = A_d + A_s$
  - A_d contains "large" elements and is in double precision
  - A_s contains "small" elements and is in single precision
  - Accesses to A_s take half the time
  - Based on the idea that "small" elements matter "less"
- **Questions:**
  - Error analysis? What is "large", what is "small"?
  - Vector x is not taken into account!
  - Impact on convergence of iterative linear solvers?
  - Use of half precision?

Sujet de stage 2

Topic: mixed precision GMRES-based iterative refinement on GPUs

- Goal: solve real-life, ill-conditioned linear systems on modern GPUs
- Main tool: GMRES-based iterative refinement with half precision factorization
- Work direction 1, in collaboration with Univ. of Tennessee (USA): implement variant of Lopez and Mary (2020) that reduces data movement $\Rightarrow$ compromise between speed and accuracy
- Work direction 2, in collaboration with IRIT (France): implement five-precision variant on GPUs $\Rightarrow$ compromise between speed and robustness