ARITH 2021

14 June 2021

Opportunities for Mixed Precision Arithmetic in Numerical Linear Algebra

Theo Mary

Sorbonne Université, CNRS, LIP6

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

Slides available at https://bit.ly/arith21

Today's floating-point landscape

		Bits Signif. (t)	Exp.	Range	$u = 2^{-t}$
bfloat16	B	8	8	$10^{\pm 38}$	4×10^{-3}
fp16	H	11	5	$10^{\pm 5}$	5×10^{-4}
fp32	S	24	8	$10^{\pm 38}$	6×10^{-8}
fp64	D	53	11	$10^{\pm 308}$	1×10^{-16}
fp128	Q	113	15	$10^{\pm 4932}$	1×10^{-34}

Low precision increasingly supported by hardware:

- Fp16 used by NVIDIA GPUs, AMD Radeon Instinct MI25 GPU, ARM NEON, Fujitsu A64FX ARM
- Bfloat16 used by Google TPU, NVIDIA GPUs, Arm, Intel

Today's floating-point landscape

		Bits Signif. (t)	Exp.	Range	$u = 2^{-t}$
bfloat16	B	8	8	$10^{\pm 38}$	4×10^{-3}
fp16	H	11	5	$10^{\pm 5}$	5×10^{-4}
fp32	S	24	8	$10^{\pm 38}$	6×10^{-8}
fp64	D	53	11	$10^{\pm 308}$	1×10^{-16}
fp128	Q	113	15	$10^{\pm 4932}$	1×10^{-34}

Great benefits:

- Reduced **storage**, data movement, and communications
- Increased **speed** on emerging hardware ($16\times$ on A100 from fp32 to fp16/bfloat16)
- Reduced **energy** consumption ($5\times$ with fp16, $9\times$ with bfloat16)

Today's floating-point landscape

		Bits Signif. (t)	Exp.	Range	$u = 2^{-t}$
bfloat16	B	8	8	$10^{\pm 38}$	4×10^{-3}
fp16	H	11	5	$10^{\pm 5}$	5×10^{-4}
fp32	S	24	8	$10^{\pm 38}$	6×10^{-8}
fp64	D	53	11	$10^{\pm 308}$	1×10^{-16}
fp128	Q	113	15	$10^{\pm 4932}$	1×10^{-34}

Great benefits:

- Reduced **storage**, data movement, and communications
- Increased **speed** on emerging hardware ($16\times$ on A100 from fp32 to fp16/bfloat16)
- Reduced **energy** consumption ($5\times$ with fp16, $9\times$ with bfloat16)

Some risks too:

- Low precision (large u)
- Narrow range

Mixed precision algorithms

Mix several precisions in the same code with the goal of

- Getting the performance benefits of low precisions
- While preserving the accuracy and stability of the high precision

Terminology varies: Mixed precision, Multiprecision, Adaptive precision, Variable precision, Transprecision, Dynamic precision, ...

Mixed precision algorithms

Mix several precisions in the same code with the goal of

- Getting the performance benefits of low precisions
- While preserving the accuracy and stability of the high precision

Terminology varies: Mixed precision, Multiprecision, Adaptive precision, Variable precision, Transprecision, Dynamic precision, ...

What space of precisions?

- Arbitrary/custom precisions
- Mixed precision algorithms: small set of widely available precisions, such as IEEE arithmetics + bfloat16

Mixed precision algorithms

Mix several precisions in the same code with the goal of

- Getting the performance benefits of low precisions
- While preserving the accuracy and stability of the high precision

Terminology varies: Mixed precision, Multiprecision, Adaptive precision, Variable precision, Transprecision, Dynamic precision, ...

What space of precisions?

- Arbitrary/custom precisions
- Mixed precision algorithms: small set of widely available precisions, such as IEEE arithmetics + bfloat16

Crux of the matter: **how** to select the right precision for the right variable/operation

How are precisions selected?

- Precision tuning: explore the space of variables of a code
  - ▲ Does not need any understanding of what the code does

How are precisions selected?

- Precision tuning: explore the space of variables of a code
  - ▲ Does not need any understanding of what the code does
  - ▼ Does not have any understanding of what the code does

How are precisions selected?

- Precision tuning: explore the space of variables of a code
  - ▲ Does not need any understanding of what the code does
  - ▼ Does not have any understanding of what the code does

- **Algorithm**-based, **analysis**-based, **data**-based approaches
  The more knowledge about the code we have, the better:
  1. Develop approach tailored to specific algorithm
  2. If possible use error analysis to determine best choice of precisions
  3. If possible take into account specific data at hand

  Illustration of this methodology for numerical linear algebra

How are precisions selected?

- Precision tuning: explore the space of variables of a code
  - ▲ Does not need any understanding of what the code does
  - ▼ Does not have any understanding of what the code does

- **Algorithm**-based, **analysis**-based, **data**-based approaches
  The more knowledge about the code we have, the better:
  1. Develop approach tailored to specific algorithm
  2. If possible use error analysis to determine best choice of precisions
  3. If possible take into account specific data at hand

  Illustration of this methodology for numerical linear algebra

- Mixed precision computing brings new life to numerical analysis (rounding error analysis)

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 $Ux = y$

Precision $u \Rightarrow$ computed $\widehat{x}$ satisfies $\|\widehat{x} - x\| \leq f(n)\kappa(A)u\|x\|$,
with $\kappa(A) = \|A\|\|A^{-1}\|$

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 $Ux = y$

Precision $u \Rightarrow$ computed $\widehat{x}$ satisfies $\|\widehat{x} - x\| \leq f(n)\kappa(A)u\|x\|$,
with $\kappa(A) = \|A\|\|A^{-1}\|$

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

Solve $Ax_1 = b$ via $x_1 = U^{-1}(L^{-1}b)$
while Not converged **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 while

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:

For a **target accuracy** u, and assuming $\kappa(A)u < 1$:

Solve $Ax_1 = b$ by LU factorization at precision $\mathbf{u_f}$
while Not converged **do**
  $r_i = b - Ax_i$ at precision $\mathbf{u_r}$
  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 $\mathbf{u}$
end while

Theorem (simplified from Carson and Higham, 2018)

Under the condition $\phi_i < 1$, the forward error converges to

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

Error analysis of general IR

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

For a **target accuracy** u, and assuming $\kappa(A)u < 1$:

Solve $Ax_1 = b$ by LU factorization at precision $\mathbf{u_f}$
while Not converged **do**
  $r_i = b - Ax_i$ at precision $\mathbf{u_r}$
  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 $\mathbf{u}$
end while

Theorem (simplified from Carson and Higham, 2018)

Under the condition $\phi_i < 1$, the forward error converges to

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

- **Limiting accuracy:** depends on u and u_r only, can be made independent of $\kappa(A)$ by taking $\mathbf{u_r} = \mathbf{u}^2$
- **Convergence condition:** depends on the choice of solver

70 years of LU-IR

LU-IR: reuse LU factors to solve for d_i:
$d_i = U^{-1}L^{-1}r_i \Rightarrow \|\widehat{d}_i - d_i\| \leq f(n)\kappa(A)\mathbf{u_f}\|d_i\| \Rightarrow \phi_i = O(\kappa(A)\mathbf{u_f})$

Solve $Ax_1 = b$ by LU factorization in precision $\mathbf{u_f}$
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ in precision $\mathbf{u_r}$
  Solve $Ad_i = r_i$ via $d_i = U^{-1}(L^{-1}r_i)$
  $x_{i+1} = x_i + d_i$ in precision $\mathbf{u}$
end for

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

70 years of LU-IR

LU-IR: reuse LU factors to solve for d_i:

$d_i = U^{-1}L^{-1}r_i \Rightarrow \|\widehat{d}_i - d_i\| \leq f(n)\kappa(A)\mathbf{u_f}\|d_i\| \Rightarrow \phi_i = O(\kappa(A)\mathbf{u_f})$

Solve $Ax_1 = b$ by LU factorization — $\mathbf{u_f} = \mathbf{double}$

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

  $r_i = b - Ax_i$ — $\mathbf{u_r} = \mathbf{double}$

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

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

end for

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

Fixed-precision

Jankowski and Wozniakowski (1977) Skeel (1980)

70 years of LU-IR

LU-IR: reuse LU factors to solve for d_i:

$d_i = U^{-1}L^{-1}r_i \Rightarrow \|\widehat{d}_i - d_i\| \leq f(n)\kappa(A)\mathbf{u_f}\|d_i\| \Rightarrow \phi_i = O(\kappa(A)\mathbf{u_f})$

Solve $Ax_1 = b$ by LU factorization — $\mathbf{u_f} = \mathbf{double}$

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

  $r_i = b - Ax_i$ — $\mathbf{u_r} = \mathbf{quadruple}$

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

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

end for

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

Traditional

Wilkinson (1948) Moler (1967)

70 years of LU-IR

LU-IR: reuse LU factors to solve for d_i:

$d_i = U^{-1}L^{-1}r_i \Rightarrow \|\widehat{d}_i - d_i\| \leq f(n)\kappa(A)\mathbf{u_f}\|d_i\| \Rightarrow \phi_i = O(\kappa(A)\mathbf{u_f})$

Solve $Ax_1 = b$ by LU factorization — $\mathbf{u_f} = \mathbf{single}$

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

  $r_i = b - Ax_i$ — $\mathbf{u_r} = \mathbf{double}$

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

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

end for

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

Low precision factorization

Langou et al (2006)

70 years of LU-IR

LU-IR: reuse LU factors to solve for d_i:
$d_i = U^{-1}L^{-1}r_i \Rightarrow \|\widehat{d}_i - d_i\| \leq f(n)\kappa(A)\mathbf{u_f}\|d_i\| \Rightarrow \phi_i = O(\kappa(A)\mathbf{u_f})$

Solve $Ax_1 = b$ by LU factorization — $\mathbf{u_f} = \mathbf{single}$
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ — $\mathbf{u_r} = \mathbf{quadruple}$
  Solve $Ad_i = r_i$ via $d_i = U^{-1}(L^{-1}r_i)$
  $x_{i+1} = x_i + d_i$ — $\mathbf{u} = \mathbf{double}$
end for

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

Three precisions

Carson and Higham (2018)

70 years of LU-IR

LU-IR: reuse LU factors to solve for d_i:
$d_i = U^{-1}L^{-1}r_i \Rightarrow \|\widehat{d}_i - d_i\| \leq f(n)\kappa(A)\mathbf{u_f}\|d_i\| \Rightarrow \phi_i = O(\kappa(A)\mathbf{u_f})$

Solve $Ax_1 = b$ by LU factorization — $\mathbf{u_f} = \mathbf{half}$
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ — $\mathbf{u_r} = \mathbf{quadruple}$
  Solve $Ad_i = r_i$ via $d_i = U^{-1}(L^{-1}r_i)$
  $x_{i+1} = x_i + d_i$ — $\mathbf{u} = \mathbf{double}$
end for

	u_f	u	u_r	max $\kappa(A)$	Forward error
Fixed	D	D	D	10^{16}	$\kappa(A) \cdot 10^{-16}$
Traditional	D	D	Q	10^{16}	10^{-16}
Low prec. fact.	H	D	D	10^{3}	$\kappa(A) \cdot 10^{-16}$
3 precisions	H	D	Q	10^{3}	10^{-16}

Only well-conditioned problems can be solved with a half precision factorization!

GMRES-based IR: Carson and Higham (2017)

- **Replace LU by GMRES solver:** solve $\widetilde{A}d_i = \widetilde{r}_i$ with GMRES, where $\widetilde{A} = U^{-1}L^{-1}A$ is preconditioned by LU factors
- Rationale:
  - $\kappa(\widetilde{A})$ often smaller than $\kappa(A)$
  - GMRES can be asked to converge to accuracy $\mathbf{u} \ll \mathbf{u_f}$

  $\Rightarrow$ $\widetilde{A}d_i = \widetilde{r}_i$ is solved with accuracy $\phi_i = \kappa(\widetilde{A})\mathbf{u}$
  - Convergence condition improved from $\kappa(A)\mathbf{u_f} < 1$ to $\kappa(\widetilde{A})\mathbf{u} < 1$

GMRES-IR

GMRES-based IR: Carson and Higham (2017)

- **Replace LU by GMRES solver:** solve $\widetilde{A}d_i = \widetilde{r}_i$ with GMRES, where $\widetilde{A} = U^{-1}L^{-1}A$ is preconditioned by LU factors
- Rationale:
  - $\kappa(\widetilde{A})$ often smaller than $\kappa(A)$
  - GMRES can be asked to converge to accuracy $\mathbf{u} \ll \mathbf{u_f}$

  $\Rightarrow$ $\widetilde{A}d_i = \widetilde{r}_i$ is solved with accuracy $\phi_i = \kappa(\widetilde{A})\mathbf{u}$
  - Convergence condition improved from $\kappa(A)\mathbf{u_f} < 1$ to $\kappa(\widetilde{A})\mathbf{u} < 1$
- **The catch**: the matrix–vector products are with $\widetilde{A} = U^{-1}L^{-1}A$, introduce an extra $\kappa(A)$ unless performed in higher precision

Solve $Ax_1 = b$ by LU factorization at precision $\mathbf{u_f}$
while Not converged **do**
  $r_i = b - Ax_i$ at precision $\mathbf{u_r}$
  Solve $U^{-1}L^{-1}Ad_i = U^{-1}L^{-1}r_i$ by GMRES at precision $\mathbf{u}$ with products with $U^{-1}L^{-1}A$ at precision $\mathbf{u}^2$
  $x_{i+1} = x_i + d_i$ at precision $\mathbf{u}$
end while

LU-IR vs GMRES-IR

Using $\kappa(\widetilde{A}) \leq (1 + \kappa(A)\mathbf{u_f})^2$ we determine the convergence condition on $\kappa(A)$

	u_f	u	u_r	max $\kappa(A)$	Forward error
LU-IR	S	D	Q	10^8	10^{-16}
GMRES-IR	S	D	Q	10^{16}	10^{-16}
LU-IR	H	D	Q	10^3	10^{-16}
GMRES-IR	H	D	Q	10^{11}	10^{-16}

GMRES-IR can handle much more ill-conditioned matrices.

LU-IR vs GMRES-IR

Using $\kappa(\widetilde{A}) \leq (1 + \kappa(A)\mathbf{u_f})^2$ we determine the convergence condition on $\kappa(A)$

	u_f	u	u_r	max $\kappa(A)$	Forward error
LU-IR	S	D	Q	10^8	10^{-16}
GMRES-IR	S	D	Q	10^{16}	10^{-16}
LU-IR	H	D	Q	10^3	10^{-16}
GMRES-IR	H	D	Q	10^{11}	10^{-16}

GMRES-IR can handle much more ill-conditioned matrices.

However:

- LU solves are performed at precision $\mathbf{u}^2$ instead of $\mathbf{u_f}$
  $\Rightarrow$ **practical limitation**
  - Increases cost per iteration
  - If u is D, requires use of quad precision
  - Practical implementations have relaxed this requirement by replacing u^2 with u, with no theoretical guarantee

Rethinking GMRES-IR

- Goal: solve $Ad_i = r_i$ with GMRES and bound $\phi_i = \|\widehat{d}_i - d_i\|/\|d_i\|$
  - In what precision do we really need to run GMRES?
  - How much extra precision is really needed in the matvec products?

Solve $Ax_1 = b$ by LU factorization at precision $\mathbf{u_f}$
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ at precision $\mathbf{u_r}$
  Solve $Ad_i = r_i$ with preconditioned GMRES at precision $\mathbf{u}$ except matvecs at precision $\mathbf{u}^2$
  $x_{i+1} = x_i + d_i$ at precision $\mathbf{u}$
end for

Rethinking GMRES-IR

- Goal: solve $Ad_i = r_i$ with GMRES and bound $\phi_i = \|\widehat{d}_i - d_i\|/\|d_i\|$
  - In what precision do we really need to run GMRES?
  - How much extra precision is really needed in the matvec products?

Solve $Ax_1 = b$ by LU factorization at precision $\mathbf{u_f}$
for $i = 1 : nsteps$ **do**
  $r_i = b - Ax_i$ at precision $\mathbf{u_r}$
  Solve $Ad_i = r_i$ with preconditioned GMRES at precision $\mathbf{u}$ except matvecs at precision $\mathbf{u}^2$
  $x_{i+1} = x_i + d_i$ at precision $\mathbf{u}$
end for

Rethinking GMRES-IR

- Goal: solve $Ad_i = r_i$ with GMRES and bound $\phi_i = \|\widehat{d}_i - d_i\|/\|d_i\|$
  - In what precision do we really need to run GMRES?
  - How much extra precision is really needed in the matvec products?

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

Relax the requirements on the GMRES precisions: run at precision $\mathbf{u_g} \leq \mathbf{u}$ with matvecs at precision $\mathbf{u_p} \leq \mathbf{u}^2$

$\Rightarrow$ **FIVE precisions** in total!

What can we say about the convergence of this GMRES-IR5?

Two precision GMRES

- Unpreconditioned GMRES in precision $\mathbf{u}$ for $Ax = b$:
  - Backward error of order $\mathbf{u}$ Paige, Rozloznik, Strakos (2006)
  - Forward error of order $\kappa(A)\mathbf{u}$

- Two precision preconditioned GMRES for $\widetilde{A}x = b$:
  - Backward error of order $\kappa(A)\mathbf{u_p} + \mathbf{u_g}$
    - The matrix–vector products are performed with $\widetilde{A} = U^{-1}L^{-1}A$: $y = U^{-1}L^{-1}Ax \Rightarrow \|\widehat{y} - y\| \lesssim \kappa(A)\mathbf{u_p}\|\widetilde{A}\|\|x\|$
    - The rest is at precision $\mathbf{u_g}$
  - Forward error of order $\kappa(\widetilde{A})\left(\kappa(A)\mathbf{u_p} + \mathbf{u_g}\right)$
  - $\kappa(\widetilde{A}) \leq (1 + \kappa(A)\mathbf{u_f})^2 \Rightarrow \phi_i \sim \kappa(A)^2\mathbf{u_f}^2\left(\kappa(A)\mathbf{u_p} + \mathbf{u_g}\right)$

Side-result: generalization of the backward stability of GMRES to a preconditioned two-precision GMRES

Amestoy, Buttari, Higham, L'Excellent, M., Vieublé (2021)

Five precision GMRES-IR

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

Theorem (convergence of GMRES-IR5)

Under the condition $(\mathbf{u_g} + \kappa(A)\mathbf{u_p})\kappa(A)^2\mathbf{u_f}^2 < 1$, the forward error converges to its limiting accuracy

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

Amestoy, Buttari, Higham, L'Excellent, M., Vieublé (2021)

Meaningful combinations

With five arithmetics (fp16, bfloat16, fp32, fp64, fp128) there are over **3000 different combinations** of GMRES-IR5!

They are not all relevant !

Meaningful combinations: those where none of the precisions can be lowered without worsening either the limiting accuracy or the convergence condition.

Filtering rules

- $\mathbf{u^2 \leq u_r \leq u \leq u_f}$
- $\mathbf{u_p \leq u_g}$
- $\mathbf{u_p < u_f}$
- $\mathbf{u_p < u}$, $\mathbf{u_p = u}$, $\mathbf{u_p > u}$ all possible
- $\mathbf{u_g \geq u}$
- $\mathbf{u_g < u_f}$, $\mathbf{u_g = u_f}$, $\mathbf{u_g > u_f}$ all possible

Theoretical results

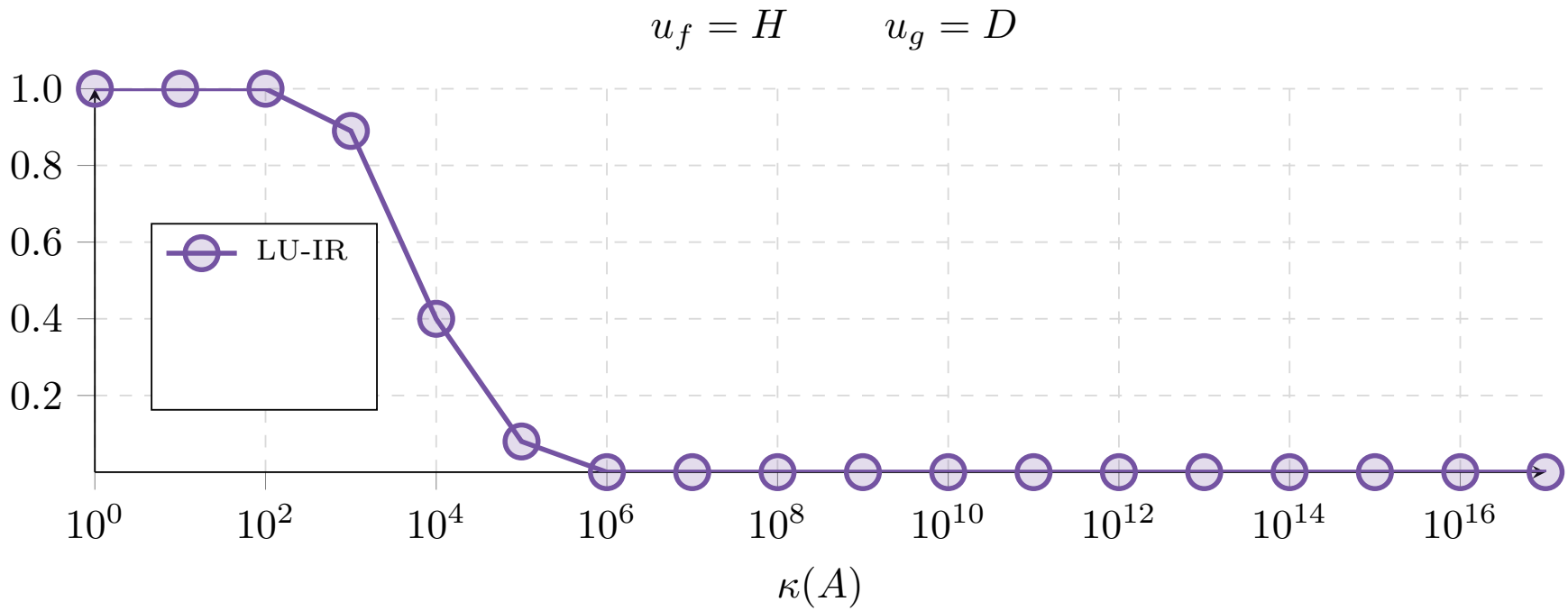
Meaningful combinations of GMRES-IR5 for $\mathbf{u_f} = H$ and $\mathbf{u} = D$.

$\mathbf{u_g}$	$\mathbf{u_p}$	Convergence Condition $\max(\kappa(A))$
LU-IR		2×10^3
B	S	3×10^4
H	S	4×10^4
H	D	9×10^4
S	D	8×10^6
D	D	3×10^7
D	Q	2×10^{11}

Five combinations between LU-IR and Carson & Higham's GMRES-IR $\Rightarrow$ More **flexible** precisions choice to fit at best the **hardware constraints** and the **problem difficulty**.

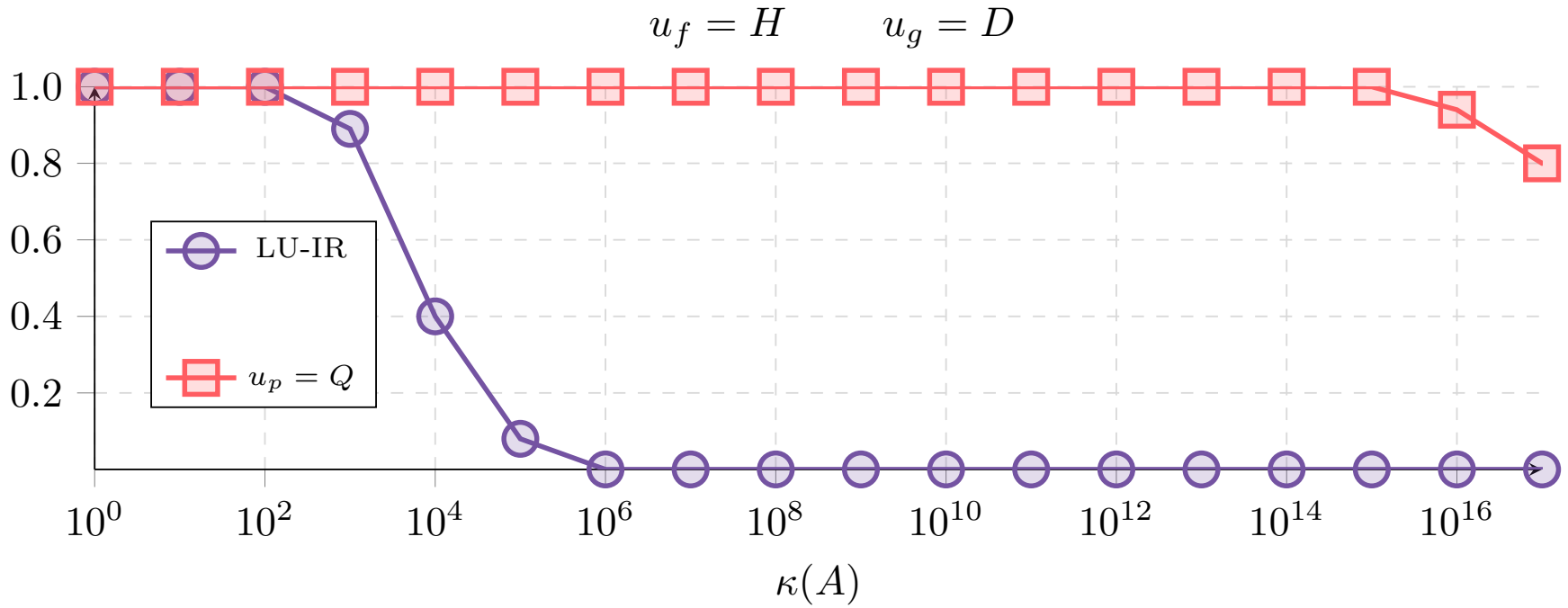
Experimental results

Take 100 random matrices with specified $\kappa(A)$ and measure the success rate: the percentage of matrices for which GMRES-IR5 converges to a small forward error



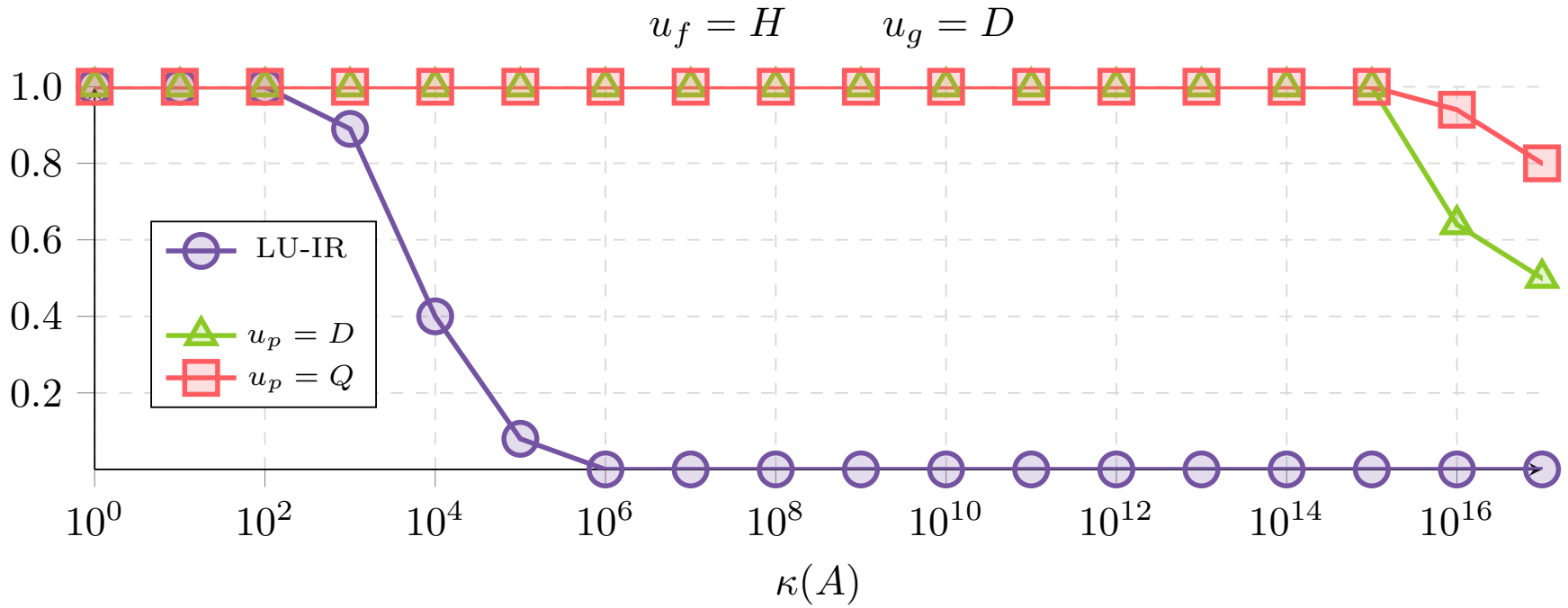
Experimental results

Take 100 random matrices with specified $\kappa(A)$ and measure the success rate: the percentage of matrices for which GMRES-IR5 converges to a small forward error



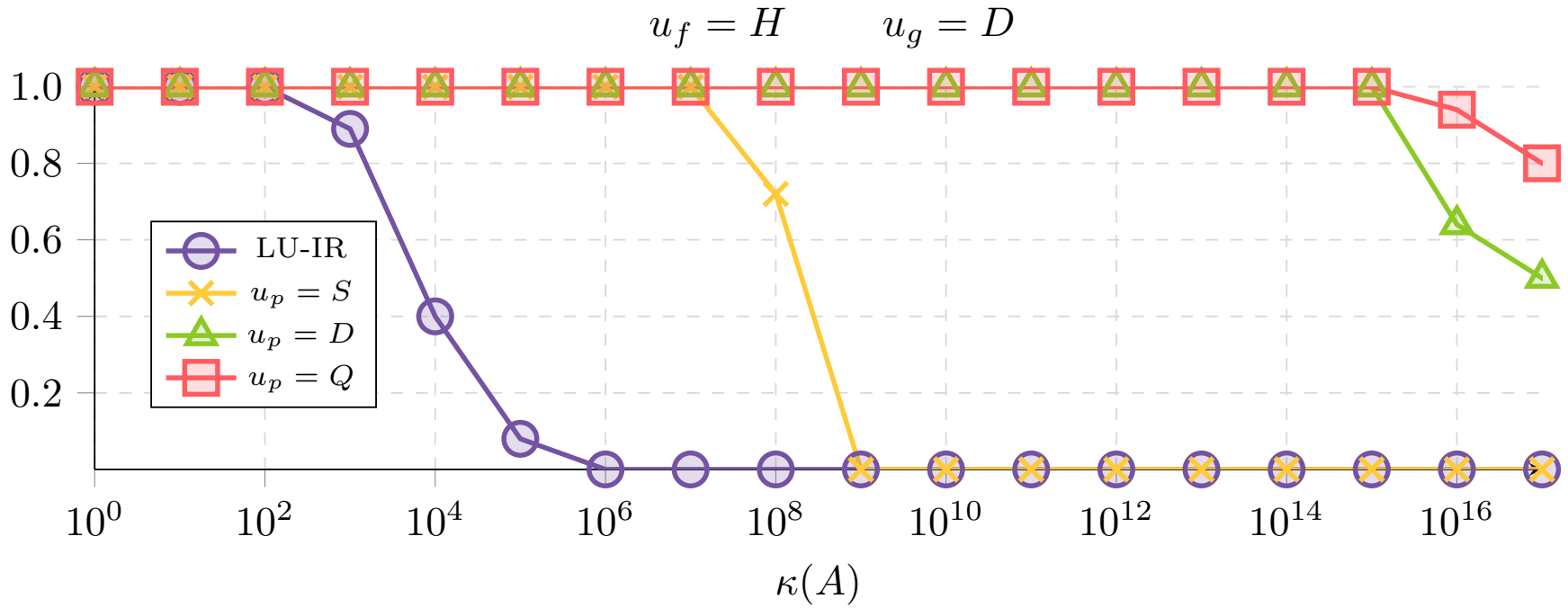
Experimental results

Take 100 random matrices with specified $\kappa(A)$ and measure the success rate: the percentage of matrices for which GMRES-IR5 converges to a small forward error



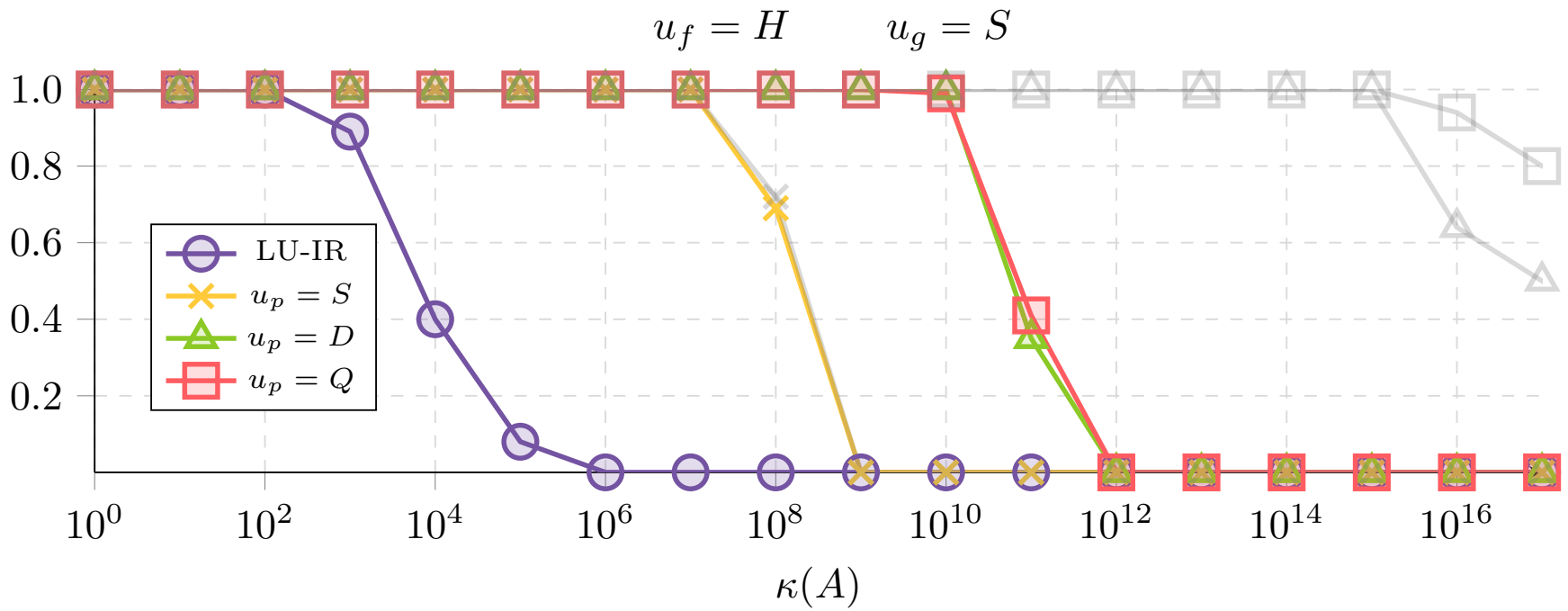
Experimental results

Take 100 random matrices with specified $\kappa(A)$ and measure the success rate: the percentage of matrices for which GMRES-IR5 converges to a small forward error



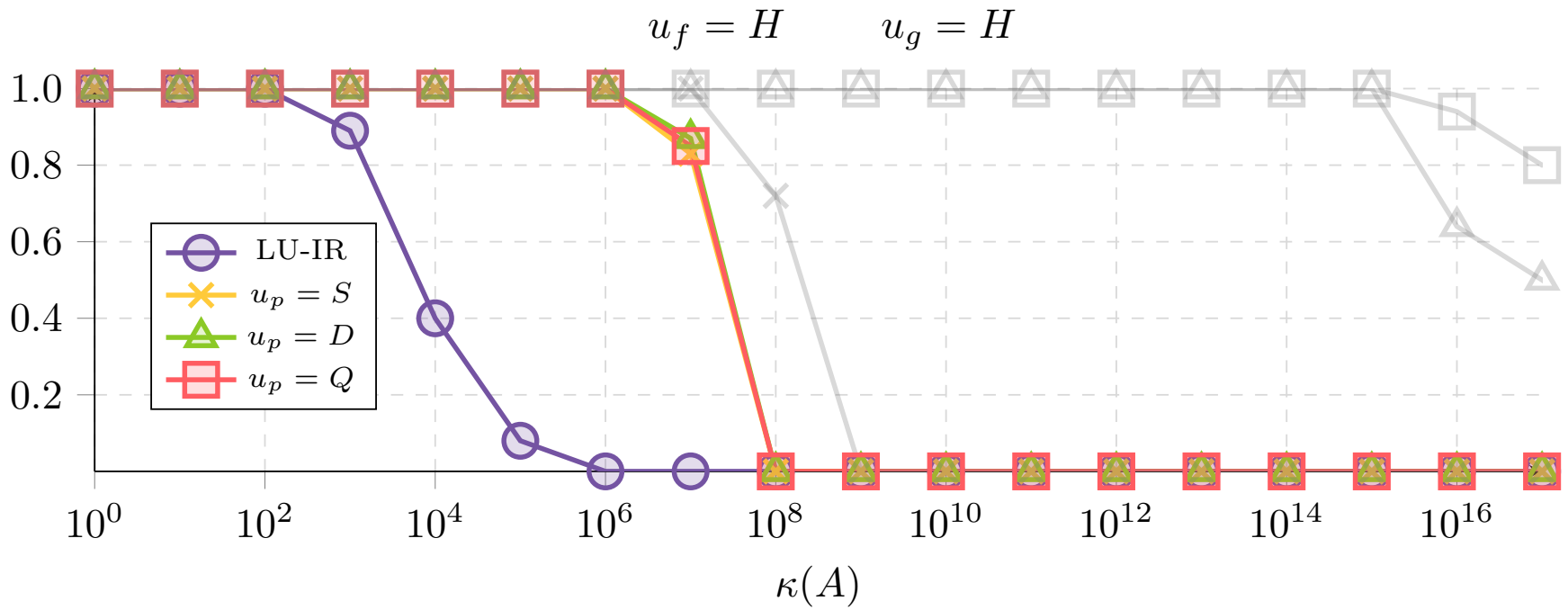
Experimental results

Take 100 random matrices with specified $\kappa(A)$ and measure the success rate: the percentage of matrices for which GMRES-IR5 converges to a small forward error



Experimental results

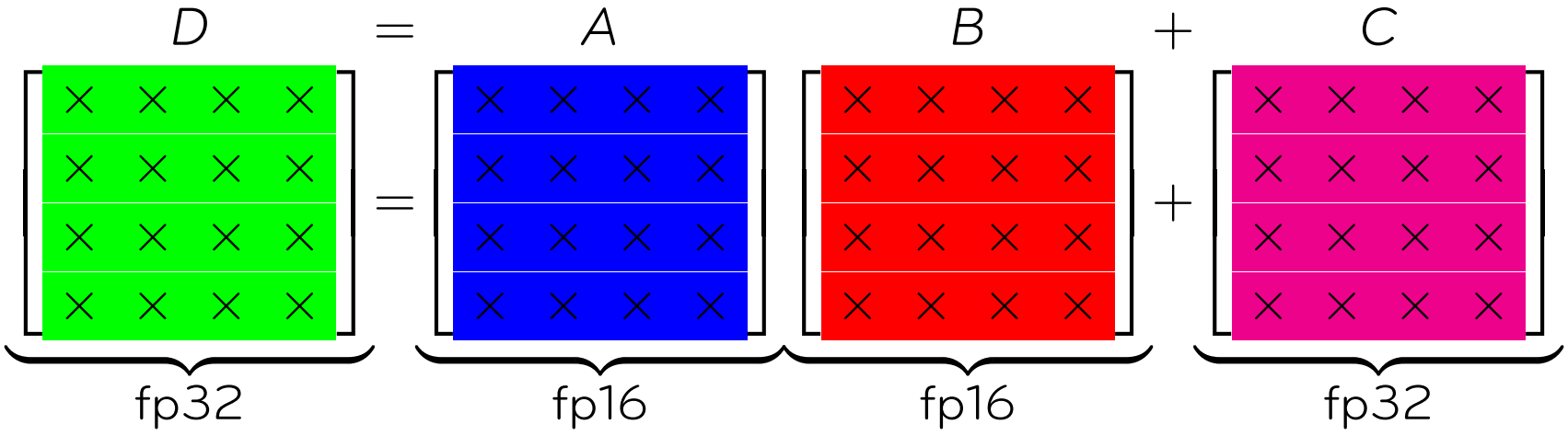
Take 100 random matrices with specified $\kappa(A)$ and measure the success rate: the percentage of matrices for which GMRES-IR5 converges to a small forward error



Similar picture on many types of matrices

NVIDIA GPU tensor cores

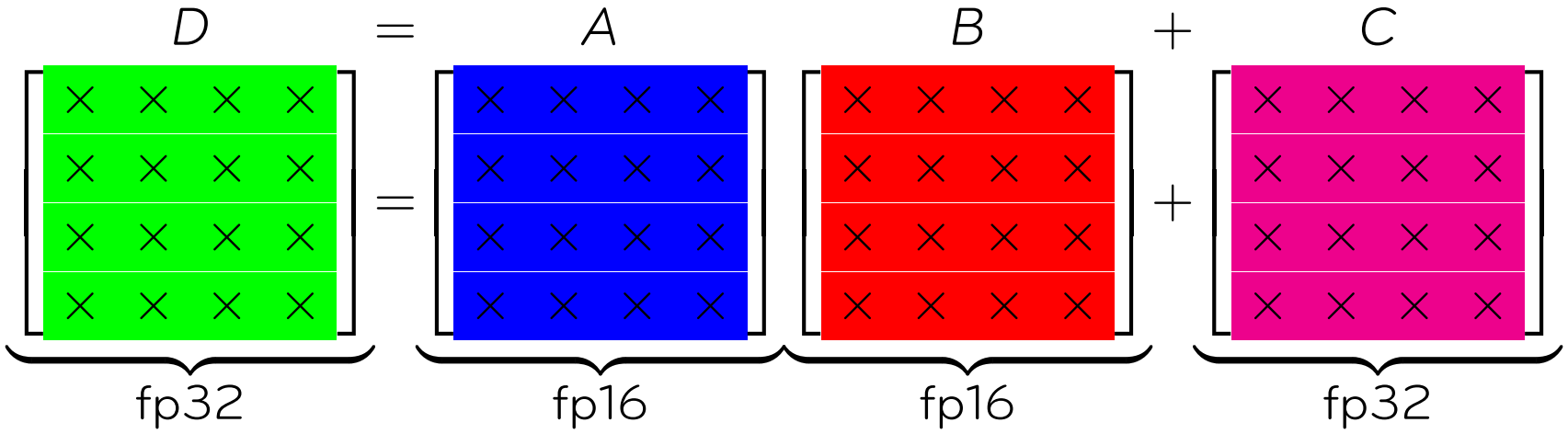
Tensor cores units available on NVIDIA GPUs V100 carry out a 4×4 matrix multiplication **in 1 clock cycle**:



- **Performance boost**: peaks at 125 TFLOPS ($8\times$ speedup vs fp32, $16\times$ on A100)

NVIDIA GPU tensor cores

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



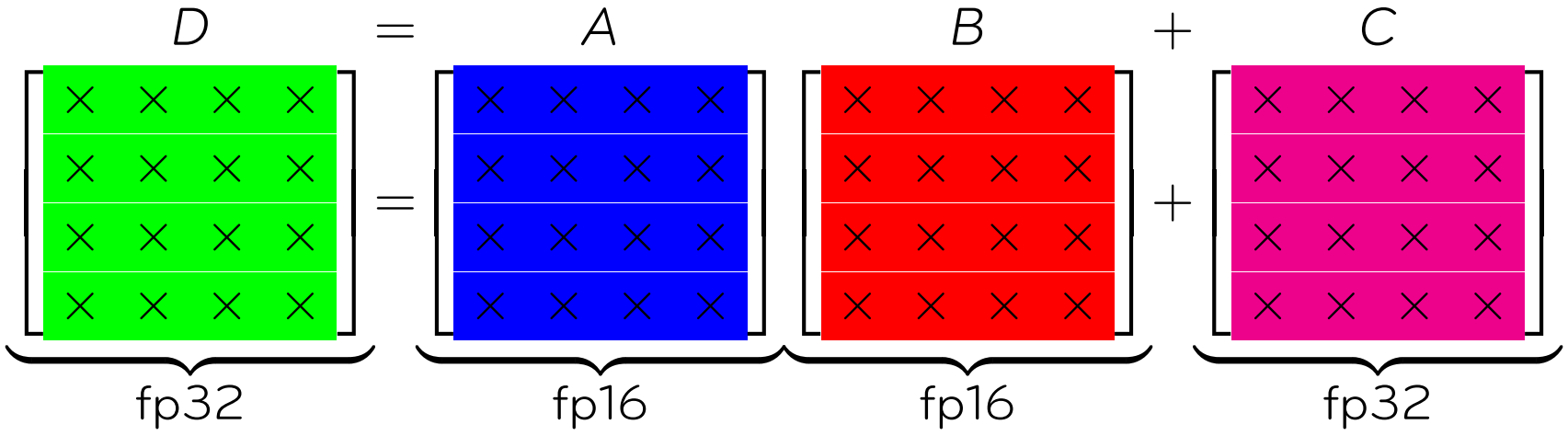
- **Performance boost**: peaks at 125 TFLOPS ($8\times$ speedup vs fp32, $16\times$ on A100)
- **Accuracy boost**: let $C = AB$, with $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, the computed $\widehat{C}$ satisfies

$$|\widehat{C} - C| \lesssim c_n |A||B|, \quad c_n = \begin{cases}$$

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

NVIDIA GPU tensor cores

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



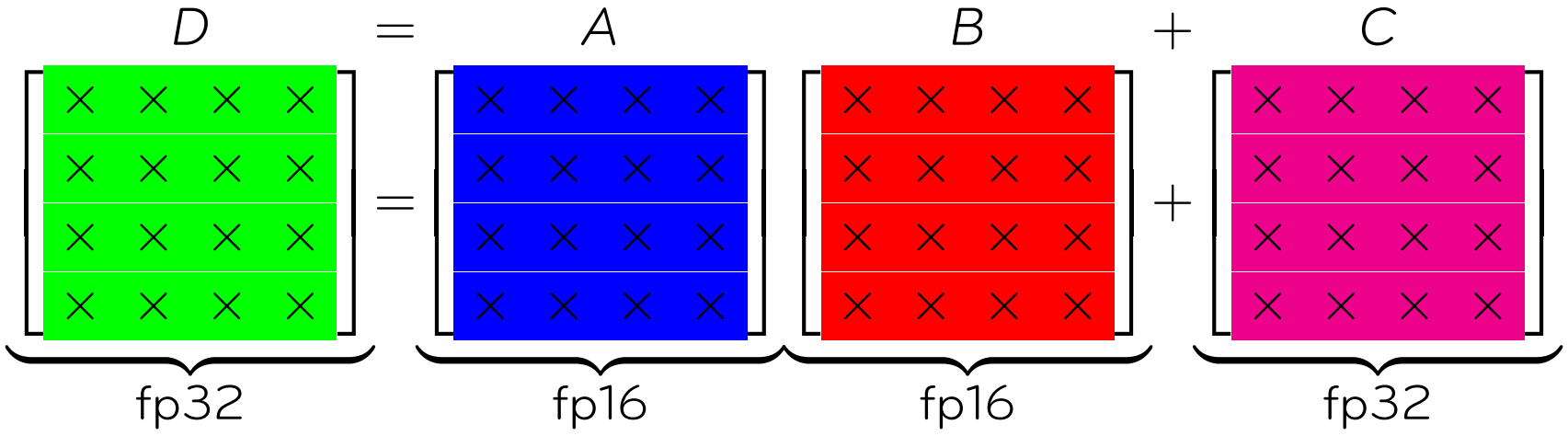
- **Performance boost**: peaks at 125 TFLOPS ($8\times$ speedup vs fp32, $16\times$ on A100)
- **Accuracy boost**: let $C = AB$, with $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, the computed $\widehat{C}$ satisfies

$$|\widehat{C} - C| \lesssim c_n |A||B|, \quad c_n = \begin{cases} nu_{16} & \text{(fp16)} \\ nu_{32} & \text{(fp32)} \end{cases}$$

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

NVIDIA GPU tensor cores

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



- **Performance boost**: peaks at 125 TFLOPS ($8\times$ speedup vs fp32, $16\times$ on A100)
- **Accuracy boost**: let $C = AB$, with $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, the computed $\widehat{C}$ satisfies

$$|\widehat{C} - C| \lesssim c_n |A||B|, \quad c_n = \begin{cases} nu_{16} & \text{(fp16)} \\ 2u_{16} + nu_{32} & \text{(tensor cores)} \\ nu_{32} & \text{(fp32)} \end{cases}$$

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

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 tensor cores

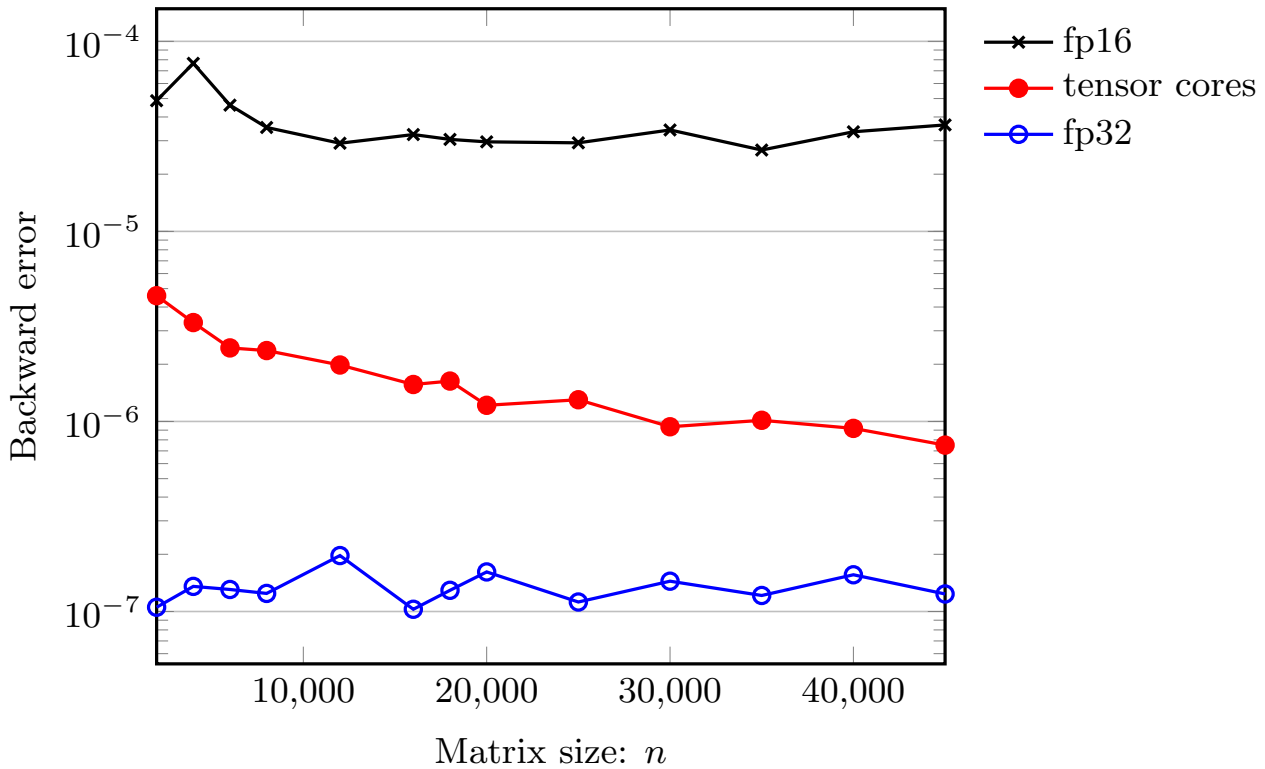
- Block version to use matrix–matrix operations
- $O(n^3)$ part of the flops done with tensor cores

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**
      $\widetilde{L}_{ik} \leftarrow \mathrm{fl}_{16}(L_{ik})$ and $\widetilde{U}_{ki} \leftarrow \mathrm{fl}_{16}(U_{ki})$
      $A_{ij} \leftarrow A_{ij} - \widetilde{L}_{ik}\widetilde{U}_{kj}$ using tensor cores
    end for
  end for
end for

LU factorization with tensor cores

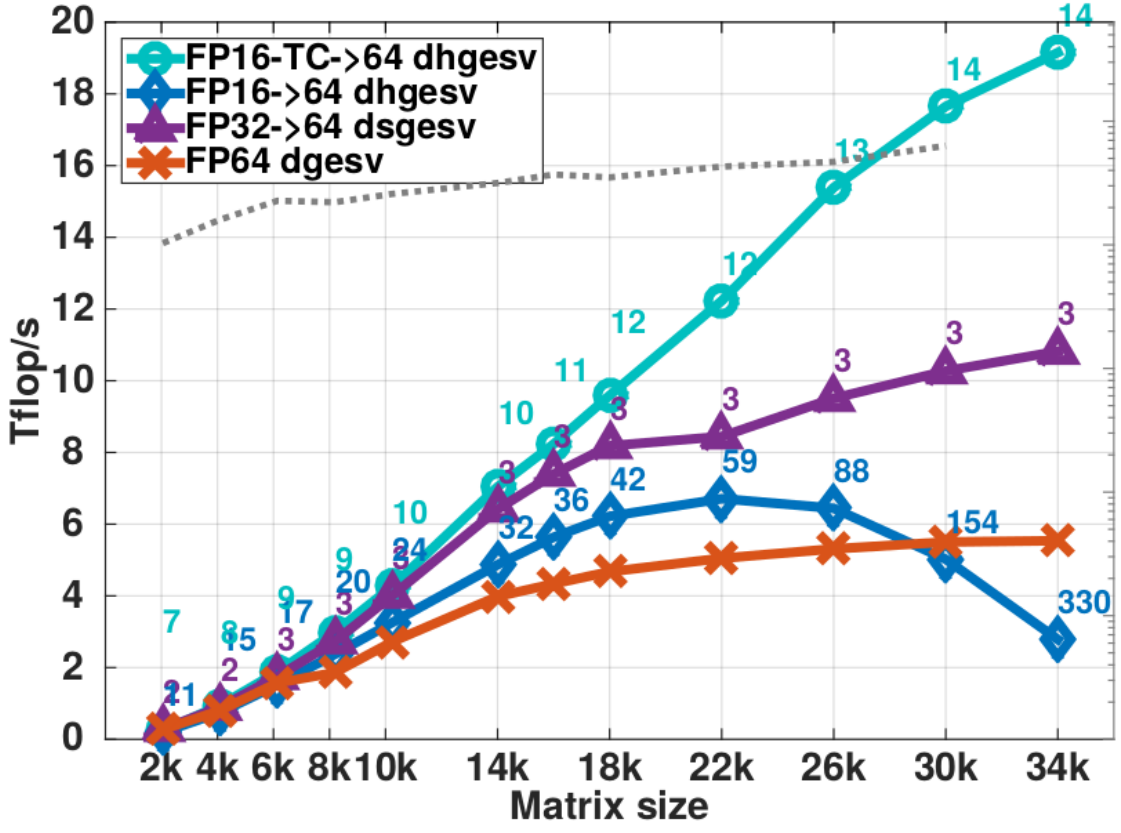
Error analysis for LU follows from matrix multiplication analysis and gives same bounds to first order Blanchard et al. (2020)

Standard fp16	Tensor cores	Standard fp32
nu_{16}	$2u_{16} + nu_{32}$	nu_{32}



Impact on iterative refinement

Results from Haidar et al. (2018)



- TC accuracy boost can be critical!
- TC performance suboptimal here, but can reach up to **50 TFLOPS** with optimized data movements Lopez and M. (2020)

Preconditioners other than LU

~~Solve $Ax_1 = b$ by LU factorization at precision $\mathbf{u_f}$~~
Compute $M^{-1} \approx A^{-1}$ and initialize x_1
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ at precision $\mathbf{u_r}$
  Solve $Ad_i = r_i$ with preconditioned GMRES at
    precision $\mathbf{u_g}$ except matvecs at precision $\mathbf{u_p}$
  $x_{i+1} = x_i + d_i$ at precision $\mathbf{u}$
end for

Preconditioners other than LU

~~Solve $Ax_1 = b$ by LU factorization at precision $\mathbf{u_f}$~~
Compute $M^{-1} \approx A^{-1}$ and initialize x_1
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ at precision $\mathbf{u_r}$
  Solve $Ad_i = r_i$ with preconditioned GMRES at precision $\mathbf{u_g}$ except matvecs at precision $\mathbf{u_p}$
  $x_{i+1} = x_i + d_i$ at precision $\mathbf{u}$
end for

A better preconditioner implies:

- ▲ Smaller $\kappa(M^{-1}A)$
- ▼ More expensive to compute/apply
- ▼ Larger error in matvecs with $M^{-1}A$ in GMRES

Convergence condition becomes

$$\kappa(M^{-1}A)\left(\frac{\|M^{-1}\|\|A\|}{\|M^{-1}A\|}u_p + u_g\right) < 1$$

Mixed precision restarted GMRES

Initialize x_1
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ at precision $\mathbf{u_{high}}$
  Solve $Ad_i = r_i$ with GMRES at precision $\mathbf{u_{low}}$
  $x_{i+1} = x_i + d_i$ at precision $\mathbf{u}$
end for

- With no preconditioner ($M = I$), GMRES-IR becomes equivalent to mixed precision restarted GMRES (inner–outer scheme)
  Turner and Walker (1992) Buttari et al. (2008)
  Lindquist et al. (2020) Loe et al. (2021)

Mixed precision restarted GMRES

Initialize x_1
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ at precision $\mathbf{u_{high}}$
  Solve $Ad_i = r_i$ with GMRES at precision $\mathbf{u_{low}}$
  $x_{i+1} = x_i + d_i$ at precision $\mathbf{u}$
end for

- With no preconditioner ($M = I$), GMRES-IR becomes equivalent to mixed precision restarted GMRES (inner–outer scheme)
  Turner and Walker (1992) Buttari et al. (2008)
  Lindquist et al. (2020) Loe et al. (2021)
- Preconditioners can exploit mixed precision too
  Anzt et al. (2018)

Mixed precision restarted GMRES

Initialize x_1
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ at precision $\mathbf{u_{high}}$
  Solve $Ad_i = r_i$ with GMRES at precision $\mathbf{u_{low}}$
  $x_{i+1} = x_i + d_i$ at precision $\mathbf{u}$
end for

- With no preconditioner ($M = I$), GMRES-IR becomes equivalent to mixed precision restarted GMRES (inner–outer scheme)
  Turner and Walker (1992) Buttari et al. (2008)
  Lindquist et al. (2020) Loe et al. (2021)
- Preconditioners can exploit mixed precision too
  Anzt et al. (2018)
- GMRES can exploit mixed precision too
  Gratton et al. (2019) Agullo et al. (2020) Aliaga et al. (2020)

Data-driven mixed precision computing?

- So far, precisions are chosen independently of input data (ex: A and b are not taken into account in $Ax = b$)
- Simple example: run $\kappa(A)$ estimator before selecting optimal GMRES-IR5 variant
- In the following, more sophisticated examples exploit the matrix structure (sparsity, data sparsity) to use even lower precisions
- Different approaches sharing a strong connection! Based on the same fundamental observation:

$$\begin{array}{rl} & 1.0\,1\,0\,1\,1\,0\,1 \times 2^{0} \\ + & \underline{\qquad\qquad 1.1\,0(1\,0\,1\,1\,0) \times 2^{-6}} \\ = & 1.0\,1\,1\,0\,0\,0\,0 \end{array}$$

$\Rightarrow$ **Small elements can be stored in lower precision**

Data-driven mixed precision SpMV

Consider the sparse matrix–vector (SpMV) product $y = Ax$

Ahmad, Sundar, Hall (2020) propose to split $A = A_d + A_s$, where A_s contains the "small" elements of A, and compute:

$$y = \underbrace{A_s x}_{\text{Compute in single}} + \underbrace{A_d x}_{\text{Compute in double}}$$

Data-driven mixed precision SpMV

Consider the sparse matrix–vector (SpMV) product $y = Ax$

Ahmad, Sundar, Hall (2020) propose to split $A = A_d + A_s$, where A_s contains the "small" elements of A, and compute:

$$y = \underbrace{A_s x}_{\text{Compute in single}} + \underbrace{A_d x}_{\text{Compute in double}}$$

Analysis: split row i of A into p buckets B_{ik} and sum elements of B_{ik} in precision u_k

$$y_i = \sum_{i=1}^{p} y_i^{(k)}, \quad y_i^{(k)} = \sum_{a_{ij}x_j \in B_{ik}} a_{ij}x_j$$

$$|\widehat{y}_i^{(k)} - y_i^{(k)}| \leq n_i^{(k)} u_k \sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|$$

Backward error (Oettli-Präger):

$$\max_i \frac{|\widehat{y}_i - y_i|}{|A||x|} \leq \max_i \sum_{k=1}^{p} n_i^{(k)} u_k \frac{\sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|}{\sum_{j=1}^{n} |a_{ij}x_j|}$$

Data-driven mixed precision SpMV

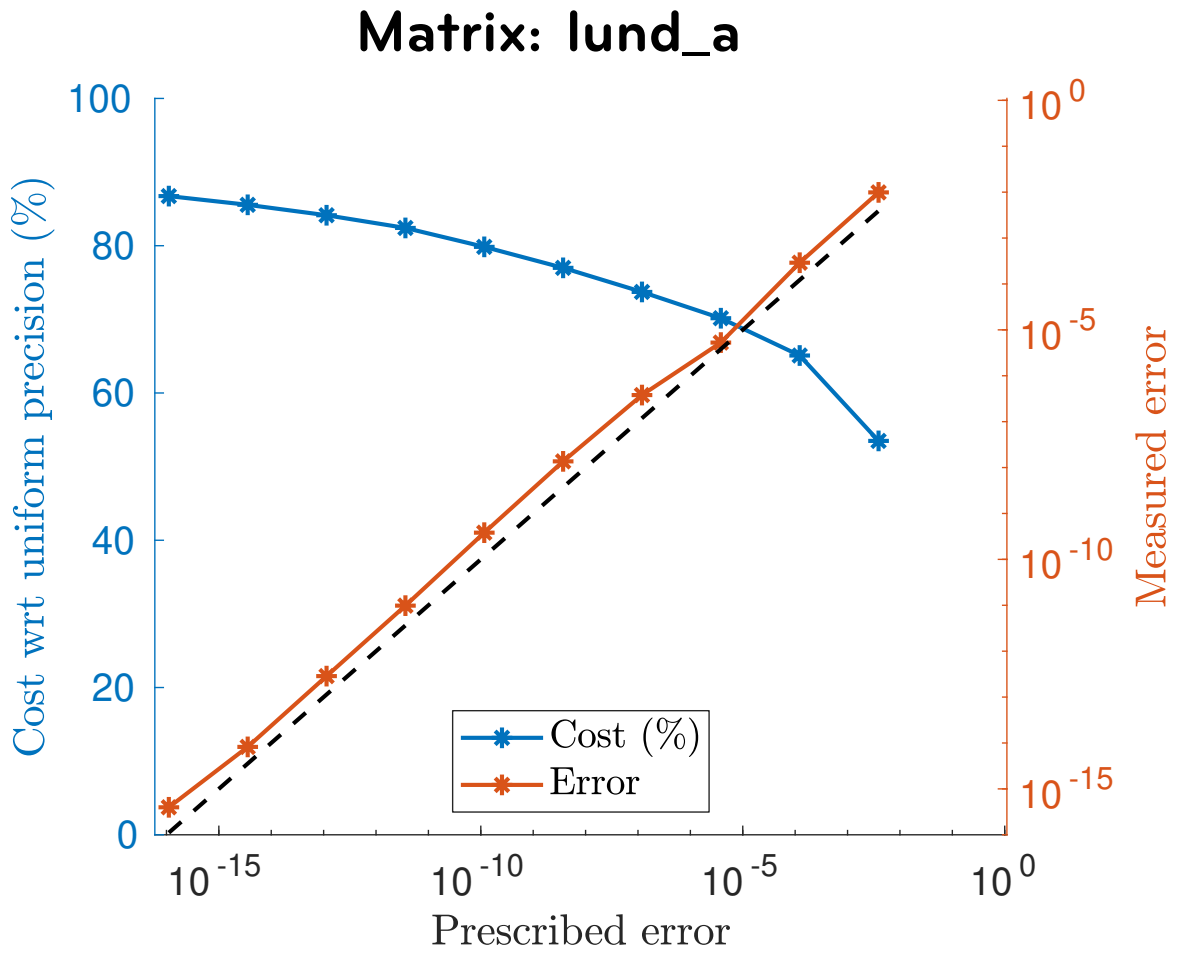
To achieve a backward error of order ε, must control the ratios

$\phi_{ik} = \frac{\sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|}{\sum_{j=1}^{n} |a_{ij}x_j|}$ $\Rightarrow$ explicit rule for building the buckets B_{ik}

Data-driven mixed precision SpMV

To achieve a backward error of order ε, must control the ratios

$\phi_{ik} = \frac{\sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|}{\sum_{j=1}^{n} |a_{ij}x_j|}$ $\Rightarrow$ explicit rule for building the buckets B_{ik}

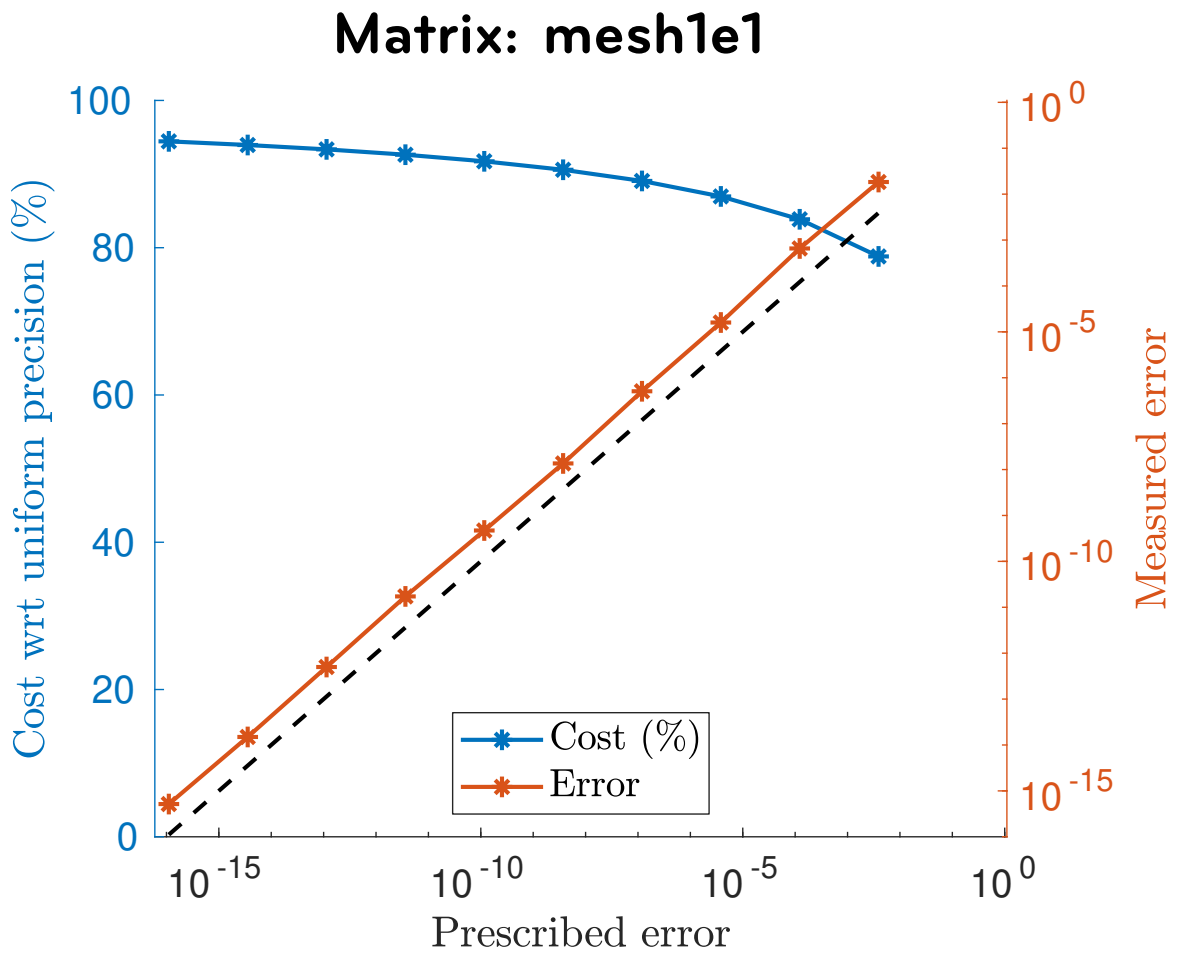


Graillat, Jézéquel, M., Molina (2021)

Data-driven mixed precision SpMV

To achieve a backward error of order ε, must control the ratios

$\phi_{ik} = \frac{\sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|}{\sum_{j=1}^{n} |a_{ij}x_j|}$ $\Rightarrow$ explicit rule for building the buckets B_{ik}

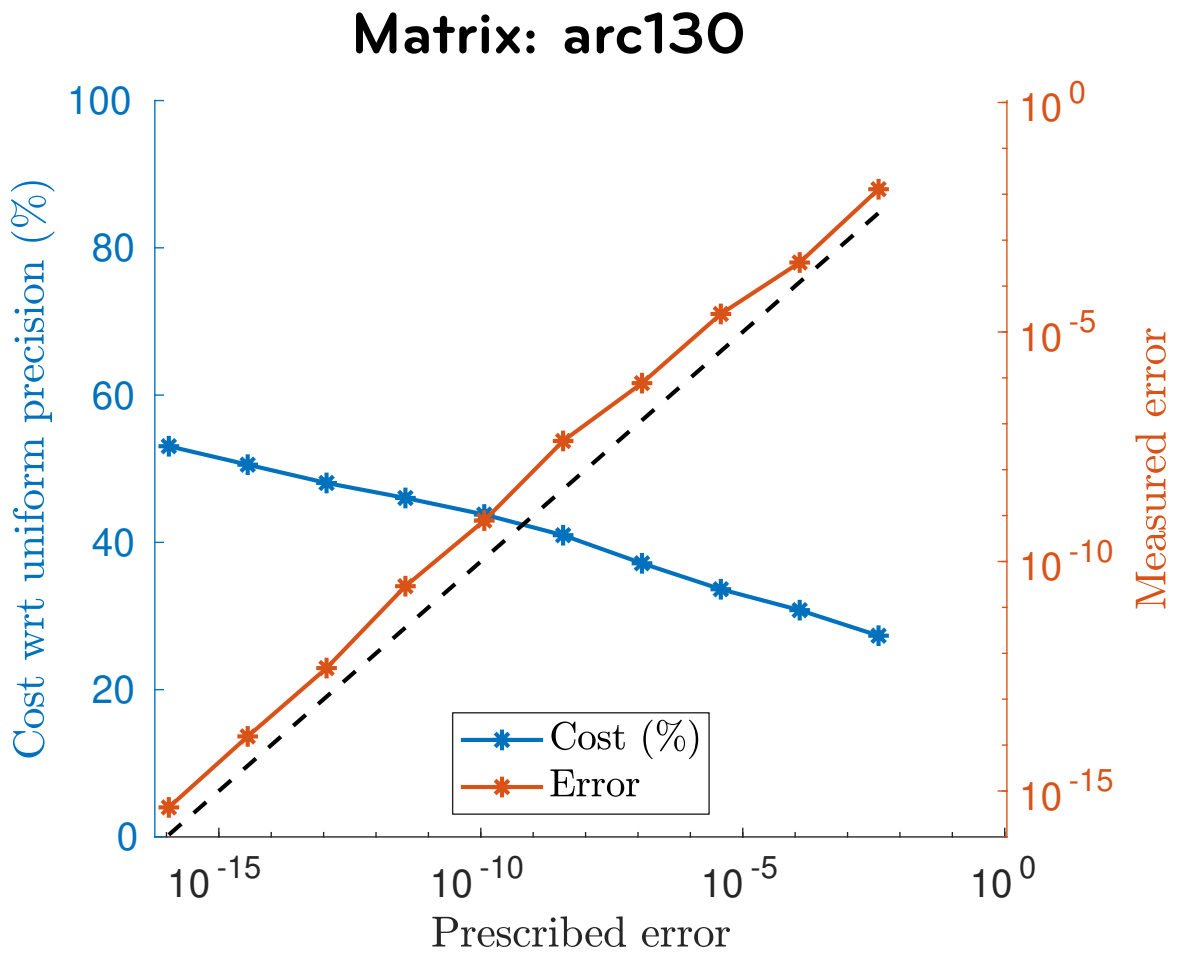


Graillat, Jézéquel, M., Molina (2021)

Data-driven mixed precision SpMV

To achieve a backward error of order ε, must control the ratios

$\phi_{ik} = \frac{\sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|}{\sum_{j=1}^{n} |a_{ij}x_j|}$ $\Rightarrow$ explicit rule for building the buckets B_{ik}

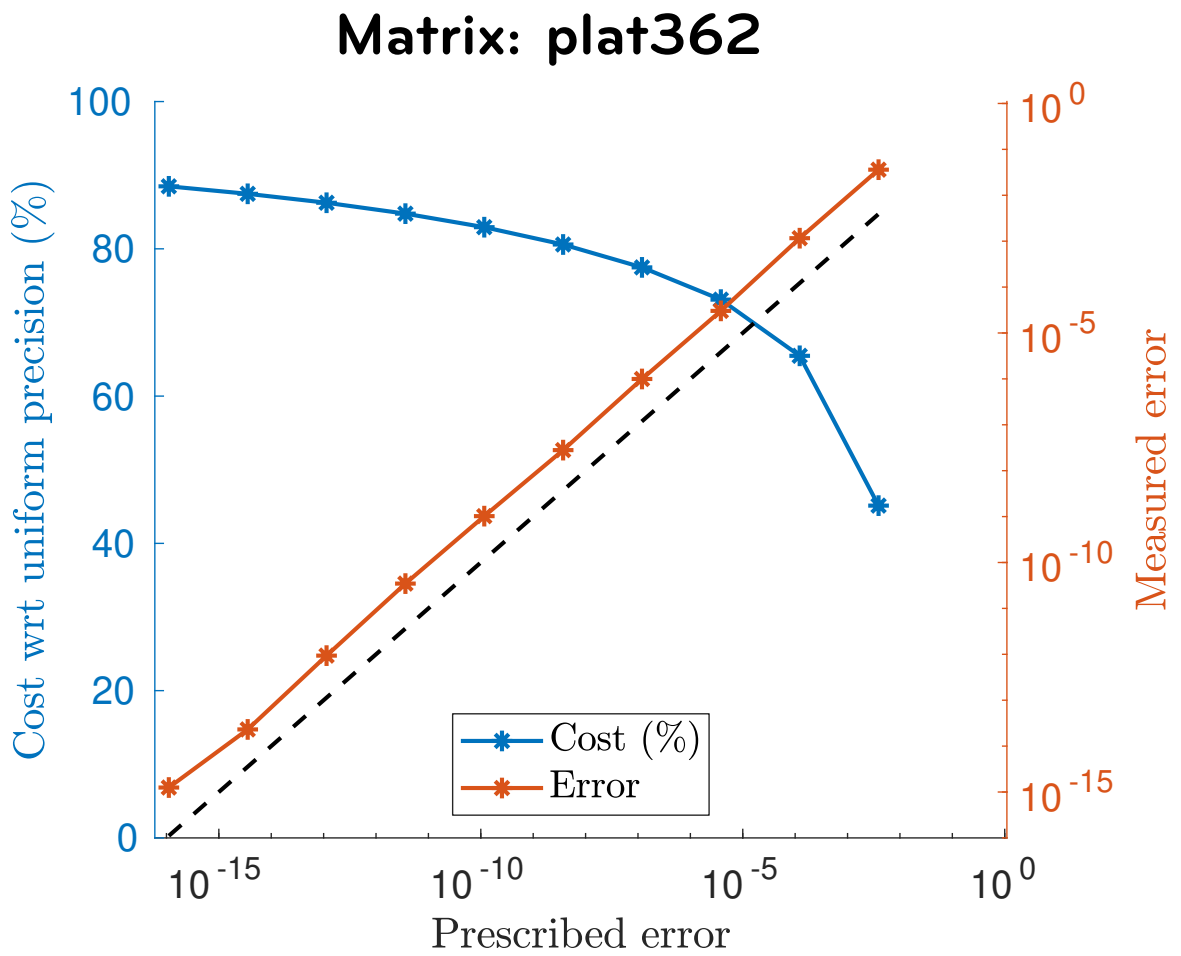


Graillat, Jézéquel, M., Molina (2021)

Data-driven mixed precision SpMV

To achieve a backward error of order ε, must control the ratios

$\phi_{ik} = \frac{\sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|}{\sum_{j=1}^{n} |a_{ij}x_j|}$ $\Rightarrow$ explicit rule for building the buckets B_{ik}

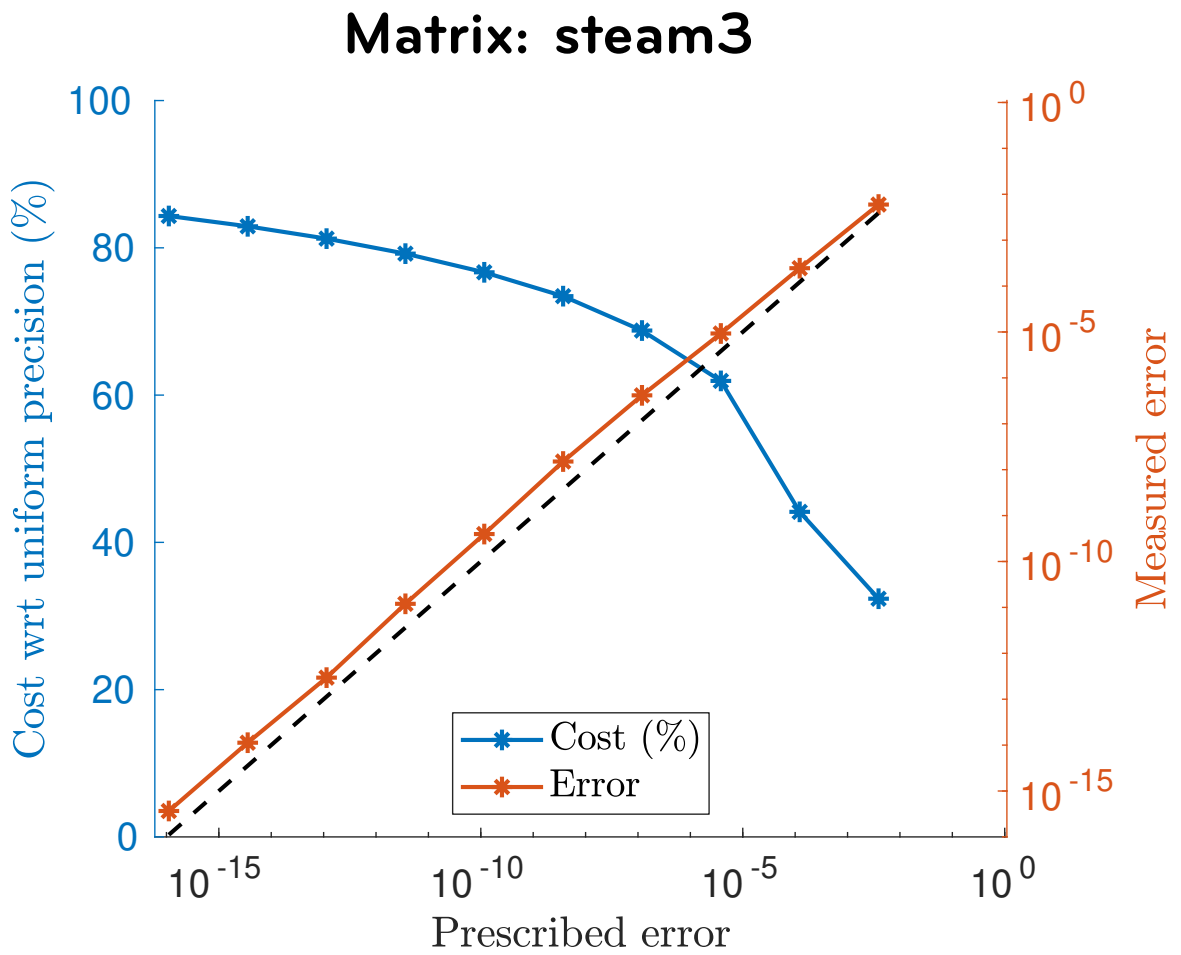


Graillat, Jézéquel, M., Molina (2021)

Data-driven mixed precision SpMV

To achieve a backward error of order ε, must control the ratios

$\phi_{ik} = \frac{\sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|}{\sum_{j=1}^{n} |a_{ij}x_j|}$ $\Rightarrow$ explicit rule for building the buckets B_{ik}

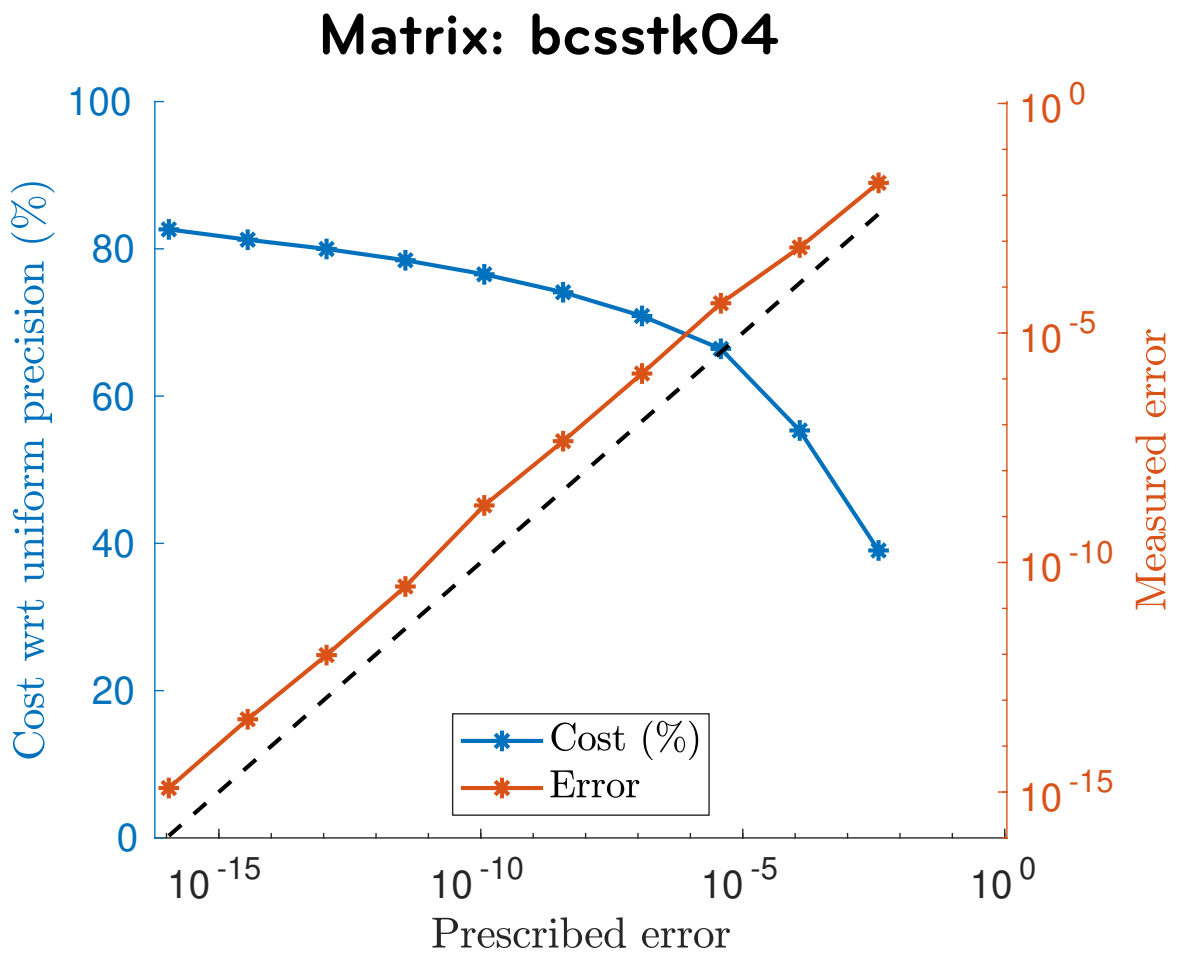


Graillat, Jézéquel, M., Molina (2021)

Data-driven mixed precision SpMV

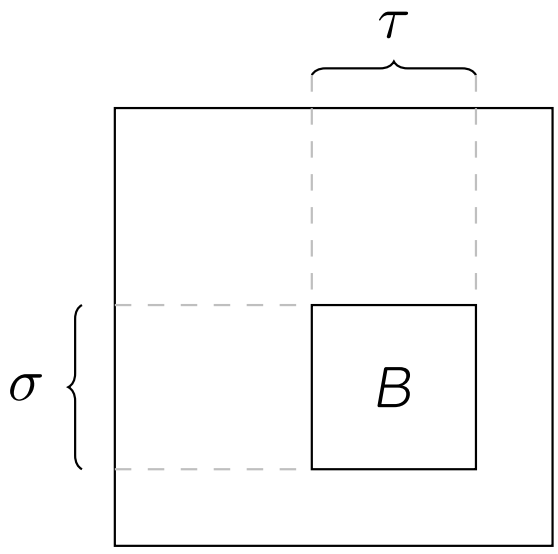
To achieve a backward error of order ε, must control the ratios

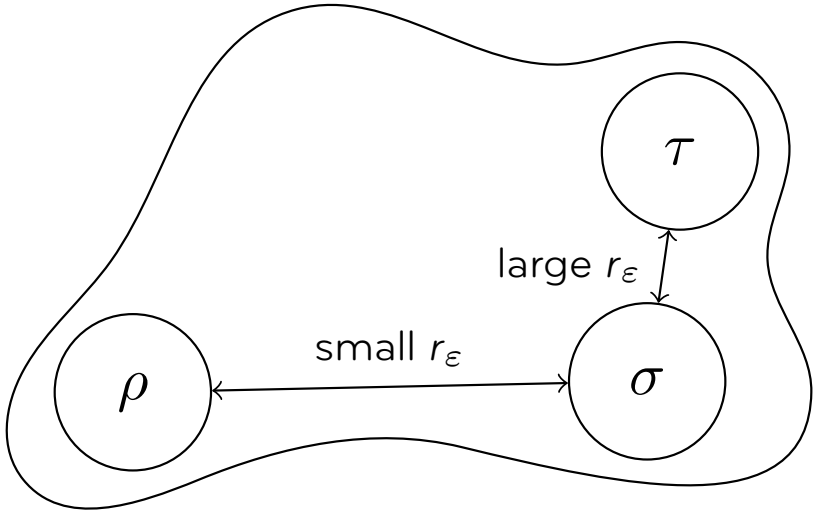
$\phi_{ik} = \frac{\sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|}{\sum_{j=1}^{n} |a_{ij}x_j|}$ $\Rightarrow$ explicit rule for building the buckets B_{ik}



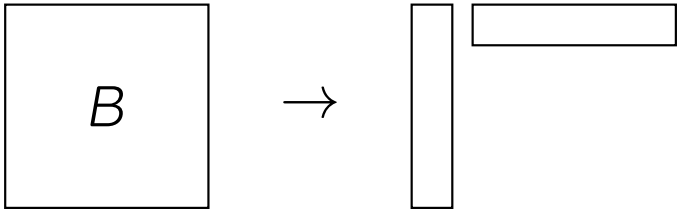
Graillat, Jézéquel, M., Molina (2021)

Data sparse matrices





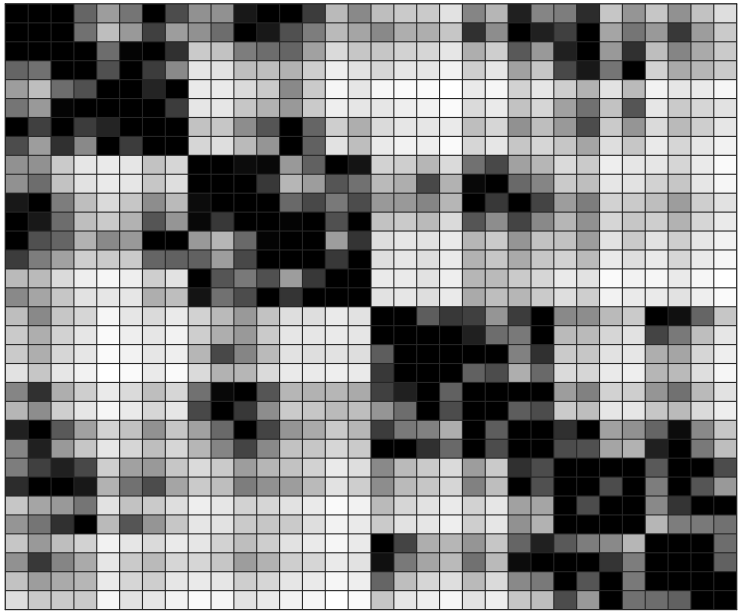
- Data sparse matrices possess a block low rank structure: a block B represents the interaction between two subdomains
  $\Rightarrow$ singular values decay rapidly for far away subdomains



BLR matrices

Block low rank (BLR) matrices use a flat 2D block partitioning

Amestoy et al. (2015) Amestoy et al. (2019)



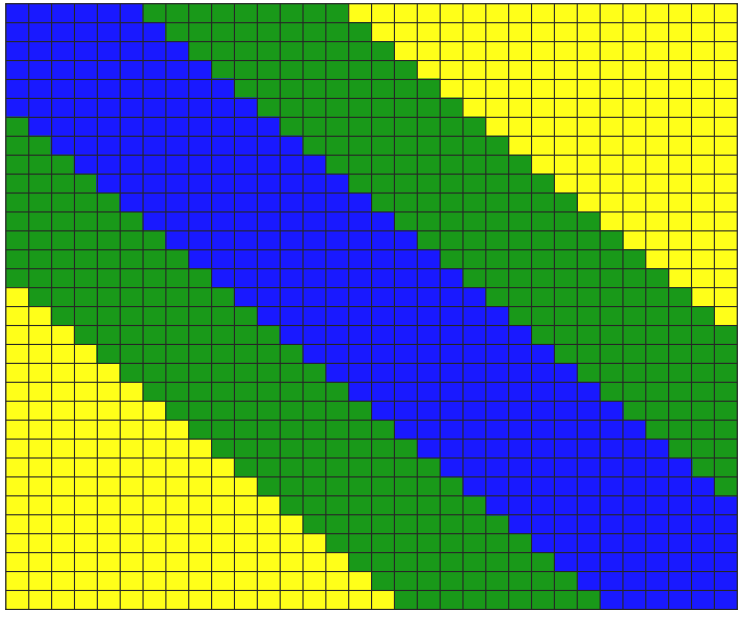
Example of a BLR matrix (Schur complement of a 64^3 Poisson problem with block size 128)

- Diagonal blocks are full rank
- Off-diagonal blocks A_{ij} are approximated by low-rank blocks T_{ij} satisfying $\|A_{ij} - T_{ij}\| \leq \varepsilon \|A\|$
- ε controls the backward error of BLR LU Higham and M. (2021)

Data-driven mixed precision BLR matrices

Idea: store blocks far away from the diagonal in lower precisions

Abdulah et al. (2019) Doucet et al. (2019) Abdulah et al. (2021)

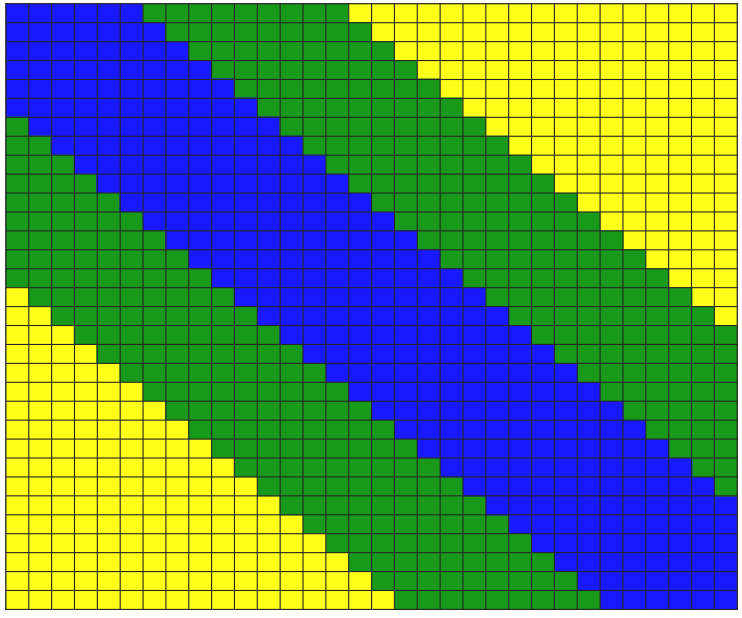


- **double**
- **single**
- **half**

Data-driven mixed precision BLR matrices

Idea: store blocks far away from the diagonal in lower precisions

Abdulah et al. (2019) Doucet et al. (2019) Abdulah et al. (2021)



- **double**
- **single**
- **half**

Analysis:

- Converting A_{ij} to precision $\mathbf{u_{low}}$ introduces an error $\mathbf{u_{low}}\|A_{ij}\|$

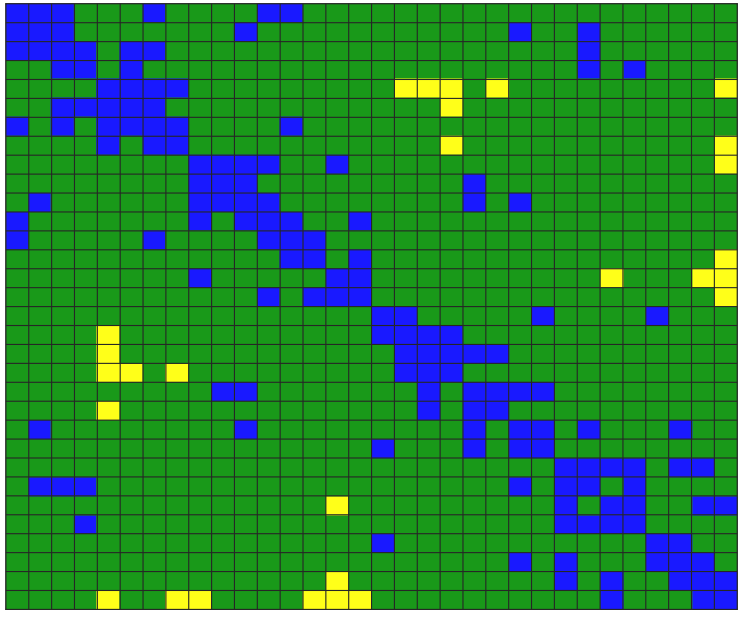
$\Rightarrow$ If $\|A_{ij}\| \leq \varepsilon\|A\|/\mathbf{u_{low}}$, block can be safely stored in precision $\mathbf{u_{low}}$

Data-driven mixed precision BLR matrices

Idea: store blocks far away from the diagonal in lower precisions

Abdulah et al. (2019) Doucet et al. (2019) Abdulah et al. (2021)

(Poisson, $\varepsilon = 10^{-10}$)



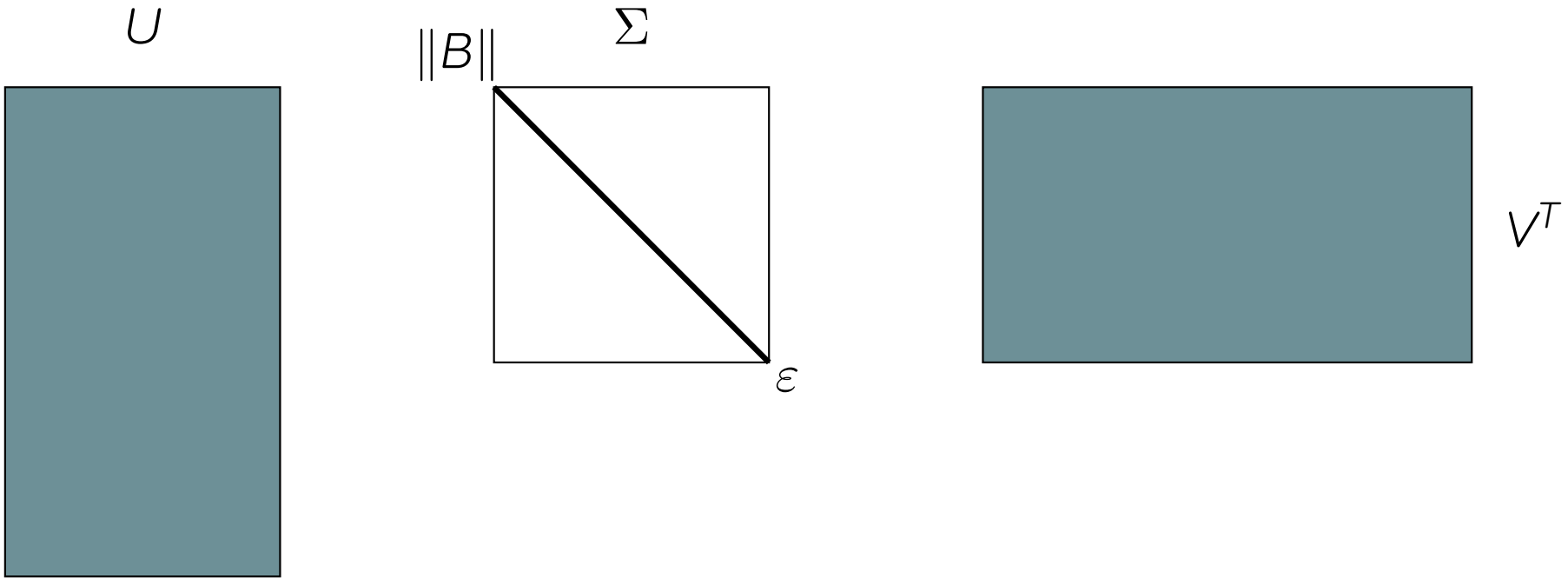
- **double**
- **single**
- **half**

Analysis:

- Converting A_{ij} to precision $\mathbf{u_{low}}$ introduces an error $\mathbf{u_{low}}\|A_{ij}\|$

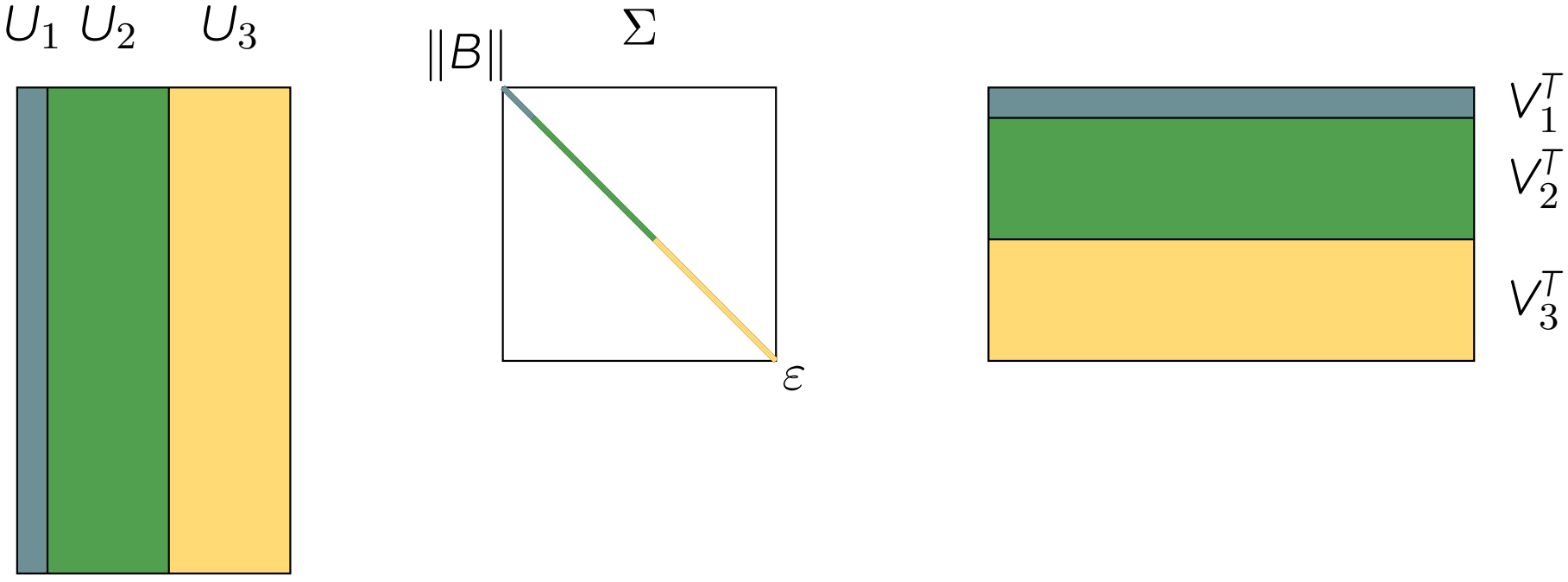
$\Rightarrow$ If $\|A_{ij}\| \leq \varepsilon\|A\|/\mathbf{u_{low}}$, block can be safely stored in precision $\mathbf{u_{low}}$

Data-driven mixed precision low rank compression



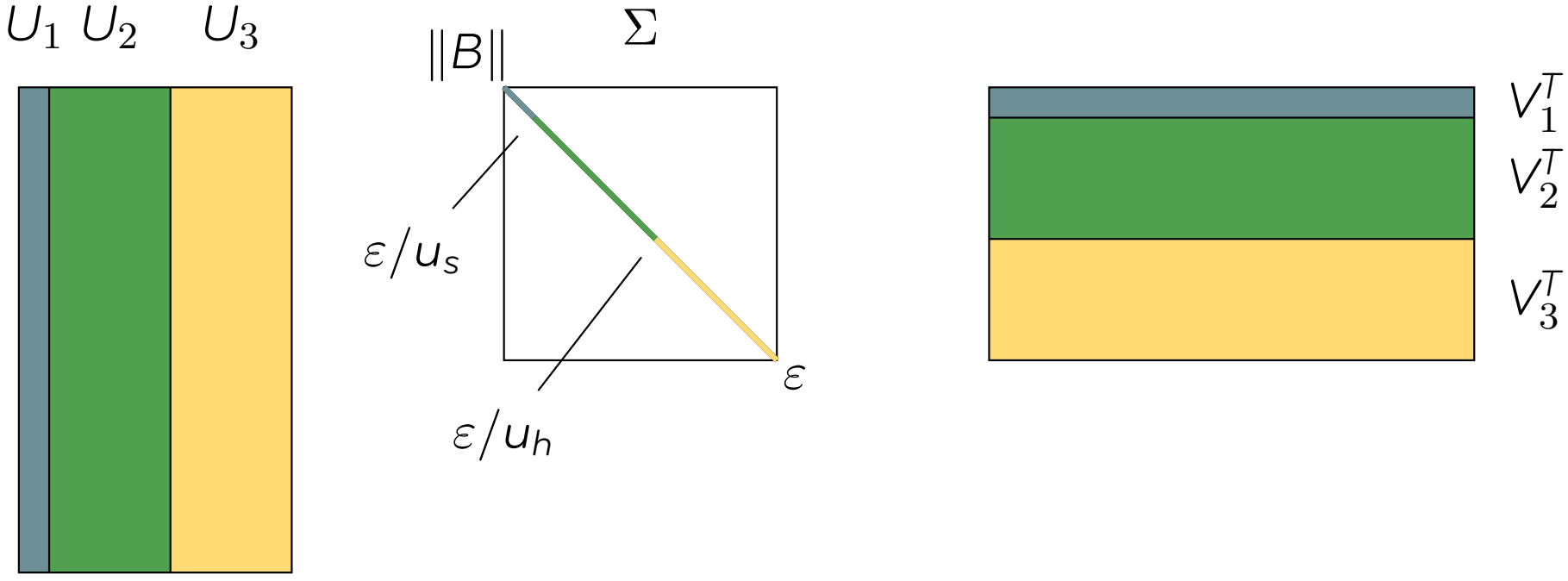
- Low-rank compress based on, e.g., SVD: $\Rightarrow \|B - U\Sigma V^T\| \leq \varepsilon$, everything stored in **double precision**

Data-driven mixed precision low rank compression



- Low-rank compress based on, e.g., SVD: $\Rightarrow \|B - U\Sigma V^T\| \leq \varepsilon$, everything stored in **double precision**
- Mixed precision compression: partition the SVD into several groups of different precision
- Converting U_i and V_i to precision u_i introduces error proportional $u_i\|\Sigma_i\|$

Data-driven mixed precision low rank compression

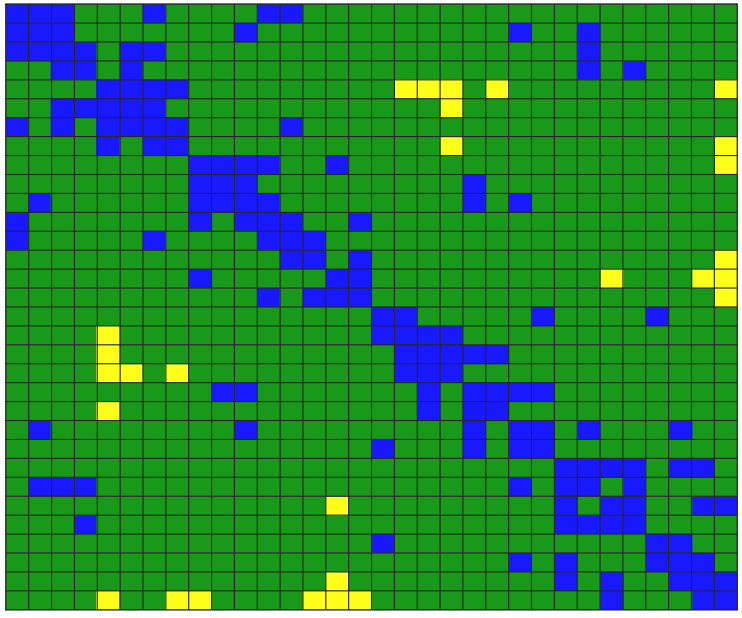


- Low-rank compress based on, e.g., SVD: $\Rightarrow \|B - U\Sigma V^T\| \leq \varepsilon$, everything stored in **double precision**
- Mixed precision compression: partition the SVD into several groups of different precision
- Converting U_i and V_i to precision u_i introduces error proportional $u_i\|\Sigma_i\|$

$\Rightarrow$ Need to partition Σ such that $\|\Sigma_i\| \leq \varepsilon/u_i$

Back to mixed precision BLR matrices

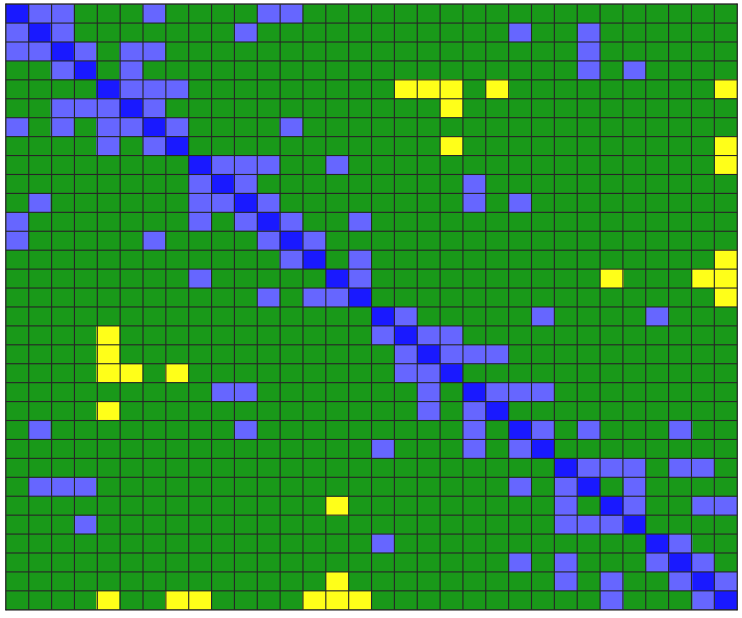
(Poisson, $\varepsilon = 10^{-10}$)



- **double**
- **single**
- **half**

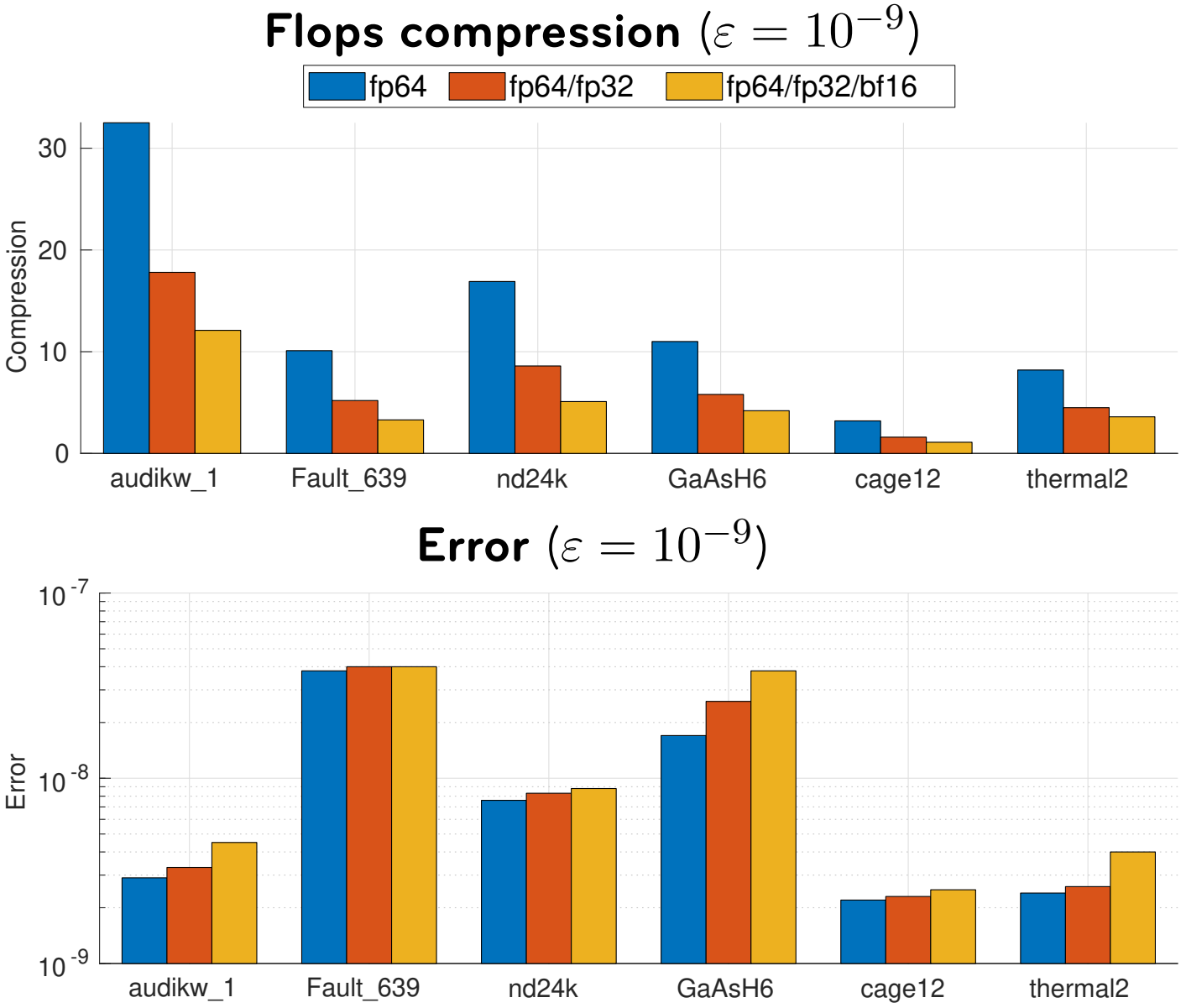
Back to mixed precision BLR matrices

(Poisson, $\varepsilon = 10^{-10}$)



- **double** $\Rightarrow$ **double**, **double/single/half**
- **single** $\Rightarrow$ **single/half**
- **half**

Results with mixed precision BLR LU



Up to $3.3\times$ flops reduction with almost no error increase

Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2021)

Conclusion: mixed precision opportunities in NLA

Compute MP-BLR approximation $M \approx A$ using
  MP-SVD on each block
Solve $Mx_1 = b$ with MP-LU factorization
for $i = 1$: *nsteps* **do**
  $r_i = b - Ax_i$ with MP-SpMV
  Solve $Ad_i = r_i$ with MP-GMRES, using
    MP-preconditioner and MP-SpMV
  $x_{i+1} = x_i + d_i$ (in uniform precision!)
end for

Conclusion: take-home messages

- Numerical linear algebra is **full of opportunities** for mixed precision arithmetic
- Expert knowledge of the algorithms is crucial: need for **cross-disciplinary research** in linear algebra, numerical analysis, and computer arithmetic
- Rounding **error analysis is a precious guide** to the mixed precision practitioner
- Should adapt precisions to the data at hand: **data-driven mixed precision computing**

Slides available at `https://bit.ly/arith21`
(references are clickable)