SIAG/LA Early Career Prize

SIAM LA, May 18, 2021

Exploiting Mixed Precision Arithmetic in the Solution of Linear Systems

Theo Mary

Sorbonne Université, CNRS, LIP6

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

Slides available at https://bit.ly/la21mix

Patrick Amestoy

Pierre Blanchard

Olivier Boiteau

Alfredo Buttari

Matthieu Gerest

Nicholas Higham

Fabienne Jézéquel

Jean-Yves L'Excellent

Florent Lopez

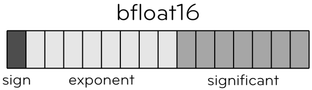
Srikara Pranesh

Bastien Vieublé

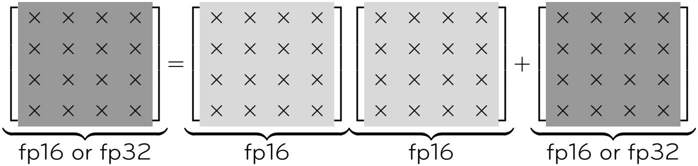
Outline

Objective: accelerate $Ax = b$ in mixed precision by exploiting...

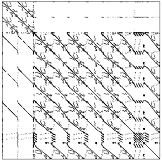
1. **Low precisions** (e.g., fp16, bfloat16)

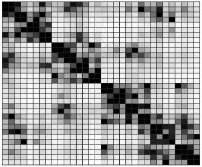


2. **Specialized hardware** (e.g., Tensor Cores)



3. **Sparsity** (both structural and data sparsity)





Low precisions
Specialized hardware
Sparsity

Low precisions
Specialized hardware
Sparsity

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× on A100 from fp32 to fp16/bfloat16)
- Reduced **energy** consumption ($5\times$ with fp16, $9\times$ with bfloat16)

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

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

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**
> $\quad r_i = b - Ax_i$
> $\quad$ Solve $Ad_i = r_i$ via $d_i = U^{-1}(L^{-1}r_i)$
> $\quad x_{i+1} = x_i + d_i$
> **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 $\|\widehat{d}_i - d_i\| \le \phi_i \|d_i\|$
  $x_{i+1} = x_i + d_i$ at precision $\mathbf{u}$
end while

Theorem (simplified from Carson and Higham, 2018)

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

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

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 $\|\widehat{d}_i - d_i\| \le \phi_i \|d_i\|$
  $x_{i+1} = x_i + d_i$ at precision $\mathbf{u}$
end while

Theorem (simplified from Carson and Higham, 2018)

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

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

- **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:

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

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

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

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

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

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

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

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

Solve $Ax_1 = b$ by LU factorization	$\mathbf{u_f} =$ **single**
for $i = 1: nsteps$ **do**	
$r_i = b - Ax_i$	$\mathbf{u_r} =$ **double**
Solve $Ad_i = r_i$ via $d_i = U^{-1}(L^{-1}r_i)$	
$x_{i+1} = x_i + d_i$	$\mathbf{u} =$ **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}
LP factorization	S	D	D	10^{8}	$\kappa(A)\cdot 10^{-16}$

Low precision factorization

Langou et al (2006)

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

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

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

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

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

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 u_f
while Not converged do
    r_i = b − Ax_i at precision u_r
    Solve U^{-1}L^{-1}Ad_i = U^{-1}L^{-1}r_i by GMRES at precision u with
      products with U^{-1}L^{-1}A at precision u^2
    x_{i+1} = x_i + d_i at precision 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** $i = 1\colon nsteps$ **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 $\mathbf{u_g} \le \mathbf{u}$ with matvecs at precision $\mathbf{u_p} \le \mathbf{u}^2$

$\Rightarrow$ **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 $\kappa(\widetilde{A})\big(\kappa(A)\mathbf{u_p} + \mathbf{u_g}\big)$
  - $\kappa(\widetilde{A}) \leq (1 + \kappa(A)\mathbf{u_f})^2 \Rightarrow \phi_i \sim \kappa(A)^2\mathbf{u_f}^2\big(\kappa(A)\mathbf{u_p} + \mathbf{u_g}\big)$

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

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

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

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

Theoretical results

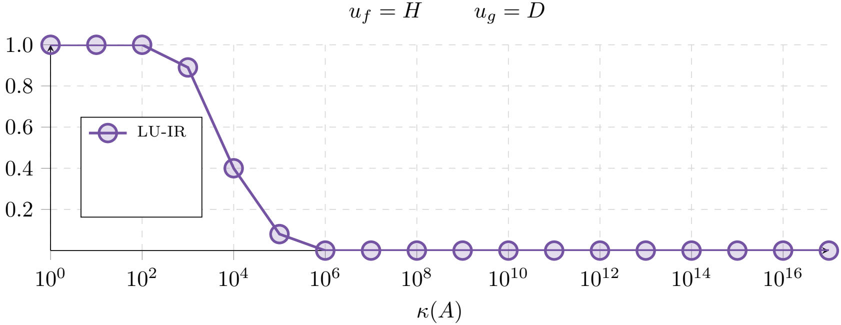
Meaningful combinations of GMRES-IR5 for $\mathbf{u_f} = H$ and $\mathbf{u} = D$.

$\mathbf{u_g}$	$\mathbf{u_p}$	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}

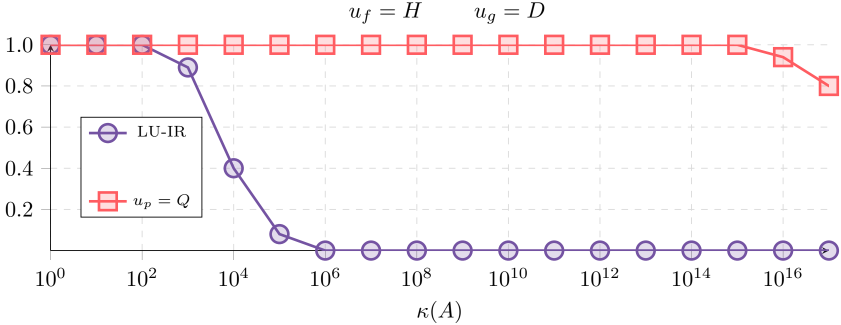
Five combinations between LU-IR and Carson & Higham's GMRES-IR $\Rightarrow$ More **flexible** precisions choice to fit at best the **hardware constraints** and the **problem difficulty**.

Experimental results

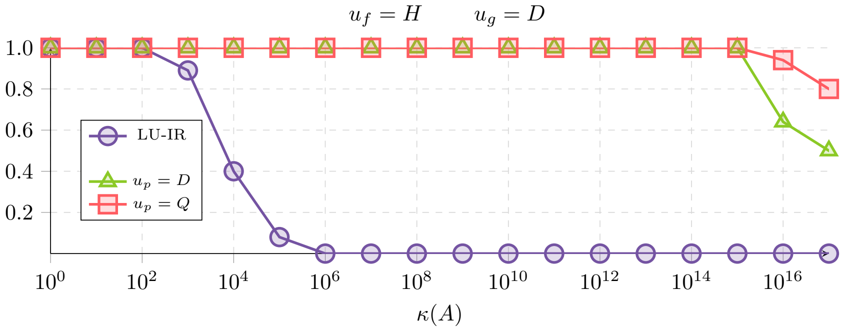
Take 100 random matrices with specified $\kappa(A)$ and measure the success rate: the percentage of matrices for which GMRES-IR5 converges to a small forward error



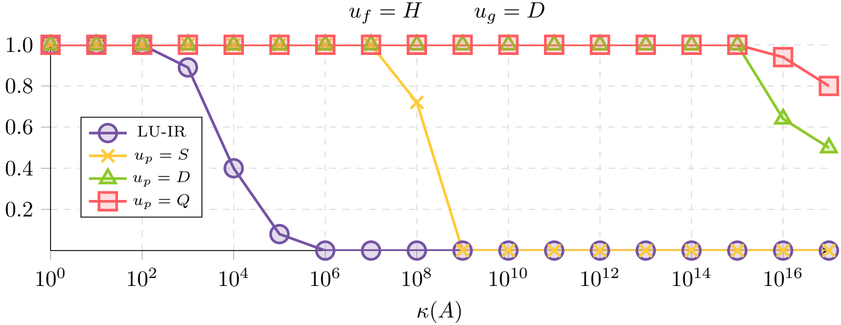
Take 100 random matrices with specified $\kappa(A)$ and measure the success rate: the percentage of matrices for which GMRES-IR5 converges to a small forward error



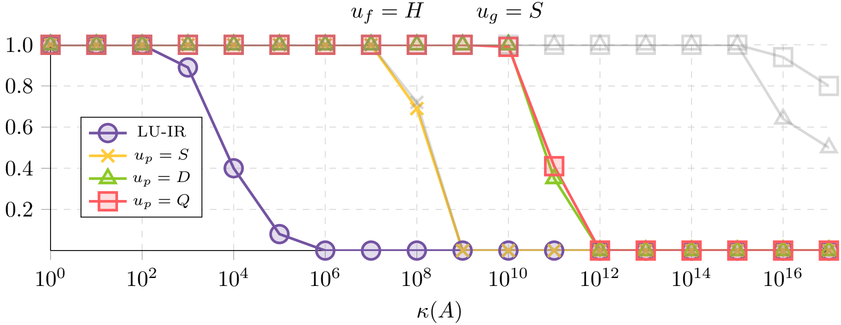
Take 100 random matrices with specified $\kappa(A)$ and measure the success rate: the percentage of matrices for which GMRES-IR5 converges to a small forward error



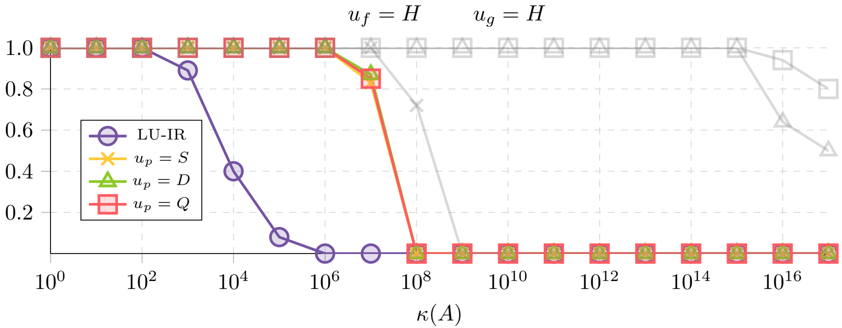
Take 100 random matrices with specified $\kappa(A)$ and measure the success rate: the percentage of matrices for which GMRES-IR5 converges to a small forward error



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



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

Low precisions
Specialized hardware
Sparsity

Low precisions

Specialized hardware

Sparsity

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

$$\underbrace{\overset{D}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp32}} = \underbrace{\overset{A}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp16}} \underbrace{\overset{B}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp16}} + \underbrace{\overset{C}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp32}}$$

- **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**:

$$\underbrace{\overset{D}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp32}} = \underbrace{\overset{A}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp16}} \underbrace{\overset{B}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp16}} + \underbrace{\overset{C}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp32}}$$

- **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

$$|\widehat{C} - C| \lesssim c_n |A||B|, \quad c_n = \left\{ \vphantom{\begin{matrix} a \\ b \\ c \end{matrix}} \right.$$

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**:

$$\begin{array}{ccccccc} D & = & A & & B & + & C \\ \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\text{fp32}} & = & \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\text{fp16}} & & \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\text{fp16}} & + & \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\text{fp32}} \end{array}$$

- **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**:

$$\underbrace{\overset{D}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp32}} = \underbrace{\overset{A}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp16}} \underbrace{\overset{B}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp16}} + \underbrace{\overset{C}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}}_{\text{fp32}}$$

- **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 $k = 1\colon n/b$ **do**
  Factorize $L_{kk}U_{kk} = A_{kk}$ (with unblocked alg.)
  for $i = k+1\colon n/b$ **do**
    Solve $L_{ik}U_{kk} = A_{ik}$ and $L_{kk}U_{ki} = A_{ki}$ for L_{ik} and U_{ki}
  end for
  for $i = k+1\colon n/b$ **do**
    for $j = k+1\colon n/b$ **do**

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

    end for
  end for
end for

Block LU factorization with tensor cores

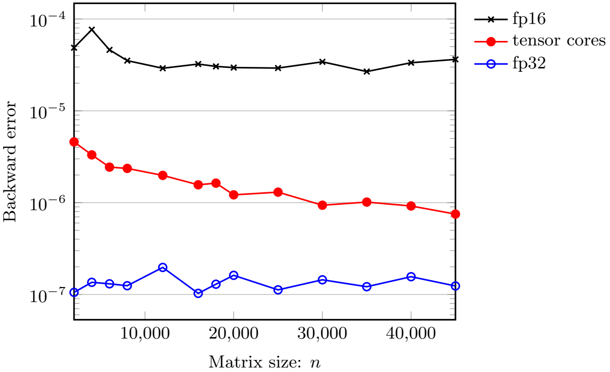
- Block version to use matrix–matrix operations
- $O(n^3)$ part of the flops done with tensor cores

for $k = 1\colon n/b$ **do**
    Factorize $L_{kk}U_{kk} = A_{kk}$ (with unblocked alg.)
    for $i = k+1\colon n/b$ **do**
        Solve $L_{ik}U_{kk} = A_{ik}$ and $L_{kk}U_{ki} = A_{ki}$ for L_{ik} and U_{ki}
    end for
    for $i = k+1\colon n/b$ **do**
        for $j = k+1\colon n/b$ **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

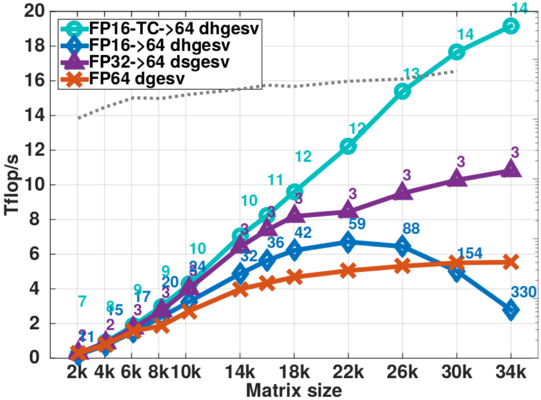
Error analysis for LU follows from matrix multiplication analysis and gives same bounds to first order Blanchard et al. (2020)

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



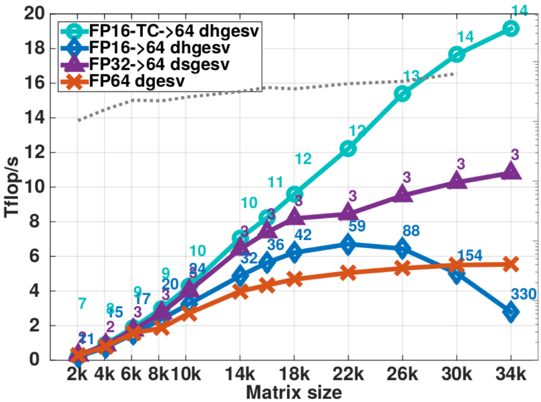
Impact on iterative refinement

Results from Haidar et al. (2018)



- TC accuracy boost can be critical!
- TC performance suboptimal here

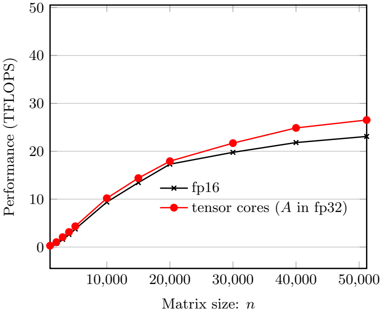
Results from Haidar et al. (2018)

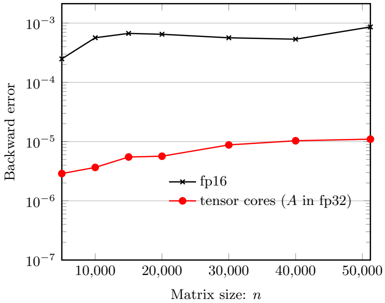


- TC accuracy boost can be critical!
- TC performance suboptimal here ⇒ **why?**

- LU factorization is traditionally a compute-bound operation...
- With Tensor Cores, flops are $8\times$ faster
- Matrix is stored in fp32 $\Rightarrow$ **data movement is unchanged !**

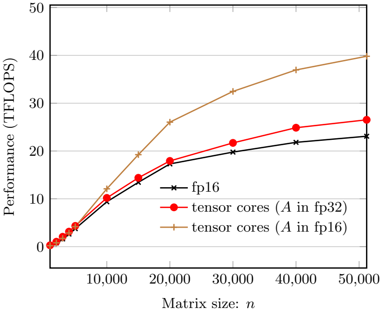
$\Rightarrow$ LU with tensor cores becomes memory-bound !

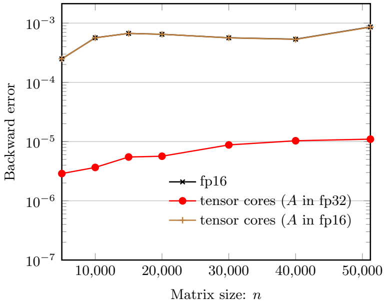




- LU factorization is traditionally a compute-bound operation...
- With Tensor Cores, flops are $8\times$ faster
- Matrix is stored in fp32 $\Rightarrow$ **data movement is unchanged !**

$\Rightarrow$ LU with tensor cores becomes memory-bound !





- Idea: store matrix in fp16
- Problem: huge accuracy loss, tensor cores accuracy boost completely negated

Reducing data movement

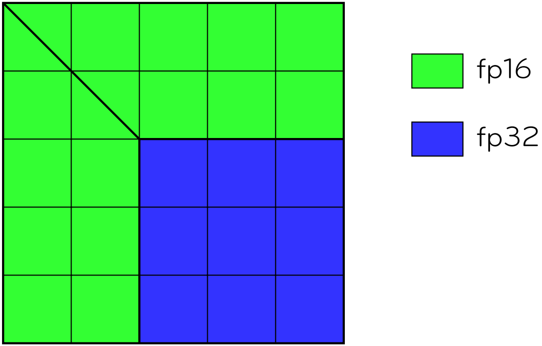
Two ingredients to reduce data movement with no accuracy loss:

Reducing data movement

Two ingredients to reduce data movement with no accuracy loss:

1. Mixed fp16/fp32 representation

Matrix after 2 steps:

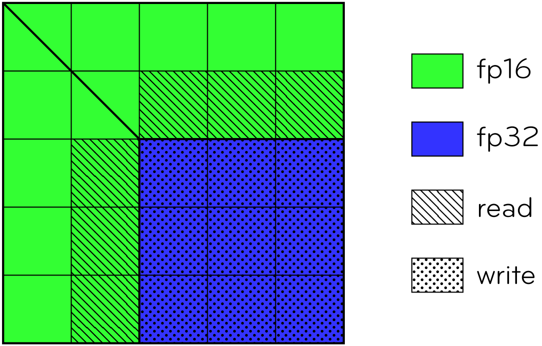


Reducing data movement

Two ingredients to reduce data movement with no accuracy loss:

1. Mixed fp16/fp32 representation

Matrix after 2 steps:

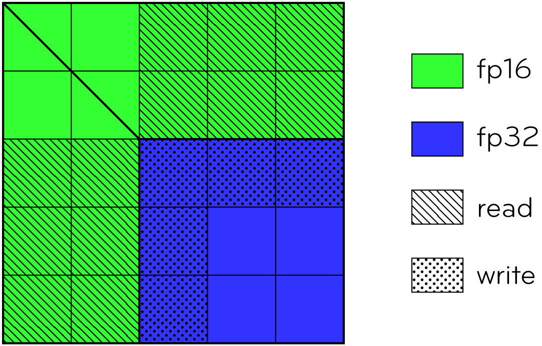


Reducing data movement

Two ingredients to reduce data movement with no accuracy loss:

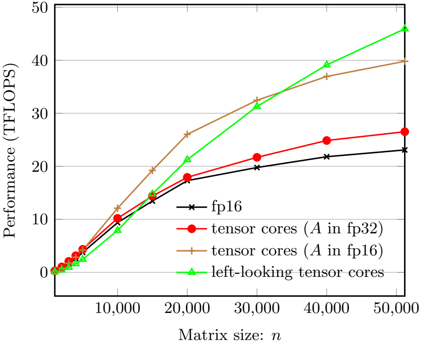
1. Mixed fp16/fp32 representation
2. Right-looking $\rightarrow$ left-looking factorization

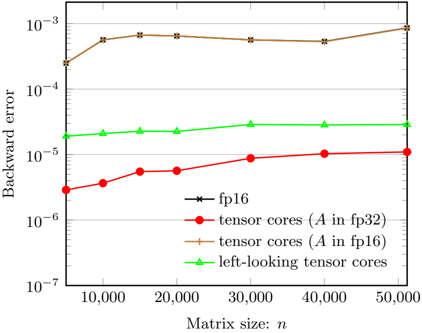
Matrix after 2 steps:



$O(n^3)$ fp32 + $O(n^2)$ fp16 $\rightarrow$ $O(n^2)$ fp32 + $O(n^3)$ fp16

Experimental results





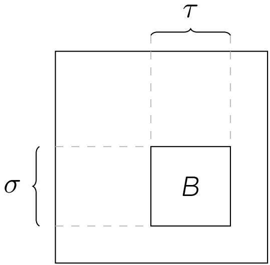
Nearly **50 TFLOPS** without significantly impacting accuracy

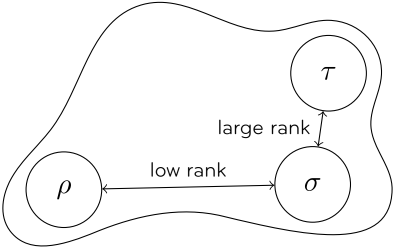
Lopez and M. (2020)

Low precisions
Specialized hardware
Sparsity

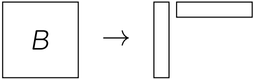
Low precisions
Specialized hardware
Sparsity

- Sparse matrices: exploit exact zeros
- Data sparse matrices: exploit numerical zeros



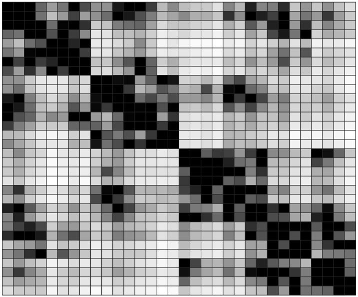


- A block B represents the interaction between two subdomains
  $\Rightarrow$ low numerical rank for far away subdomains



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 $\|A_{ij} - T_{ij}\| \le \varepsilon \|A\|$
- ε controls the backward error of BLR LU Higham and M. (2021)

Complexity of LU factorization

- Crucial to exploit sparsity to tackle large scale problems

	Flops	Storage
Dense	$O(n^3)$	$O(n^2)$
Sparse (3D domain)	$O(n^2)$	$O(n^{4/3})$
BLR (constant ranks)	$O(n^2)$	$O(n^{3/2})$
Sparse+BLR	$O(n^{4/3})$	$O(n \log n)$

Amestoy, Buttari, L'Excellent, M. (2017)

Complexity of LU factorization

- Crucial to exploit sparsity to tackle large scale problems

	Flops	Storage
Dense	$O(n^3)$	$O(n^2)$
Sparse (3D domain)	$O(n^2)$	$O(n^{4/3})$
BLR (constant ranks)	$O(n^2)$	$O(n^{3/2})$
Sparse+BLR	$O(n^{4/3})$	$O(n \log n)$

Amestoy, Buttari, L'Excellent, M. (2017)

- **In mixed precision**, is sparsity **a challenge or an opportunity?**

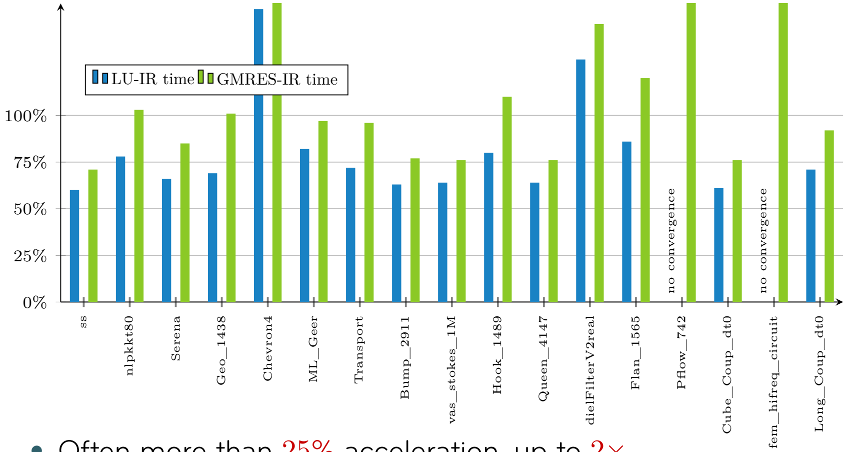
$\Rightarrow$ A little bit of both

Challenge: ratio LU factorization cost / LU solve cost

$$\begin{array}{ccccc} \text{Dense} & \rightarrow & \text{Sparse} & \rightarrow & \text{Sparse+BLR} \\ O(n) & \rightarrow & O(n^{2/3}) & \rightarrow & O(n^{1/3}) \end{array}$$

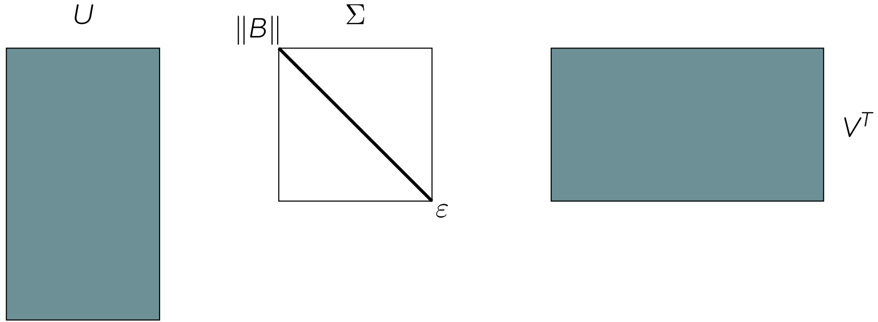
$\Rightarrow$ **less room to amortize iterations**

fp32 LU (MUMPS) + IR on large sparse ill-conditioned matrices
Time (%) w.r.t. fp64 MUMPS solver



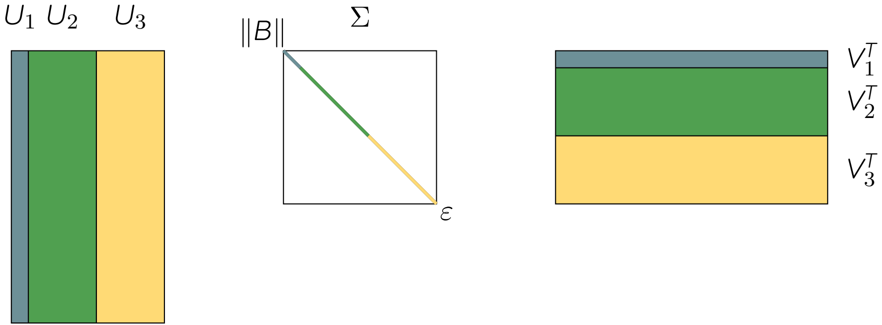
- Often more than 25% acceleration, up to $2\times$
- GMRES-IR slower than LU-IR but more robust

Mixed precision low rank compression



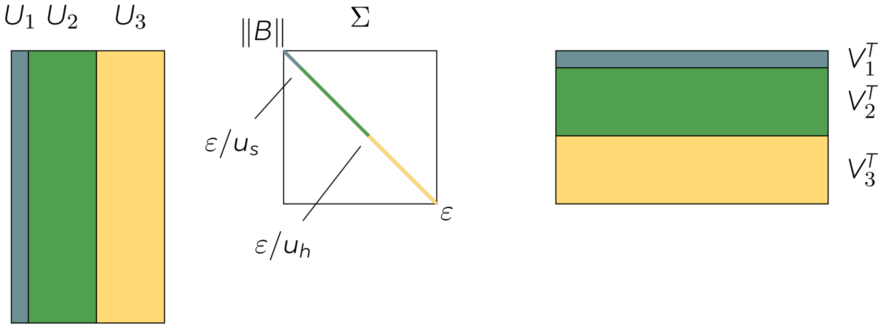
- Low-rank compress based on, e.g., SVD: $\Rightarrow \|B - U\Sigma V^T\| \leq \varepsilon$, everything stored in **double precision**

Mixed precision 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 $u_i\|\Sigma_i\|$

Mixed precision low rank compression

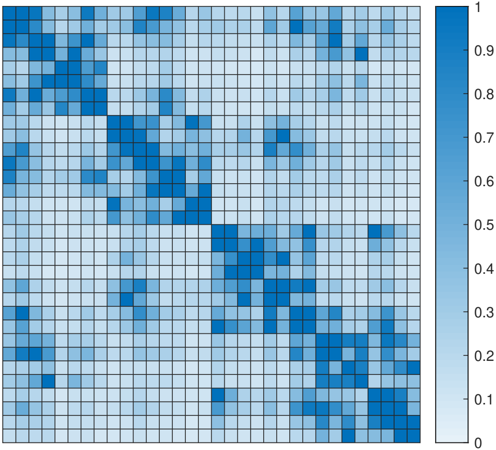


- Low-rank compress based on, e.g., SVD: $\Rightarrow \|B - U\Sigma V^T\| \leq \varepsilon$, everything stored in **double precision**
- Mixed precision compression: partition the SVD into several groups of different precision
- Converting U_i and V_i to precision u_i introduces error proportional $u_i\|\Sigma_i\|$

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

Mixed precision BLR matrices

(Poisson, $\varepsilon = 10^{-12}$)

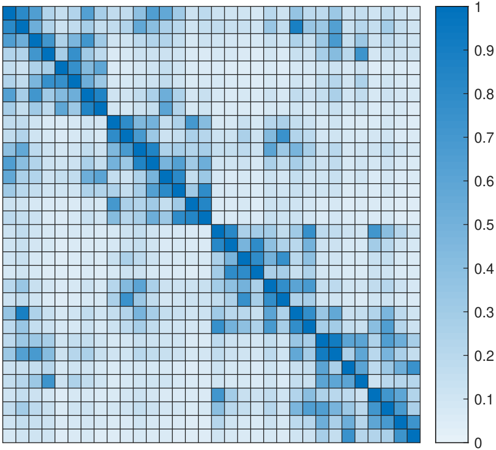


Double

100%

Mixed precision BLR matrices

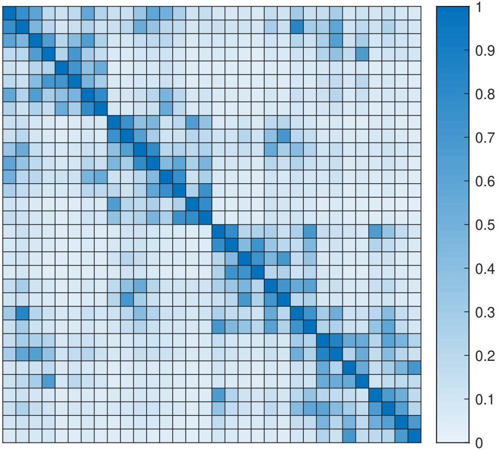
(Poisson, $\varepsilon = 10^{-12}$)



Double 26% Single 74%

Mixed precision BLR matrices

(Poisson, $\varepsilon = 10^{-12}$)



Double	Single	Half
26%	44%	30%

Most entries can be stored in precision much lower than ε !

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

Conclusions

- Emerging **low precisions** provide new opportunities for high performance NLA
- **Mixed precision algorithms** have proven highly successful at $Ax = b$, even for ill-conditioned A
- **Specialized hardware** helps, both for speed and accuracy
- **Sparsity** can make things more challenging... but data sparsity creates new mixed precision opportunities!

Slides available at `https://bit.ly/la21mix`
(references on next slides)

References (mixed precision algorithms)

- E. Carson and N. J. Higham. Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions. *SIAM J. Sci. Comput.*, 40(2), A817–A847 (2018)
- E. Carson and N. J. Higham. A New Analysis of Iterative Refinement and Its Application to Accurate Solution of Ill-Conditioned Sparse Linear Systems. *SIAM J. Sci. Comput.*, 39(6), A2834–A2856 (2017).
- P. R. Amestoy, A. Buttari, N. J. Higham, J.-Y. L'Excellent, T. Mary, and B. Vieublé. Five-precision GMRES-based Iterative Refinement. MIMPS EPrint 2021.5.
- A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham. Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers. *SC'18*.
- P. Blanchard, N. J. Higham, F. Lopez, T. Mary, and S. Pranesh. Mixed Precision Block Fused Multiply-Add: Error Analysis and Application to GPU Tensor Cores. *SIAM J. Sci. Comput.* 42(3), C124–C141 (2020).
- F. Lopez and T. Mary. Mixed Precision LU Factorization on GPU Tensor Cores: Reducing Data Movement and Memory Footprint. MIMS EPrint 2020.20.

References (BLR matrices)

- P. R. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker. Improving Multifrontal Methods by Means of Block Low-Rank Representations *SIAM J. Sci. Comput.*, 37(3), A1451–A1474 (2015).
- P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. On the Complexity of the Block Low-Rank Multifrontal Factorization. *SIAM J. Sci. Comput.*, 39(4), A1710–A1740 (2017).
- P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures. *ACM Trans. Math. Softw.*, 45(1), 2:1–2:26 (2019).
- N. J. Higham and T. Mary. Solving Block Low-Rank Linear Systems by LU Factorization is Numerically Stable. *IMA J. Numer. Anal.*, drab020 (2021).
- P. R. Amestoy, O. Boiteau, A. Buttari, M. Gerest, F. Jézéquel, J.-Y. L'Excellent, and T. Mary. Mixed Precision Low Rank Approximations and their Application to Block Low Rank Matrix Factorization. In preparation.