Mixed precision iterative refinement for the solution of large sparse linear systems

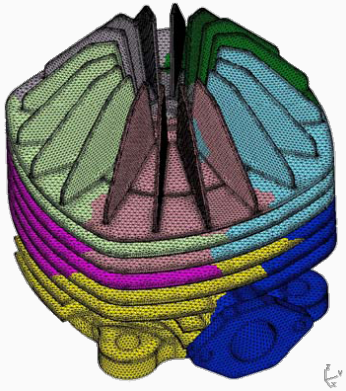
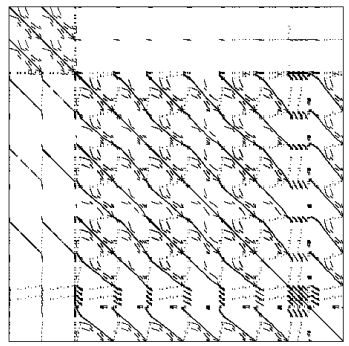
Speaker: Bastien Vieublé
Supervisors: Alfredo Buttari and Théo Mary

30/11/2022

INPT-IRIT, Toulouse

Solving Large Sparse Linear Systems

Context


Sparse Linear System $Ax = b$

At the foundations of many scientific computing applications (e.g., discretization of PDEs).

Large-scale sparse linear systems...

Up to billions of unknowns, applications demanding TeraBytes of memory and Exaflops of computation.

...require large-scale computers.

Increasingly large numbers of cores available, high heterogeneity in the computation (CPU, GPU, FPGA, TPU, etc), and high heterogeneity in data motions (RAM to cache, out-of-core, node to node transfer, etc).

Solvers

What are the ways to solve a sparse $Ax = b \in \mathbb{R}^n$ on computers?

Iterative solvers

Compute a sequence of x_k converging towards x.

Examples: Gauss-Seidel, SOR, Krylov subspace methods, etc.

- ➤ Low computational cost and memory consumption if the convergence is quick (about $\mathcal{O}(n)$ operations per iteration)...
- ➤ BUT convergence depends on the matrix properties.

Direct solvers

Based on a factorization of A.

Examples: LDL^T, LU, QR, etc.

- ➤ High computational cost and memory consumption...
- ➤ BUT they are robust and easy to use.

Solvers

What are the ways to solve a sparse $Ax = b \in \mathbb{R}^n$ on computers?

Iterative solvers

Compute a sequence of x_k converging towards x.

Examples: Gauss-Seidel, SOR, Krylov subspace methods, etc.

- ➤ Low computational cost and memory consumption if the convergence is quick (about $\mathcal{O}(n)$ operations per iteration)...
- ➤ BUT convergence depends on the matrix properties.

Direct solvers

Based on a factorization of A.

Examples: LDL^T, LU, QR, etc.

- ➤ High computational cost and memory consumption...
- ➤ BUT they are robust and easy to use.

$\Rightarrow$ **For both, the reduction of the computational cost is the focus of much research.**

Reduce the cost by reducing the complexity

Approximate computing: deliberately approximate the computations in order to improve the performance at the cost of introducing a perturbation.

- The perturbed problem should be close to the original one and should reduce time and/or memory!
- In general the larger the perturbations the larger the savings...
- BUT large perturbations = low accuracy!

In this PhD we explore two approximate computing techniques: low precision arithmetics, numerical approximations.

Low precision arithmetics

Commonly available arithmetics

	ID	Signif. bits	Exp. bits	Range	Unit roundoff u
fp128	Q	113	15	$10^{\pm 4932}$	1×10^{-34}
double-fp64	DD	107	11	$10^{\pm 308}$	6×10^{-33}
fp64	D	53	11	$10^{\pm 308}$	1×10^{-16}
fp32	S	24	8	$10^{\pm 38}$	6×10^{-8}
tfloat32	T	11	8	$10^{\pm 38}$	5×10^{-4}
fp16	H	11	5	$10^{\pm 5}$	5×10^{-4}
bfloat16	B	8	8	$10^{\pm 38}$	4×10^{-3}
fp8 (E4M3)	R	4	4	$10^{\pm 2}$	6.3×10^{-2}
fp8 (E5M2)	R*	3	5	$10^{\pm 5}$	1.3×10^{-1}

➤ The unit roundoff is the largest relative distance between any number and its closest floating point representation.

➤ The range is the interval of representable numbers by a given arithmetic.

➤ Recent announcement of 8-bit arithmetics: fp8 E4M3 and E5M2.

Why using low precision arithmetics?

Low precision arithmetics are less accurate and present a narrower range. BUT there are 3 main benefits of using low precision arithmetics:

- ➤ Storage, data movement and communications are all proportional to the total number of bits. ⇒ Time and memory savings!
- ➤ Speed of computation is also at least proportional to the total number of bits. ⇒ Time savings!
- ➤ Power consumption is dependent on the number of bits[1]. ⇒ Energy savings!

[1] Tong, Nagle and Rutenbar, *Reducing power by optimizing the necessary precision/range of floating-point arithmetic*, 2000

Numerical approximations

What is a numerical approximation?

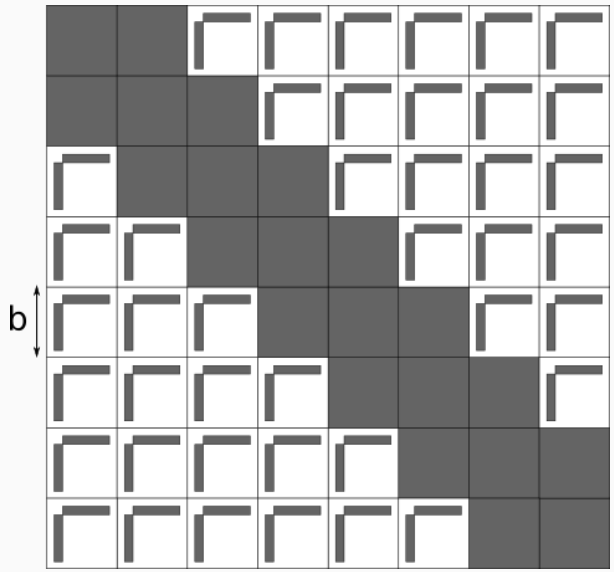
Numerical approximations refer to a class of approaches that relax certain constraints on the quality of the solution at the algorithm level to leverage resource savings.

- They are independent of the floating point arithmetic used.
- They introduce arbitrary or controllable perturbations that affect the accuracy of the solution.
- Many numerical approximations exist on various kinds of algorithms.

We focus on two examples of numerical approximations for direct solvers: block low-rank and static pivoting.

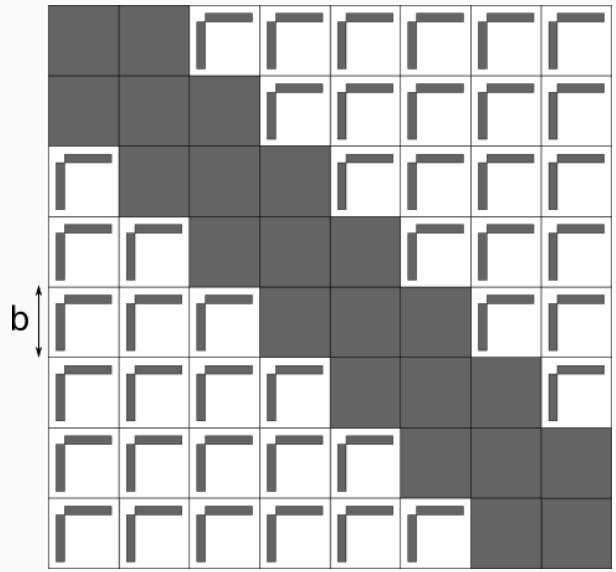
Numerical approximations: Block Low-Rank

Block Low-Rank: Decompose dense matrices into regular blocks of size b.
Try to compress each block with a low rank approximation at precision ϵ_{BLR}.



Numerical approximations: Block Low-Rank

Block Low-Rank: Decompose dense matrices into regular blocks of size b.
Try to compress each block with a low rank approximation at precision ϵ_{BLR}.



	Flops	Memory
Classic[2]	$O(n^3)$	$O(n^2)$
BLR[3]	$O(n^{2.5})$	$O(n^{1.5})$

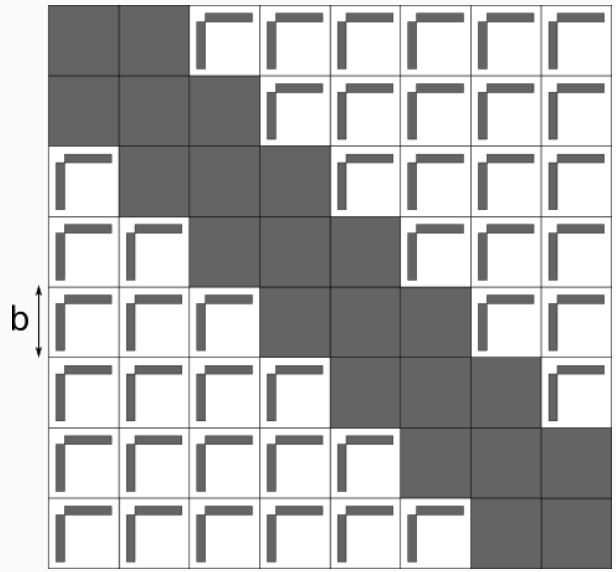
Complexities of dense factorization

[2] J. A. George, *Nested dissection of a regular finite element mesh*, 1973

[3] Amestoy, Buttari, L'Excellent, and Mary, *On the Complexity of the Block Low-Rank Multifrontal Factorization*, 2017

Numerical approximations: Block Low-Rank

Block Low-Rank: Decompose dense matrices into regular blocks of size b.
Try to compress each block with a low rank approximation at precision ϵ_{BLR}.



	Flops	Memory
Classic	$O(n^3)$	$O(n^2)$
BLR	$O(n^{2.5})$	$O(n^{1.5})$

Complexities of dense factorization

Pros:

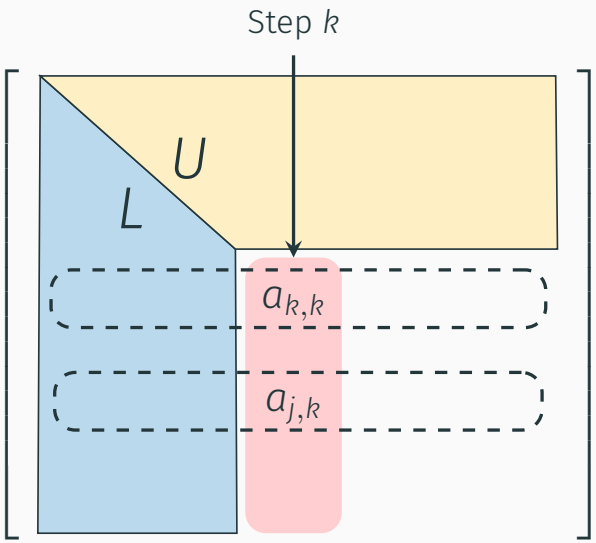
- Reduction of asymptotic complexity...
- ... which is translated in time and memory savings!

Cons:

- Introduce a perturbation ϵ_{BLR}.
- Compression is problem dependent.

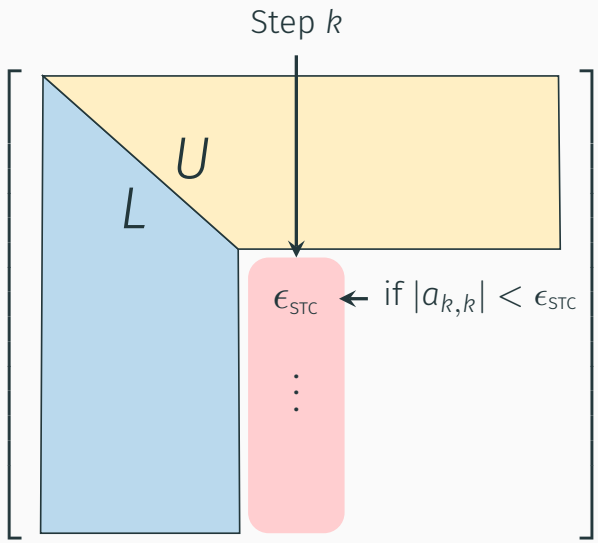
Numerical approximations: Static pivoting

Numerical pivoting is essential for stability and accuracy of direct linear solvers. Different methods exist, they achieve different trade-offs.



$$|a_{j_k}| = \max_{l \in \{k,n\}} |a_{l,k}|$$

Partial Pivoting

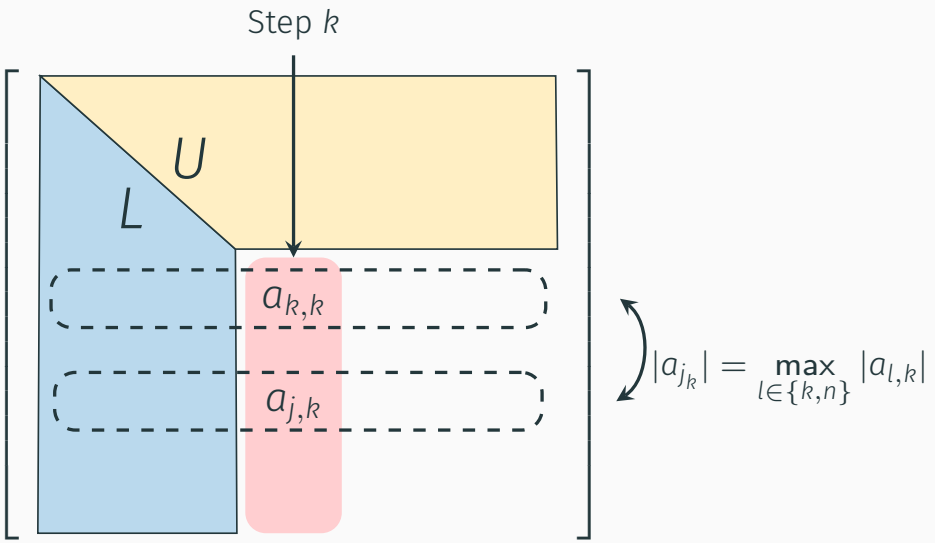


Static Pivoting[2]

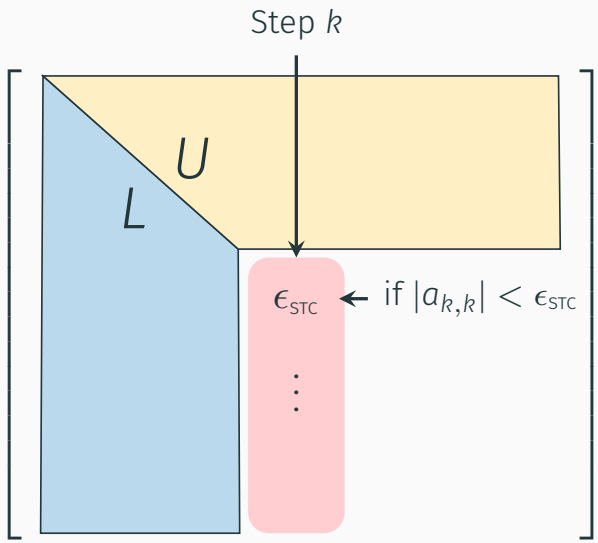
[2] X. S. Li and J. W. Demmel, *A Scalable Sparse Direct Solver Using Static Pivoting*, 1998

Numerical approximations: Static pivoting

Numerical pivoting is essential for stability and accuracy of direct linear solvers. Different methods exist, they achieve different trade-offs.



Partial Pivoting



Static Pivoting

Pros:

- More BLAS 3.
- No communication for synchronization.

Cons:

- Large ϵ_{STC} less accuracy.
- Small ϵ_{STC} less stability.

The fundamental issue of approximate computing

Problem

- ➤ Low precision arithmetics and approximations can greatly improve performance of sparse linear solvers...
- ➤ BUT they degrade their accuracy at the same time.
- ➤ Unfortunately application experts generally require high accuracy on the solution (i.e. most commonly double precision accuracy).

Idea: What if we could use low precisions and approximations to accelerate the most expensive parts of the computation, and use higher precision only on some strategic operations to recover the lost accuracy at low cost?

The fundamental issue of approximate computing

Problem

- ➤ Low precision arithmetics and approximations can greatly improve performance of sparse linear solvers...
- ➤ BUT they degrade their accuracy at the same time.
- ➤ Unfortunately application experts generally require high accuracy on the solution (i.e. most commonly double precision accuracy).

Idea: What if we could use low precisions and approximations to accelerate the most expensive parts of the computation, and use higher precision only on some strategic operations to recover the lost accuracy at low cost?

$\Rightarrow$ This is the goal of mixed precision algorithms!

Mixed precision iterative refinement(s) for LU direct solver

LU-IR3: Recover the accuracy on linear systems

Algorithm LU-based iterative refinement in three precisions[a]

1: Compute the LU factorization $A = LU$ (u_f)
2: Solve $Ax_0 = b$ (u_f)
3: **while not** converged **do**
4: Compute $r_i = b - Ax_i$ (u_r)
5: Solve $Ad_i = r_i$ by $d_i = \hat{U}^{-1}\hat{L}^{-1}r_i$ (u_f)
6: Compute $x_{i+1} = x_i + d_i$ (u)
7: **end while**

[a] E. Carson and N. J. Higham, *Accelerating the solution of linear systems by iterative refinement in three precisions*, 2018

LU-IR3: Recover the accuracy on linear systems

Algorithm LU-based iterative refinement in three precisions[a]

1:	Compute the LU factorization $A = LU$	$\mathcal{O}(n^3)$	(u_f)
2:	Solve $Ax_0 = b$	$\mathcal{O}(n^2)$	(u_f)
3:	**while not** converged **do**		
4:	Compute $r_i = b - Ax_i$	$\mathcal{O}(n^2)$	(u_r)
5:	Solve $Ad_i = r_i$ by $d_i = \hat{U}^{-1}\hat{L}^{-1}r_i$	$\mathcal{O}(n^2)$	(u_f)
6:	Compute $x_{i+1} = x_i + d_i$	$\mathcal{O}(n)$	(u)
7:	**end while**		

The strategy is to accelerate with low precisions and/or numerical approximations the factorization $\mathbf{O(n^3)}$ and recover a good accuracy by using higher precisions for the residual and update $\mathbf{O(n^2)}$.

[a] E. Carson and N. J. Higham, *Accelerating the solution of linear systems by iterative refinement in three precisions*, 2018

LU-IR3: Recover the accuracy on linear systems

Algorithm LU-based iterative refinement in three precisions

1:	Compute the LU factorization $A = LU$	$\mathcal{O}(n^3)$	(u_f)
2:	Solve $Ax_0 = b$	$\mathcal{O}(n^2)$	(u_f)
3:	**while not** converged **do**		
4:	Compute $r_i = b - Ax_i$	$\mathcal{O}(n^2)$	(u_r)
5:	Solve $Ad_i = r_i$ by $d_i = \hat{U}^{-1}\hat{L}^{-1}r_i$	$\mathcal{O}(n^2)$	(u_f)
6:	Compute $x_{i+1} = x_i + d_i$	$\mathcal{O}(n)$	(u)
7:	**end while**		

	Convergence condition	Forward error
LU-IR3	$\kappa(A)u_f \ll 1$	$u_r\kappa(A) + u$

Limit: Very low precision factorization leads to a very restrictive convergence condition for LU-IR3 (e.g. with $u_f =$ fp16 we have $\kappa(A) \ll 2 \times 10^3$).

LU-GMRES-IR3: Get more robust

Algorithm GMRES-based iterative refinement in three precisions[a]

1: Compute the LU factorization $A = LU$ (u_f)
2: Solve $Ax_0 = b$ (u_f)
3: **while not** converged **do**
4: Compute $r_i = b - Ax_i$ (u_r)
5: Solve $\widetilde{A}d_i = \hat{U}^{-1}\hat{L}^{-1}Ad_i = \hat{U}^{-1}\hat{L}^{-1}r_i$ by GMRES at precision (u) with matrix vector products with $\widetilde{A}$ at precision (u^2)
6: Compute $x_{i+1} = x_i + d_i$ (u)
7: **end while**

➤ LU-GMRES-IR3 is a more robust form of iterative refinement capable of tackling higher condition numbers $\kappa(A)$ than LU-IR3.

➤ Based on GMRES solver which is a well-known Krylov subspace based iterative solver.

[a] E. Carson and N. J. Higham, *A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems*, 2017

LU-GMRES-IR3: Get more robust

Algorithm GMRES-based iterative refinement in three precisions

1: Compute the LU factorization $A = LU$ (u_f)
2: Solve $Ax_0 = b$ (u_f)
3: **while not** converged **do**
4: Compute $r_i = b - Ax_i$ (u_r)
5: Solve $\widetilde{A}d_i = \hat{U}^{-1}\hat{L}^{-1}Ad_i = \hat{U}^{-1}\hat{L}^{-1}r_i$ by GMRES at precision (u) with matrix vector products with $\widetilde{A}$ at precision (u^2)
6: Compute $x_{i+1} = x_i + d_i$ (u)
7: **end while**

	Convergence condition	Forward error
LU-IR3	$\kappa(A)u_f \ll 1$	$u_r\kappa(A) + u$
LU-GMRES-IR3	$\kappa(A)u^{1/2}u_f \ll 1$	$u_r\kappa(A) + u$

Example: If $u_f =$ fp16, the condition on LU-IR3 is 2×10^3, on LU-GMRES-IR3 it is 2×10^{11}!

Promises and open questions

Theoretical promises of modern IR

- Can use low precisions to accelerate the computation of the solution of sparse systems.
- Recover high accuracy at low cost (forward error $= 10^{-16}$).
- Process ill-conditioned matrices (i.e. big $\kappa(A)$).

Major open questions:

1. Is the application of the preconditioned matrix–vector product in precision u^2 costless in LU-GMRES-IR3? If not, can we trade off robustness and performance by relaxing the requirements on the precisions in LU-GMRES-IR3?

2. How can we efficiently translate the theoretical complexity reduction in actual performance gains for the parallel direct solution of large sparse systems and how to combine them with numerical approximations?

3. How these modern IR algorithms, focused on the improvement of direct solvers, can be extended for the improvement of iterative solvers (in particular Krylov subspace based solvers)?

Promises and open questions

Theoretical promises of modern IR

- Can use low precisions to accelerate the computation of the solution of sparse systems.
- Recover high accuracy at low cost (forward error $= 10^{-16}$).
- Process ill-conditioned matrices (i.e. big $\kappa(A)$).

Major open questions:

1. Is the application of the preconditioned matrix–vector product in precision u^2 costless in LU-GMRES-IR3? If not, can we trade off robustness and performance by Relaxing the requirements on the precisions in LU-GMRES-IR3?

2. How can we efficiently translate the theoretical complexity reduction in actual Performance gains for the parallel direct solution of large sparse systems and how to combine them with numerical approximations ?

3. How these Modern IR algorithms, focused on the improvement of direct solvers, can be extended for the improvement of iterative solvers (in particular Krylov subspace based solvers)?

Relax the precisions of LU-GMRES-IR3

Practical issues of LU-GMRES-IR3

LU-GMRES-IR3 is more robust on $\kappa(A)$ than LU-IR3. However, the LU solves are performed in precision u^2 for the application of the preconditioner: this is a major practical issue.

- It increases cost per iteration compared with LU-IR3.
- If $u = \text{fp64}$ then $u^2 = \text{fp128} \Rightarrow$ It requires a quad precision LU solver (not widely available on commonly used parallel sparse direct solvers). Moreover, if fp128 is not supported by the hardware, it can be really slow.
- Need to cast the LU factors from precision u_f to precision u^2 $\Rightarrow$ Huge memory consumption increase if we keep a full copy of the factors.

Other issue: Do we need to run the other GMRES operations in precision u?

Practical issues of LU-GMRES-IR3

LU-GMRES-IR3 is more robust on $\kappa(A)$ than LU-IR3. However, the LU solves are performed in precision u^2 for the application of the preconditioner: this is a major practical issue.

- ➤ It increases cost per iteration compared with LU-IR3.
- ➤ If $u =$ fp64 then $u^2 =$ fp128 $\Rightarrow$ It requires a quad precision LU solver (not widely available on commonly used parallel sparse direct solvers). Moreover, if fp128 is not supported by the hardware, it can be really slow.
- ➤ Need to cast the LU factors from precision u_f to precision u^2 $\Rightarrow$ Huge memory consumption increase if we keep a full copy of the factors.

Other issue: Do we need to run the other GMRES operations in precision u?

$\Rightarrow$ Can we relax the precision u^2 on the preconditioning and u on the rest of the operations?

LU-GMRES-IR5

Algorithm LU-GMRES-IR3

1: Compute the LU factorization $A = LU$ (u_f)
2: Solve $Ax_0 = b$ (u_f)
3: **while not** converged **do**
4: Compute $r_i = b - Ax_i$ (u_r)
5: Solve $\widetilde{A}d_i = \hat{U}^{-1}\hat{L}^{-1}Ad_i = \hat{U}^{-1}\hat{L}^{-1}r_i$ by GMRES at precision $(u\ )$ with matrix vector products with $\widetilde{A}$ at precision (u^2).
6: Compute $x_{i+1} = x_i + d_i$ (u)
7: **end while**

LU-GMRES-IR5

Algorithm LU-GMRES-IR3

1: Compute the LU factorization $A = LU$ (u_f)
2: Solve $Ax_0 = b$ (u_f)
3: **while not** converged **do**
4: Compute $r_i = b - Ax_i$ (u_r)
5: Solve $\widetilde{A}d_i = \hat{U}^{-1}\hat{L}^{-1}Ad_i = \hat{U}^{-1}\hat{L}^{-1}r_i$ by GMRES at precision $(u\ \ )$ with matrix vector products with $\widetilde{A}$ at precision (u^2).
6: Compute $x_{i+1} = x_i + d_i$ (u)
7: **end while**

LU-GMRES-IR5

Algorithm LU-GMRES-IR5

1: Compute the LU factorization $A = LU$ (u_f)
2: Solve $Ax_0 = b$ (u_f)
3: **while not** converged **do**
4: Compute $r_i = b - Ax_i$ (u_r)
5: Solve $\widetilde{A}d_i = \hat{U}^{-1}\hat{L}^{-1}Ad_i = \hat{U}^{-1}\hat{L}^{-1}r_i$ by GMRES at precision (u_g) with matrix vector products with $\widetilde{A}$ at precision (u_p).
6: Compute $x_{i+1} = x_i + d_i$ (u)
7: **end while**

- $\mathbf{u_p}$: precision at which we apply the **preconditioned** matrix-vector products.
- $\mathbf{u_g}$: precision at which we apply the other **GMRES** operations.

Remark: Possibly $u_p > u^2$ (and $u_g > u$).

A key result in the error analysis

Theorem (Stability of preconditioned MGS-GMRES in 2 precisions)

Consider solving a preconditioned linear system

$$\widetilde{A}d = s, \quad \widetilde{A} = \hat{U}^{-1}\hat{L}^{-1}A, \quad A \in \mathbb{R}^{n\times n},$$

with a MGS-GMRES in precision u_g except for the products with $\widetilde{A}$ applied in precision u_p.

The computed solution $\hat{d}$ achieves a backward error of order

$$u_g + u_p\kappa(A)$$

$\Rightarrow$ It generalizes the **backward stability** of **MGS-GMRES**[2] to a preconditioned MGS-GMRES in **2 precisions**.

[2] Paige, Rozložník and Strakoš, *Modified Gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES*, 2006

Convergence condition of LU-GMRES-IR5

	Convergence condition	Forward error
LU-IR3	$\kappa(A)u_f \ll 1$	$u_r\kappa(A) + u$
LU-GMRES-IR5	$(u_g + u_p\kappa(A))\kappa(A)^2u_f^2 \ll 1$	$u_r\kappa(A) + u$
LU-GMRES-IR3	$\kappa(A)u^{1/2}u_f \ll 1$	$u_r\kappa(A) + u$

If $u_f =$ fp16, the condition on LU-IR3 is 2×10^3, on LU-GMRES-IR3 it is 2×10^{11}, and on LU-GMRES-IR5 with $u_g = u_p =$ fp64 it is 3×10^7.

Meaningful combinations

With six arithmetics (fp8, bfloat16, fp16, fp32, fp64, fp128), LU-GMRES-IR5 can be declined in over **15000 different combinations**!

They are not all relevant.

Filter principle: Useless to have high precision when we can use low precision without impacting the numerical properties.

Filtering rules

- $u^2 \leq u_r \leq u \leq u_f$
- $u_p \leq u_g$
- $u_p < u_f$
- $u_p < u$, $u_p = u$, and $u_p > u$
- $u_g = u$ and $u_g > u$
- $u_g < u_f$, $u_g = u_f$, and $u_g > u_f$

Remark: These rules are based on the limiting accuracy of the forward error and the convergence condition formulas.

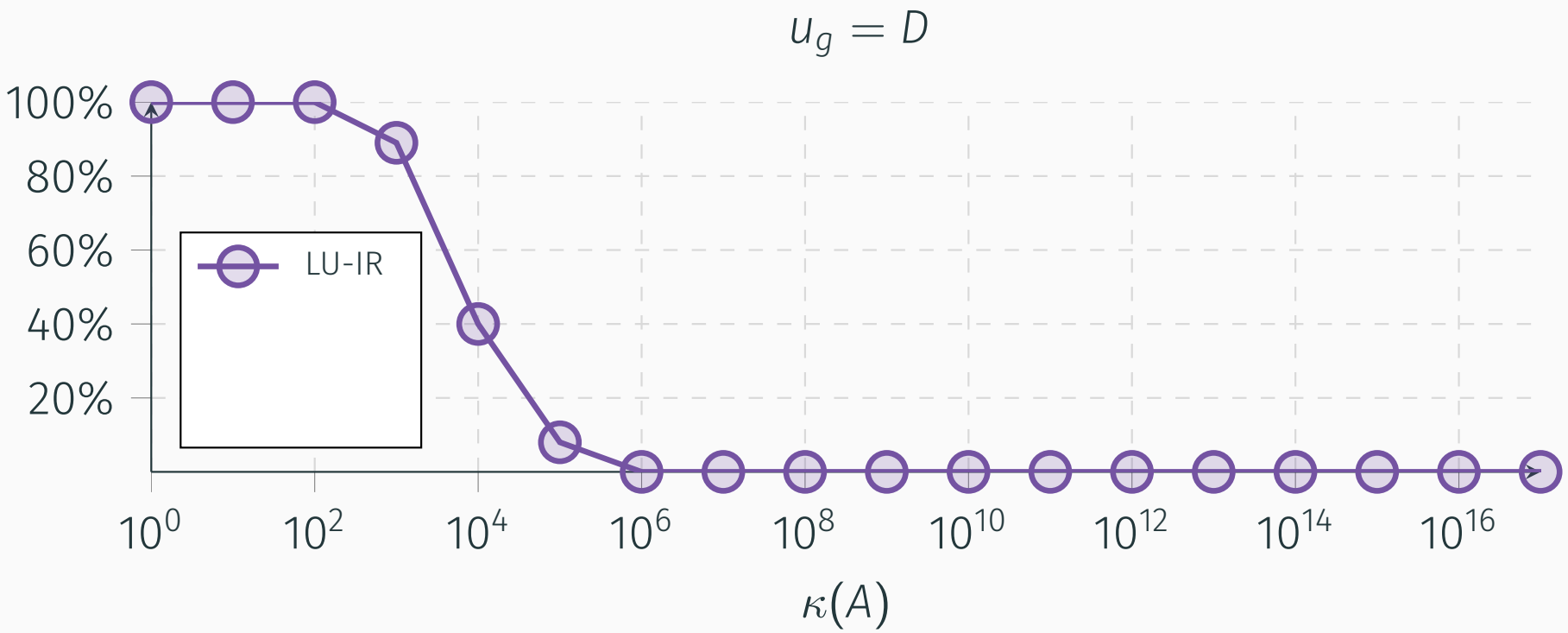
Theoretical robustness over $\kappa(A)$

u_g	u_p	Convergence Condition ($\max(\kappa(A))$)
LU-IR3		2×10^3
R	S	8×10^3
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
LU-GMRES-IR3		2×10^{11}

Meaningful combinations of LU-GMRES-IR5 for $u_f = H$ and $u = D$.

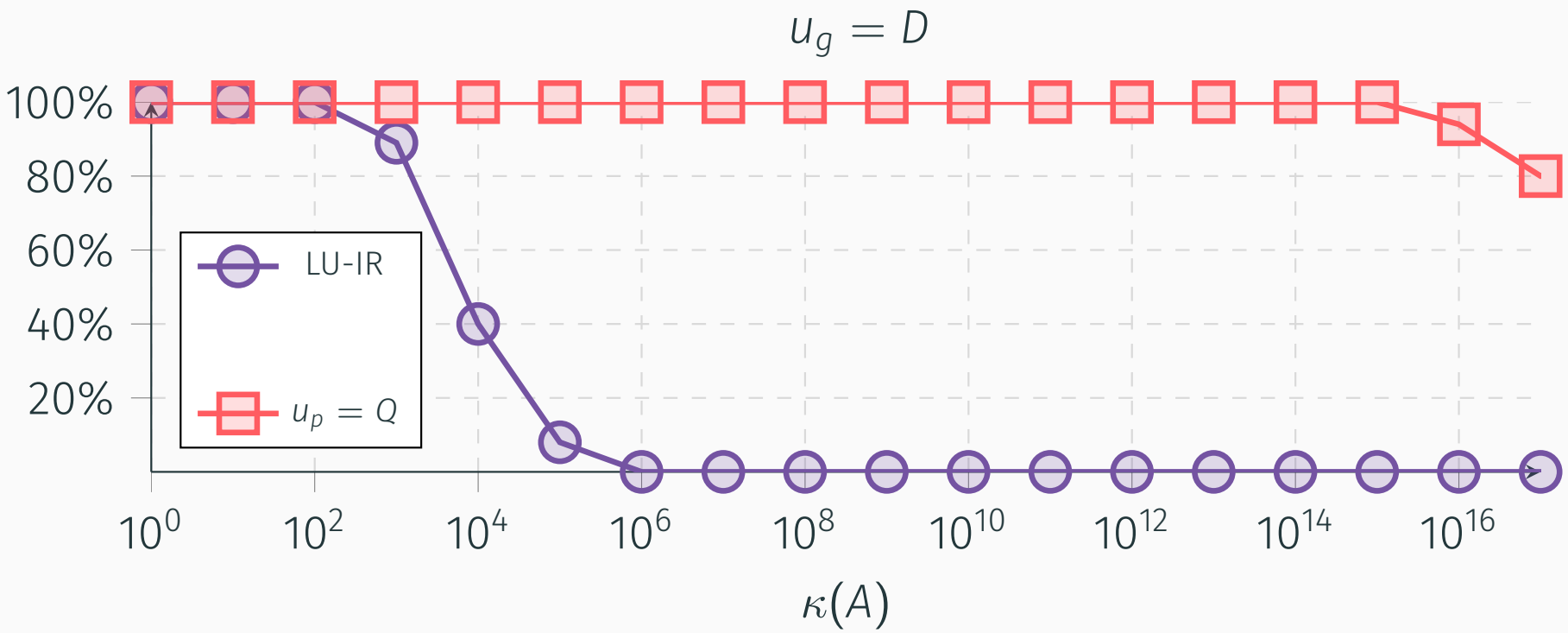
- LU-GMRES-IR5 is a **trade-off** between LU-IR3 and LU-GMRES-IR3.
- The more we **increase the precisions** u_g and u_p, the more **robust** we are.
- LU-GMRES-IR5 is **flexible** regarding the conditioning of the problems and the choice of precisions.

Experimental robustness over $\kappa(A)$



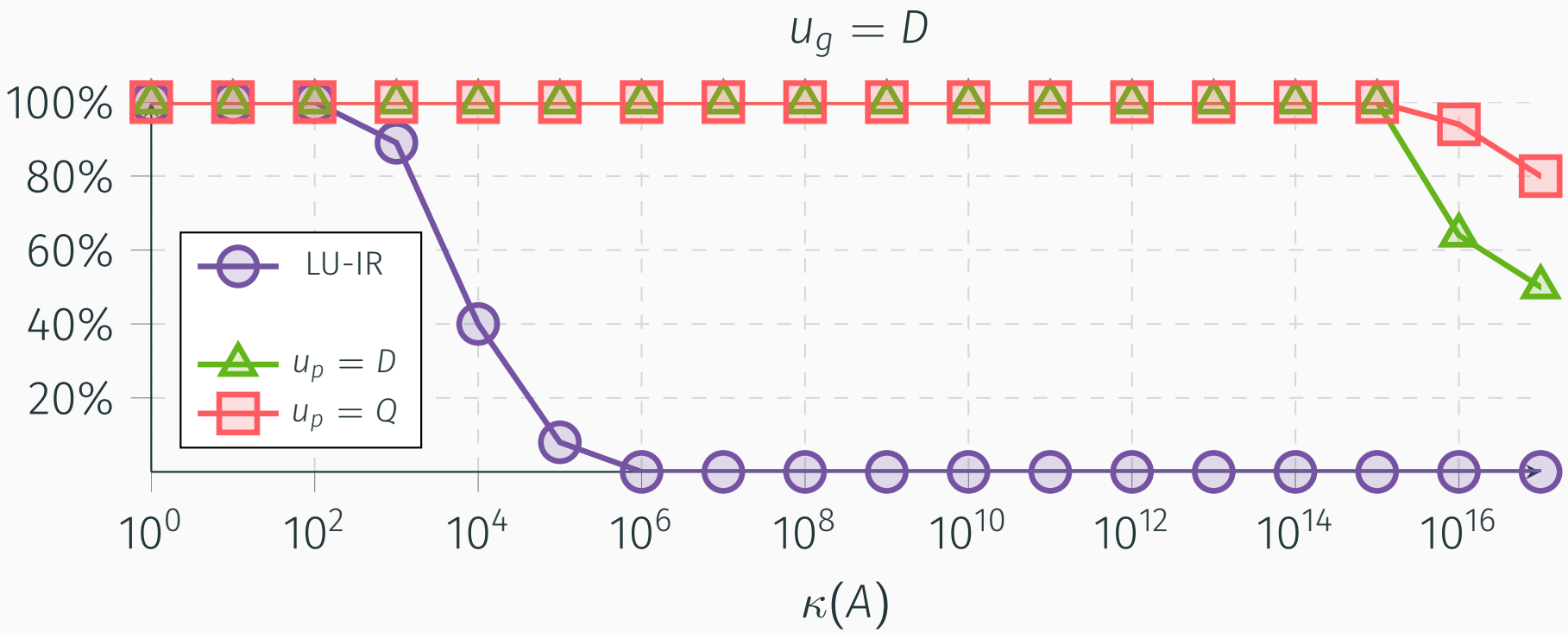
Percentage of convergence according to the condition number $\kappa(A)$. We fix $u_f =$ fp16, $u =$ fp64, and $u_r =$ fp128. The matrices are randomly generated with `randsvd(mode=2)`. The percentage for each $\kappa(A)$ is computed from 100 matrices.

Experimental robustness over $\kappa(A)$



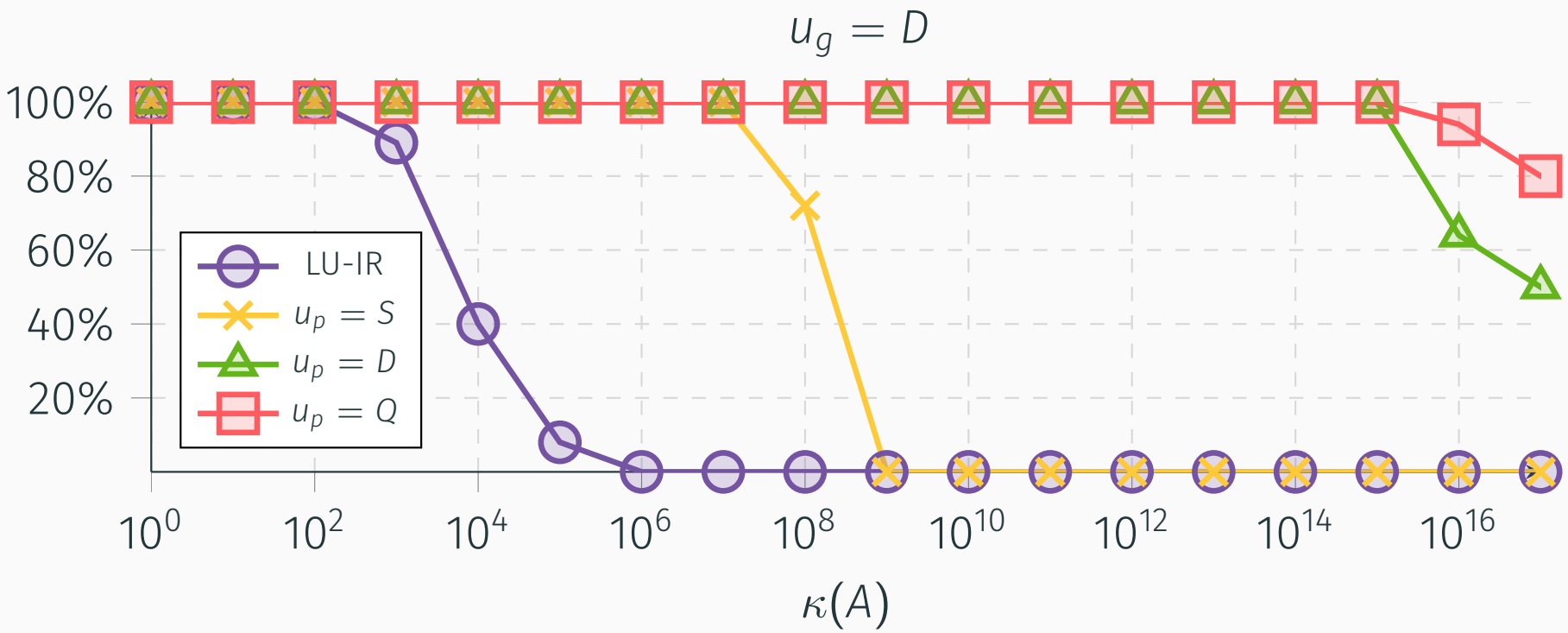
Percentage of convergence according to the condition number $\kappa(A)$. We fix $u_f = \text{fp16}$, $u = \text{fp64}$, and $u_r = \text{fp128}$. The matrices are randomly generated with `randsvd(mode=2)`. The percentage for each $\kappa(A)$ is computed from 100 matrices.

Experimental robustness over $\kappa(A)$



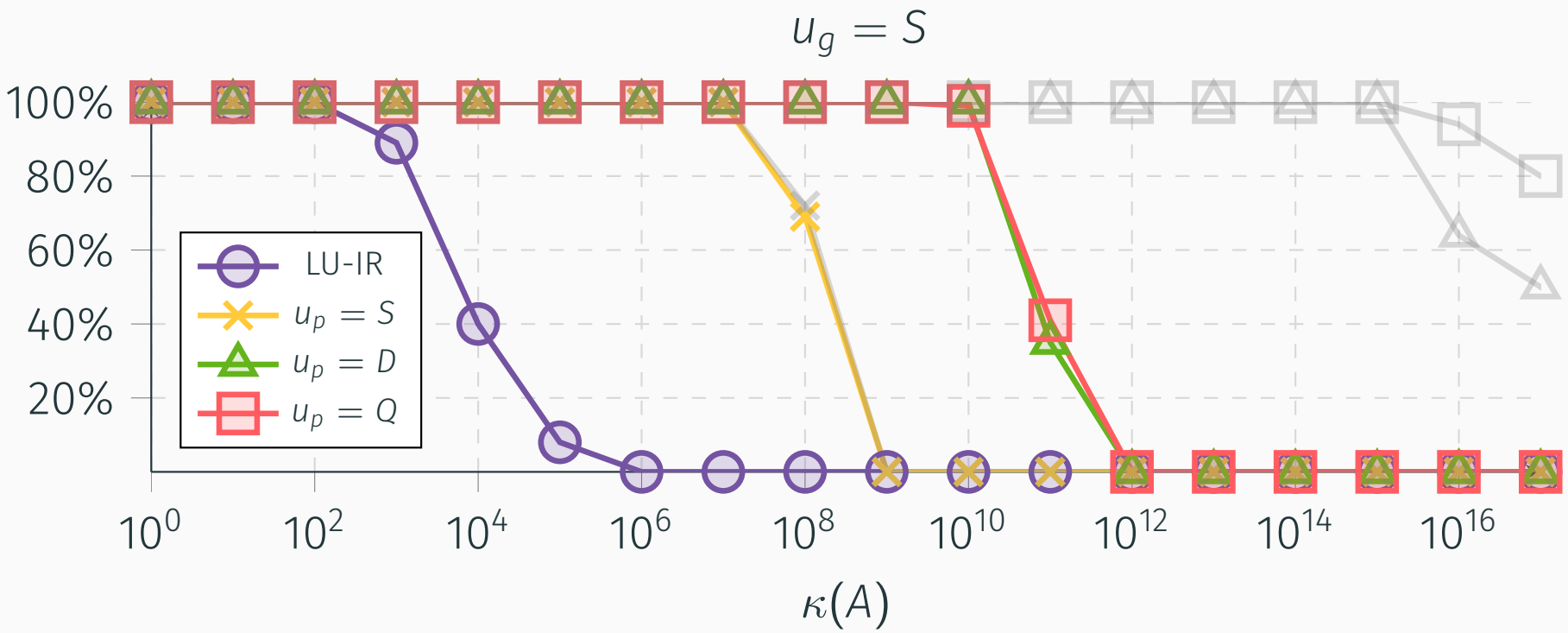
Percentage of convergence according to the condition number $\kappa(A)$. We fix $u_f = \text{fp16}$, $u = \text{fp64}$, and $u_r = \text{fp128}$. The matrices are randomly generated with `randsvd(mode=2)`. The percentage for each $\kappa(A)$ is computed from 100 matrices.

Experimental robustness over $\kappa(A)$



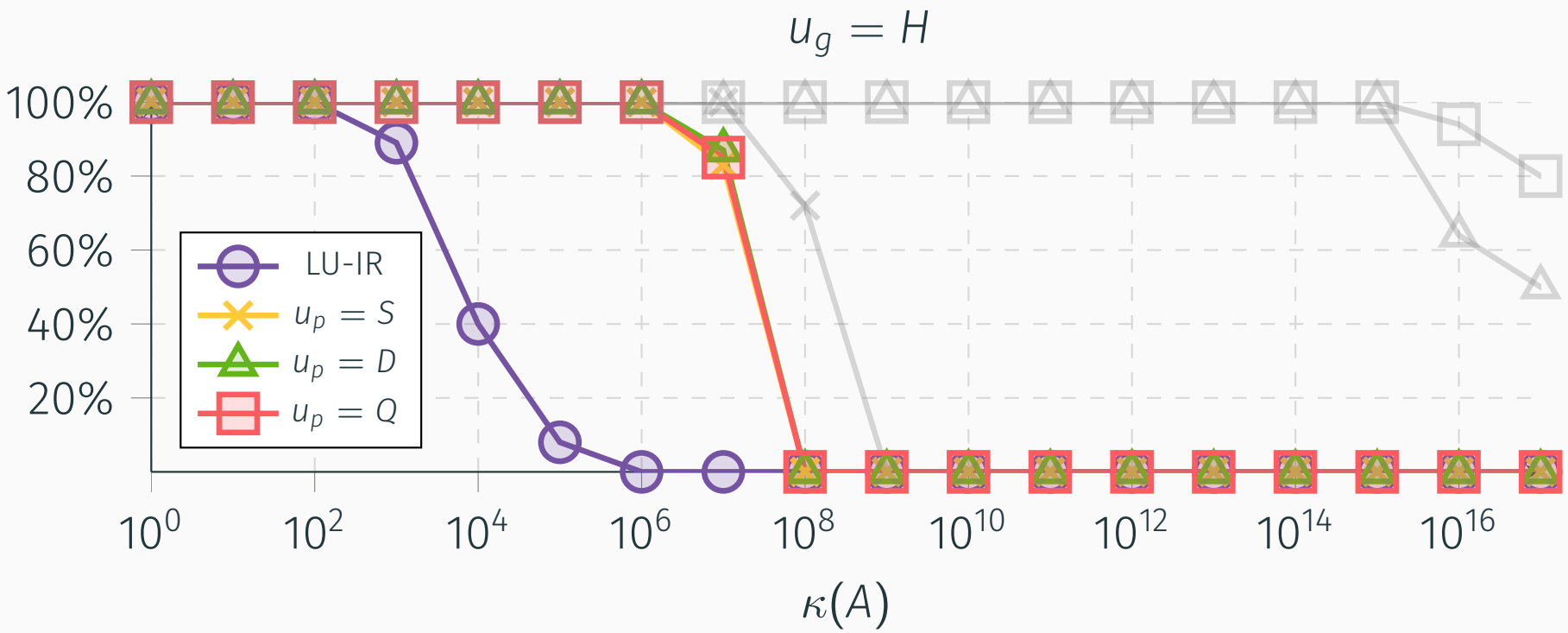
Percentage of convergence according to the condition number $\kappa(A)$. We fix $u_f = \text{fp16}$, $u = \text{fp64}$, and $u_r = \text{fp128}$. The matrices are randomly generated with `randsvd(mode=2)`. The percentage for each $\kappa(A)$ is computed from 100 matrices.

Experimental robustness over $\kappa(A)$



Percentage of convergence according to the condition number $\kappa(A)$. We fix $u_f = \text{fp16}$, $u = \text{fp64}$, and $u_r = \text{fp128}$. The matrices are randomly generated with `randsvd(mode=2)`. The percentage for each $\kappa(A)$ is computed from 100 matrices.

Experimental robustness over $\kappa(A)$



Percentage of convergence according to the condition number $\kappa(A)$. We fix $u_f =$ fp16, $u =$ fp64, and $u_r =$ fp128. The matrices are randomly generated with `randsvd(mode=2)`. The percentage for each $\kappa(A)$ is computed from 100 matrices.

Conclusion on LU-GMRES-IR5

Contributions

➤ **New algorithm:** LU-GMRES-IR5 which relaxes restrictive requirements on the precisions in LU-GMRES-IR3.

➤ **Error analysis:** New convergence condition for LU-GMRES-IR5 that demonstrates a high versatility regarding trade-offs between performance, problem difficulty, and hardware constraints.

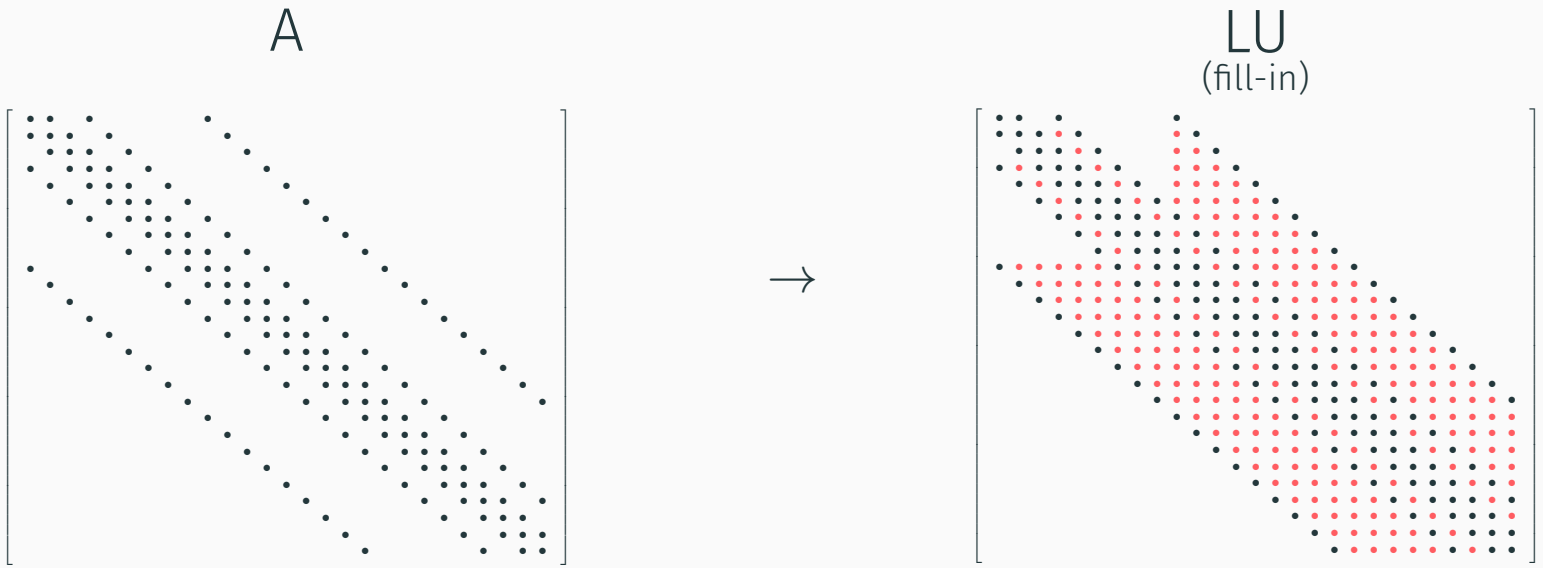
➤ **Numerical experiments:** Validate the theoretical convergence condition.

Amestoy, Buttari, Higham, L'Excellent, Mary, Vieublé. " *Five-Precision GMRES-based iterative refinement*". In: Submitted to a journal, preprint available on HAL (ID: hal-03190686).

Next: LU-GMRES-IR5 is better suited for the solution of large sparse problems since $u_p \geq u^2$. BUT we still need to consider the combined use of state-of-the-art iterative refinements with state-of-the-art sparse factorizations.

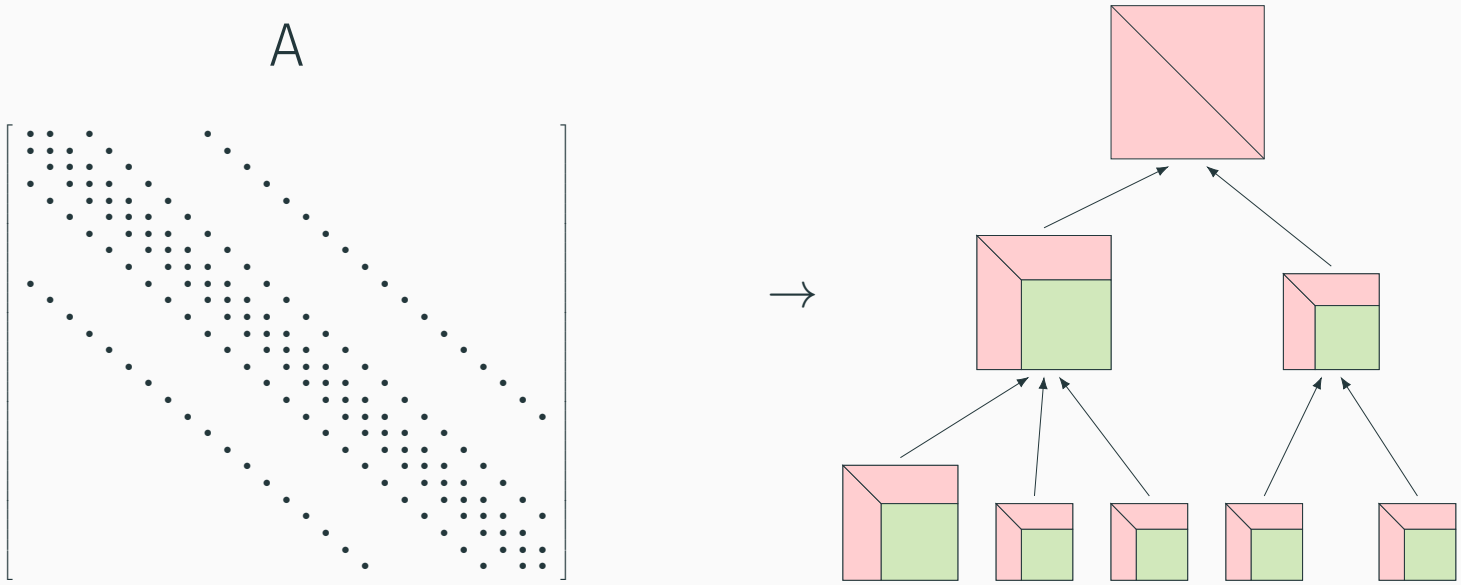
Error and performance analysis of LU-IR3 and LU-GMRES-IR5 on sparse systems with numerical approximations

LU Sparse Direct Factorization: Fill-in



Multifrontal LU Sparse Direct Factorization

A multifrontal sparse factorization can be decomposed into a series of factorizations of dense matrices whose dependencies are represented by an assembly tree:



➤ The red parts are the LU entries, the green part are temporary data.

➤ In multifrontal factorization the total memory consumption is higher than the factors in memory. The difference is called the active memory overhead.

Specific features of sparse iterative refinement

Algorithm Iterative refinement: complexities Dense VS Sparse

1:	Compute the LU factorization $A = LU$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	(u_f)
2:	Solve $Ax_0 = b$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{4/3})$	(u_f)
3:	**while not** converged **do**			
4:	Compute $r_i = b - Ax_i$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	(u_r)
5:	Solve $Ad_i = r_i$.	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{4/3})$	(u_s)
6:	Compute $x_{i+1} = x_i + d_i$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	(u)
7:	**end while**			

Fill-in in sparse direct solvers, i.e. $\mathrm{NNZ}(A) \ll \mathrm{NNZ}(LU)$!

Specific features of sparse iterative refinement

Algorithm Iterative refinement: complexities Dense VS Sparse

1:	Compute the LU factorization $A = LU$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	(u_f)
2:	Solve $Ax_0 = b$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{4/3})$	(u_f)
3:	**while not** converged **do**			
4:	Compute $r_i = b - Ax_i$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	(u_r)
5:	Solve $Ad_i = r_i$.	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{4/3})$	(u_s)
6:	Compute $x_{i+1} = x_i + d_i$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	(u)
7:	**end while**			

Fill-in in sparse direct solvers, i.e. $\mathrm{NNZ}(A) \ll \mathrm{NNZ}(LU)$!

➤ SpMV much cheaper than solve $\Rightarrow$ $u_r \ll u$ has limited impact on performance (even for $u_r =$ fp128).

Specific features of sparse iterative refinement

Algorithm Iterative refinement: complexities Dense VS Sparse

1:	Compute the LU factorization $A = LU$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	(u_f)
2:	Solve $Ax_0 = b$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{4/3})$	(u_f)
3:	**while not** converged **do**			
4:	Compute $r_i = b - Ax_i$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	(u_r)
5:	Solve $Ad_i = r_i$.	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{4/3})$	(u_s)
6:	Compute $x_{i+1} = x_i + d_i$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	(u)
7:	**end while**			

Fill-in in sparse direct solvers, i.e. $\mathrm{NNZ}(A) \ll \mathrm{NNZ}(LU)$!

➤ Memory space of A in u_r ($\mathcal{O}(n)$ entries) negligible compared with the LU factors in u_f ($\mathcal{O}(n^{4/3})$ entries) $\Rightarrow$ LU-IR3 saves memory over a direct solver in u!

Specific features of sparse iterative refinement

Algorithm Iterative refinement: complexities Dense VS Sparse

1:	Compute the LU factorization $A = LU$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	(u_f)
2:	Solve $Ax_0 = b$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{4/3})$	(u_f)
3:	**while not** converged **do**			
4:	Compute $r_i = b - Ax_i$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	(u_r)
5:	Solve $Ad_i = r_i$.	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{4/3})$	(u_s)
6:	Compute $x_{i+1} = x_i + d_i$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	(u)
7:	**end while**			

Fill-in in sparse direct solvers, i.e. $\mathrm{NNZ}(A) \ll \mathrm{NNZ}(LU)$!

➤ **(Multifrontal only)** Even if LU-GMRES-IR5 fully stores the factors in $u_p = u$, it does not need to store the active memory in $u_p = u$
$\Rightarrow$ LU-GMRES-IR5 can save memory over a direct solver in u.

Specific features of approximate factorization

Algorithm LU-IR3: complexities Sparse VS Approximations

1:	Compute the LU factorization $A = \hat{L}\hat{U}$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^\alpha)$	$(?)$
2:	Solve $Ax_0 = b$	$\mathcal{O}(n^{4/3})$	$\mathcal{O}(n^\beta)$	$(?)$
3:	**while not** converged **do**			
4:	Compute $r_i = b - Ax_i$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	(u_r)
5:	Solve $Ad_i = r_i$ by $d_i = \hat{U}^{-1}\hat{L}^{-1}r_i$.	$\mathcal{O}(n^{4/3})$	$\mathcal{O}(n^\beta)$	$(?)$
6:	Compute $x_{i+1} = x_i + d_i$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	(u)
7:	**end while**			

- Where $2 \geq \alpha$ and $4/3 \geq \beta$.
- **Question:** What can be said on the accuracy of the factorization and the solve?

Error analysis with numerical approximations

Two main changes on the accuracies compared with classic LU with partial pivoting:

➤ Need to handle numerical approximations $\Rightarrow$ We consider a generic model of numerical approximations introducing a perturbation ϵ. It includes BLR and static pivoting.

➤ We must take into account the growth factor ρ_n ($\approx$ difference of scale between the entries of A and its factors L and U) which might not be negligible without stable pivoting strategy.

Error analysis with numerical approximations

Two main changes on the accuracies compared with classic LU with partial pivoting:

➤ Need to handle numerical approximations ⇒ We consider a **generic model of numerical approximations** introducing a perturbation ϵ. It includes BLR and static pivoting.

➤ We must take into account the **growth factor** ρ_n ($\approx$ difference of scale between the entries of A and its factors L and U) which might **not be negligible** without stable pivoting strategy.

Theorem (Convergence conditions)

Let $Ax = b$ be solved by LU-IR3 or GMRES-IR5 using an approximate LU factorization. Then the forward error will converge provided that

$$(u_f \rho_n + \epsilon)\kappa(A) \ll 1 \qquad (LU - IR3)$$

$$(u_g + u_p \rho_n \kappa(A))(u_f \rho_n + \epsilon)^2 \kappa(A)^2 \ll 1 \qquad (LU - GMRES - IR5)$$

Implemented parallel methods

Solver	u_f	u	u_r	u_g	u_p	$\max(\kappa(A))$ $(\epsilon = 0)$	forward error
DMUMPS	fp64 LU direct solver					—	$\kappa(A) \times 10^{-16}$
LU-IR	S	D	D	—	—	2×10^{7}	$\kappa(A) \times 10^{-16}$
LU-GMRES-IR	S	D	D	D	D	1×10^{10}	$\kappa(A) \times 10^{-16}$

➤ LU-IR and LU-GMRES-IR use single precision (fp32) factorization, BLR, and static pivoting to save resources.

➤ We use a multifrontal sparse solver. While we expect our conclusions on the execution time to hold for all direct sparse solvers. Our conclusions on the memory consumption related to the active memory are specific to the multifrontal solvers.

Implementation details and design choices

➤ We use the MUMPS multifrontal sparse solvers for factorization and solve. MUMPS supports BLR, static pivoting, and threshold partial pivoting.

Implementation details and design choices

➤ We use the MUMPS multifrontal sparse solvers for factorization and solve. MUMPS supports BLR, static pivoting, and threshold partial pivoting.

➤ The default pivoting strategy is not partial pivoting, but is threshold partial pivoting.

Implementation details and design choices

➤ We use the MUMPS multifrontal sparse solvers for factorization and solve. MUMPS supports BLR, static pivoting, and threshold partial pivoting.

➤ The default pivoting strategy is not partial pivoting, but is threshold partial pivoting.

➤ For LU-GMRES-IR: we cast in-place the factors fully from fp32 to fp64. The active memory is not cast and is overwritten by the factors in fp64!

Implementation details and design choices

➤ We use the MUMPS multifrontal sparse solvers for factorization and solve. MUMPS supports BLR, static pivoting, and threshold partial pivoting.

➤ The default pivoting strategy is not partial pivoting, but is threshold partial pivoting.

➤ For LU-GMRES-IR: we cast in-place the factors fully from fp32 to fp64. The active memory is not cast and is overwritten by the factors in fp64!

➤ In-house GMRES implementation and SpMV kernel running in parallel on the master MPI process.

Implementation details and design choices

➤ We use the MUMPS multifrontal sparse solvers for factorization and solve. MUMPS supports BLR, static pivoting, and threshold partial pivoting.

➤ The default pivoting strategy is not partial pivoting, but is threshold partial pivoting.

➤ For LU-GMRES-IR: we cast in-place the factors fully from fp32 to fp64. The active memory is not cast and is overwritten by the factors in fp64!

➤ In-house GMRES implementation and SpMV kernel running in parallel on the master MPI process.

➤ The MUMPS factorization and solve are distributed over the MPI processes.

Matrix set

Name	N	NNZ	Arith.	Sym.	$\kappa(A)$	Fact. (flops)	Slv. (flops)
ElectroPhys10M	1.02E+07	1.41E+08	R	1	1.10E+01	4E+14	9E+10
DrivAer6M	6.11E+06	4.97E+07	R	1	9.40E+05	6E+13	3E+10
Queen_4147	4.14E+06	3.28E+08	R	1	4.30E+06	3E+14	6E+10
tminlet3M	2.84E+06	1.62E+08	C	0	2.70E+07	1E+14	2E+10
perf009ar	5.41E+06	2.08E+08	R	1	3.70E+08	2E+13	2E+10
elasticity-3d	5.18E+06	1.16E+08	R	1	3.60E+09	2E+14	5E+10
lfm_aug5M	5.52E+06	3.71E+07	C	1	5.80E+11	2E+14	5E+10
CarBody25M	2.44E+07	7.06E+08	R	1	8.60E+12	1E+13	3E+10
thmgas	5.53E+06	3.71E+07	R	0	8.28E+13	1E+14	4E+10

Set of **industrial** and SuiteSparse matrices.

➤ The matrices are **ordered in increasing $\kappa(A)$**, the higher $\kappa(A)$ is, the slower the convergence (if reached at all).

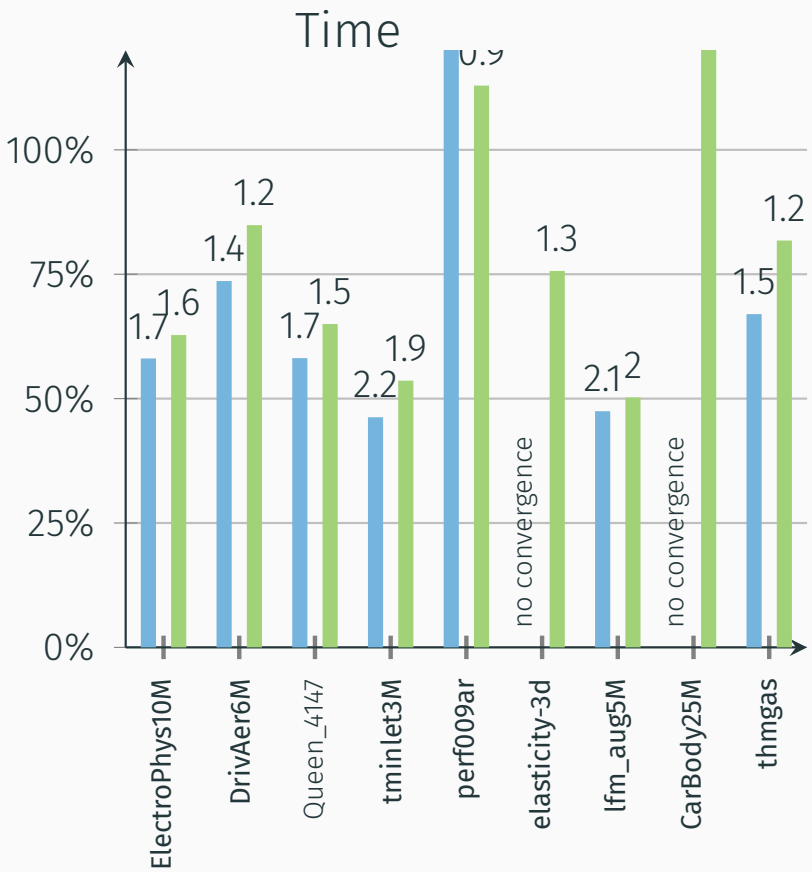
Matrix set

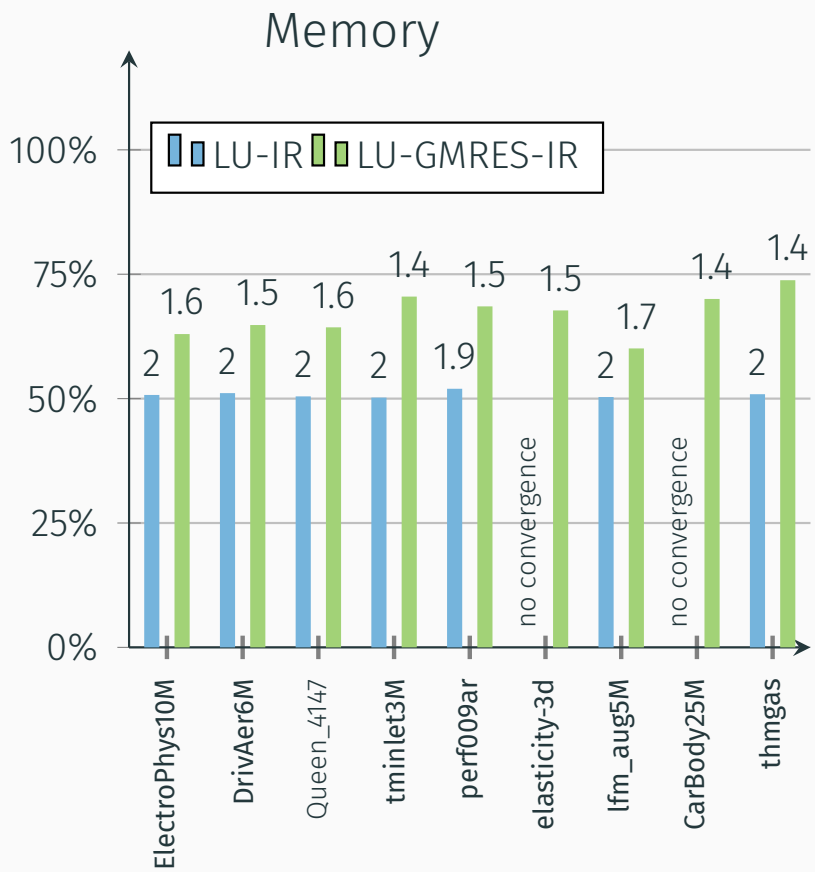
Name	N	NNZ	Arith.	Sym.	$\kappa(A)$	Fact. (flops)	Slv. (flops)
ElectroPhys10M	1.02E+07	1.41E+08	R	1	1.10E+01	4E+14	9E+10
DrivAer6M	6.11E+06	4.97E+07	R	1	9.40E+05	6E+13	3E+10
Queen_4147	4.14E+06	3.28E+08	R	1	4.30E+06	3E+14	6E+10
tminlet3M	2.84E+06	1.62E+08	C	0	2.70E+07	1E+14	2E+10
perf009ar	5.41E+06	2.08E+08	R	1	3.70E+08	2E+13	2E+10
elasticity-3d	5.18E+06	1.16E+08	R	1	3.60E+09	2E+14	5E+10
lfm_aug5M	5.52E+06	3.71E+07	C	1	5.80E+11	2E+14	5E+10
CarBody25M	2.44E+07	7.06E+08	R	1	8.60E+12	1E+13	3E+10
thmgas	5.53E+06	3.71E+07	R	0	8.28E+13	1E+14	4E+10

Set of **industrial** and SuiteSparse matrices.

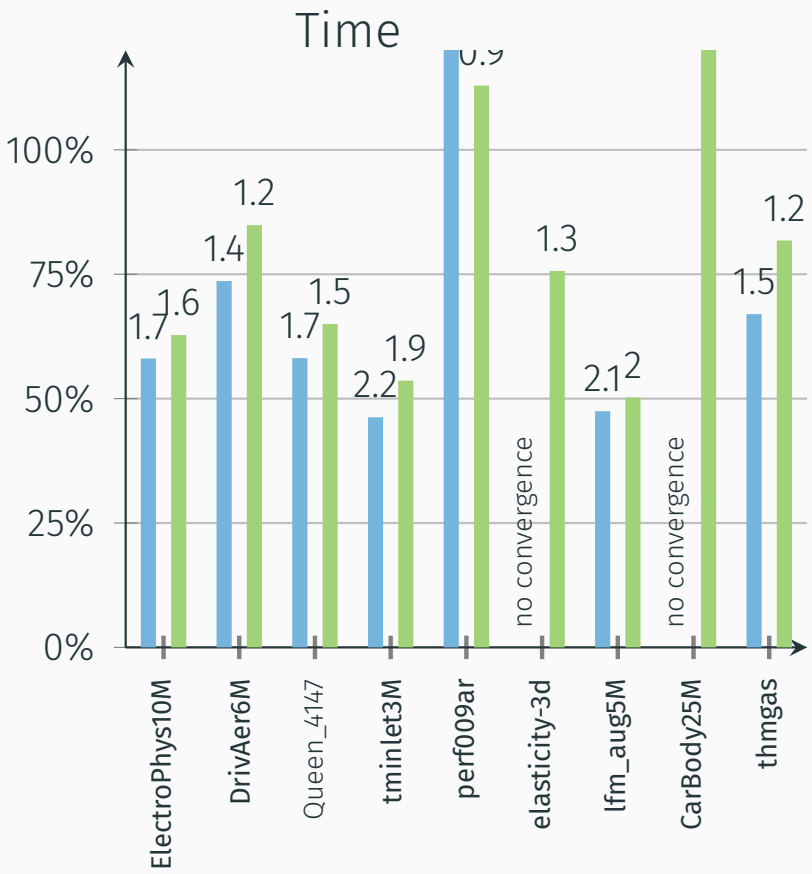
➤ We run on OLYMPE supercomputer nodes (two Intel 18-cores Skylake/node), 1 node (2MPI×18threads) or 2 nodes (4MPI×18threads) depending on the matrix size.

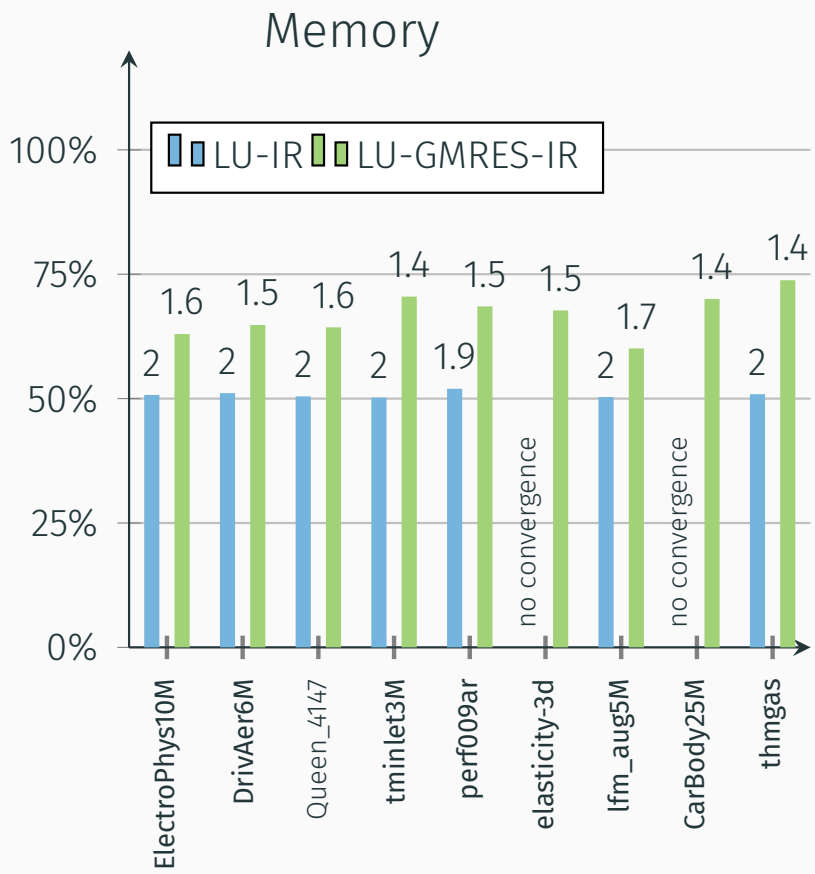
Time and memory performance w.r.t. DMUMPS





Time and memory performance w.r.t. DMUMPS

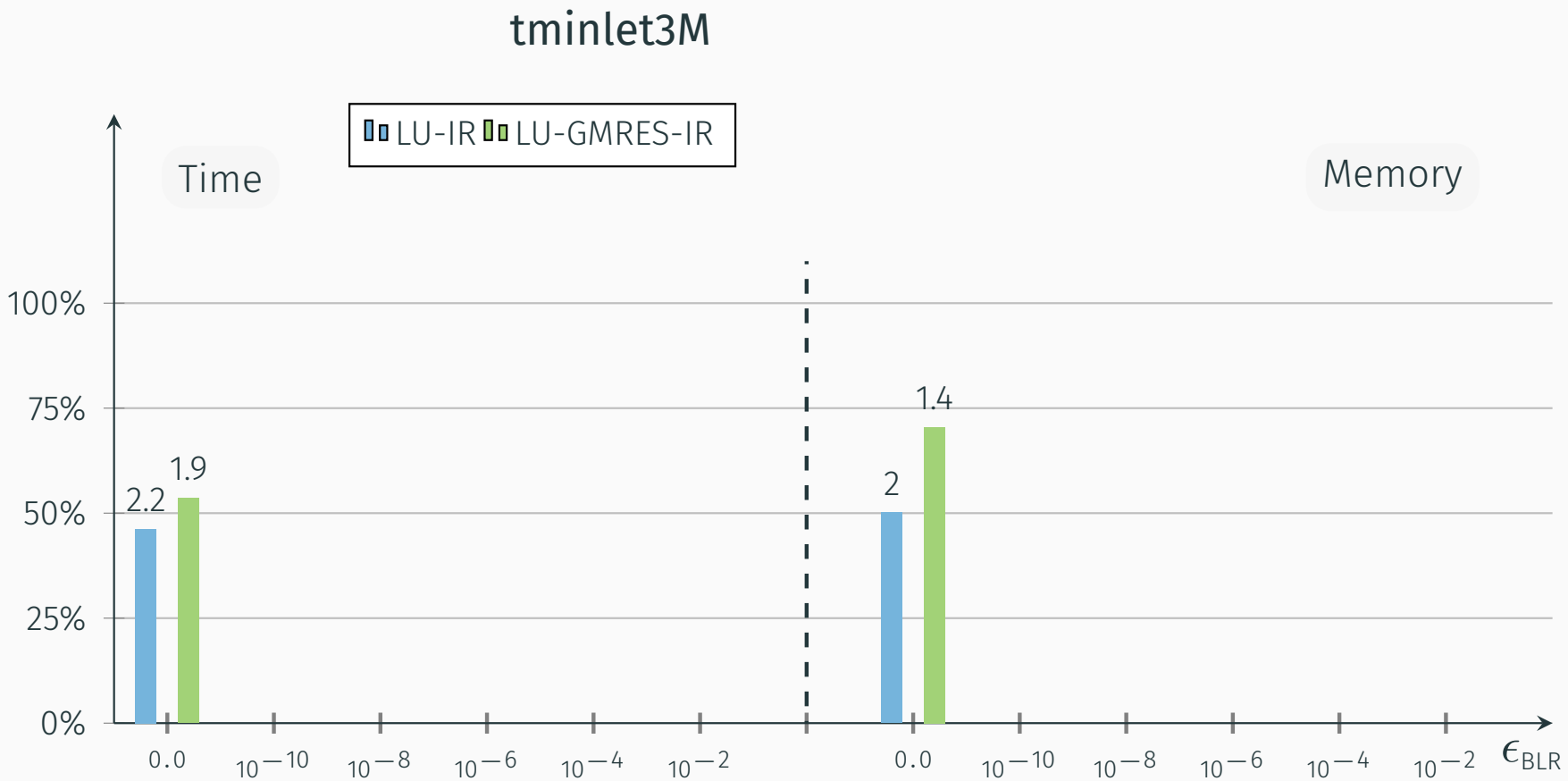




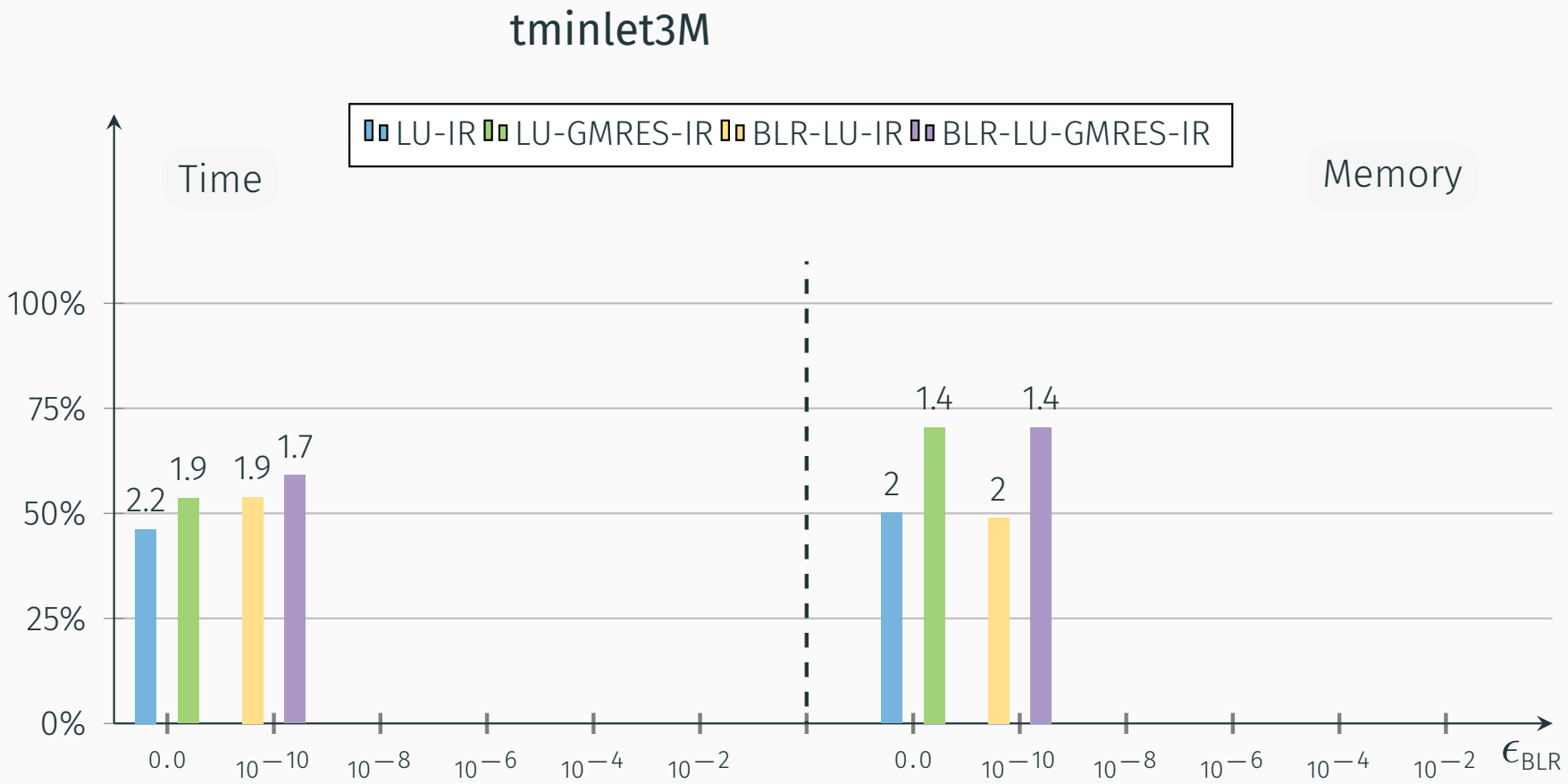
- LU-IR up to 2.2× faster.
- LU-GMRES-IR up to 1.9× faster. Slower than LU-IR, but more robust.

- LU-IR consumes 2× less memory.
- LU-GMRES-IR consumes at best 1.7× less despite factors in double ⇒ save active memory.

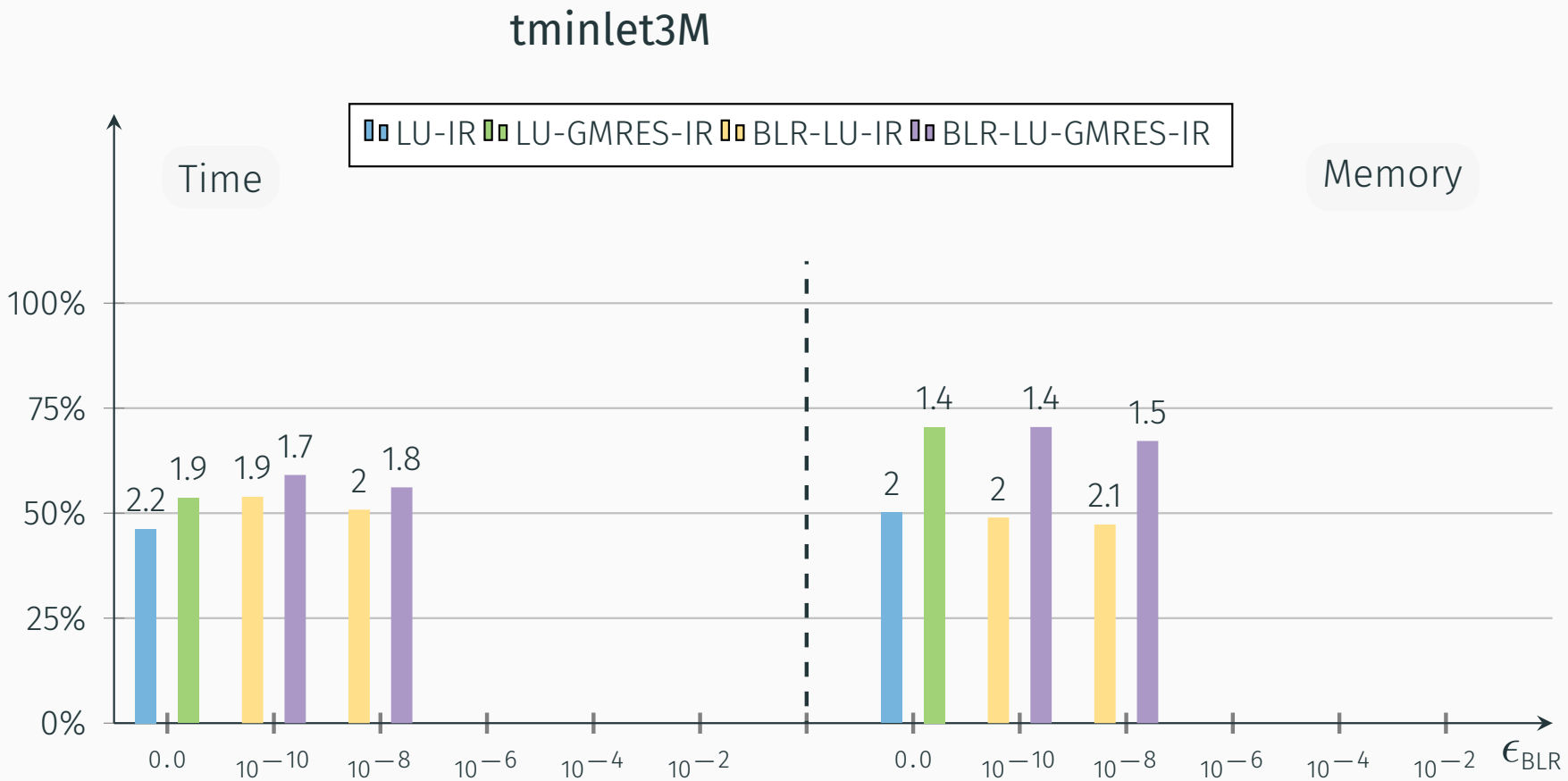
Time and memory performance with BLR w.r.t. DMUMPS



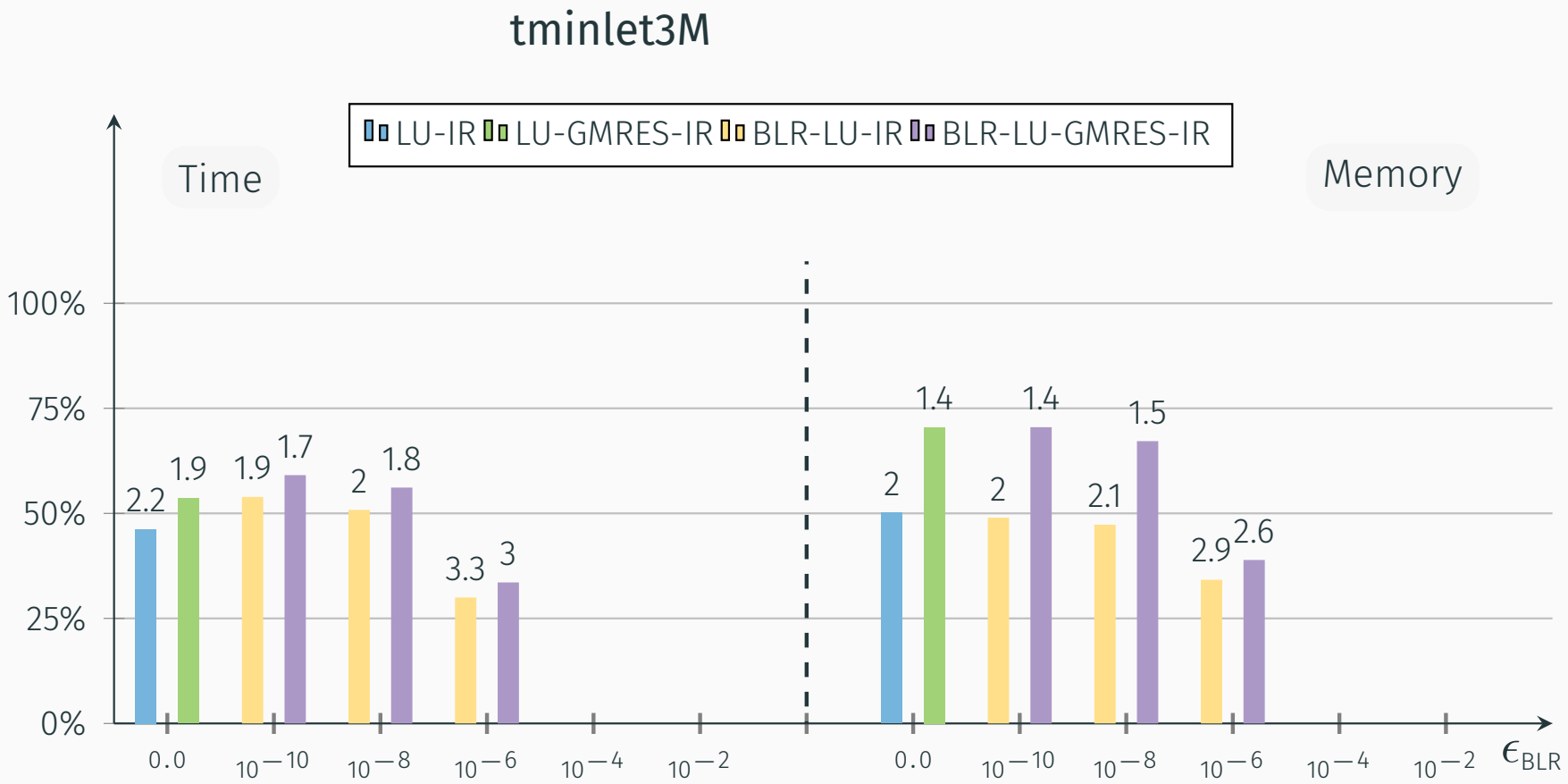
Time and memory performance with BLR w.r.t. DMUMPS



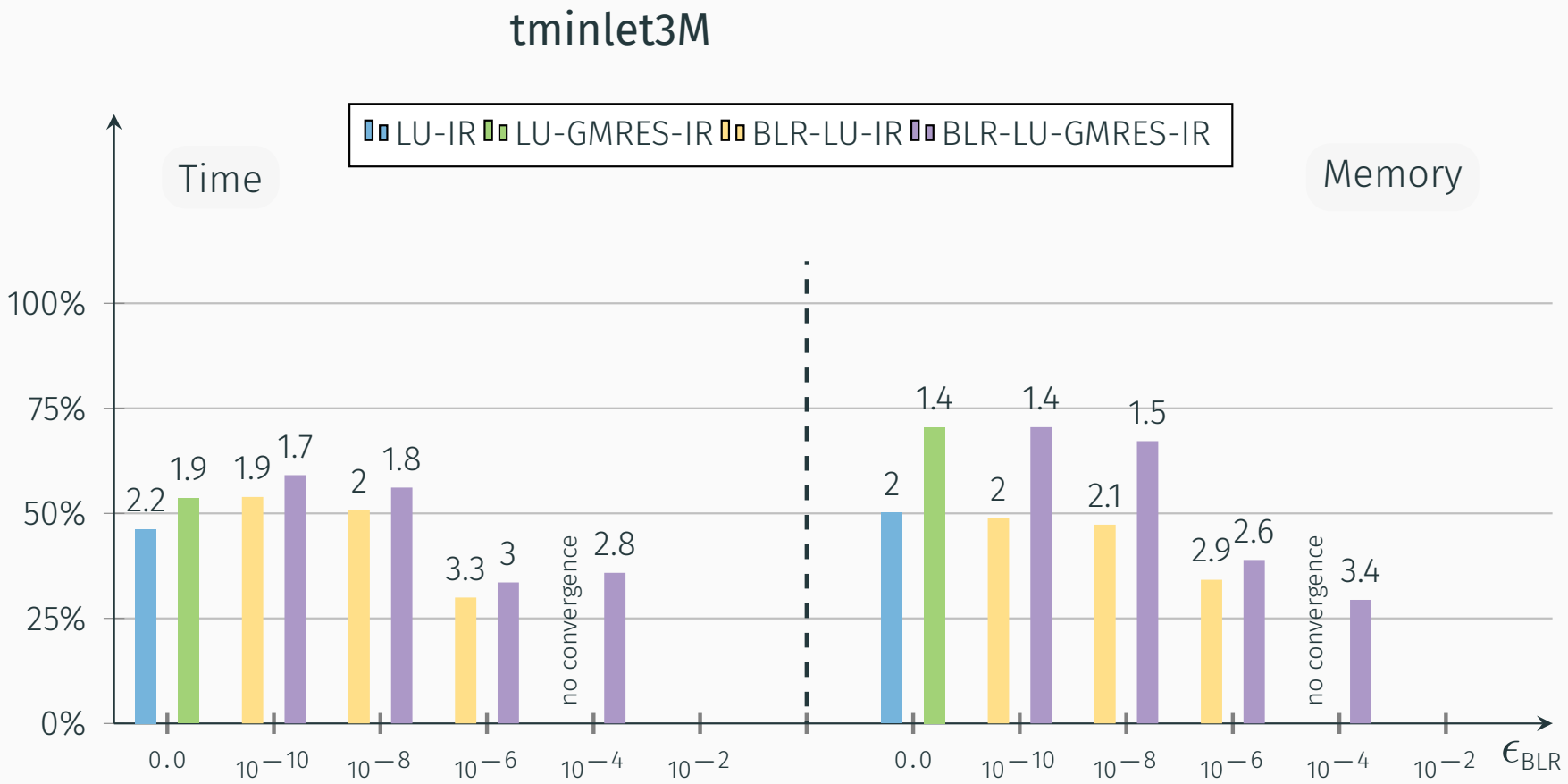
Time and memory performance with BLR w.r.t. DMUMPS



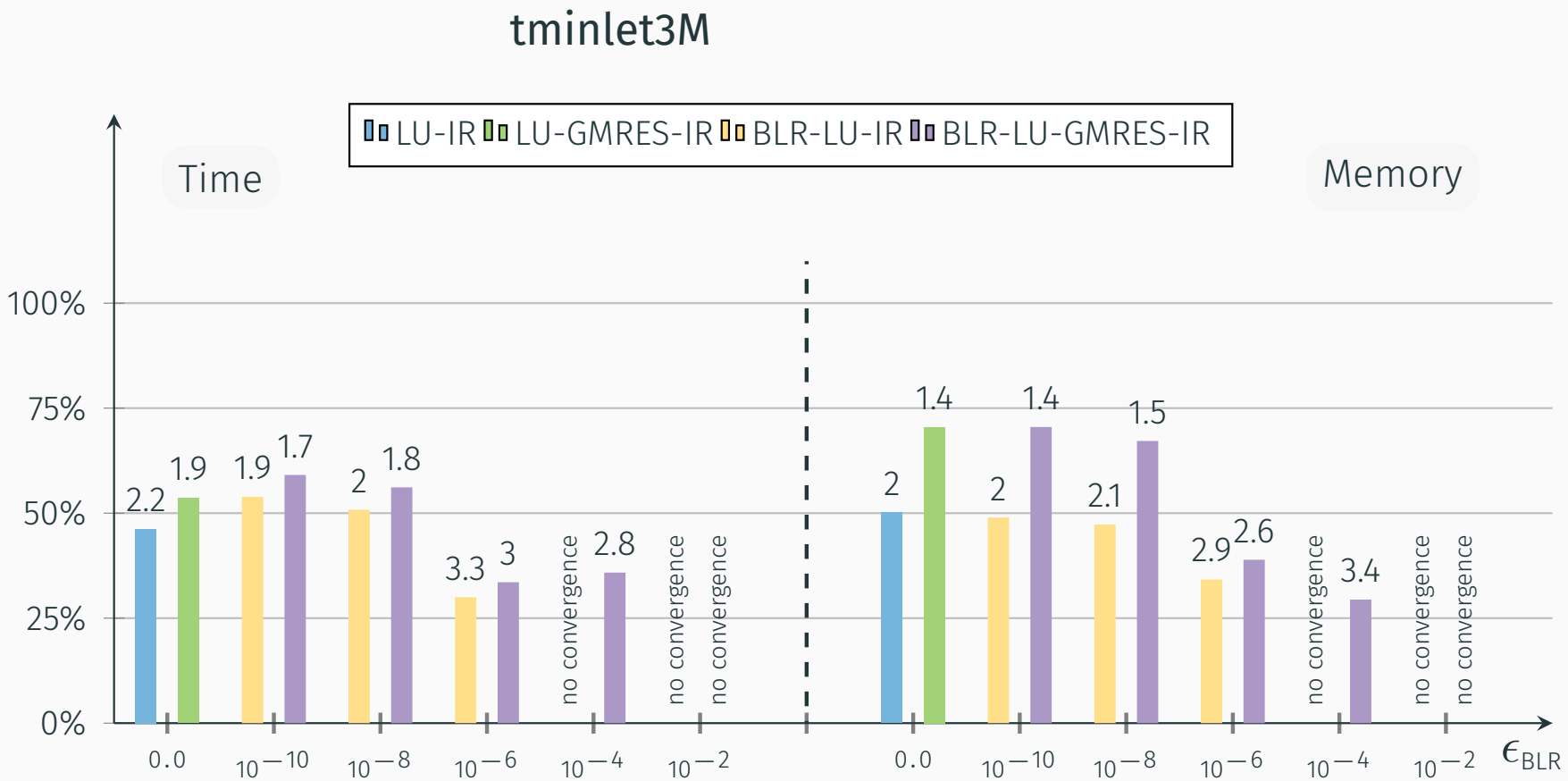
Time and memory performance with BLR w.r.t. DMUMPS



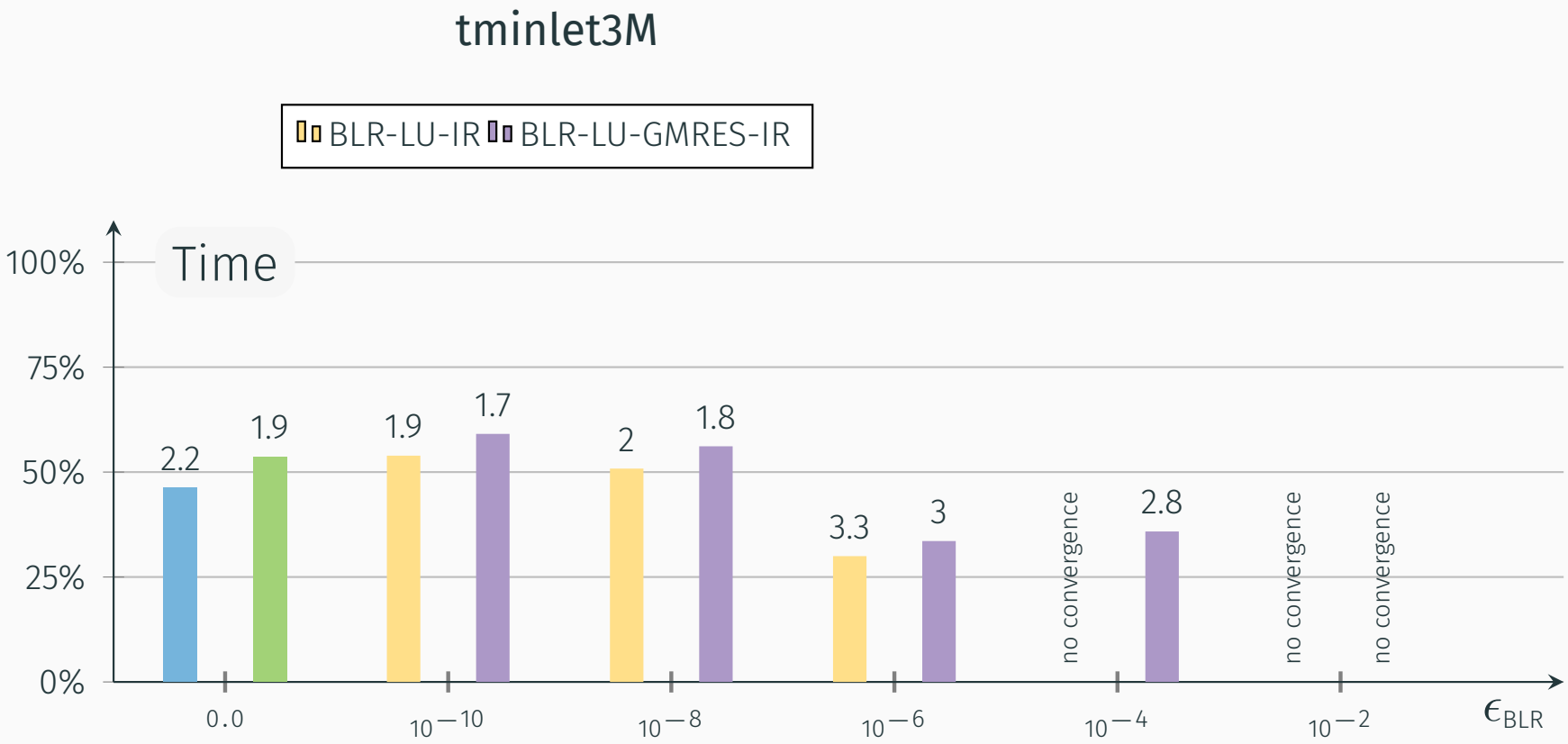
Time and memory performance with BLR w.r.t. DMUMPS



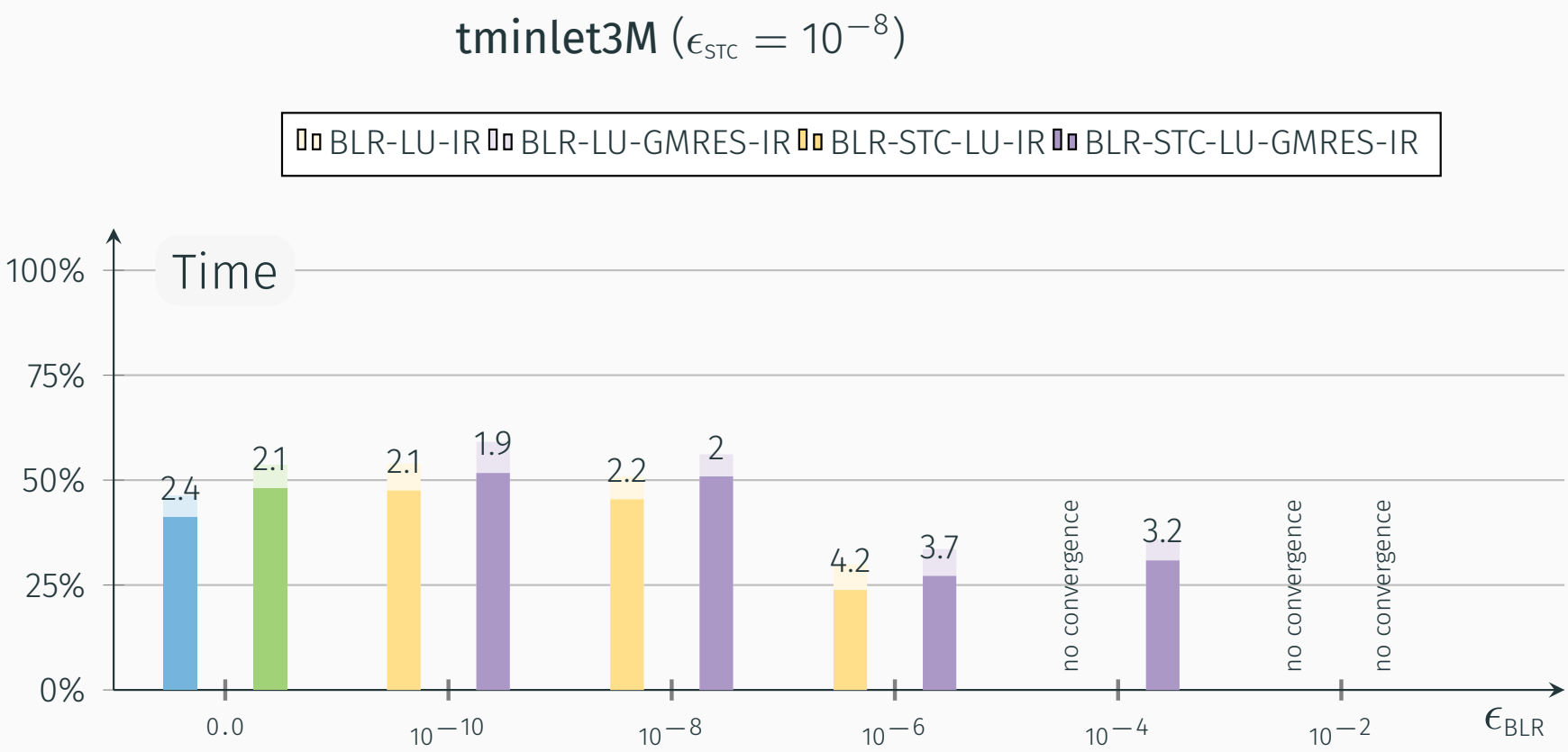
Time and memory performance with BLR w.r.t. DMUMPS



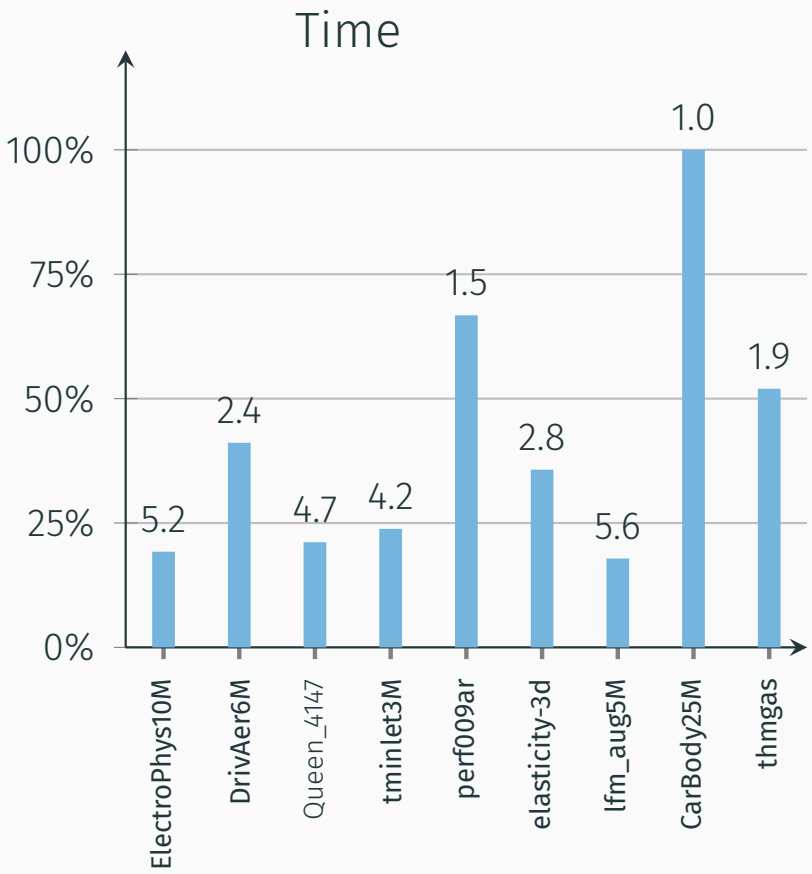
Time performance with BLR + static pivoting w.r.t. DMUMPS

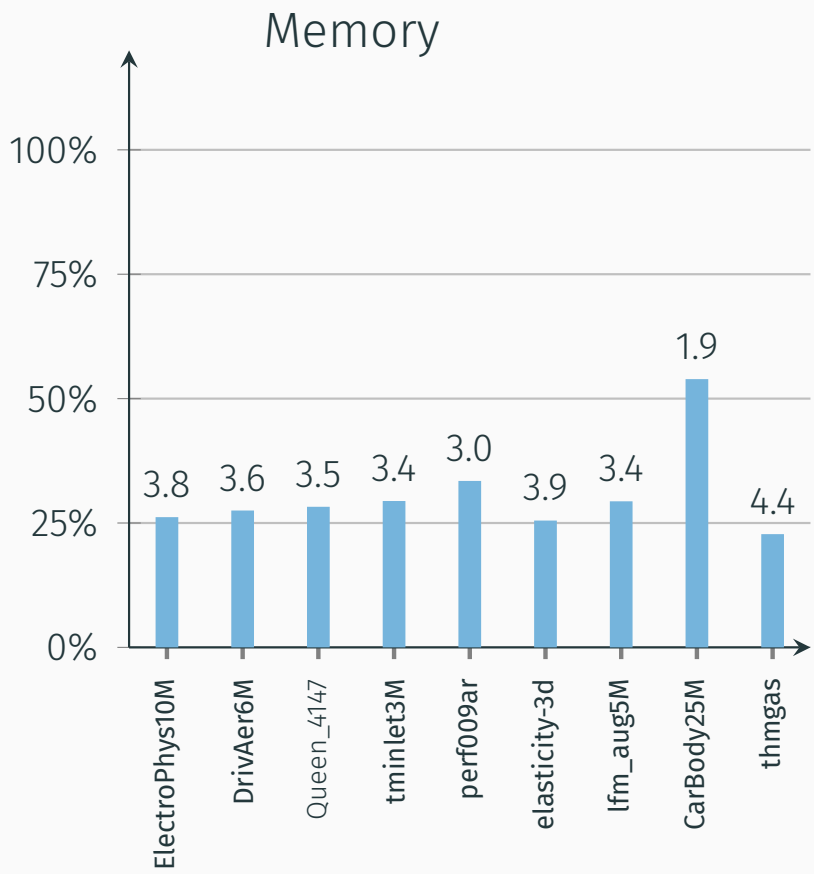


Time performance with BLR + static pivoting w.r.t. DMUMPS



Gather it all: Best time and memory w.r.t. DMUMPS





$\Rightarrow$ Up to 5.6$\times$ faster and 4.4$\times$ less memory with the same accuracy on the solution than DMUMPS!

Conclusion on sparse iterative refinement

Contributions

➤ **Error analysis:** New convergence conditions for LU-IR3 and LU-GMRES-IR5 taking into account numerical approximations used in LU direct solvers.

➤ **Performance analysis:** Demonstrate heavy resource savings while preserving the accuracy on sparse problems from a wide range of industrial and real-life applications.

Amestoy, Buttari, Higham, L'Excellent, Mary, Vieublé. " *Combining sparse approximate factorizations with mixed precision iterative refinement*". In: Accepted in ACM TOMS, preprint available on HAL (ID: hal-03536031).

Next: Until now, we showed that state-of-the-art iterative refinements could greatly improve sparse direct solvers. Can we improve sparse iterative solvers in the same way, in particular Krylov subspace based solvers?

Arbitrary preconditioned GMRES in mixed precision

Three criteria: Inner/Outer solvers

Algorithm: Refinement loop

1:
2: **repeat**
3: $x_{i+1} = \text{GMRES}(A, b, x_i, \tau)$
4: **until** convergence

1st criterion: Outer solver in high precision and inner solver in low precision ($u \ll u_i$).

➤ [Turner & Walker, 92]: outer iterative refinement (in precision u) with inner GMRES (in precision u_i).

➤ [Buttari et al., 08]: outer FGMRES (in precision u) with inner GMRES as a preconditioner (in precision u_i).

Algorithm: GMRES(A, b, x_0, τ)

Require: $A \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1:		
2:	$r_0 = b - Ax_0$	u
3:	$\beta = \|r_0\|,\ v_1 = r_0/\beta,\ k = 1$	u_i
4:	**repeat**	
5:	$w_k = Av_k$	u_i
6:		
7:	**for** $i = 1, \ldots, k$ **do**	
8:	$h_{i,k} = v_i^T w_k$	u_i
9:	$w_k = w_k - h_{i,k} v_i$	u_i
10:	**end for**	
11:	$h_{k+1,k} = \|w_k\|, v_{k+1} = w_k/h_{k+1,k}$	u_i
12:	$V_k = [v_1, \ldots, v_k]$	
13:	$H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$	
14:	$y_k = \textbf{argmin}_y \|\beta e_1 - H_k y\|$	u_i
15:	$k = k + 1$	
16:	**until** $\|\beta e_1 - H_k y_k\| \leq \tau$	
17:	$x_k = x_0 + V_k y_k$	u

Three criteria: Inner/Outer solvers

Algorithm: Refinement loop

1:
2: **repeat**
3: $x_{i+1} = \text{GMRES}(A, b, x_i, \tau)$
4: **until** convergence

1st criterion: Outer solver in high precision and inner solver in low precision ($u \ll u_i$).

➤ [Turner & Walker, 92]: outer iterative refinement (in precision u) with inner GMRES (in precision u_i).

➤ [Buttari et al., 08]: outer FGMRES (in precision u) with inner GMRES as a preconditioner (in precision u_i).

Algorithm: FGMRES(A, b, x_0, τ)

Require: $A \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1:
2: $r_0 = b - Ax_0$ u
3: $\beta = \|r_0\|,\ v_1 = r_0/\beta,\ k = 1$ u
4: **repeat**
5: $z_k = \text{GMRES}(Az_k = v_k)$ u_i
6: $w_k = Az_k$ u
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u
9: $w_k = w_k - h_{i,k} v_i$ u
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k/h_{k+1,k}$ u
12: $V_k = [v_1, \ldots, v_k],\ Z_k = [z_1, \ldots, z_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \textbf{argmin}_y \|\beta e_1 - H_k y\|$ u
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + Z_k y_k$ u

Three criteria: Other precision in the preconditioner

Algorithm: Refinement loop

1: Compute $A = \hat{L}\hat{U}$ u_m
2: **repeat**
3: $x_{i+1} = \text{FGMRES}(A, \hat{L}\hat{U}, b, x_i, \tau)$
4: **until** convergence

2nd criterion: Application of the preconditioner in low or high precisions.

➤ [Arioli & Duff, 08]: restarted FGMRES (in precision u) preconditioned by the LU factors (computed and applied in precision u_m), where $u \ll u_m$.

➤ LU-GMRES-IR5: GMRES (in precision u_g) with restart (in precision u and u_r) left-preconditioned by the LU factors (computed in precision u_f and applied with A in precision u_p), where $ur \leq u$ and $u_p \leq u_g \leq u_f$.

Algorithm: FGMRES(A, $\hat{L}\hat{U}$, b, x_0, τ)

Require: $A, \hat{L}\hat{U} \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1:
2: $r_0 = b - Ax_0$ u
3: $\beta = \|r_0\|,\ v_1 = r_0/\beta,\ k = 1$ u
4: **repeat**
5: $z_k = \hat{U} \setminus \hat{L} \setminus v_k$ u_m
6: $w_k = Az_k$ u
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u
9: $w_k = w_k - h_{i,k} v_i$ u
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k/h_{k+1,k}$ u
12: $V_k = [v_1, \ldots, v_k],\ Z_k = [z_1, \ldots, z_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \mathbf{argmin}_y \|\beta e_1 - H_k y\|$ u
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + Z_k y_k$ u

Three criteria: Other precision in the preconditioner

Algorithm: Refinement loop

1: Compute $A = \hat{L}\hat{U}$ u_f
2: **repeat**
3: $x_{i+1} = \text{LGMRES}(A, \hat{L}\hat{U}, b, x_i, \tau)$
4: **until** convergence

2nd criterion: Application of the preconditioner in low or high precisions.

➤ [Arioli & Duff, 08]: restarted FGMRES (in precision u) preconditioned by the LU factors (computed and applied in precision u_m), where $u \ll u_m$.

➤ LU-GMRES-IR5: GMRES (in precision u_g) with restart (in precision u and u_r) left-preconditioned by the LU factors (computed in precision u_f and applied with A in precision u_p), where $ur \leq u$ and $u_p \leq u_g \leq u_f$.

Algorithm: LGMRES(A, $\hat{L}\hat{U}$, b, x_0, τ)

Require: $A, \hat{L}\hat{U} \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1: $r_0 = b - Ax$ u_r
2: $s_0 = \hat{U} \setminus \hat{L} \setminus r_0$ u_p
3: $\beta = \|s_0\|,\ v_1 = s_0/\beta,\ k = 1$ u_g
4: **repeat**
5: $z_k = Av_k$ u_p
6: $w_k = \hat{U} \setminus \hat{L} \setminus z_k$ u_p
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u_g
9: $w_k = w_k - h_{i,k} v_i$ u_g
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k/h_{k+1,k}$ u_g
12: $V_k = [v_1, \ldots, v_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \mathbf{argmin}_y \|\beta e_1 - H_k y\|$ u_g
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + V_k y_k$ u

Three criteria: Adaptive precision

Algorithm: Refinement loop

1:
2: **repeat**
3: $x_{i+1} = \text{GMRES}(A, b, x_i, \tau)$
4: **until** convergence

3rd criterion: Adapt the precisions as the iterations go.

➤ [Gratton et al., 20]: inexact Krylov GMRES with decreasing precisions u_a^k on the matrix–vector products and u_i^k on the inner products, where $u_a^k \leq u_a^{k+1}$ and $u_i^k \leq u_i^{k+1}$.

➤ [Oktay & Carson, 21]: $u_r \leq u$ and u_f are fixed. Increasing precisions inside GMRES if the convergence stagnates, where $u_g^k \geq u_g^{k+1}$ and $u_p^k \geq u_p^{k+1}$.

Algorithm: GMRES(A, b, x_0, τ)

Require: $A \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1:
2: $r_0 = b - Ax_0$ u
3: $\beta = \|r_0\|,\ v_1 = r_0/\beta,\ k = 1$ u
4: **repeat**
5: $w_k = Av_k$ u_a^k
6:
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u_i^k
9: $w_k = w_k - h_{i,k} v_i$ u
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k/h_{k+1,k}$ u
12: $V_k = [v_1, \ldots, v_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \mathbf{argmin}_y \|\beta e_1 - H_k y\|$ u
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + V_k y_k$ u

Three criteria: Adaptive precision

Algorithm: Refinement loop

1: Compute $A = \hat{L}\hat{U}$ u_f
2: **repeat**
3: $x_{i+1} = \text{LGMRES}(A, \hat{L}\hat{U}, b, x_i, \tau)$
4: **until** convergence

3rd criterion: Adapt the precision as the iterations go.

➤ [Gratton et al., 20]: inexact Krylov GMRES with decreasing precisions u_a^k on the matrix–vector products and u_i^k on the inner products, where $u_a^k \leq u_a^{k+1}$ and $u_i^k \leq u_i^{k+1}$.

➤ [Oktay & Carson, 21]: $u_r \leq u$ and u_f are fixed. Increasing precisions inside GMRES if the convergence stagnates, where $u_g^k \geq u_g^{k+1}$ and $u_p^k \geq u_p^{k+1}$.

Algorithm: LGMRES$(A, \hat{L}\hat{U}, b, x_0, \tau)$

Require: $A, \hat{L}\hat{U} \in \mathbb{R}^{n\times n}, b, x_0 \in \mathbb{R}^n, \tau \in \mathbb{R}$

1: $r_0 = b - Ax$ u_r
2: $s_0 = \hat{U} \setminus \hat{L} \setminus r_0$ u_p^k
3: $\beta = \|s_0\|,\ v_1 = s_0/\beta,\ k = 1$ u_g^k
4: **repeat**
5: $z_k = Av_k$ u_p^k
6: $w_k = \hat{U} \setminus \hat{L} \setminus z_k$ u_p^k
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u_q^k
9: $w_k = w_k - h_{i,k} v_i$ u_q^k
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k/h_{k+1,k}$ u_q^k
12: $V_k = [v_1, \ldots, v_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \textbf{argmin}_y \|\beta e_1 - H_k y\|$ u_q^k
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + V_k y_k$ u

State of the art: Other works

Just the tip of the iceberg...

- Emmanuel Agullo, Franck Cappello, Sheng Di, Luc Giraud, Xin Liang, and Nick Schenkels, "*Exploring variable accuracy storage through lossy compression techniques in numerical linear algebra: a first application to flexible GMRES*", 2020.
- Hartwig Anzt, Vincent Heuveline, and Björn Rocker, "*An Error Correction Solver for Linear Systems: Evaluation of Mixed Precision Implementations*", 2011.
- Mario Arioli, Iain S. Duff, Serge Gratton, and Stéphane Pralet, "*A Note on GMRES Preconditioned by a Perturbed LDL^T Decomposition with Static Pivoting*", 2007.
- Erin Carson and Nicholas J. Higham, "*A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems*", 2017.
- Erin Carson and Noaman Khan, "*Mixed Precision Iterative Refinement with Sparse Approximate Inverse Preconditioning*", 2022.
- Neil Lindquist, Piotr Luszczek, and Jack Dongarra, "*Improving the performance of the GMRES method using mixed-precision techniques*", 2020.
- Jennifer A. Loe, Christian A. Glusa, Ichitaro Yamazaki, Erik G. Boman, and Sivasankaran Rajamanickam, "*A Study of Mixed Precision Strategies for GMRES on GPUs*", 2021.
- José Aliaga, Hartwig Anzt, Thomas Grützmacher, Enrique Quintana-Ortí, and Andrés Tomás, "*Compressed basis GMRES on high performance GPUs*", 2020.
- . . .

BUT most of these works have a lot of overlap with one of the previous configurations.

Current limitations we want to tackle

Limitations

- Several error analyses in these works are specialized for one type of preconditioner (LU, ILU, Block Jacobi, etc): theoretical results are not meant to be extended to other preconditioners.
- Too many different mixed precision strategies: how to choose one? which one is the best? are they linked? are they coherent between each other?
- Lack of general advice/discussions helping the user to choose a certain strategy according to its use case.

Questions (and objectives)

- One GMRES to rule them all $\Rightarrow$ Can we gather all these propositions under a same coherent mixed precision GMRES?
- Can we provide an efficient way for a user to set his/her precisions according to his/her application?
- Can we provide other mixed precision opportunities for GMRES?

Our proposition: M-GMRES-IR6

Algorithm: Refinement loop

1: (Optional) Compute M^{-1} u_f
2: **repeat**
3: $x_{i+1} = \text{GMRES}(A, M^{-1}, b, x_i, \tau)$
4: **until** convergence

Algorithm: GMRES(A,M^{-1}, b, x_0, τ)

Require: $A, M^{-1} \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1: $r_0 = b - Ax$ u_r
2: $s_0 = M^{-1} r_0$ u_m
3: $\beta = \|s_0\|,\ v_1 = s_0/\beta,\ k = 1$ u_g
4: **repeat**
5: $z_k = Av_k$ u_a
6: $w_k = M^{-1} z_k$ u_m
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u_g
9: $w_k = w_k - h_{i,k} v_i$ u_g
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k/h_{k+1,k}$ u_g
12: $V_k = [v_1, \ldots, v_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \textbf{argmin}_y \|\beta e_1 - H_k y\|$ u_g
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + V_k y_k$ u

Our proposition: M-GMRES-IR6

Algorithm: Refinement loop

1: (Optional) Compute M^{-1} u_f
2: **repeat**
3: $x_{i+1} = \text{GMRES}(A, M^{-1}, b, x_i, \tau)$
4: **until** convergence

➤ Generalization of LU-GMRES-IR5.

Algorithm: GMRES(A,M^{-1}, b, x_0, τ)

Require: $A, M^{-1} \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1: $r_0 = b - Ax$ u_r
2: $s_0 = M^{-1} r_0$ u_m
3: $\beta = \|s_0\|,\ v_1 = s_0/\beta,\ k = 1$ u_g
4: **repeat**
5: $z_k = Av_k$ u_a
6: $w_k = M^{-1} z_k$ u_m
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u_g
9: $w_k = w_k - h_{i,k} v_i$ u_g
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k / h_{k+1,k}$ u_g
12: $V_k = [v_1, \ldots, v_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \textbf{argmin}_y \|\beta e_1 - H_k y\|$ u_g
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + V_k y_k$ u

Our proposition: M-GMRES-IR6

Algorithm: Refinement loop

1: (Optional) Compute $\boldsymbol{M^{-1}}$ u_f
2: **repeat**
3: $x_{i+1} = \text{GMRES}(A, \boldsymbol{M^{-1}}, b, x_i, \tau)$
4: **until** convergence

- Generalization of LU-GMRES-IR5.
- Arbitrary preconditioner M^{-1}.

Algorithm: GMRES(A,$\boldsymbol{M^{-1}}$, b, x_0, τ)

Require: $A, \boldsymbol{M^{-1}} \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1: $r_0 = b - Ax$ u_r
2: $s_0 = \boldsymbol{M^{-1}} r_0$ u_m
3: $\beta = \|s_0\|,\ v_1 = s_0/\beta,\ k = 1$ u_g
4: **repeat**
5: $z_k = Av_k$ u_a
6: $w_k = \boldsymbol{M^{-1}} z_k$ u_m
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u_g
9: $w_k = w_k - h_{i,k} v_i$ u_g
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k/h_{k+1,k}$ u_g
12: $V_k = [v_1, \ldots, v_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \textbf{argmin}_y \|\beta e_1 - H_k y\|$ u_g
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + V_k y_k$ u

Our proposition: M-GMRES-IR6

Algorithm: Refinement loop

1: (Optional) Compute M^{-1} u_f
2: **repeat**
3: $x_{i+1} = \text{GMRES}(A, M^{-1}, b, x_i, \tau)$
4: **until** convergence

- Generalization of LU-GMRES-IR5.
- Arbitrary preconditioner M^{-1}.
- Dissociating the precision u_p in two precisions $u_m \neq u_a$.

Algorithm: GMRES(A,M^{-1}, b, x_0, τ)

Require: $A, M^{-1} \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1: $r_0 = b - Ax$ u_r
2: $s_0 = M^{-1} r_0$ u_m
3: $\beta = \|s_0\|,\ v_1 = s_0/\beta,\ k = 1$ u_g
4: **repeat**
5: $z_k = Av_k$ u_a
6: $w_k = M^{-1} z_k$ u_m
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u_g
9: $w_k = w_k - h_{i,k} v_i$ u_g
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k / h_{k+1,k}$ u_g
12: $V_k = [v_1, \ldots, v_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \textbf{argmin}_y \|\beta e_1 - H_k y\|$ u_g
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + V_k y_k$ u

Our proposition: M-GMRES-IR6

Algorithm: Refinement loop

1: (Optional) Compute M^{-1} u_f
2: **repeat**
3: $x_{i+1} = \text{GMRES}(A, M^{-1}, b, x_i, \tau)$
4: **until** convergence

- Generalization of LU-GMRES-IR5.
- Arbitrary preconditioner M^{-1}.
- Dissociating the precision u_p in two precisions $u_m \neq u_a$.
- Up to 6 independent precisions: u_f, u, u_r, u_g, u_m, and u_a !

Algorithm: GMRES(A,M^{-1}, b, x_0, τ)

Require: $A, M^{-1} \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1: $r_0 = b - Ax$ u_r
2: $s_0 = M^{-1} r_0$ u_m
3: $\beta = \|s_0\|,\ v_1 = s_0/\beta,\ k = 1$ u_g
4: **repeat**
5: $z_k = Av_k$ u_a
6: $w_k = M^{-1} z_k$ u_m
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u_g
9: $w_k = w_k - h_{i,k} v_i$ u_g
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k / h_{k+1,k}$ u_g
12: $V_k = [v_1, \ldots, v_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \textbf{argmin}_y \|\beta e_1 - H_k y\|$ u_g
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + V_k y_k$ u

Our proposition: M-GMRES-IR6

Algorithm: Refinement loop

1: (Optional) Compute M^{-1} u_f
2: **repeat**
3: $x_{i+1} = \text{GMRES}(A, M^{-1}, b, x_i, \tau)$
4: **until** convergence

- Generalization of LU-GMRES-IR5.
- Arbitrary preconditioner M^{-1}.
- Dissociating the precision u_p in two precisions $u_m \neq u_a$.
- Up to 6 independent precisions: u_f, u, u_r, u_g, u_m, and u_a !
- Covers many other works on mixed precision GMRES...

Algorithm: GMRES(A,M^{-1}, b, x_0, τ)

Require: $A, M^{-1} \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1: $r_0 = b - Ax$ u_r
2: $s_0 = M^{-1} r_0$ u_m
3: $\beta = \|s_0\|,\ v_1 = s_0/\beta,\ k = 1$ u_g
4: **repeat**
5: $z_k = Av_k$ u_a
6: $w_k = M^{-1} z_k$ u_m
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u_g
9: $w_k = w_k - h_{i,k} v_i$ u_g
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k / h_{k+1,k}$ u_g
12: $V_k = [v_1, \ldots, v_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \textbf{argmin}_y \|\beta e_1 - H_k y\|$ u_g
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + V_k y_k$ u

Our proposition: M-GMRES-IR6

Algorithm: Refinement loop

1: (Optional) Compute M^{-1} u_f
2: **repeat**
3: $x_{i+1} = \text{GMRES}(A, M^{-1}, b, x_i, \tau)$
4: **until** convergence

- Generalization of LU-GMRES-IR5.
- Arbitrary preconditioner M^{-1}.
- Dissociating the precision u_p in two precisions $u_m \neq u_a$.
- Up to 6 independent precisions: u_f, u, u_r, u_g, u_m, and u_a !
- Covers many other works on mixed precision GMRES...
- ... and offers new opportunities.

Algorithm: GMRES(A,M^{-1}, b, x_0, τ)

Require: $A, M^{-1} \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$

1: $r_0 = b - Ax$ u_r
2: $s_0 = M^{-1} r_0$ u_m
3: $\beta = \|s_0\|,\ v_1 = s_0/\beta,\ k = 1$ u_g
4: **repeat**
5: $z_k = Av_k$ u_a
6: $w_k = M^{-1} z_k$ u_m
7: **for** $i = 1, \ldots, k$ **do**
8: $h_{i,k} = v_i^T w_k$ u_g
9: $w_k = w_k - h_{i,k} v_i$ u_g
10: **end for**
11: $h_{k+1,k} = \|w_k\|, v_{k+1} = w_k/h_{k+1,k}$ u_g
12: $V_k = [v_1, \ldots, v_k]$
13: $H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$
14: $y_k = \textbf{argmin}_y \|\beta e_1 - H_k y\|$ u_g
15: $k = k + 1$
16: **until** $\|\beta e_1 - H_k y_k\| \leq \tau$
17: $x_k = x_0 + V_k y_k$ u

Stability of restarted left-preconditioned GMRES

Theorem (Stability of M-GMRES-IR6)

Let $Ax = b$ be solved by the mixed precision left-preconditioned restarted MGS-GMRES. Provided that A and M are not singular, the forward error

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq n u_r \, \mathrm{cond}(A, x) + u, \tag{1}$$

and we guarantee that the backward error

$$\frac{\|A\hat{x} - b\|}{\|A\|\|x\| + \|b\|} \leq n u_r + u, \tag{2}$$

Stability of restarted left-preconditioned GMRES

Theorem (Stability of M-GMRES-IR6)

Let $Ax = b$ be solved by the mixed precision left-preconditioned restarted MGS-GMRES. Provided that A and M are not singular, the ***forward error***

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq n u_r \, \mathrm{cond}(A, x) + u, \quad \text{if} \quad \max\left(u_g, u_a\rho_A, u_m\rho_M, \tau\right)\kappa(\tilde{A}) \ll 1, \tag{1}$$

and we guarantee that the ***backward error***

$$\frac{\|A\hat{x} - b\|}{\|A\|\|x\| + \|b\|} \leq n u_r + u, \quad \text{if} \quad \max\left(u_g, u_a\rho_A, u_m\rho_M, \tau\right)\kappa(M) \ll 1. \tag{2}$$

Where

$$u_a\rho_A \equiv \frac{\|M^{-1}\Delta A v_k\|}{\|M^{-1}Av_k\|} \leq u_a\bar{\rho}_A, \quad u_m\rho_M \equiv \frac{\|\Delta M A v_k\|}{\|M^{-1}Av_k\|} \leq u_m\bar{\rho}_M.$$

Stability of restarted left-preconditioned GMRES

Theorem (Stability of M-GMRES-IR6)

Let $Ax = b$ be solved by the mixed precision left-preconditioned restarted MGS-GMRES. Provided that A and M are not singular, the ***forward error***

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq n u_r \, \mathrm{cond}(A, x) + u, \quad \text{if} \quad \max\left(u_g, u_a \rho_A, u_m \rho_M, \tau\right) \kappa(\tilde{A}) \ll 1, \tag{1}$$

and we guarantee that the ***backward error***

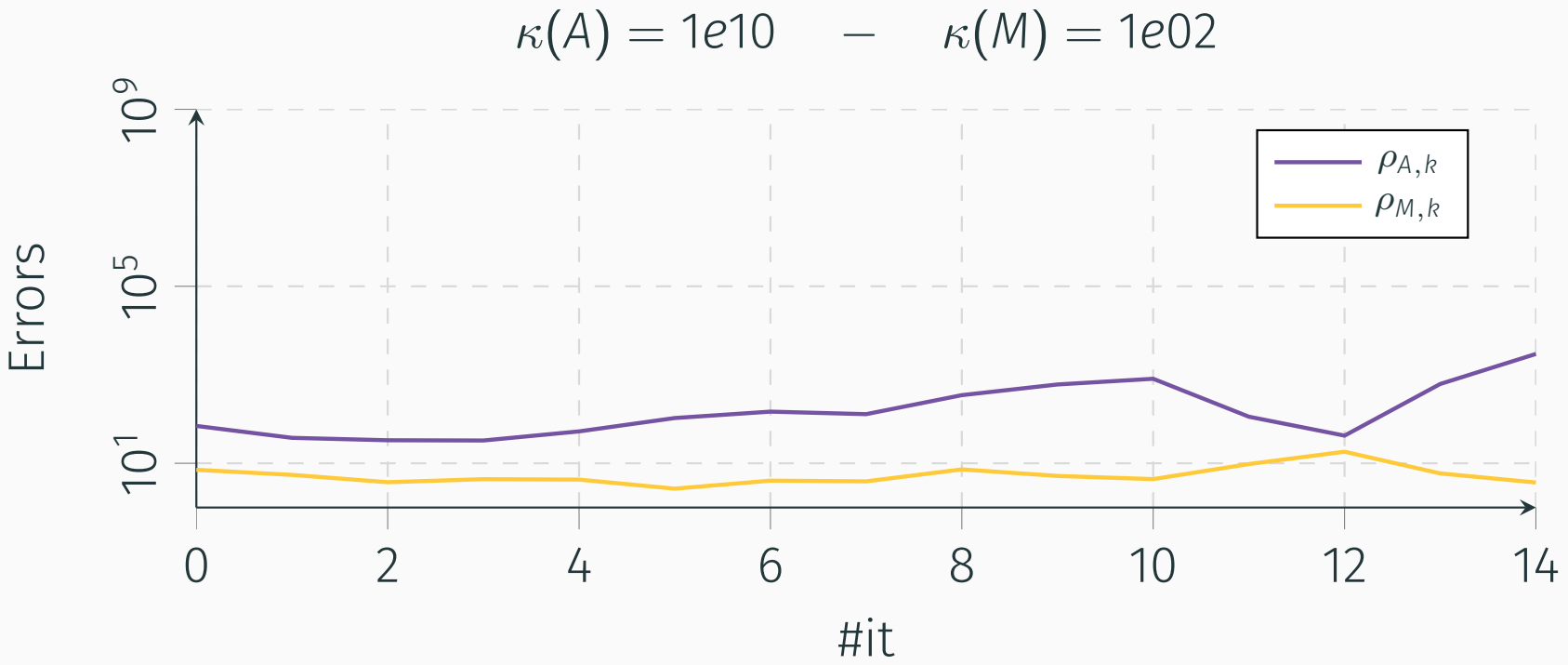
$$\frac{\|A\hat{x} - b\|}{\|A\|\|x\| + \|b\|} \leq n u_r + u, \quad \text{if} \quad \max\left(u_g, u_a \rho_A, u_m \rho_M, \tau\right) \kappa(M) \ll 1. \tag{2}$$

Where

$$u_a \rho_A \equiv \frac{\|M^{-1}\Delta A v_k\|}{\|M^{-1} A v_k\|} \leq u_a \bar{\rho}_A, \quad u_m \rho_M \equiv \frac{\|\Delta M A v_k\|}{\|M^{-1} A v_k\|} \leq u_m \bar{\rho}_M.$$

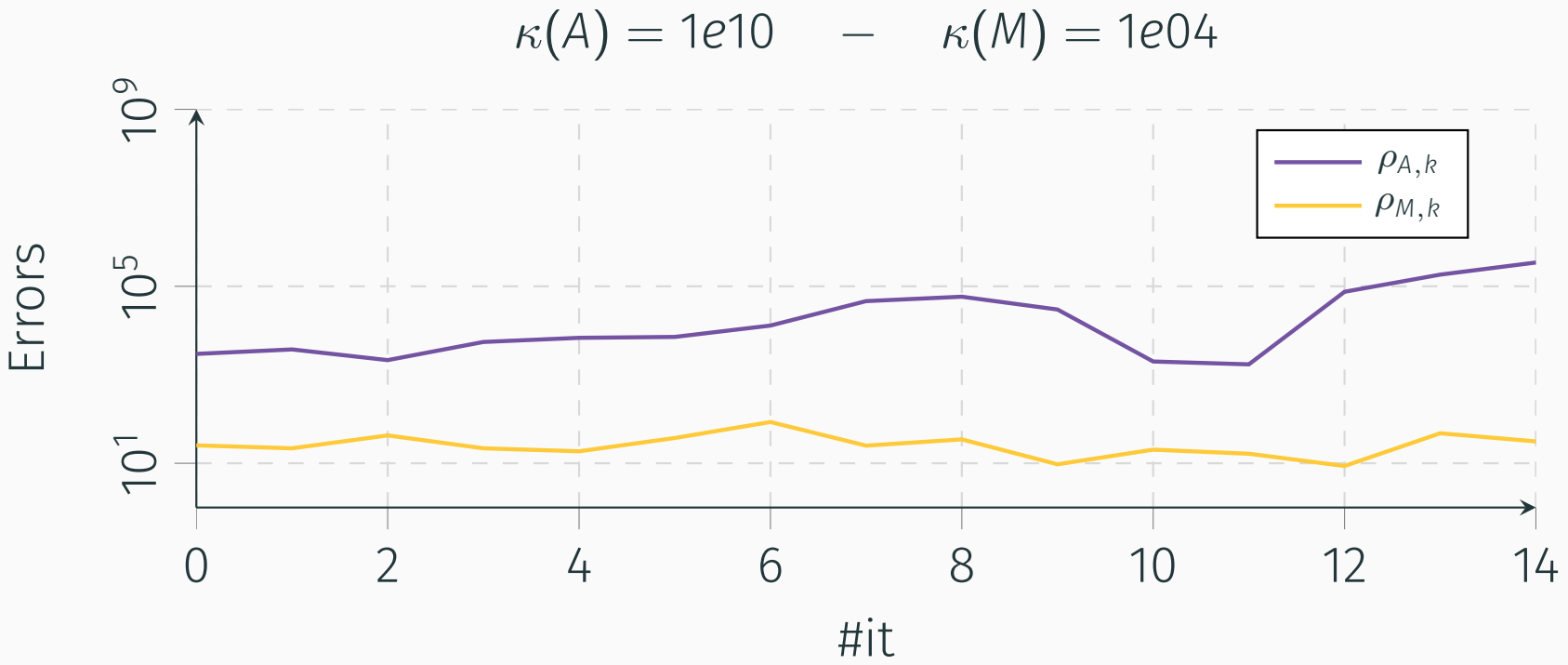
$\Rightarrow$ Ability to set $u_a \neq u_m$ relies on "$\rho_A \ll$ or $\gg \rho_M$".

Dissociate the left preconditioned product



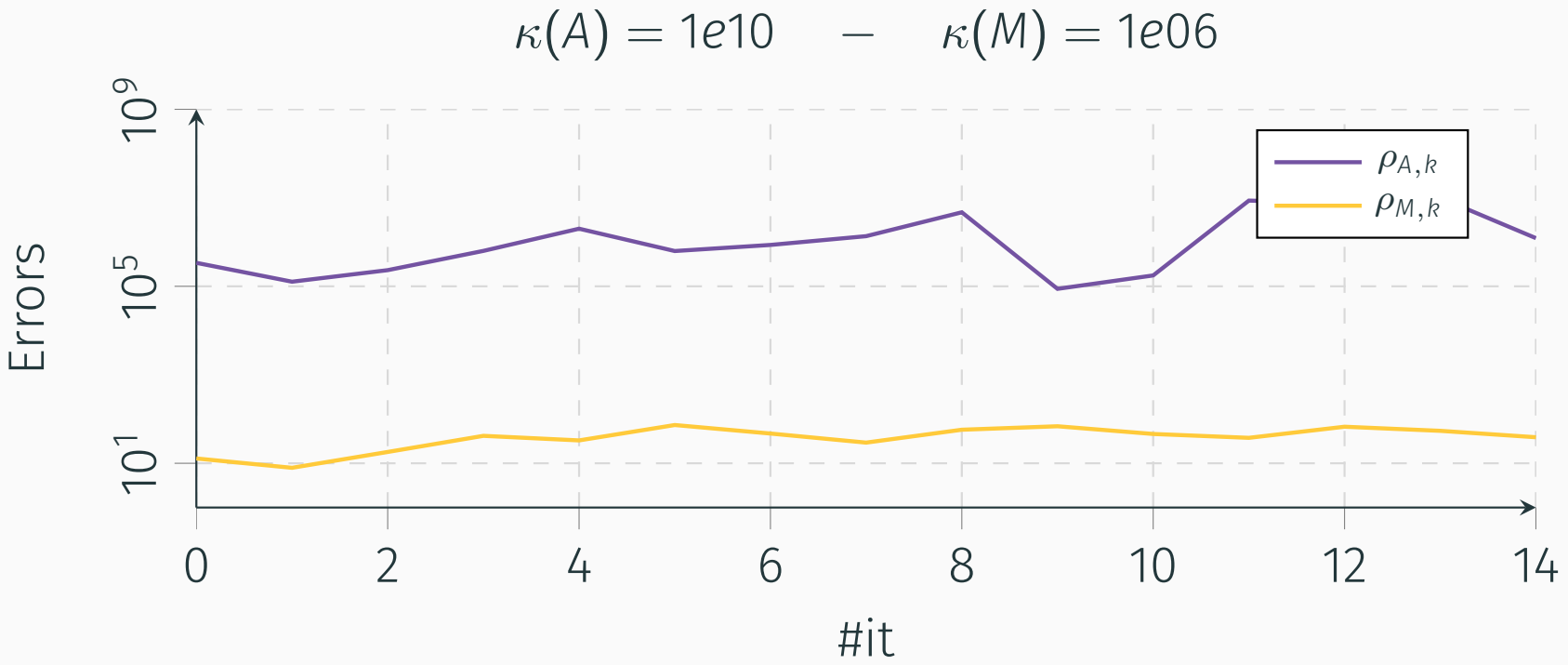
Evolution of $\rho_{A,k}$ and $\rho_{M,k}$ over 15 iterations of GMRES. (Randomly generated $A, M \in \mathbb{R}^{50\times 50}$ with targeted condition numbers)

Dissociate the left preconditioned product



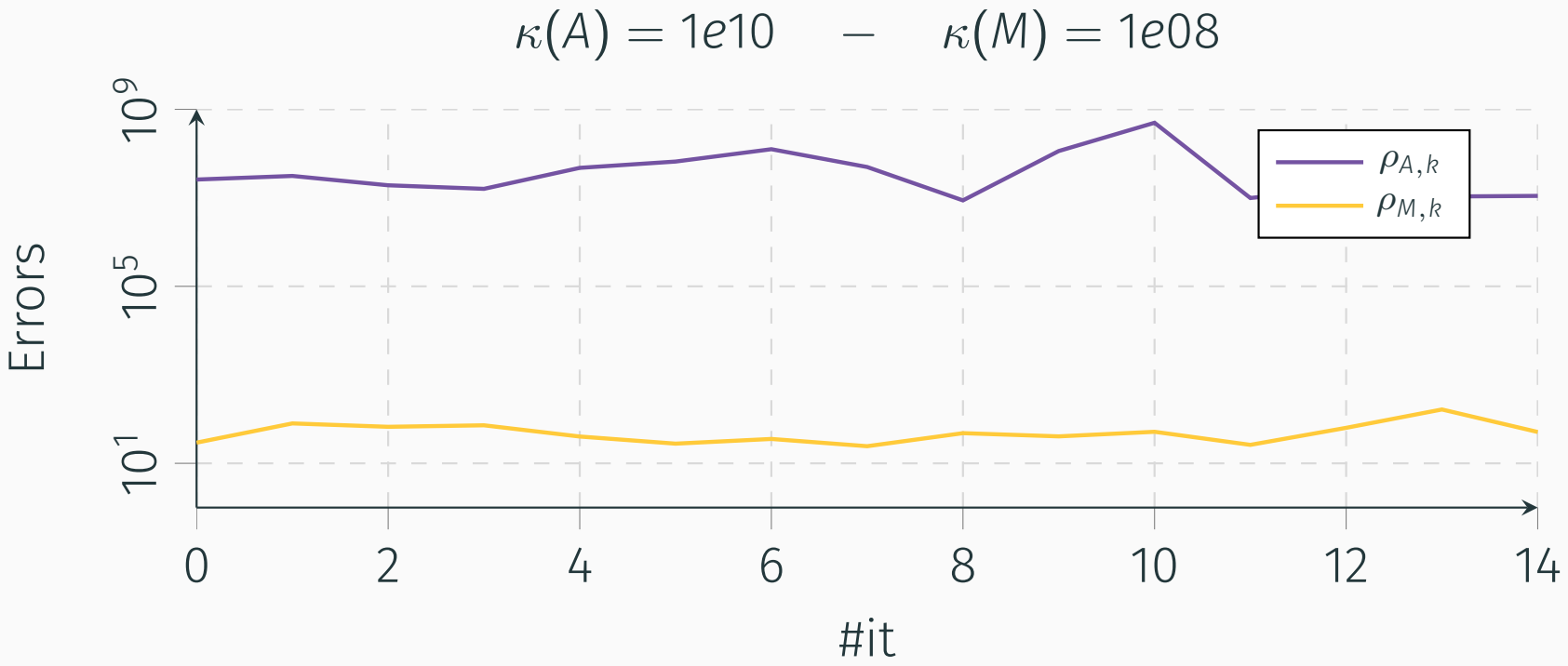
Evolution of $\rho_{A,k}$ and $\rho_{M,k}$ over 15 iterations of GMRES. (Randomly generated $A, M \in \mathbb{R}^{50\times 50}$ with targeted condition numbers)

Dissociate the left preconditioned product



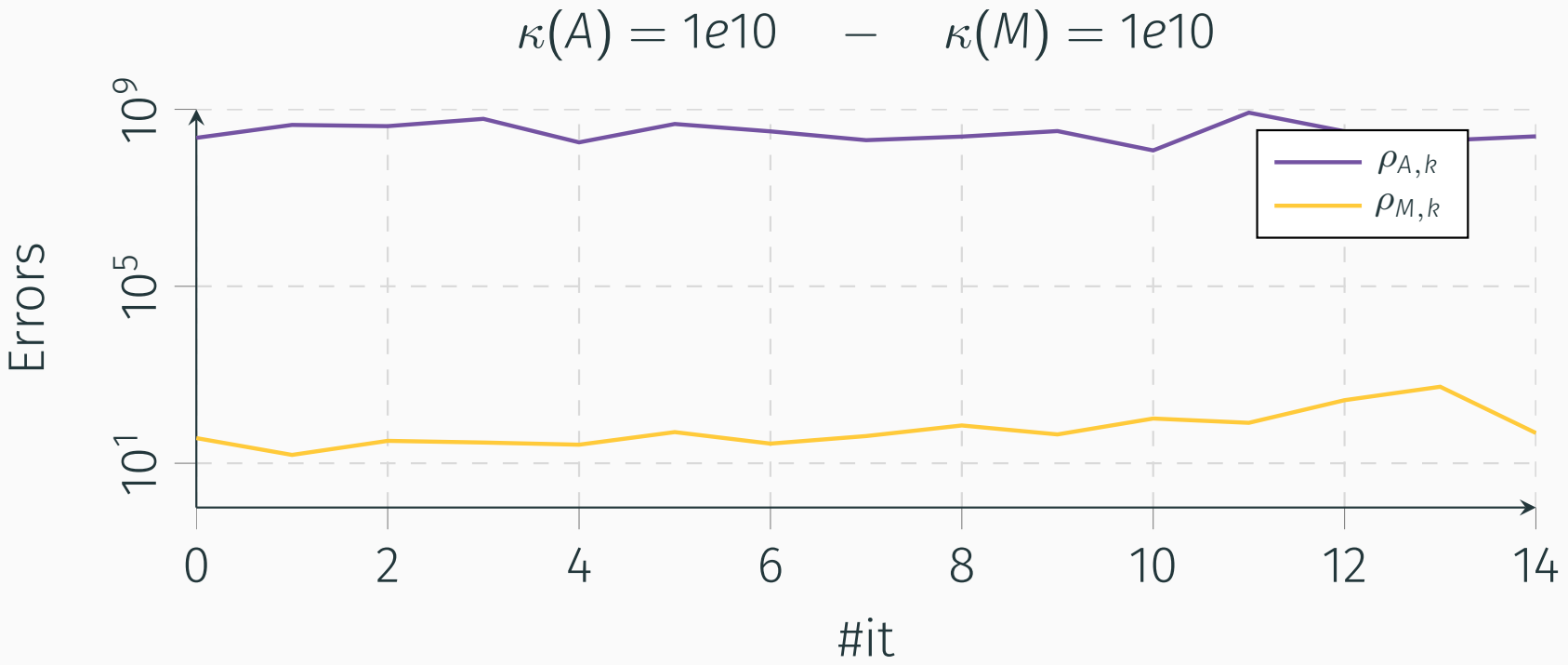
Evolution of $\rho_{A,k}$ and $\rho_{M,k}$ over 15 iterations of GMRES. (Randomly generated $A, M \in \mathbb{R}^{50\times 50}$ with targeted condition numbers)

Dissociate the left preconditioned product



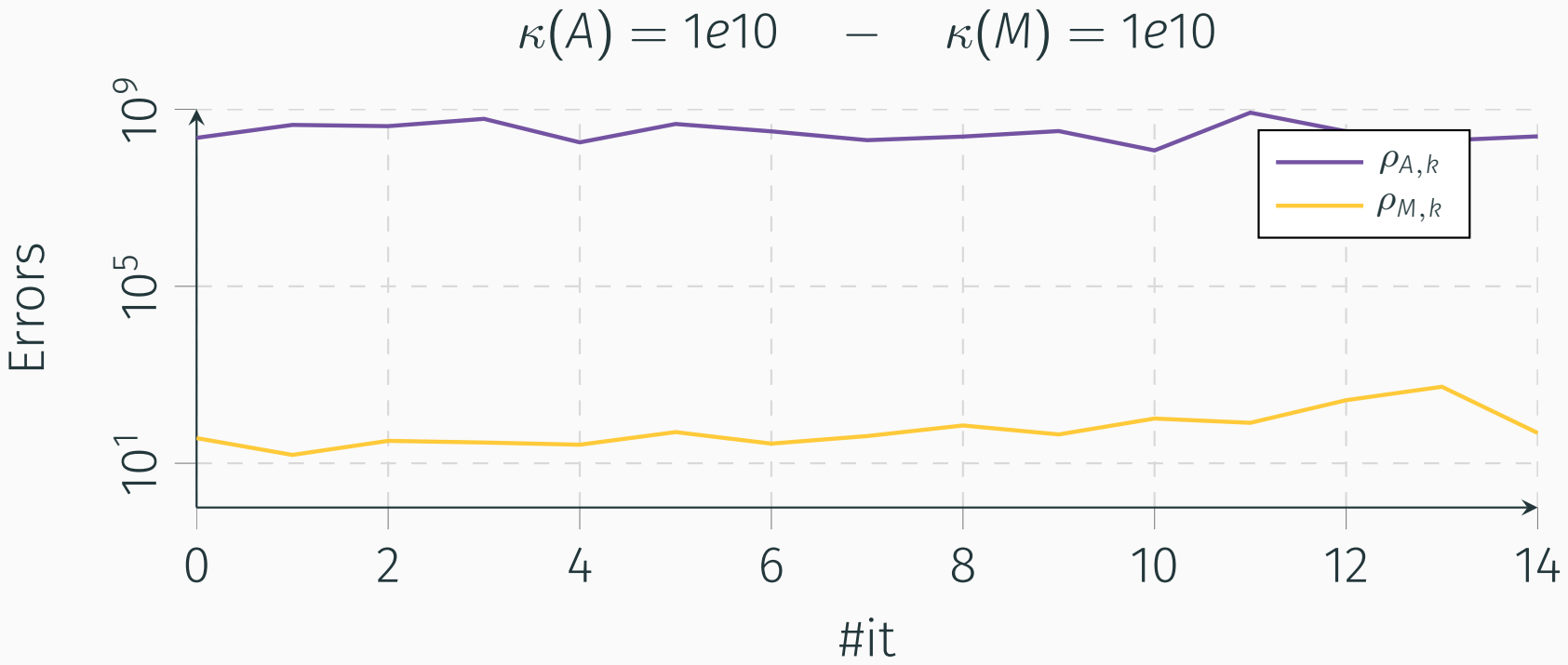
Evolution of $\rho_{A,k}$ and $\rho_{M,k}$ over 15 iterations of GMRES. (Randomly generated $A, M \in \mathbb{R}^{50\times 50}$ with targeted condition numbers)

Dissociate the left preconditioned product



Evolution of $\rho_{A,k}$ and $\rho_{M,k}$ over 15 iterations of GMRES. (Randomly generated $A, M \in \mathbb{R}^{50\times 50}$ with targeted condition numbers)

Dissociate the left preconditioned product



Evolution of $\rho_{A,k}$ and $\rho_{M,k}$ over 15 iterations of GMRES. (Randomly generated $A, M \in \mathbb{R}^{50\times 50}$ with targeted condition numbers)

$\Rightarrow$ We can set $u_m \gg u_a$!

Experimental results on real problems with M-GMRES-IR6

u_g = inner GMRES, u_m = application of M^{-1}, u_a = application of A

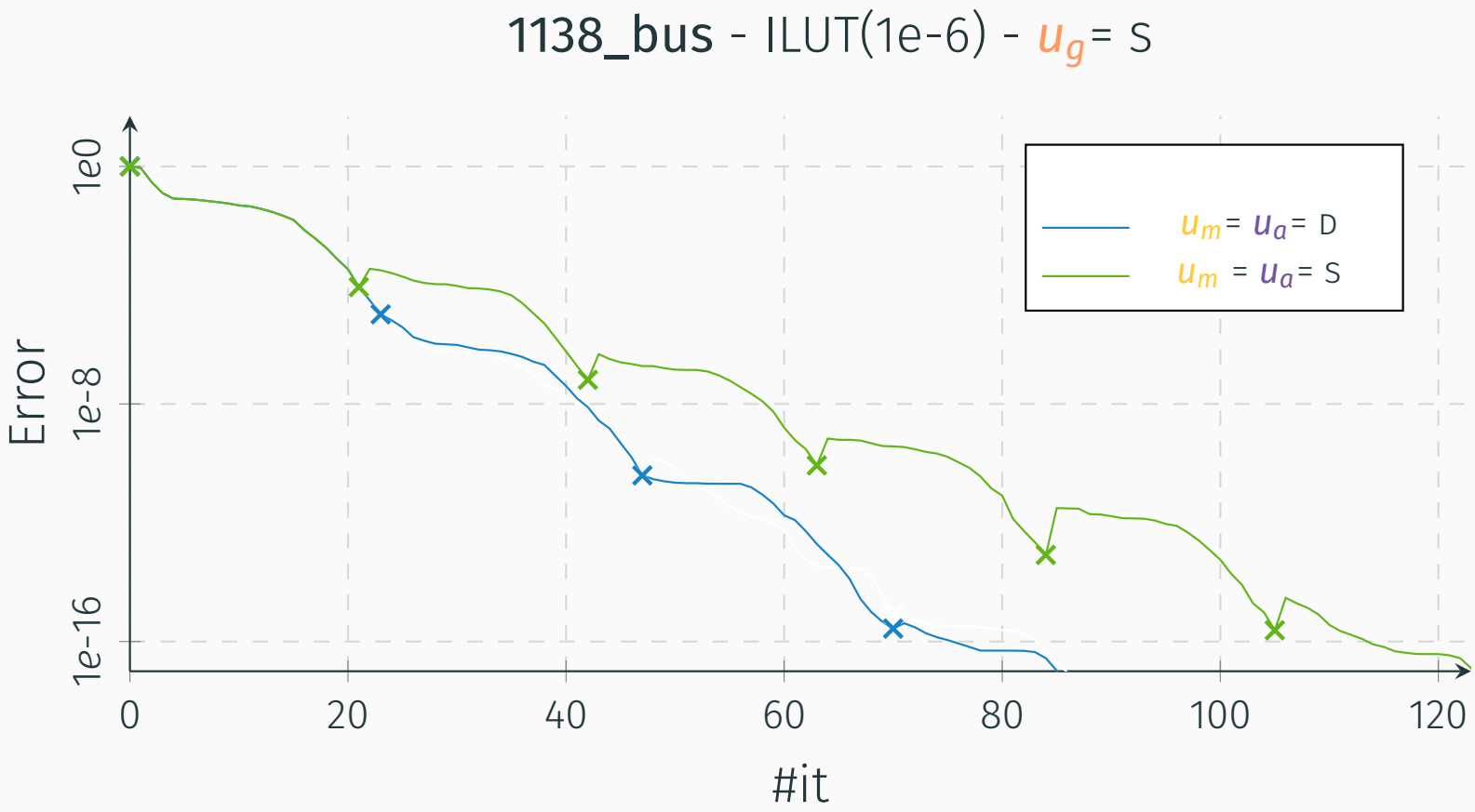
We will focus on the variant: $u_g \geq u_m \gg u_a$

- It is meaningful and has not been studied before.
- Interesting for performance in configurations where the application of the preconditioner M^{-1} is more costly than the application of A (e.g. ILU).
- Will be compared to already known variants $u_g = u_m = u_a$ and $u_g \gg u_m = u_a$.

Experimental setting:

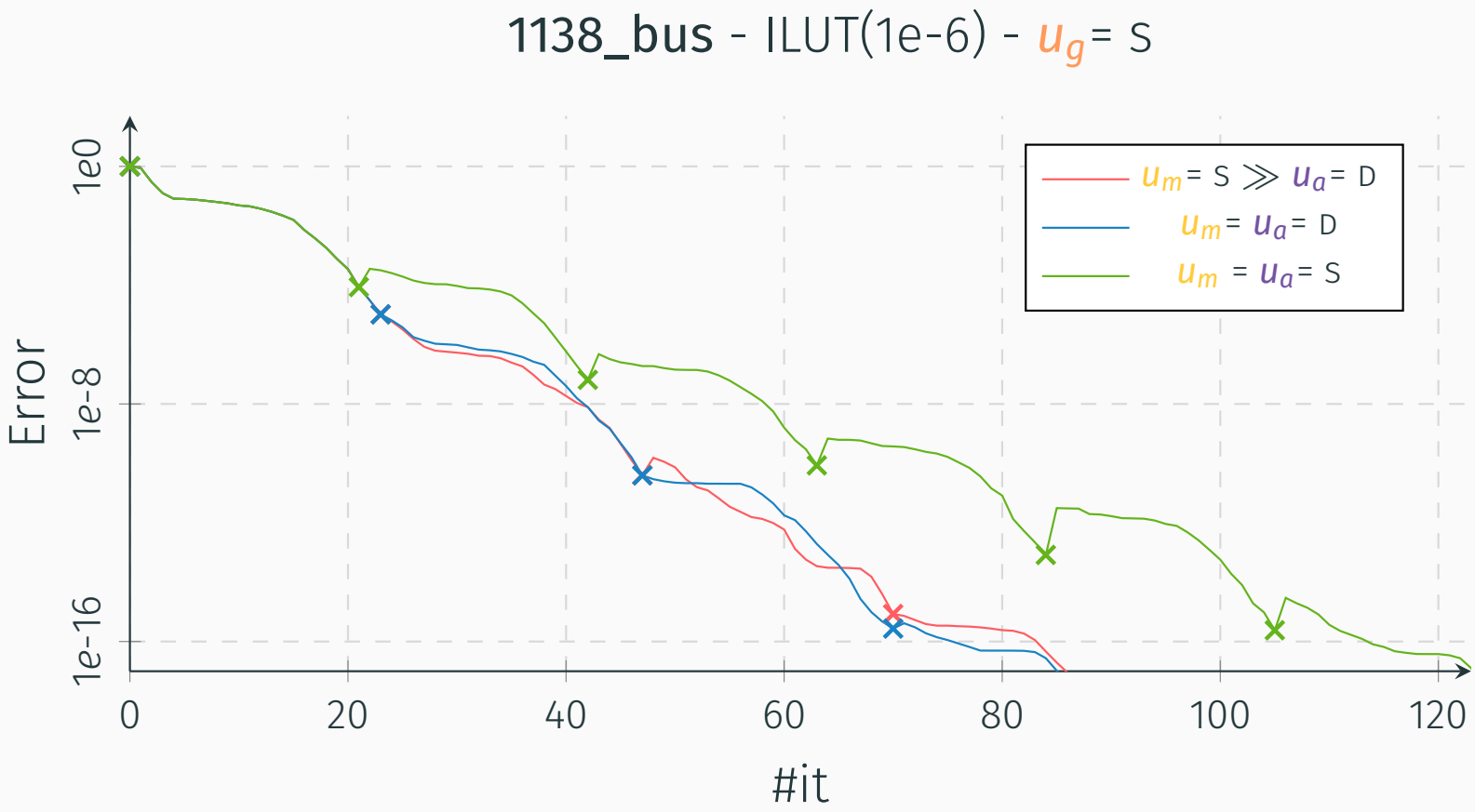
- Convergence behavior on small Suite Sparse matrices.
- Preconditioner = threshold ILU.
- fp128 = Q, fp64 = D, fp32 = S, and fp16 = H.
- u = D and u_r = Q fixed $\Rightarrow \|\hat{x} - x\|/\|x\| = 10^{-16}$.
- u_g = H or S fixed.

Experimental results on real problems with M-GMRES-IR6



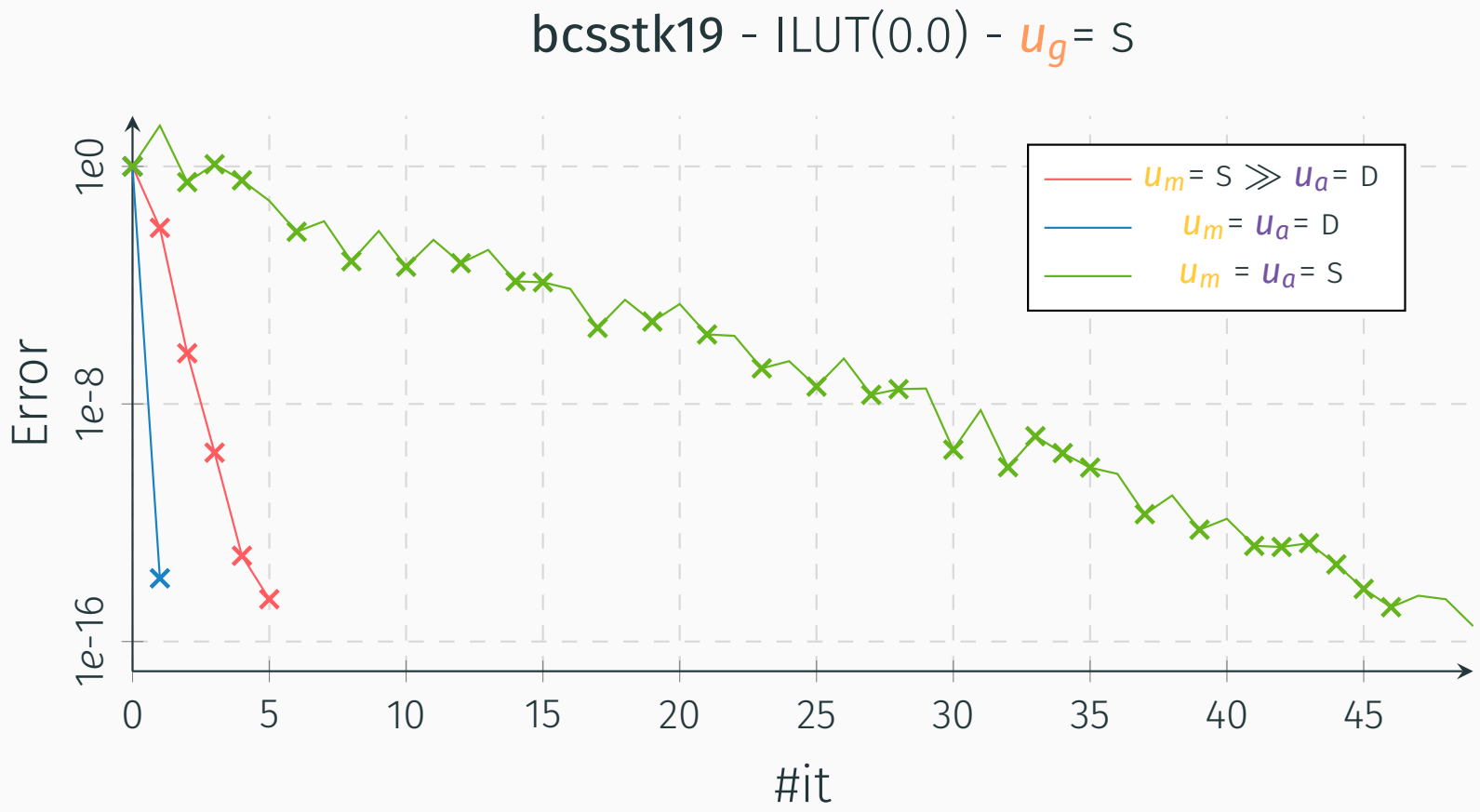
Evolution of the error $\|\hat{x} - x\|/\|x\|$ according to the number of iterations with $u = \text{D}$ and $u_r = \text{Q}$.

Experimental results on real problems with M-GMRES-IR6



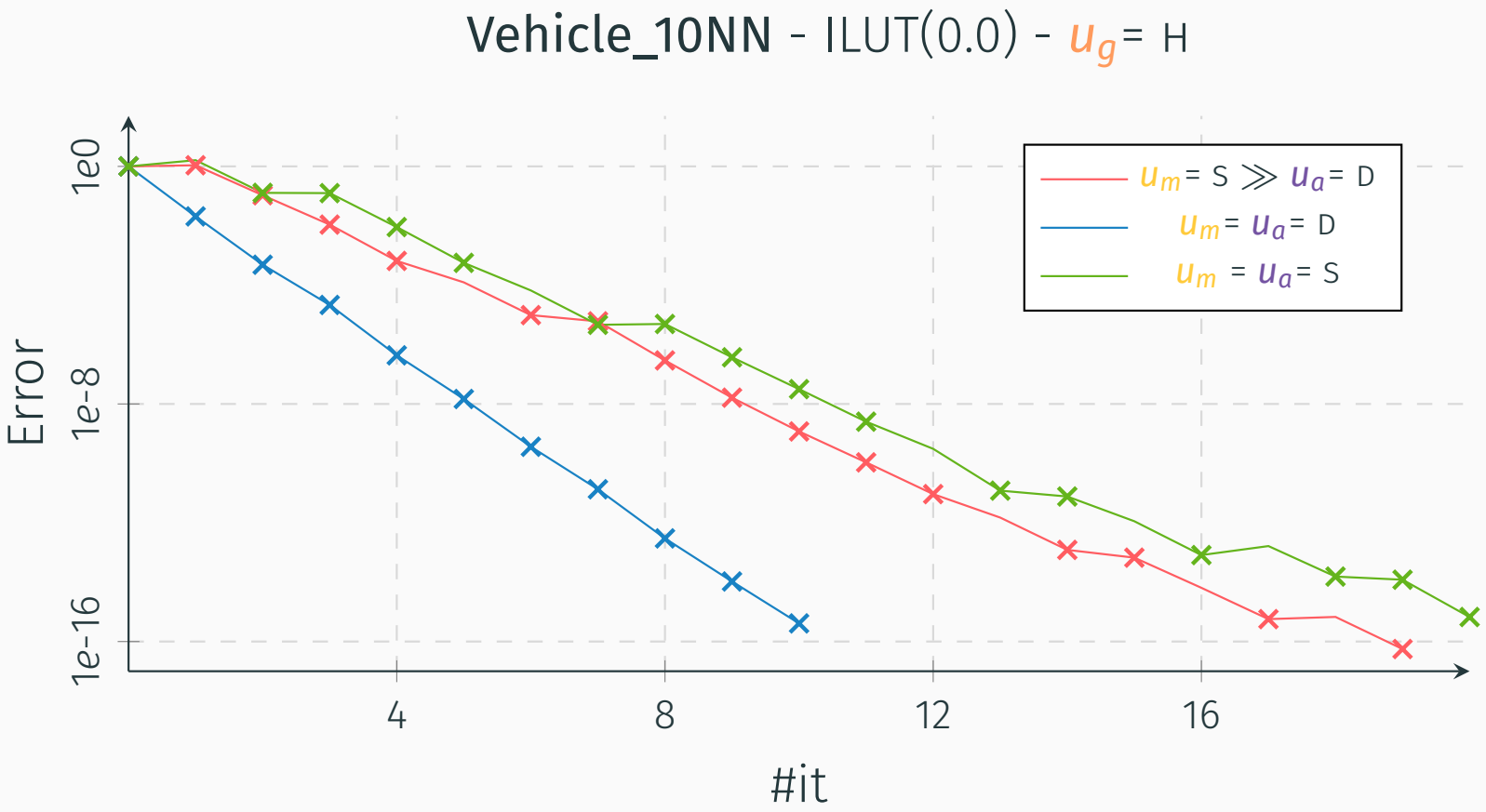
Evolution of the error $\|\hat{x} - x\|/\|x\|$ according to the number of iterations with $u = \text{D}$ and $u_r = \text{Q}$.

Experimental results on real problems with M-GMRES-IR6



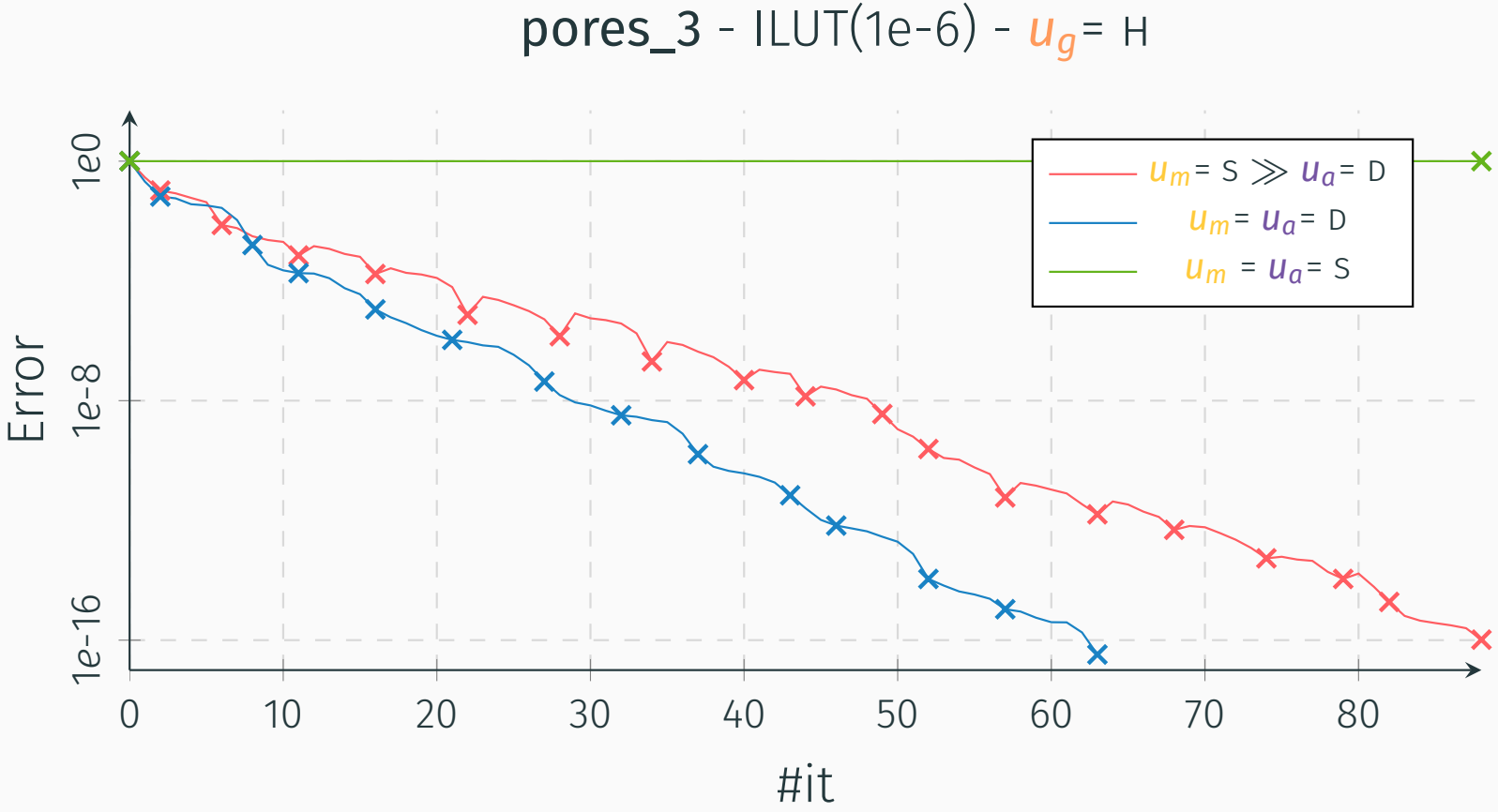
Evolution of the error $\|\hat{x} - x\|/\|x\|$ according to the number of iterations with $u = \text{D}$ and $u_r = \text{Q}$.

Experimental results on real problems with M-GMRES-IR6



Evolution of the error $\|\hat{x} - x\|/\|x\|$ according to the number of iterations with $u = \text{D}$ and $u_r = \text{Q}$.

Experimental results on real problems with M-GMRES-IR6



Evolution of the error $\|\hat{x} - x\|/\|x\|$ according to the number of iterations with u = D and u_r = Q.

Conclusion on M-GMRES-IR6

Contributions

- **New algorithm:** M-GMRES-IR6 which is a new mixed precision framework for GMRES and covers most of the previous works on mixed precision GMRES.
- **Error analysis:** Provides new convergence conditions for M-GMRES-IR6 and assesses the relevance of a new mixed precision strategy.
- **Numerical experiments:** Validate the practical relevance of a new mixed precision strategy on real-life matrices.

Early work: Few things still need to be explored and completed (e.g. right preconditioned case + FGMRES, the use of other preconditioners in the experiments, evaluate the sharpness of the bounds $\bar{\rho}_A$ and $\bar{\rho}_M$).

Article in preparation.

Conclusion

Summary of the contributions presented in this talk

LU-GMRES-IR5: relaxing the precisions (Chap 5)

Extension of LU-GMRES-IR3 to a more versatile algorithm allowing finer trade-offs between performance and robustness on numerically difficult problems. In particular LU-GMRES-IR5 is more suited to the solution of large sparse problems.

IR with sparse approximate factorization (Chap 6)

Performance analysis of state-of-the-art iterative refinement combined with state-of-the-art sparse factorizations for the parallel solution of sparse linear systems coming from real-life applications. We demonstrated gains up to a factor 5.6 in time and 4.4 in memory on our set.

Mixed precision GMRES framework (Chap 7)

New mixed precision framework for GMRES that makes use of an arbitrary preconditioner and 6 independent precision parameters. Error analysis and first numerical experiments.

Two main research directions

1. Transfer state-of-the-art approximate computing techniques into usable practical software:

   ➤ MUMPS in half precision? The increasing availability of half precision in CPU will make it easier to target a fully efficient half sparse factorization BUT it might bring new challenges.

   ➤ Would like to collaborate with a mature GMRES parallel solver (e.g. HPDDM/PETSc) to propose a usable implementation of M-GMRES-IR6.

2. Develop generic and modular theoretical analyses to address the increasing number of approximate computing propositions.

E.g.: In M-GMRES-IR6, as u_a represents the independent accuracy of the SpMV, any "approximate SpMV" can be plugged in the analysis.

Acknowledgements

- We thank the reviewers of the manuscript for their detailed feedbacks, corrections, and advice which will surely enhance the document.
- We thank the CALMIP computing center for providing access to the OLYMPE supercomputer.
- We thank our industrial partners and the EoCoE project for providing access to their matrices.
- We thank Nick Higham and the NLA group of the University of Manchester for their warm welcome and their support on the making of this thesis.
- We thank Serge Gratton and Pierre Jolivet for their kind advice.