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

Speaker : Bastien Vieublé
Joint work with : Alfredo Buttari, Theo Mary, Patrick Amestoy, Jean-Yves L'Excellent, Nicholas J. Higham

SPARSE DAYS 2020

Widening range of arithmetics

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

- Half precision increasingly **supported by hardware** (NVIDIA, Arm, AMD, Intel soon, ...)
- **Great benefits** :
  - Reduced storage, data movement, and communications
  - Faster computations : fp32 $\rightarrow$ fp16 speedup evolution with **GPU Tensor Cores** : P100 : 2× V100 : 8× A100 : 16×
  - Reduced energy consumption
- However, low precision = low accuracy
- Motivation for **mixed precision algorithms**.

Wilkinson, 1948

Algorithm Original iterative refinement

1 : Compute the LU factorization $A = LU$ (u)
2 : Solve $Ax_0 = b$ (u)
3 : **while not** converged **do**
4 : Compute $r_i = b - Ax_i$ (u^2)
5 : Solve $Ad_i = r_i$ by $d_i = L\backslash U\backslash r_i$. (u)

6 : Compute $x_{i+1} = x_i + d_i$ (u)
7 : **end while**

Originally meant to correct rounding errors of a direct LU solving.

J. H. Wilkinson. *"Rounding errors in algebraic processes"*. In : Notes Appl. Sci. 32, 1963.

C. B. Moler. *"Iterative refinement in floating point"*. In : J. Assoc. Comput. Mach., 1967.

Wilkinson, 1948

Algorithm Original iterative refinement

1 : Compute the LU factorization $A = LU$ (u)
2 : Solve $Ax_0 = b$ (u)
3 : **while not** converged **do**
4 : Compute $r_i = b - Ax_i$ (u^2)
5 : Solve $Ad_i = r_i$ by $d_i = L\backslash U\backslash r_i$. (u)

6 : Compute $x_{i+1} = x_i + d_i$ (u)
7 : **end while**

Originally meant to correct rounding errors of a direct LU solving.

J. H. Wilkinson. *"Rounding errors in algebraic processes"*. In : Notes Appl. Sci. 32, 1963.

C. B. Moler. *"Iterative refinement in floating point"*. In : J. Assoc. Comput. Mach., 1967.

Jankowski and Wozniakowski, 1977

Algorithm Fixed precision iterative refinement

1 : Compute the LU factorization $A = LU$ (u)
2 : Solve $Ax_0 = b$ (u)
3 : **while not** converged **do**
4 : Compute $r_i = b - Ax_i$ (u)
5 : Solve $Ad_i = r_i$ by $d_i = L\backslash U\backslash r_i$. (u)

6 : Compute $x_{i+1} = x_i + d_i$ (u)
7 : **end while**

This variant is available in MUMPS (ICNTL(10)).

M. Jankowski and H. Wozniakowski. *"Iterative refinement implies numerical stability"*. In : BIT, 1977.

Langou et al, 2006

Algorithm Two precision acceleration iterative refinement

1 : Compute the LU factorization $A = LU$ $(u_f > u)$
2 : Solve $Ax_0 = b$ (u_f)
3 : **while not** converged **do**
4 : Compute $r_i = b - Ax_i$ (u)
5 : Solve $Ad_i = r_i$ by $d_i = L\backslash U\backslash r_i$. (u_f)

6 : Compute $x_{i+1} = x_i + d_i$ (u)
7 : **end while**

IR has been reconsidered as an accelerator on modern architectures where single precision computation is twice as fast as double precision.

J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, and J. Dongarra. *"Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems)"*. In : ACM, 2006.

LU-IR3, Carson and Higham, 2018

Algorithm LU-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 $Ad_i = r_i$ by $d_i = L\backslash U\backslash r_i$. (u_f)

6 : Compute $x_{i+1} = x_i + d_i$ (u)
7 : **end while**

Introduction of a third precision u_r to improve the error accuracy of the solution.

E. Carson and N. J. Higham. *"Accelerating the solution of linear systems by iterative refinement in three precisions"*. In : SIAM, 2018.

LU-IR3, Carson and Higham, 2018

Algorithm LU-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 $Ad_i = r_i$ by $d_i = L\backslash U\backslash r_i$. (u_f)
6: Compute $x_{i+1} = x_i + d_i$ (u)
7: **end while**

	Convergence condition	ferr
LU-IR3	$\kappa(A) < u_f^{-1}$	$u_r\kappa(A) + u$

Very low precision factorization (e.g fp16, bfloat16) leads to a to a very restrictive convergence condition for LU-IR3 (e.g 2×10^3).

GMRES-IR3, Carson and Higham, 2018

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**

E. Carson and N. J. Higham. *"Accelerating the solution of linear systems by iterative refinement in three precisions"*. In : SIAM, 2018.

GMRES-IR3, Carson and Higham, 2018

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	ferr
LU-IR3	$\kappa(A) < u_f^{-1}$	$u_r\kappa(A) + u$
GMRES-IR3	$\kappa(A) < u^{-1/2}u_f^{-1}$	$u_r\kappa(A) + u$

If u_f is fp16, then the condition on LU-IR3 is 2×10^3, on GMRES-IR3 is 2×10^{11} !

Sparse specific features of LU-IR3

Fill-in in sparse direct solvers : $\mathrm{nnz}(A) \ll \mathrm{nnz}(LU)$.

- $n \times n$ matrix from a regular 3D geometry $\Rightarrow \mathrm{nnz}(A) = \mathcal{O}(n)$ and $\mathrm{nnz}(LU) = \mathcal{O}(n^{4/3})$.
- Operations complexities : facto $\mathcal{O}(n^2)$, solve $\mathcal{O}(n^{4/3})$, SpMV $\mathcal{O}(n)$.

Sparse specific features of LU-IR3

Fill-in in sparse direct solvers : $\mathrm{nnz}(A) \ll \mathrm{nnz}(LU)$.

- $n \times n$ matrix from a regular 3D geometry $\Rightarrow \mathrm{nnz}(A) = \mathcal{O}(n)$ and $\mathrm{nnz}(LU) = \mathcal{O}(n^{4/3})$.
- Operations complexities : facto $\mathcal{O}(n^2)$, solve $\mathcal{O}(n^{4/3})$, SpMV $\mathcal{O}(n)$.

Thus, sparse LU-IR3 has **important specific features** :

- SpMV much cheaper than solve $\Rightarrow u_r \ll u$ has limited impact on performance (even for $u_r =$ fp128).
- Memory space of A in $u_r \leq u$ negligible in comparison of the LU factors in $u_f \Rightarrow$ LU-IR3 saves memory on LU factors in u !

Practical issues of GMRES-IR3

In GMRES-IR3, LU solves are performed at precision u^2 : this is a major practical issue.

Practical issues of GMRES-IR3

In GMRES-IR3, LU solves are performed at precision u^2 : this is a major practical issue.

- Increases cost per iteration.
- If u^2 is fp128, requires a quad precision sparse solver.
- Cast the LU factors from precision u_f to precision u^2
  $\Rightarrow$ huge memory increase
  (Alternative : cast LU factors on the fly, chunk by chunk $\Rightarrow$ cost per iteration explodes)

Practical issues of GMRES-IR3

In GMRES-IR3, LU solves are performed at precision u^2 : this is a major practical issue.

- Increases cost per iteration.
- If u^2 is fp128, requires a quad precision sparse solver.
- Cast the LU factors from precision u_f to precision u^2
  $\Rightarrow$ huge memory increase
  (Alternative : cast LU factors on the fly, chunk by chunk $\Rightarrow$ cost per iteration explodes)

$\Rightarrow$ What if we relax the precision u^2 on the preconditioning?

GMRES-IR5

Algorithm 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**

GMRES-IR5

Algorithm 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**

GMRES-IR5

Algorithm 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**

Possibly $u_p \gg u^2$ (and $u_g \gg u$).

Theoretical results

IR	Convergence condition	(u_g, u_p)
LU-IR3	$\kappa(A)u_f \ll 1$	Eq. $u_g = u_f$ and $u_p = u_f$
GMRES-IR5	$(\mathbf{u_g} + \mathbf{u_p}\kappa(\mathbf{A}))\kappa(\mathbf{A})^2\mathbf{u_f^2} \ll 1$	**u_g and u_p indep.**
GMRES-IR3	$\kappa(A)u^{1/2}u_f \ll 1$	$u_g = u$ and $u_p = u^2$

The limiting accuracy does not change. It is solver independent.

Implemented methods for double precision solution

H = fp16 ; S = fp32 ; D = fp64 ; Q = fp128.

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

Implemented methods for double precision solution

H = fp16; S = fp32; D = fp64; Q = fp128.

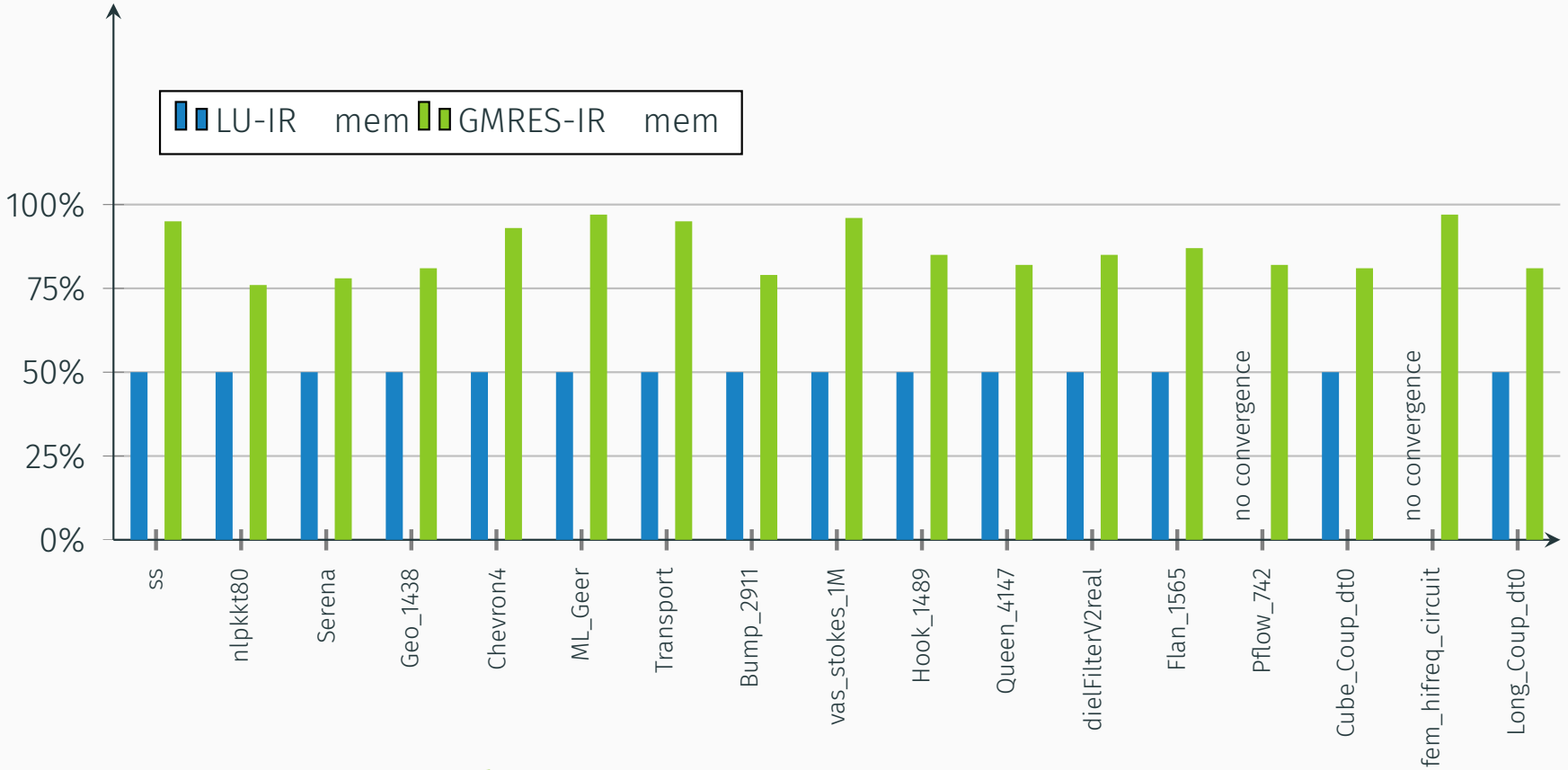
Solver	u_f	u	u_r	u_g	u_p	$\max(\kappa(A))$	ferr
DMUMPS	fp64 LU standard solver					—	$\kappa(A) \times 10^{-16}$
LU-IR	S	D	D	—	—	2×10^7	$\kappa(A) \times 10^{-16}$
GMRES-IR	S	D	D	D	D	1×10^{10}	$\kappa(A) \times 10^{-16}$
LU-IR+	S	D	Q	—	—	2×10^7	10^{-16}
GMRES-IR+	S	D	Q	D	D	1×10^{10}	10^{-16}

For the GMRES variants since $u_p \neq u_f$, the LU factors need to be casted.

All IR variants use single precision factorization to accelerate.

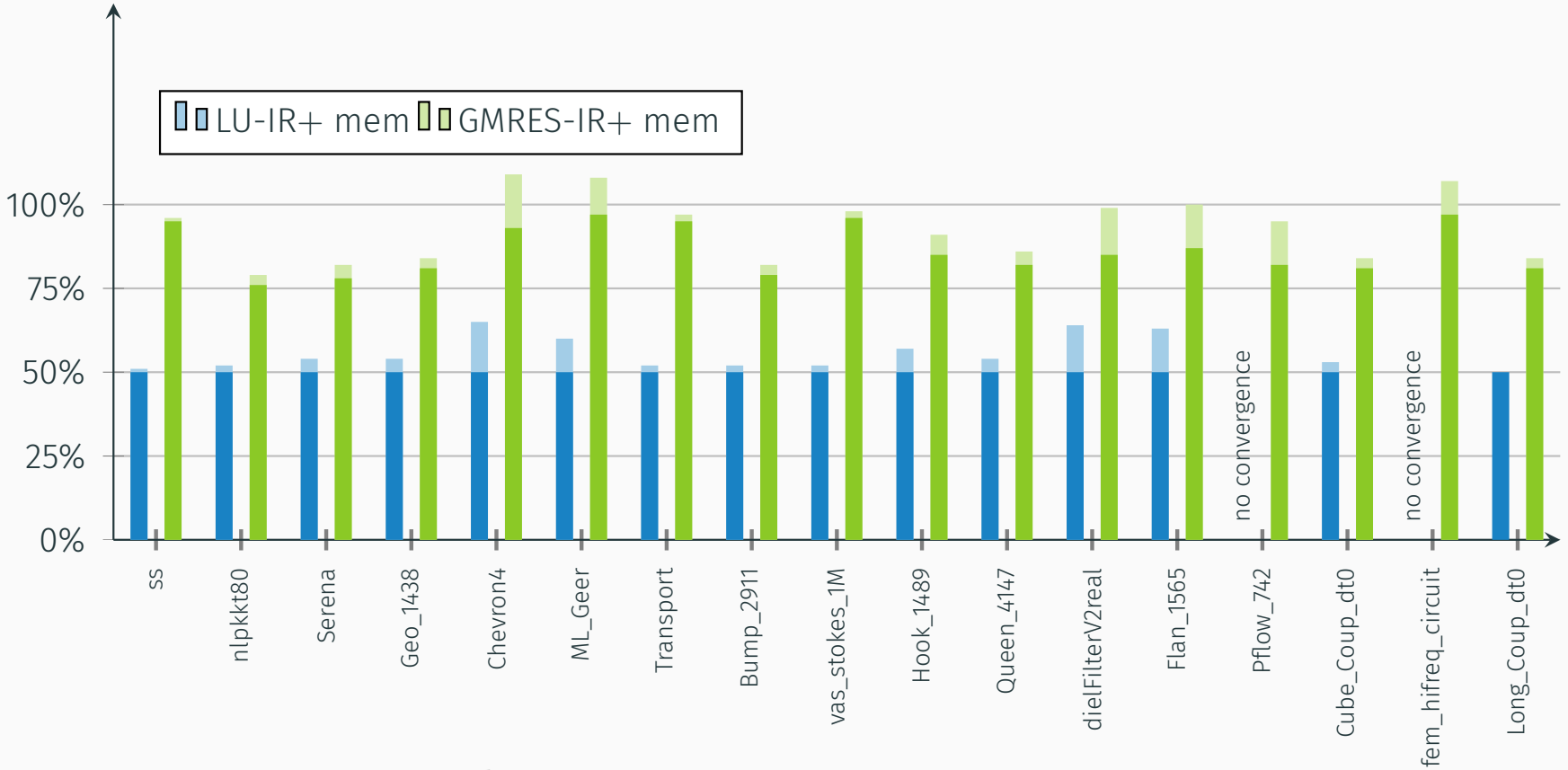
We use the multifrontal solver MUMPS for factorization and LU solve operations.

Memory Performance (No MPI - 18 threads)



- LU-IR consumes 2× less memory!
- GMRES-IR gains at best 25% despite factors in double ⇒ multifrontal solver mem peak > mem factors.

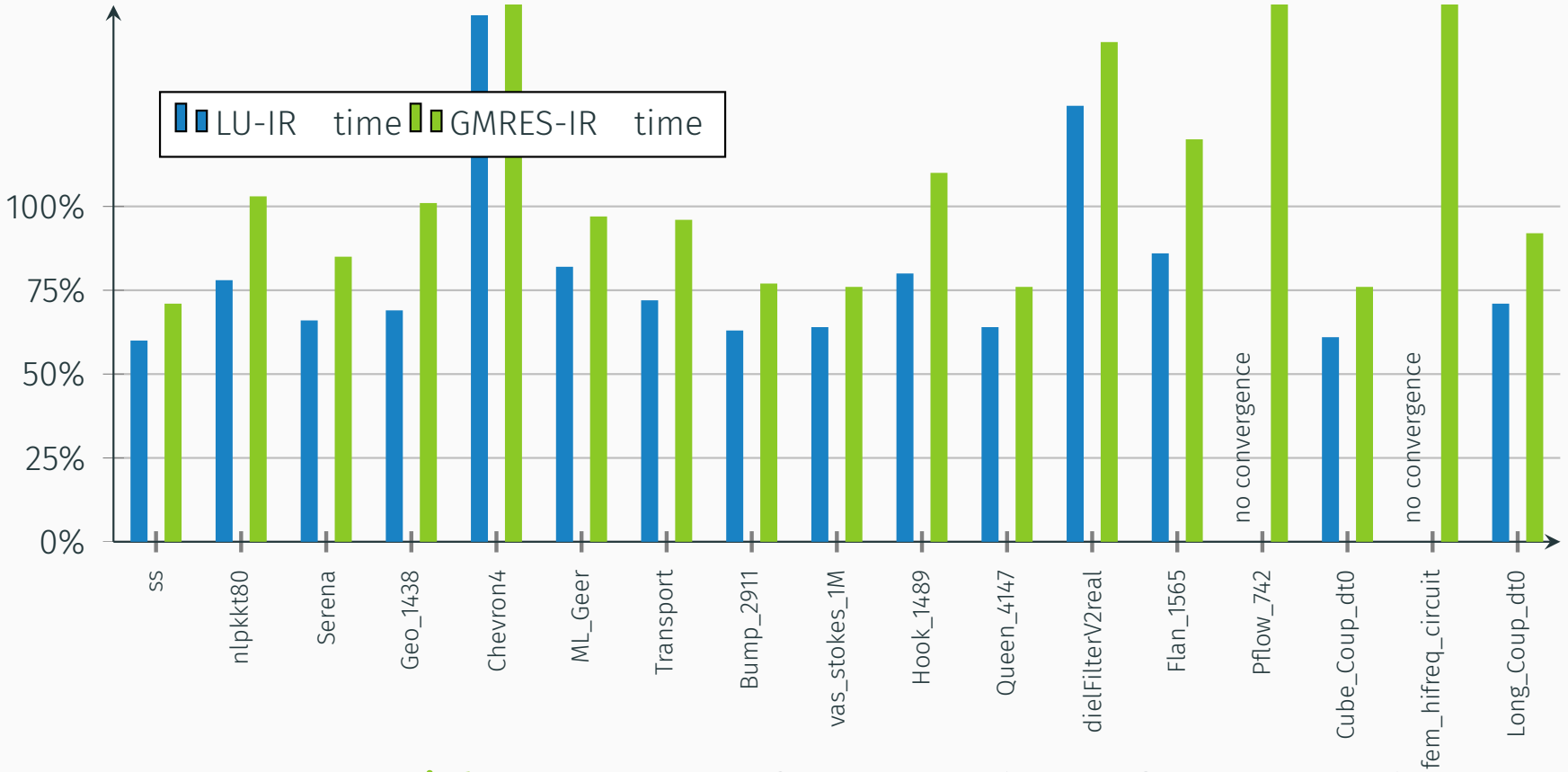
Memory Performance (No MPI - 18 threads)



- LU-IR consumes **2× less memory**!
- **GMRES-IR gains** at best 25% despite factors in double ⇒ multifrontal solver mem peak > mem factors.

- **Overhead** to reach a 10^{-16} ferr is on most matrices negligible.
- Heavy overhead when : small matrices, small fill-in.

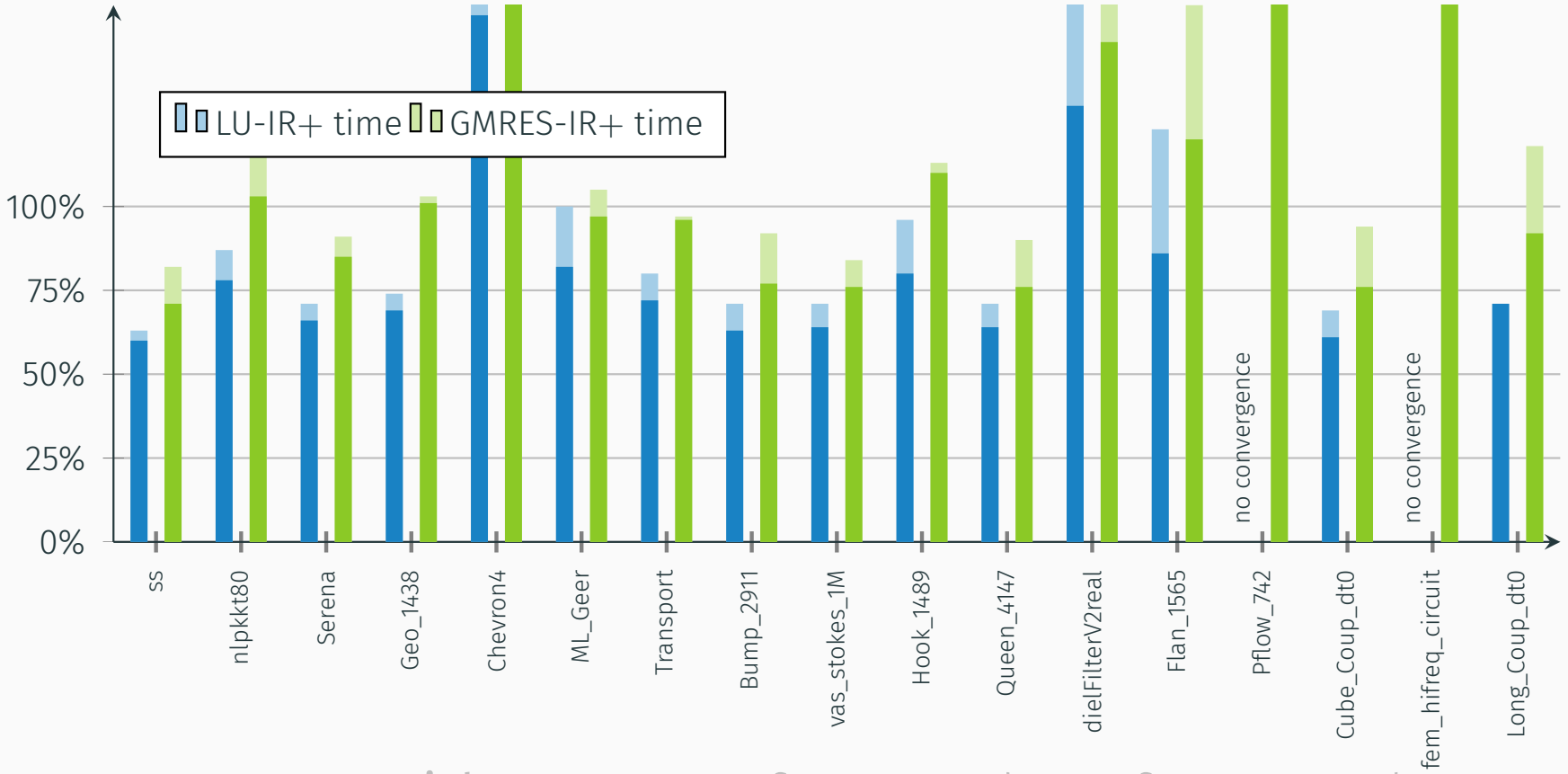
Time Performance (No MPI - 18 threads)



- LU-IR 1.4–1.7× quicker on most of the matrices!
- GMRES-IR slower than LU-IR (ask more LU solves), but converges on all the matrices ⇒ more robust.

- Underperformance when : small matrices, small fill-in, high number of iterations.

Time Performance (No MPI - 18 threads)



- LU-IR **1.4–1.7× quicker** on most of the matrices!
- GMRES-IR **slower** than LU-IR (ask more LU solves), but converges on all the matrices ⇒ **more robust**.
- Underperformance when : **small matrices**, **small fill-in**, **high number of iterations**.
- **Overhead** to reach a 10^{-16} ferr is on most matrices reasonable.

Accelerate with half precision factorization ?

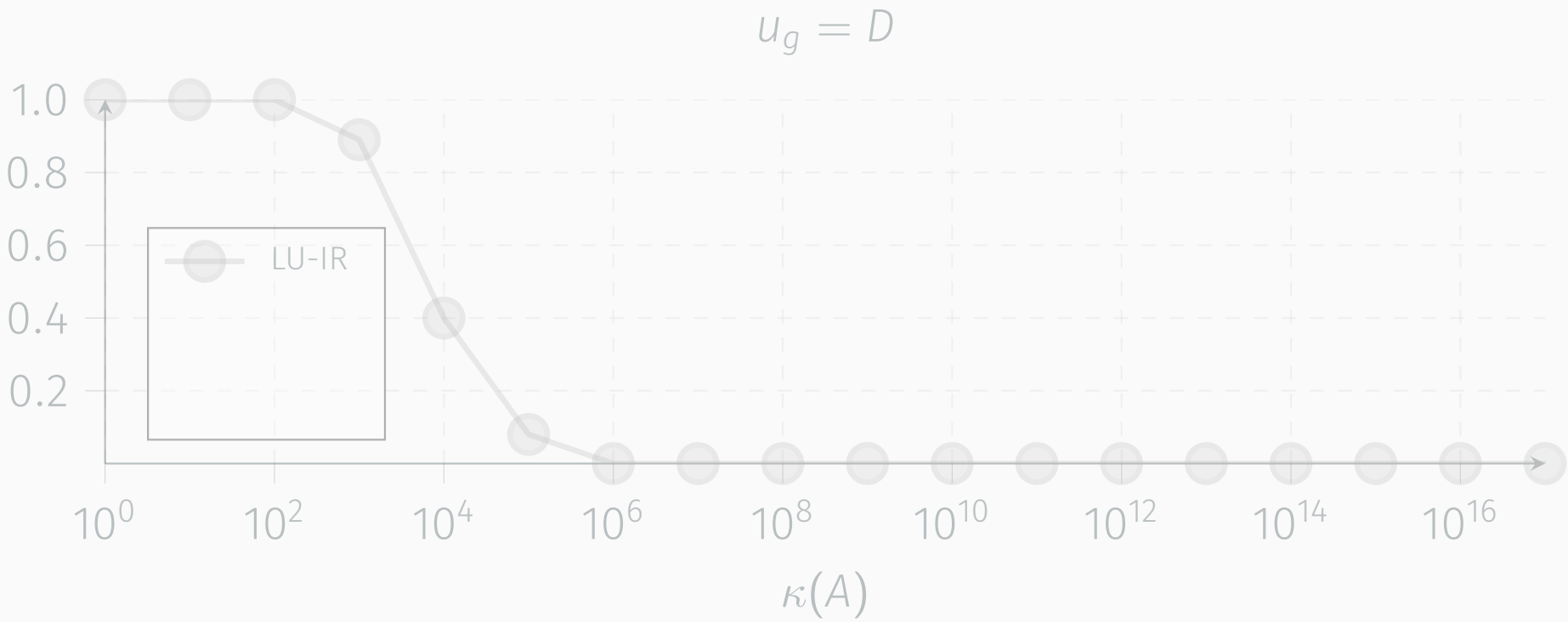
Performance study with $u_f = H$ is complicated :

- Not easy access to half precision on CPUs.
- Direct sparse solving strategies using GPUs are not straightforward.

To anticipate the availability of half on CPUs we made a **numerical study** of GMRES-IR5 with $u_f = H$:

- Focus on robustness on $\kappa(A) \Rightarrow$ main interest of GMRES-IR5 over LU-IR3.
- Made in **Julia** lang : powerful modularity on the precisions, can simulate half and quad.
- Made on **small dense matrices** of different $\kappa(A)$.

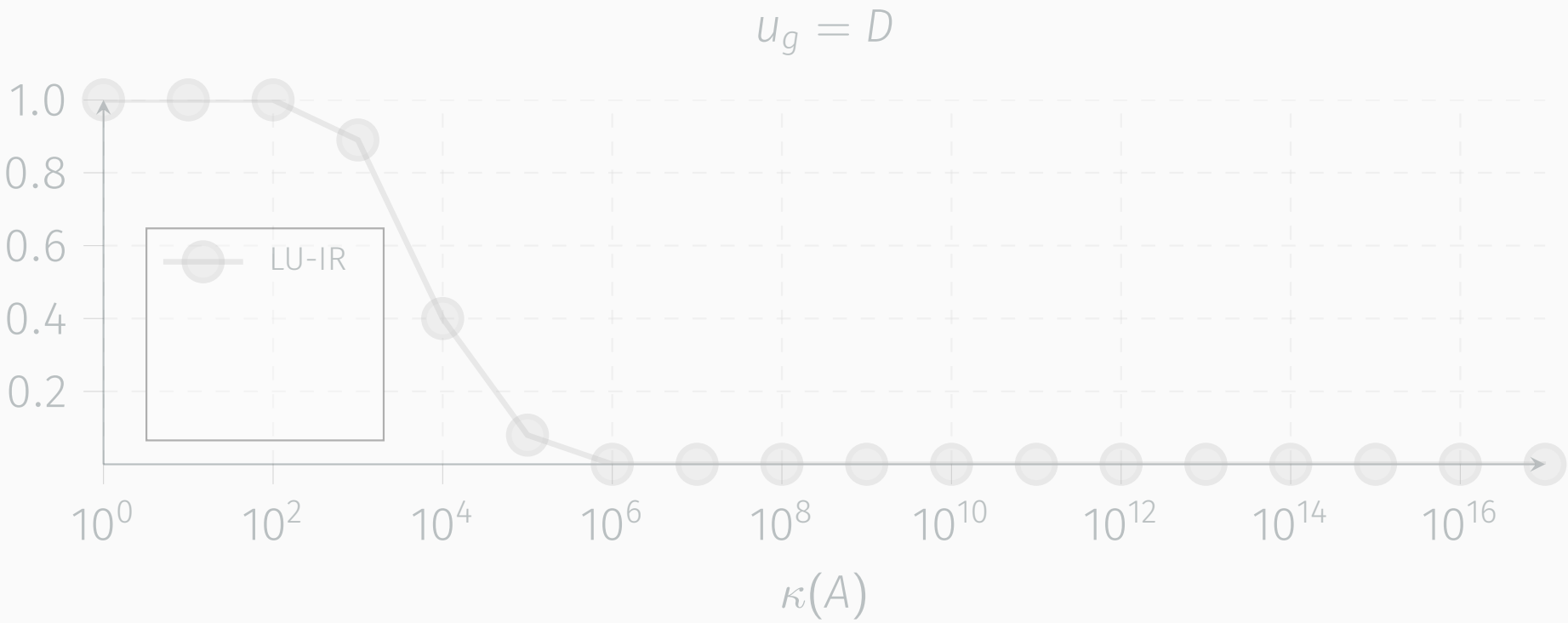
Numerical experiments $u_f = H$: robustness on $\kappa(A)$



We want to study the convergence **robustness** on $\kappa(A)$ of the following variants :

- $\mathbf{u_f} = H$, $\mathbf{u} = D$, $\mathbf{u_r} = Q$ **fixed**.
- GMRES precision $\mathbf{u_g}$ and preconditioning precision $\mathbf{u_p}$ **varying**.

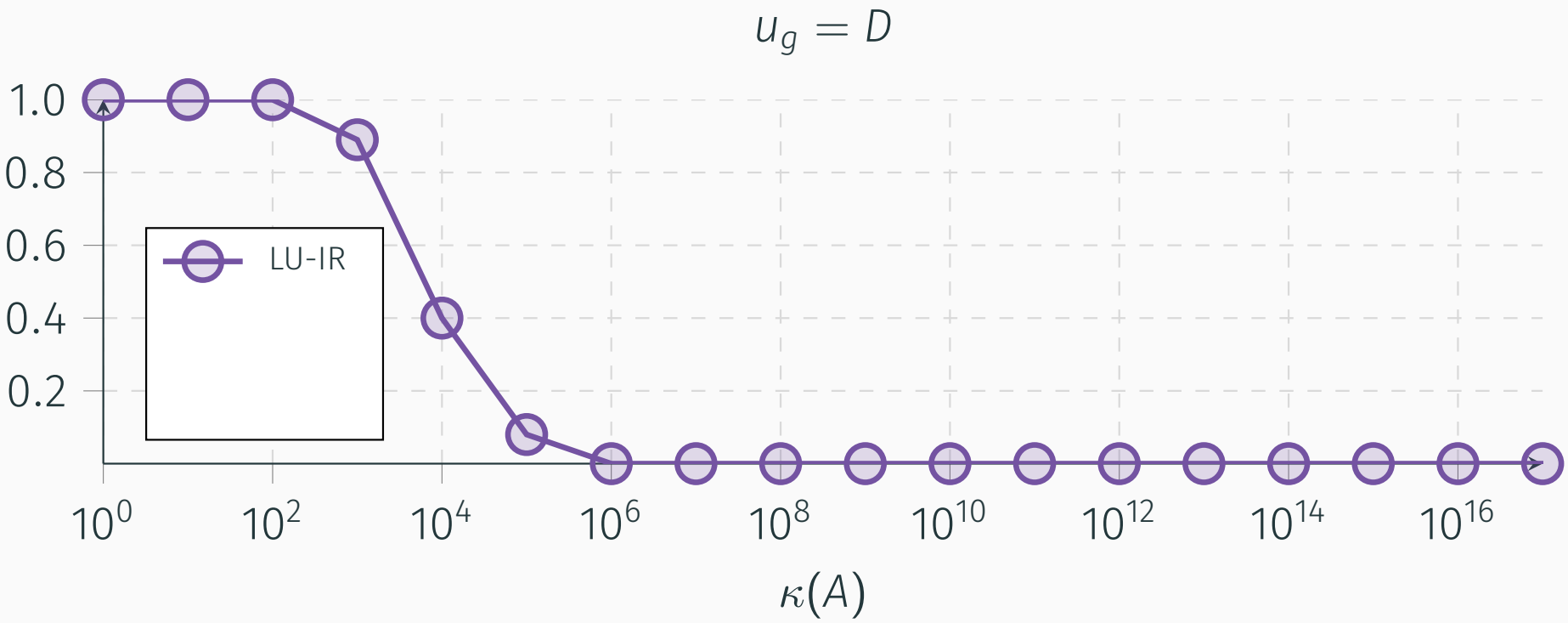
Numerical experiments $u_f = H$: robustness on $\kappa(A)$



Evaluate the robustness ?

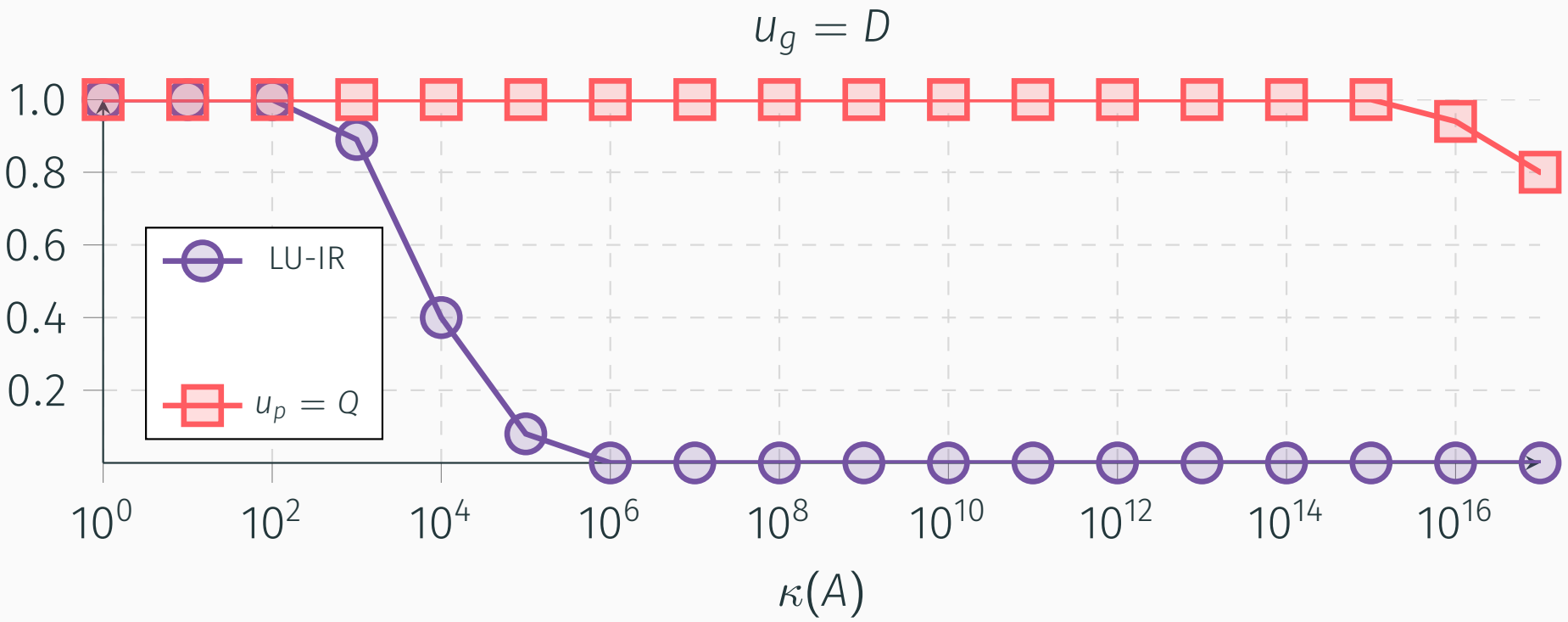
For $u_r = Q$ ferr should converge to 10^{-16} $\Rightarrow$ Easy to determine if it converges well or not. **Success rate of convergence** seems to be a good measure.

Numerical experiments $u_f = H$: robustness on $\kappa(A)$



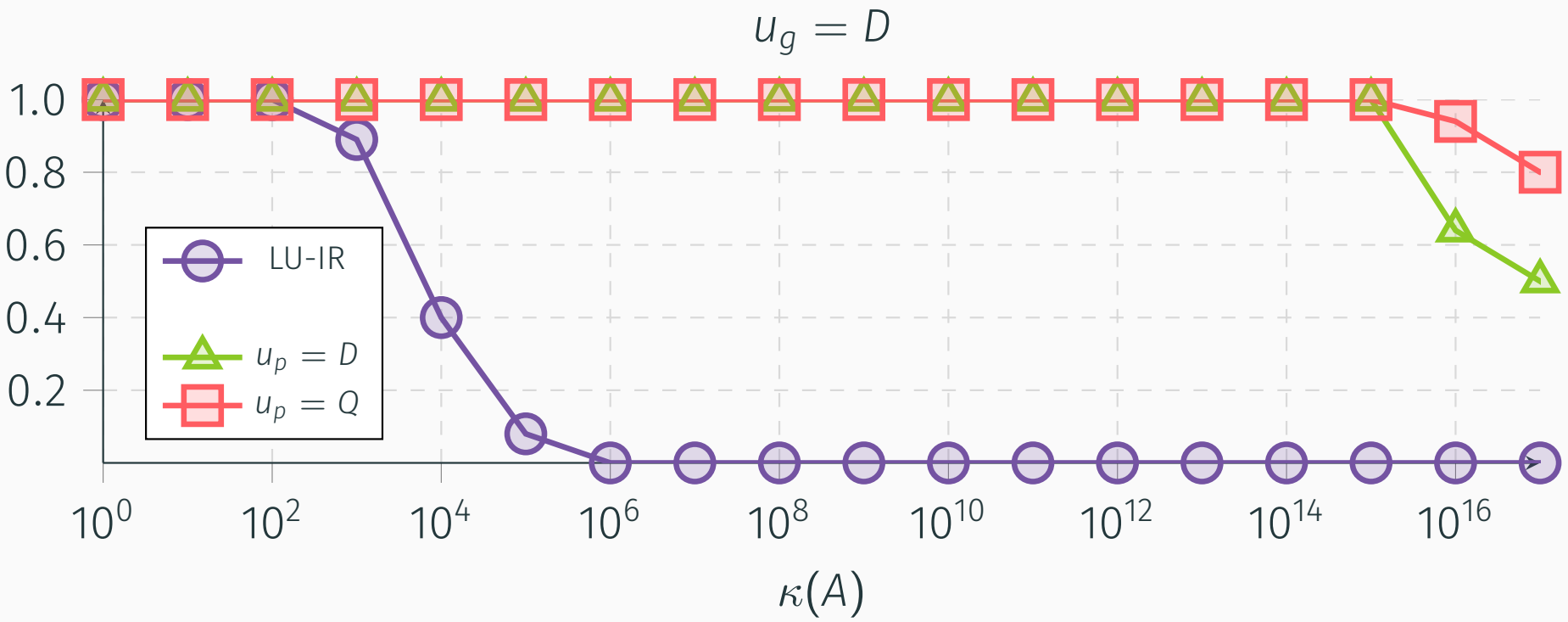
- Success rate of convergence of LU-IR3 and GMRES-IR5. Each success rate computed from 100 50×50 randsvd dense matrices.
- The more **the breaking point** of the success rate is high the more the method is robust (ex : LU-IR $\approx 10^2 - 10^3$).

Numerical experiments $u_f = H$: robustness on $\kappa(A)$



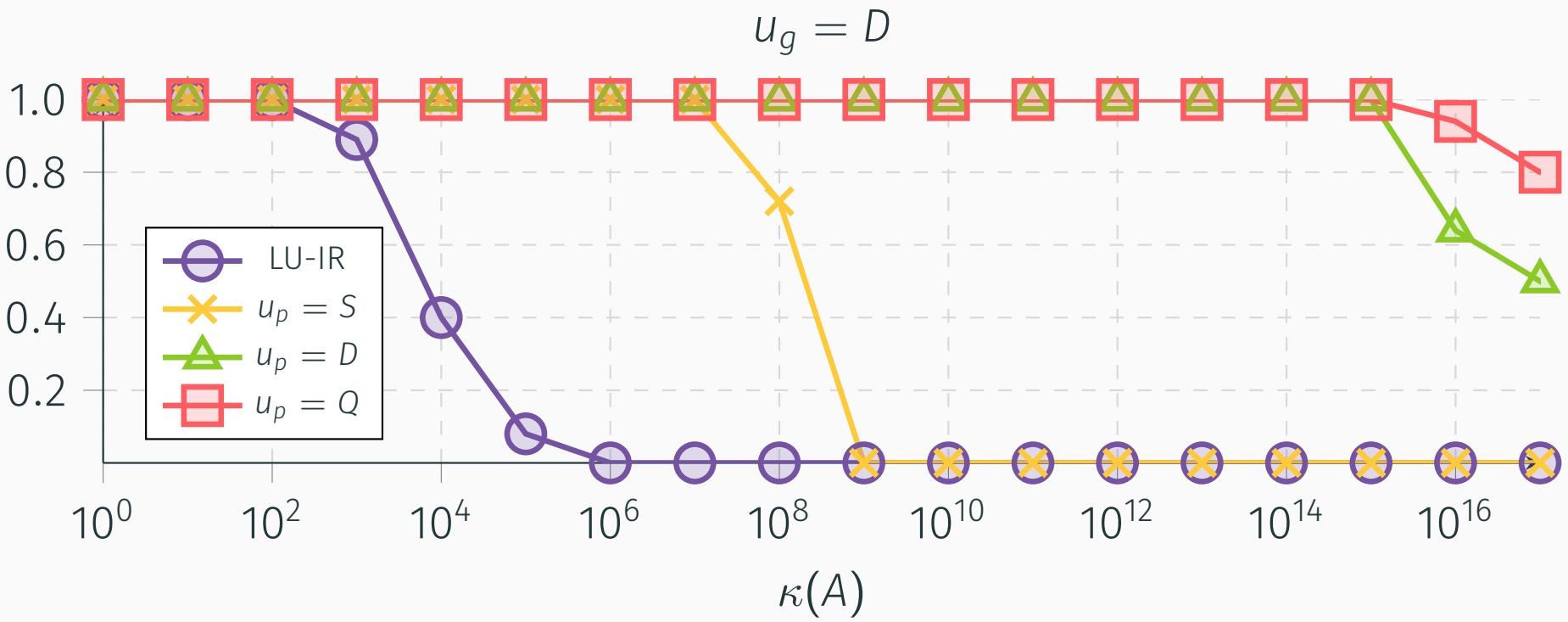
- Success rate of convergence of LU-IR3 and GMRES-IR5. Each success rate computed from 100 50×50 randsvd dense matrices.
- The more **the breaking point** of the success rate is high the more the method is robust (ex : LU-IR $\approx 10^2 - 10^3$).

Numerical experiments $u_f = H$: robustness on $\kappa(A)$



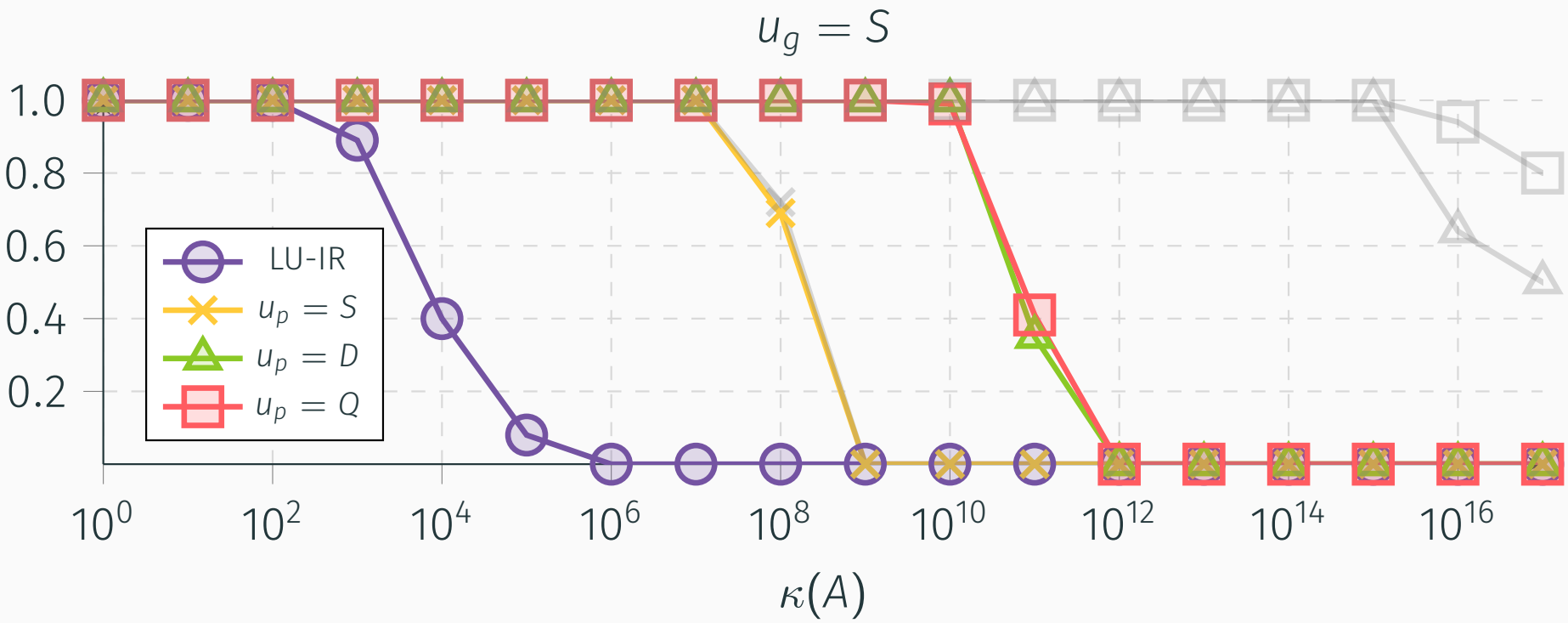
- Success rate of convergence of LU-IR3 and GMRES-IR5. Each success rate computed from 100 50×50 randsvd dense matrices.
- The more **the breaking point** of the success rate is high the more the method is robust (ex : LU-IR $\approx 10^2 - 10^3$).

Numerical experiments $u_f = H$: robustness on $\kappa(A)$



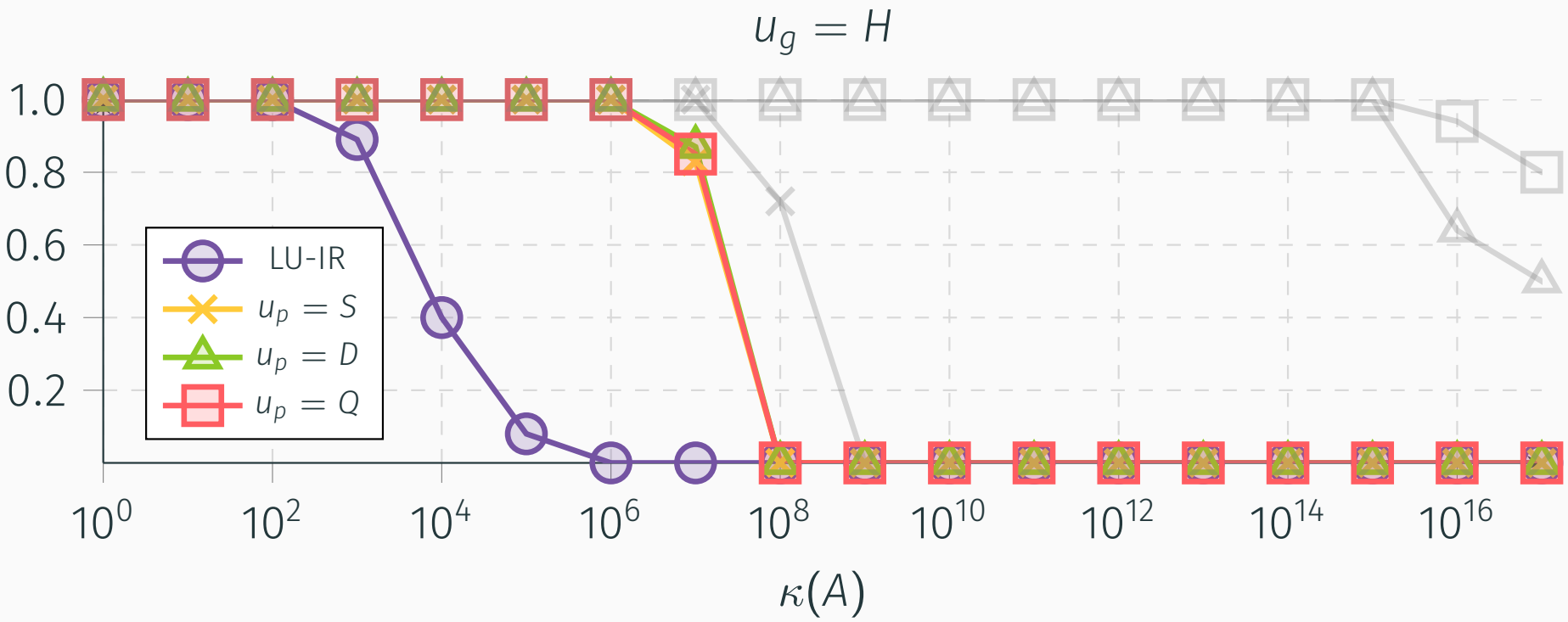
- Success rate of convergence of LU-IR3 and GMRES-IR5. Each success rate computed from 100 50×50 randsvd dense matrices.
- The more **the breaking point** of the success rate is high the more the method is robust (ex : LU-IR $\approx 10^2 - 10^3$).

Numerical experiments $u_f = H$: robustness on $\kappa(A)$



- Success rate of convergence of LU-IR3 and GMRES-IR5. Each success rate computed from 100 50×50 randsvd dense matrices.
- The more **the breaking point** of the success rate is high the more the method is robust (ex : LU-IR $\approx 10^2 - 10^3$).

Numerical experiments $u_f = H$: robustness on $\kappa(A)$



- Success rate of convergence of LU-IR3 and GMRES-IR5. Each success rate computed from 100 50×50 randsvd dense matrices.
- The more **the breaking point** of the success rate is high the more the method is robust (ex : LU-IR $\approx 10^2 - 10^3$).

Conclusion

Contributions

- **GMRES-IR5 + error analysis :** use better combinations of precisions for sparse solving.
- **Large experimental analysis :** Highlight of memory and time gains + limitations of LU-IR3 and GMRES-IR5.

In-Coming : Distributed memory, weak and strong scaling study.

- Increase the gap memory peak–factors in **strong scaling** $\Rightarrow$ increase memory gain of GMRES-IR5.
- Better asymptotic properties in **weak scaling** $\Rightarrow$ increase time gain of GMRES-IR5.

Further experiments : Half precision on CPUs and GPUs.

Amestoy, Buttari, Higham, L'Excellent, Mary, Vieublé. *"GMRES-based iterative refinement in five precisions : error analysis and application to sparse direct solvers"*. In : Submission soon.