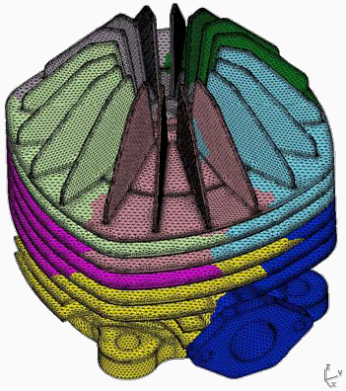
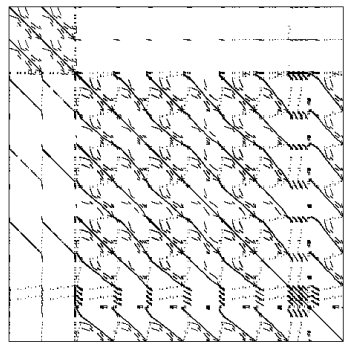
Combining sparse approximate factorizations with mixed precision iterative refinement

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

ISC 2022

Sparse Direct Solver

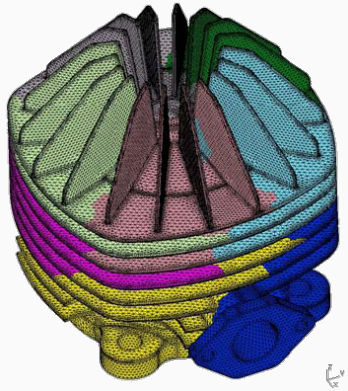
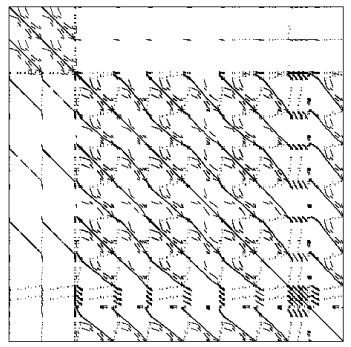

Large Sparse Linear System $Ax = b$

At the foundations of many scientific computing applications (discretization of PDEs, step of an optimzation method, ...).

Direct solver properties

- Pros : Robust, easy to use, accurate.
- Cons : Compute and memory intensive, limited scalability.

Sparse Direct Solver


Large Sparse Linear System $Ax = b$

At the foundations of many scientific computing applications (discretization of PDEs, step of an optimzation method, ...).

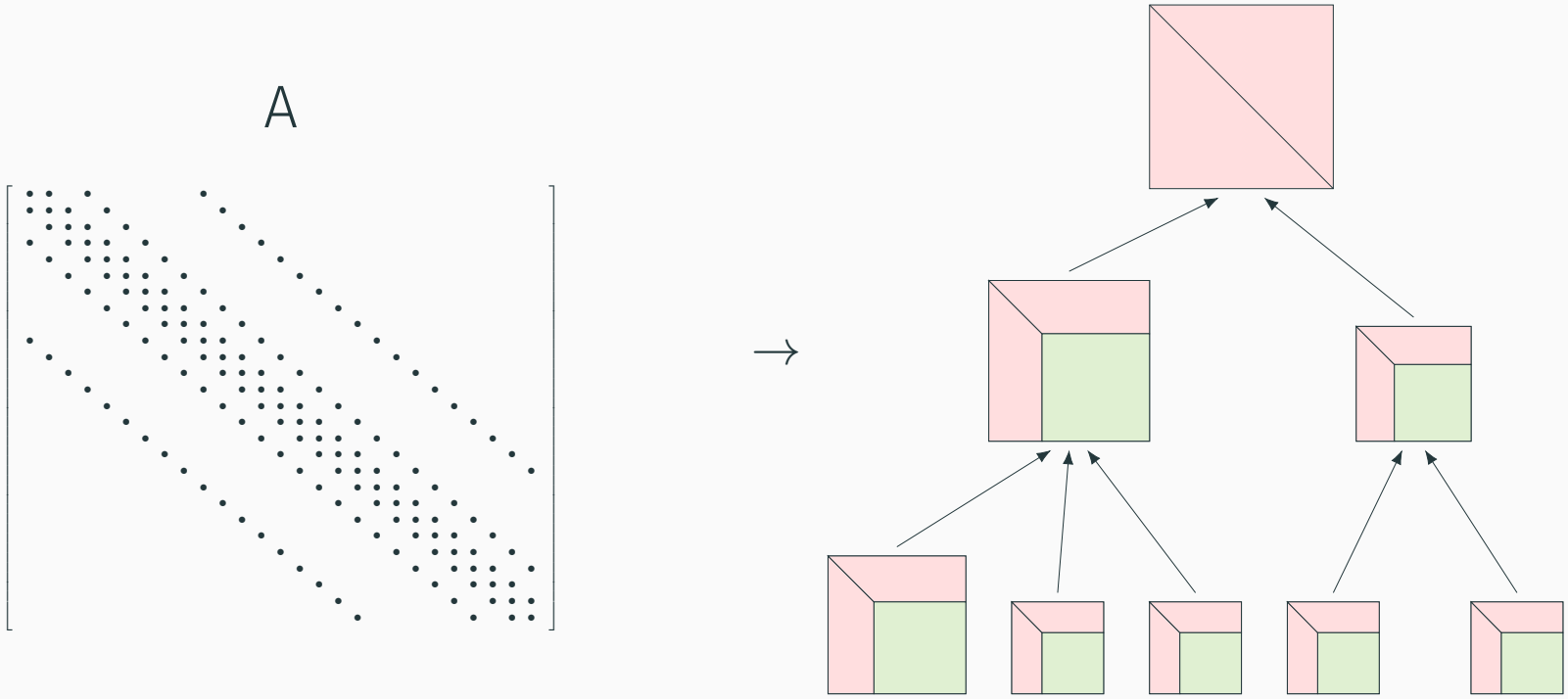
Direct solver properties

- Pros : Robust, easy to use, accurate.
- Cons : Compute and memory intensive, limited scalability.

$\Rightarrow$ **Reduce the complexity :** Numerical approximations, low arithmetic precisions.

Multifrontal Sparse Direct Factorization

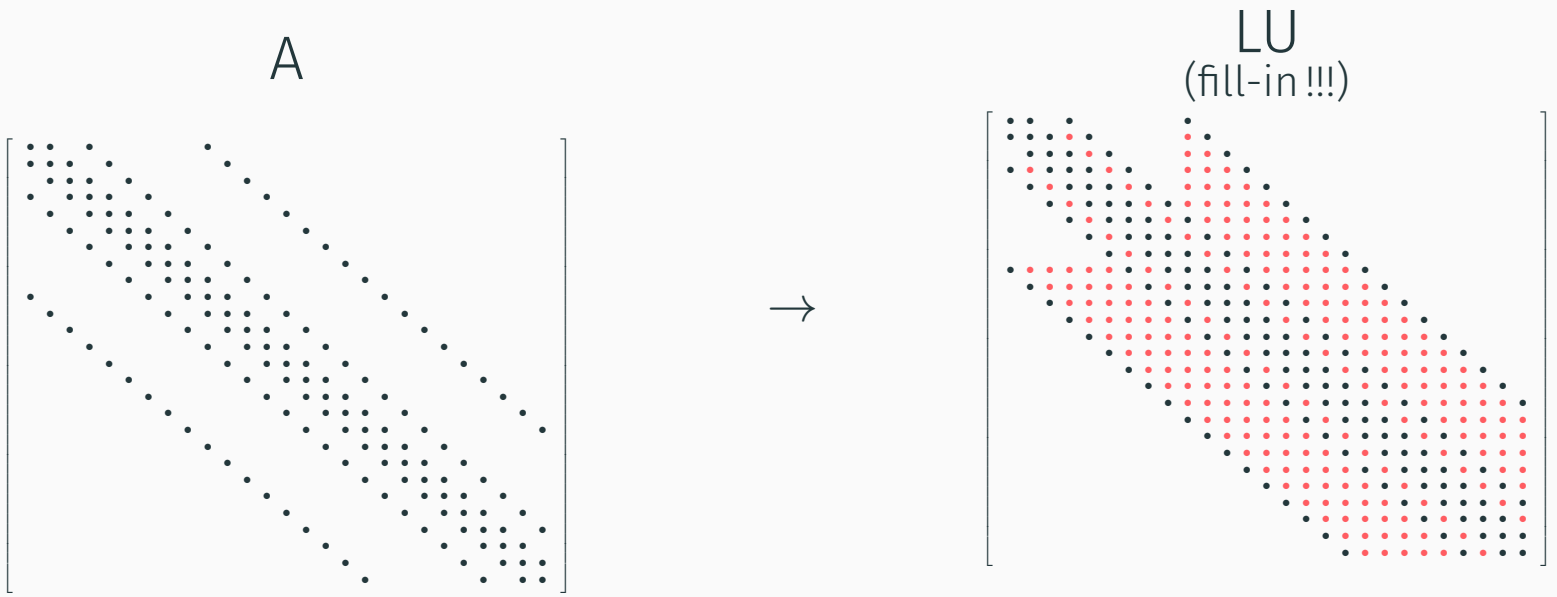
A sparse factorization can be decomposed into a series of factorizations of dense matrices (fronts) :



- 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.

Multifrontal Sparse Direct Factorization

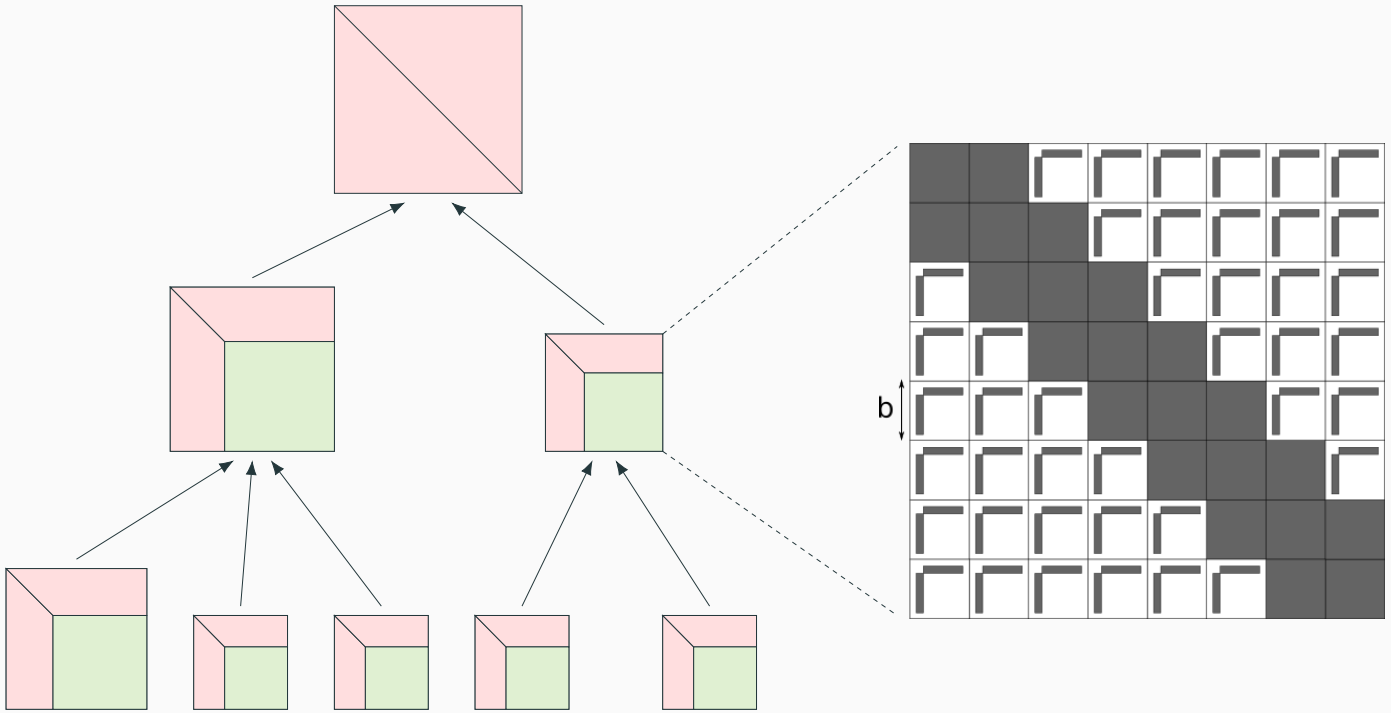
A sparse factorization can be decomposed into a series of factorizations of dense matrices (fronts) :



- 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.

Numerical approximations : Block-Low-Rank

Block-Low-Rank[1] **:** Decompose the dense matrices into regular blocks of size b. Try to compress each block with a low rank approximation at precision ϵ_{BLR}.



1. Théo Mary, *Block Low-Rank multifrontal solvers : complexity, performance, and scalability*, 2017

Numerical approximations : Block-Low-Rank

Block-Low-Rank : Decompose the 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	$O(n^2)$	$O(n^{\frac{4}{3}})$
BLR	$O(n^{\frac{4}{3}})$	$O(n \log(n))$

Complexities on cubic domain

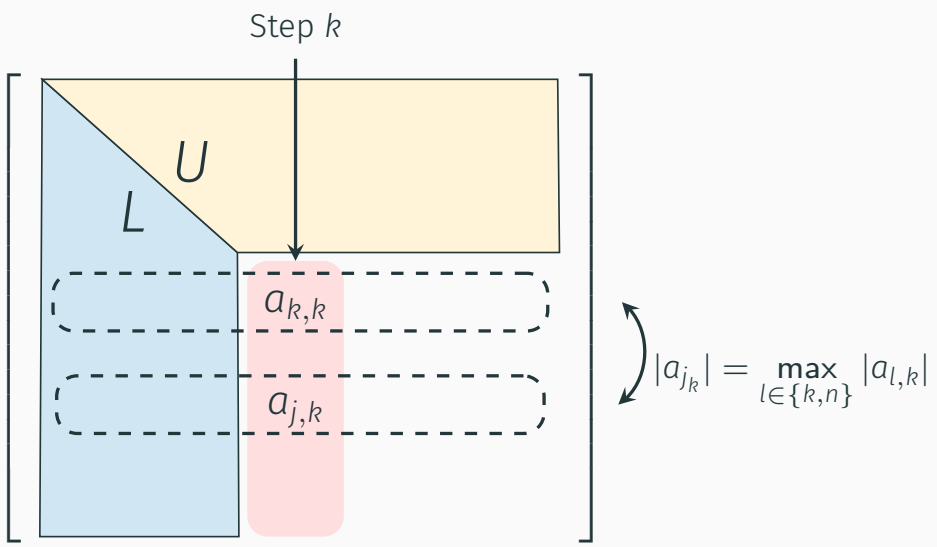
Pros :

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

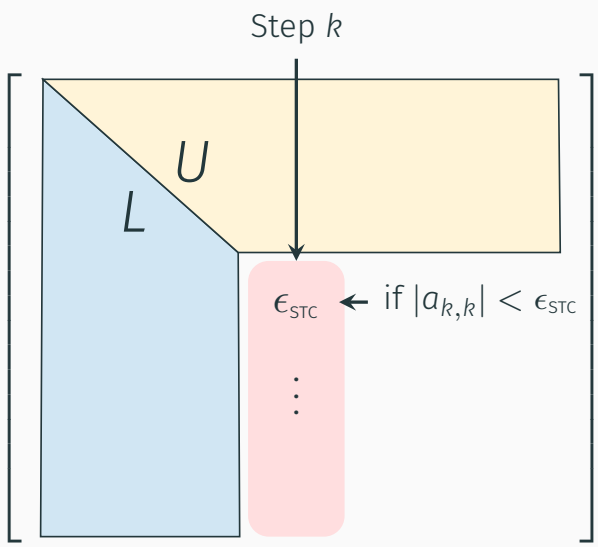
Cons :

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

Numerical approximations : Static pivoting



Partial Pivoting



Static Pivoting [2]

Pros :

- ➤ No pivot search overhead.
- ➤ No synch communications for pivoting.
- ➤ Keep the ordering of the analysis.

Cons :

- ➤ Large ϵ_{STC} less accuracy.
- ➤ Low ϵ_{STC} less stability.

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

Why using numerical approximations?

The philosophy is to 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!
- Sometimes the bigger the perturbations the bigger the savings (e.g. BLR).

BUT big perturbations = low accuracy

Mixed-precision iterative refinement for direct solvers

Algorithm Iterative refinement

1 : Compute the LU factorization $A = \hat{L}\hat{U}$ (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 $Ad_i = r_i$. (u_s)
6 : Compute $x_{i+1} = x_i + d_i$ (u)
7 : **end while**

The strategy is to accelerate with low precisions the factorization and recover a good accuracy with higher precisions on the correction iterations.

Mixed-precision iterative refinement for direct solvers

Algorithm Iterative refinement : **LU-IR3** [a]

1 : Compute the LU factorization $A = \hat{L}\hat{U}$ (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**

We can change the solver used for the correction equation :

➤ LU-IR3 : the classical form of refinement for direct solvers.

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

Mixed-precision iterative refinement for direct solvers

Algorithm Iterative refinement : **GMRES-IR5** [a]

1 : Compute the LU factorization $A = \hat{L}\hat{U}$ (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**

We can change the solver used for the correction equation :

➤ GMRES-IR5 : a more robust form capable of tackling higher condition number $\kappa(A)$.

a. Amestoy, Buttari, Higham, L'Excellent, Mary, and Vieublé, *Five-Precision GMRES-based iterative refinement*, 2021

Mixed-precision iterative refinement for direct solvers

Algorithm Iterative refinement : **GMRES-IR5**

1 : Compute the LU factorization $A = \hat{L}\hat{U}$ (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**

$\Rightarrow$ We want to use iterative refinement to improve the accuracy of sparse approximate solvers !

Question : What are the specificities of IR with sparse approximate solvers ?

Specificities with sparse factorization

Algorithm Iterative refinement : complexities Dense VS Sparse

1 : Compute the LU factorization $A = \hat{L}\hat{U}$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	(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. $\mathbf{NNZ}(A) \ll \mathbf{NNZ}(LU)$!

Specificities with sparse factorization

Algorithm Iterative refinement : complexities Dense VS Sparse

1 : Compute the LU factorization $A = \hat{L}\hat{U}$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	(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. $\mathbf{NNZ}(A) \ll \mathbf{NNZ}(LU)$!

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

Specificities with sparse factorization

Algorithm Iterative refinement : complexities Dense VS Sparse

1 : Compute the LU factorization $A = \hat{L}\hat{U}$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	(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. $\mathbf{NNZ}(A) \ll \mathbf{NNZ}(LU)$!

➤ Memory space of A in u_r negligible compared with the LU factors in $u_f \Rightarrow$ LU-IR3 stores LU factors in precision u_f, direct solver stores in precision u, then LU-IR3 saves memory !

Specificities with sparse factorization

Algorithm Iterative refinement : complexities Dense VS Sparse			
1 : Compute the LU factorization $A = \hat{L}\hat{U}$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	(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. $\mathbf{NNZ}(A) \ll \mathbf{NNZ}(LU)$!

➤ If no cast on the fly, GMRES-IR5 needs to store the factors in u_p.
GMRES-IR5 saves memory on the active memory even if $u_p = u$!

Specificities with approximate factorization

Algorithm LU-IR3 : complexities Sparse VS Approximations [a]

1 :	Compute the LU factorization $A = \hat{L}\hat{U}$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^\alpha)$	$(u_f\rho_n + \epsilon)$
2 :	Solve $Ax_0 = b$	$\mathcal{O}(n^{4/3})$	$\mathcal{O}(n^\beta)$	$(u_f\rho_n + \epsilon)$
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 $d_i = \hat{U}^{-1}\hat{L}^{-1}r_i$.	$\mathcal{O}(n^{4/3})$	$\mathcal{O}(n^\beta)$	$(u_f\rho_n + \epsilon)$
6 :	Compute $x_{i+1} = x_i + d_i$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	(u)
7 :	**end while**			

- Where $2 \geq \alpha$ and $4/3 \geq \beta$.
- Where ϵ refers to the perturbation introduced, and ρ_n is the growth factor ($\approx$ difference of scale between the values of A and its factors L and U).

a. Amestoy, Buttari, Higham, L'Excellent, Mary, and Vieublé, *Combining sparse approximate factorizations with mixed precision iterative refinement*, 2022

Error analysis with numerical approximations

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

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

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

Where ϵ refers to the perturbation introduced, and ρ_n is the growth factor ($\approx$ difference of scale between the values of A and its factors L and U).

Remark : This theorem applies for generic numerical approximations, it includes static pivoting and BLR.

Implemented parallel methods

Solver	u_f	u	u_r	u_g	u_p	$\max(\kappa(A))$ $(\epsilon = 0)$	forward error
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}	$\kappa(A) \times 10^{-16}$

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

➤ LU-IR and GMRES-IR use **single precision (fp32) factorization** to accelerate. When using **BLR and/or static pivoting** we name the variants (BLR/STC)-(GMRES/LU)-IR.

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).

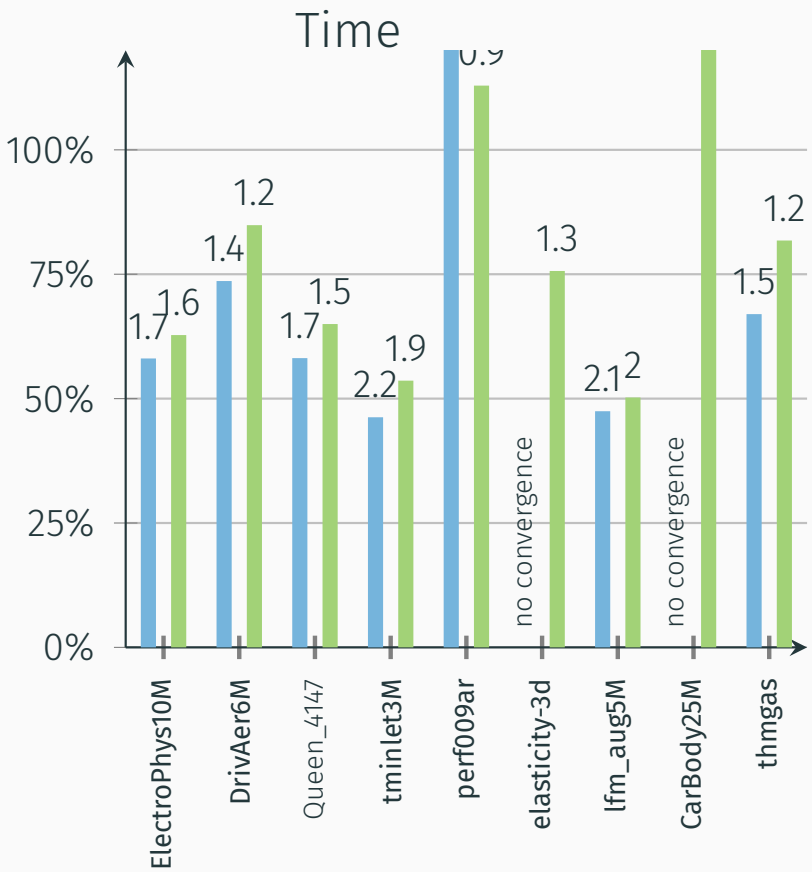
Matrix set

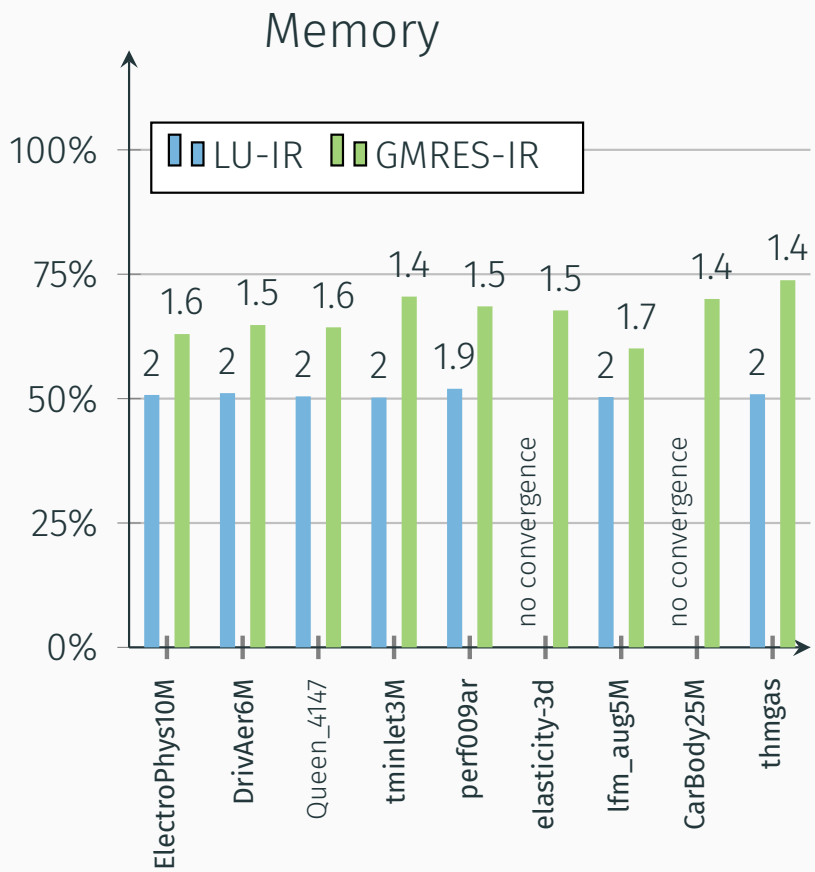
Name	N	NNZ	Arith.	Sym.	$\kappa(A)$	Fact. (flops)	Slv. (flops)
ElectroPhys10M	1.02E+07	1.41E+08	R	1	1.10E+01	4E+14	9E+10
DrivAer6M	6.11E+06	4.97E+07	R	1	9.40E+05	6E+13	3E+10
Queen_4147	4.14E+06	3.28E+08	R	1	4.30E+06	3E+14	6E+10
tminlet3M	2.84E+06	1.62E+08	C	0	2.70E+07	1E+14	2E+10
perf009ar	5.41E+06	2.08E+08	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.

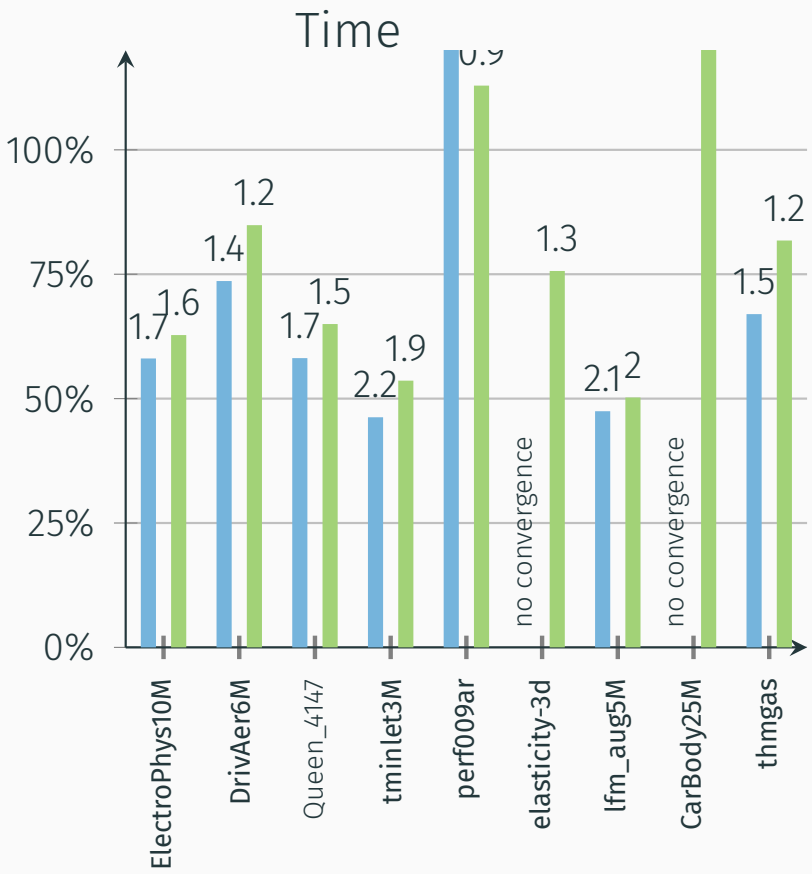
➤ We run on OLYMPE supercomputer nodes (two Intel 18-cores Skylake/node), 1 node (2MPI×18threads) or 2 nodes (4MPI×18threads) depending on the matrix size.

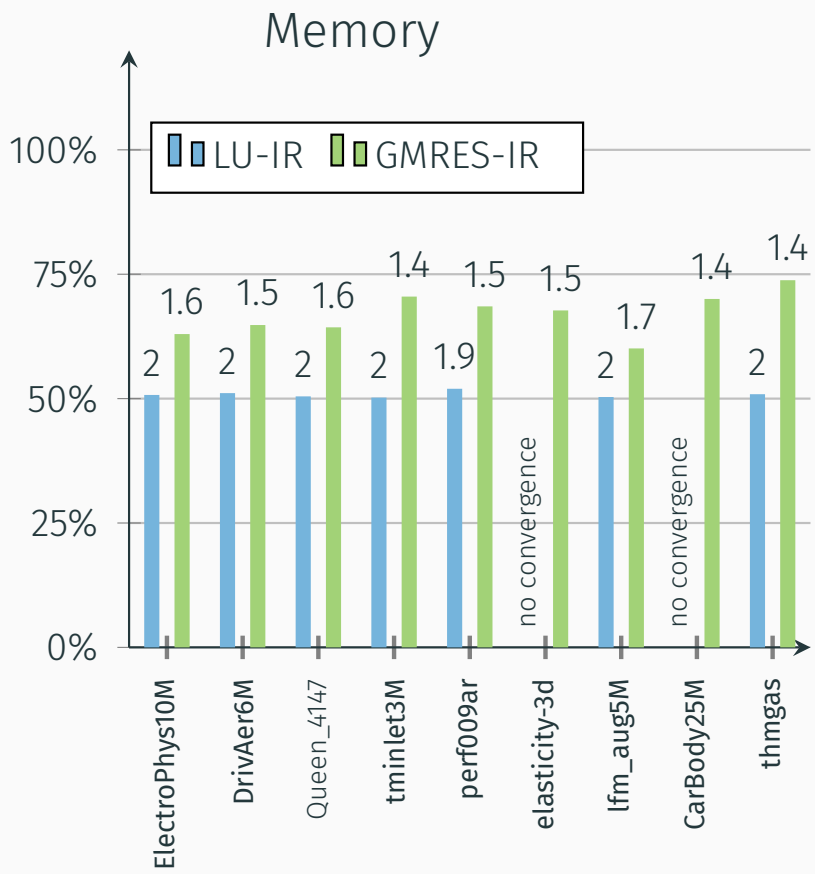
Time and memory performance w.r.t. DMUMPS





Time and memory performance w.r.t. DMUMPS

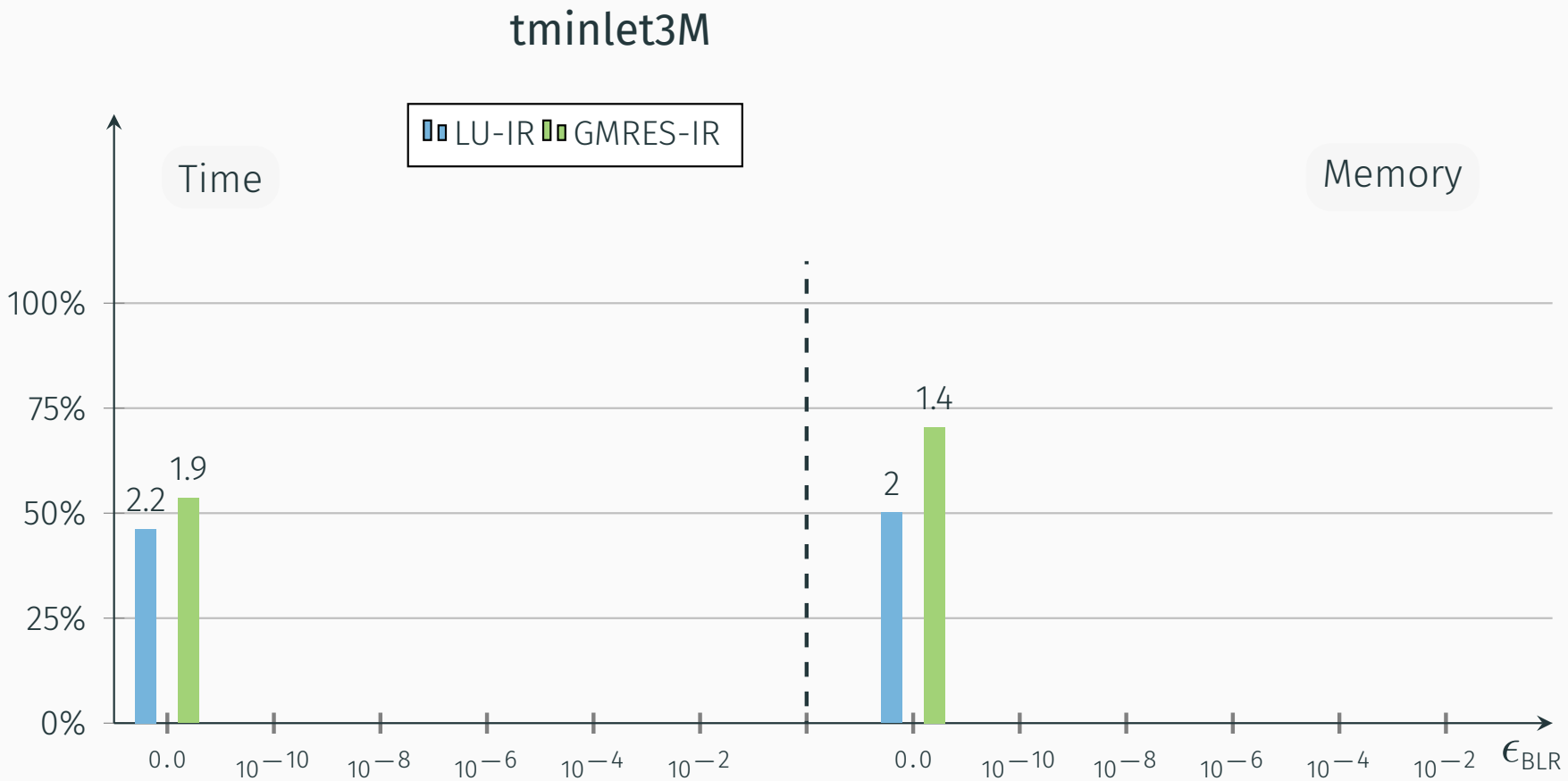




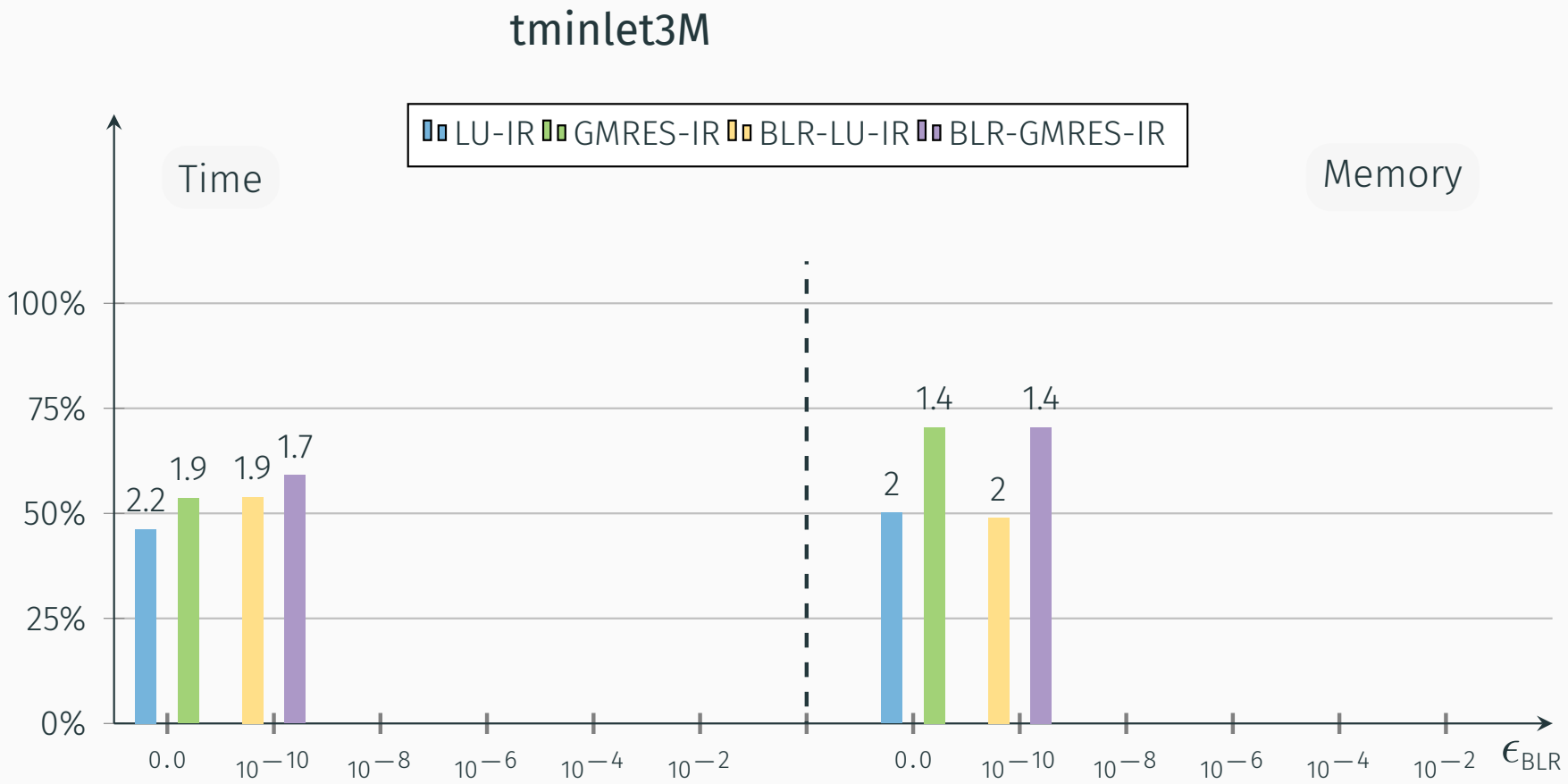
- LU-IR up to 2× faster.
- GMRES-IR slower (requires more LU solves), but more robust on $\kappa(A)$.

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

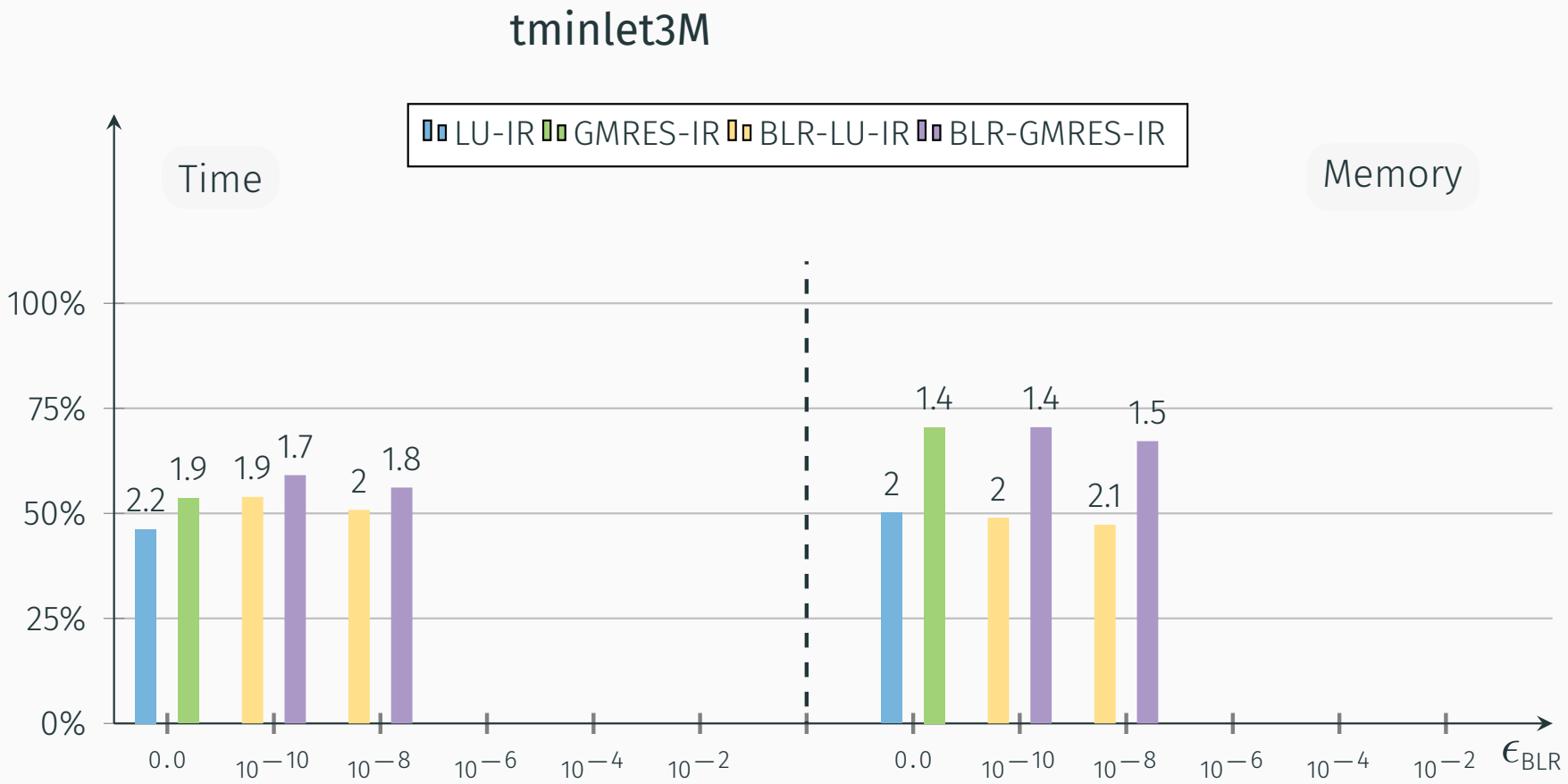
Time and memory performance with BLR w.r.t. DMUMPS



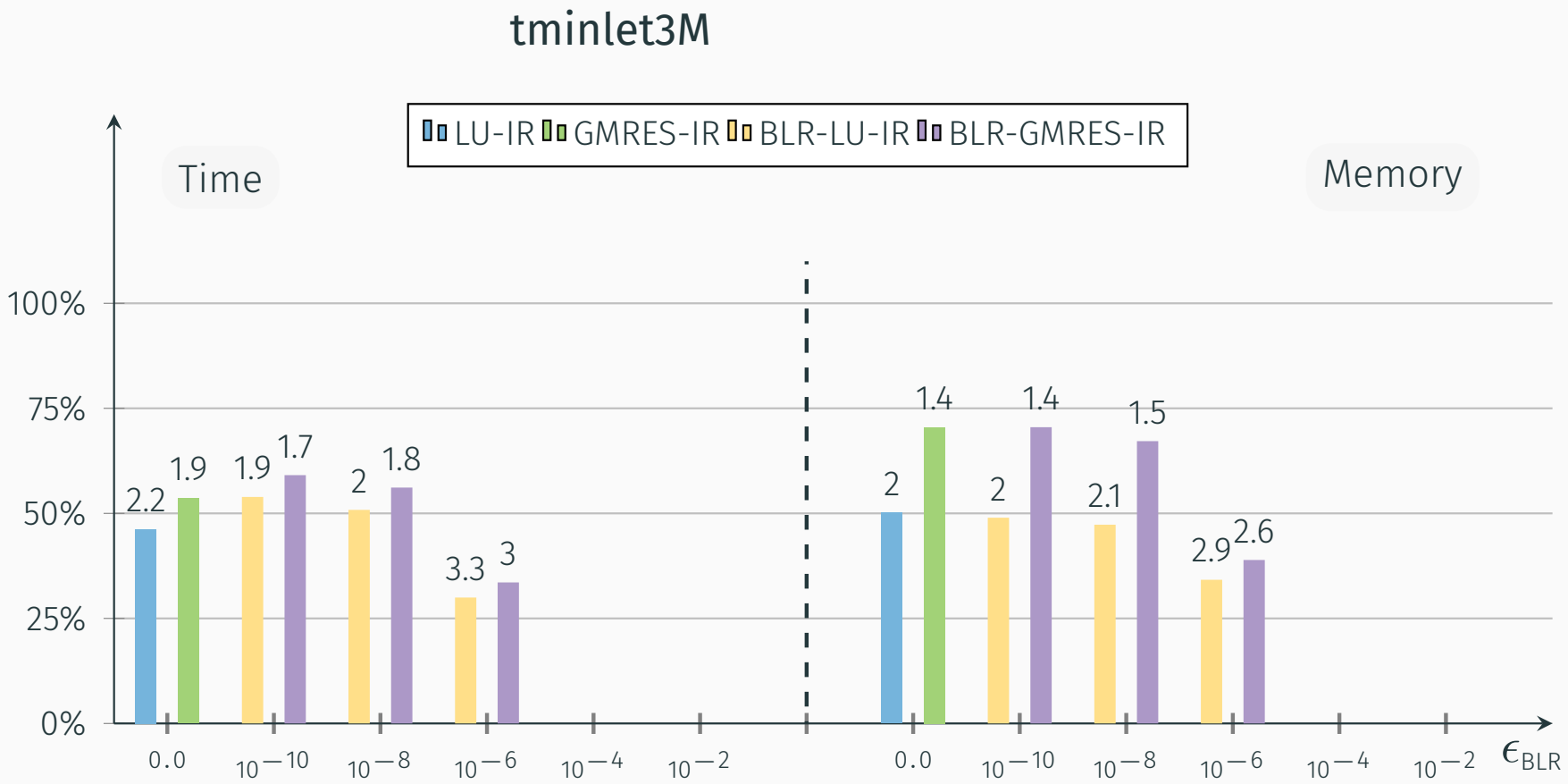
Time and memory performance with BLR w.r.t. DMUMPS



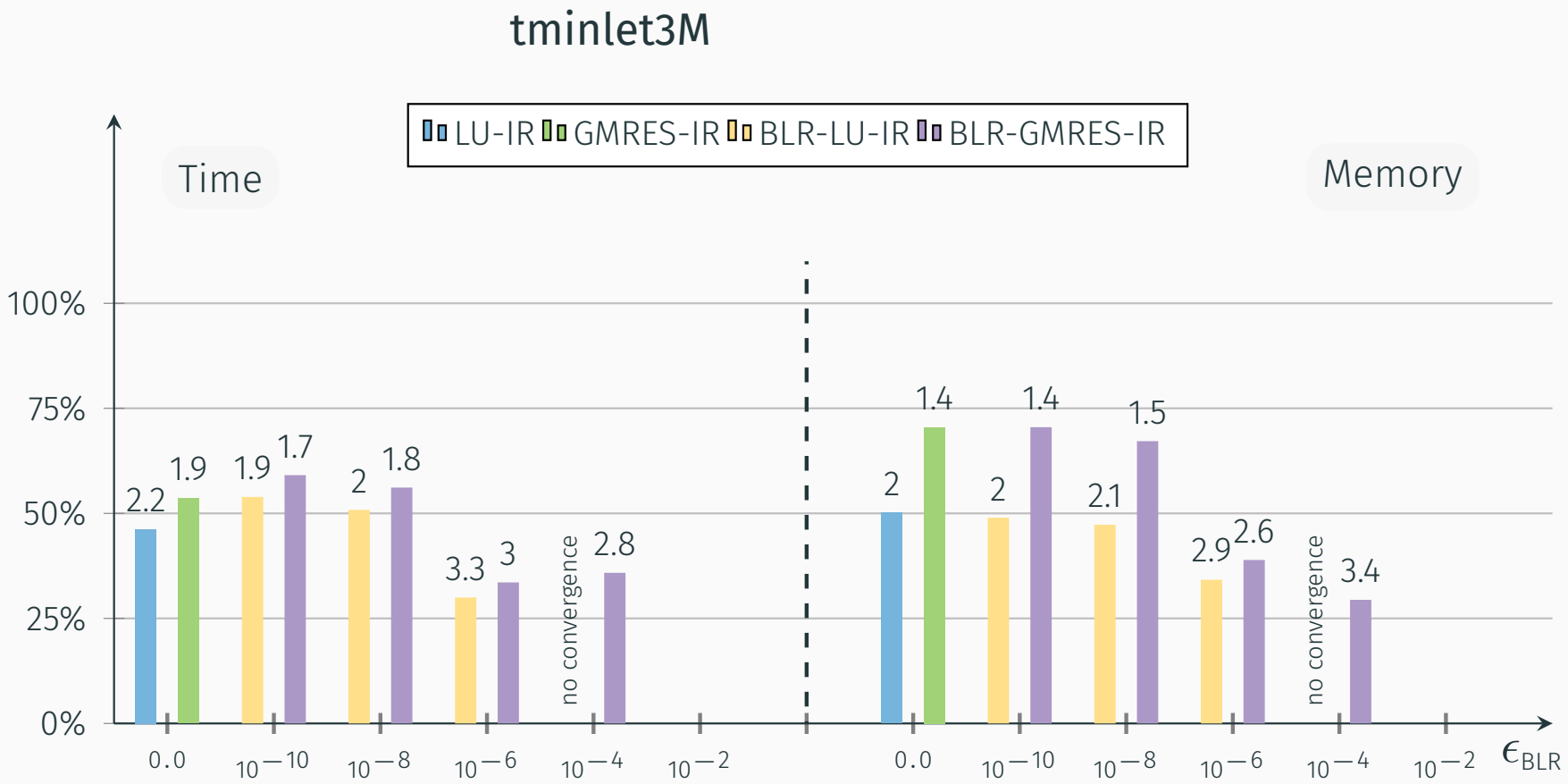
Time and memory performance with BLR w.r.t. DMUMPS



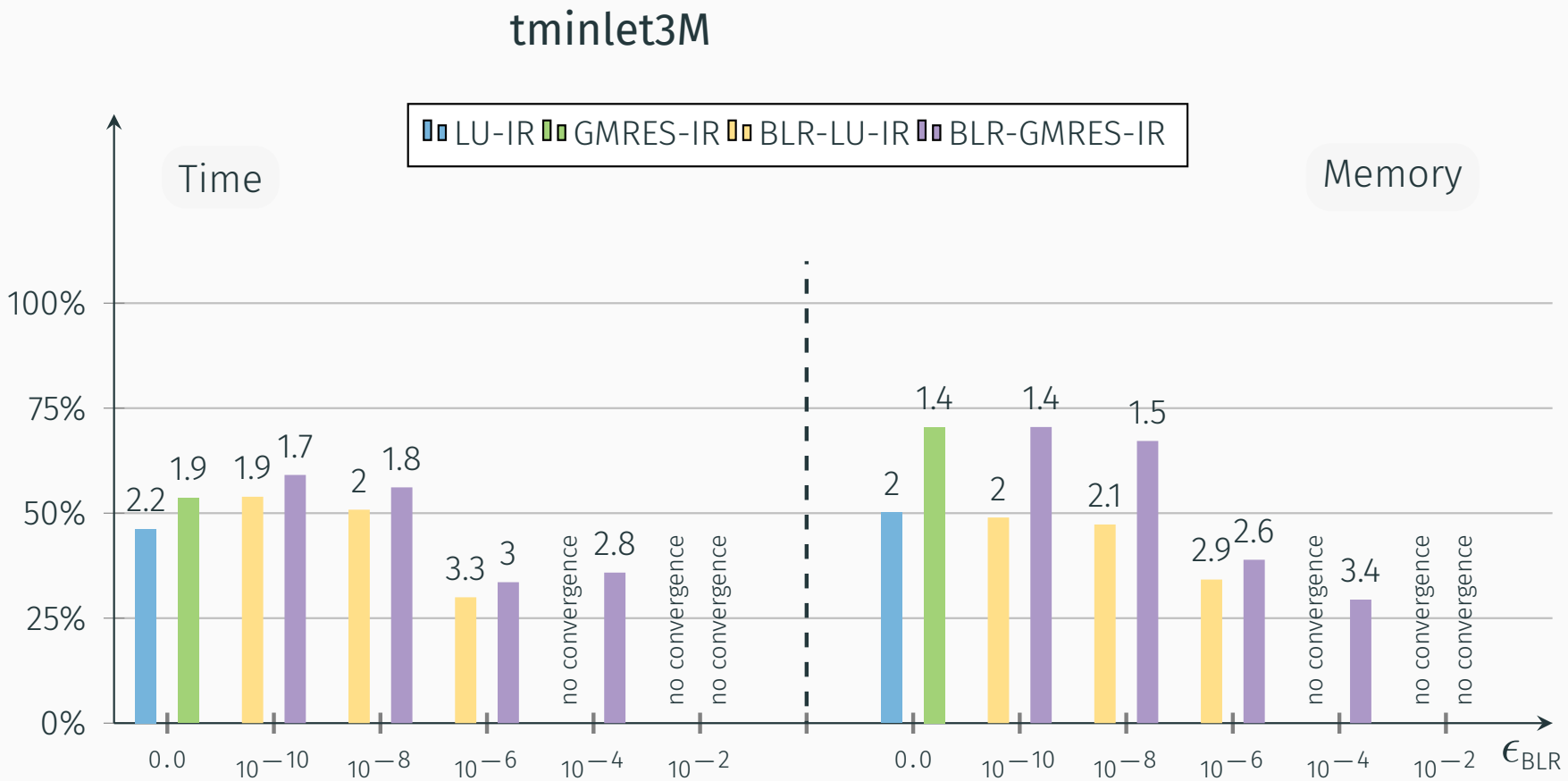
Time and memory performance with BLR w.r.t. DMUMPS



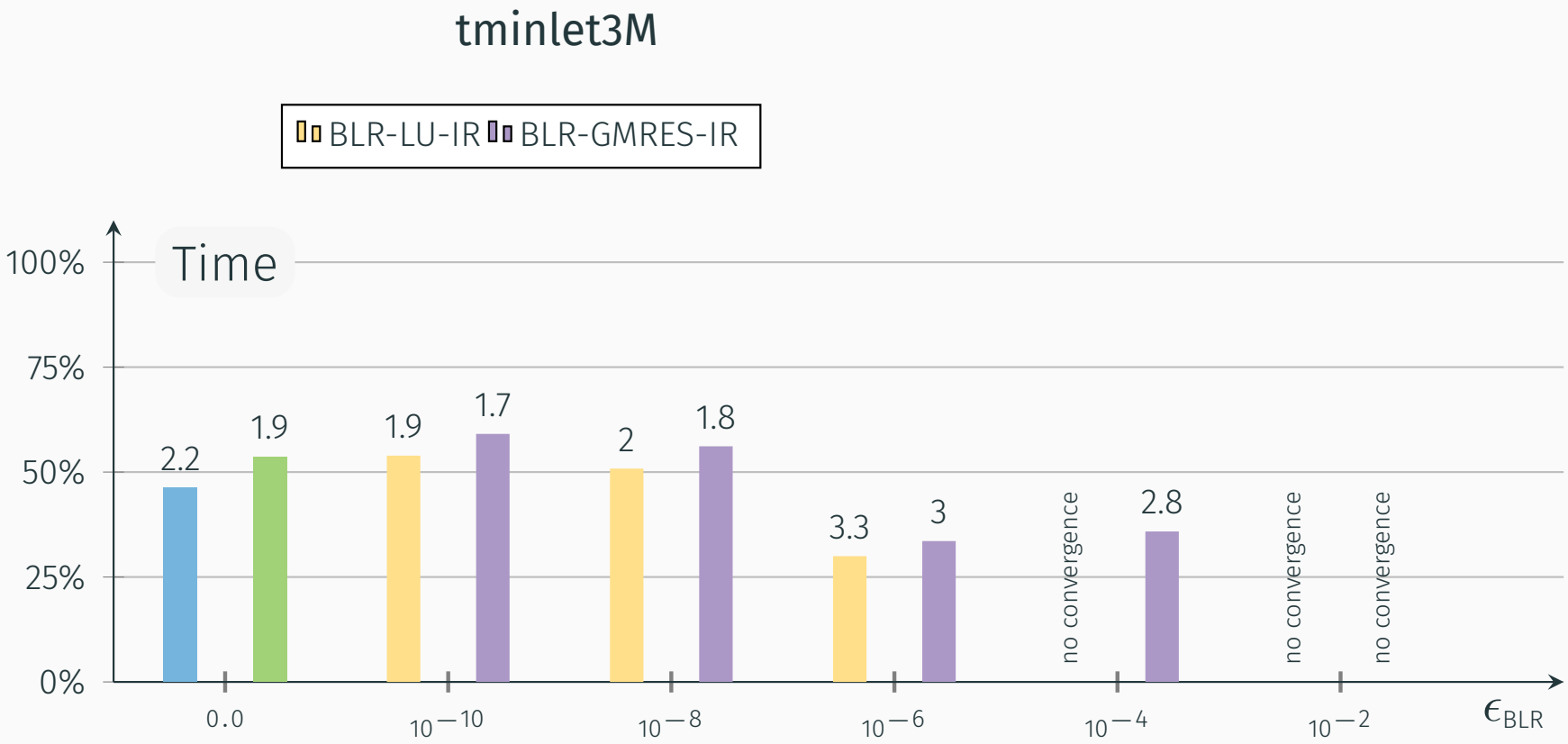
Time and memory performance with BLR w.r.t. DMUMPS



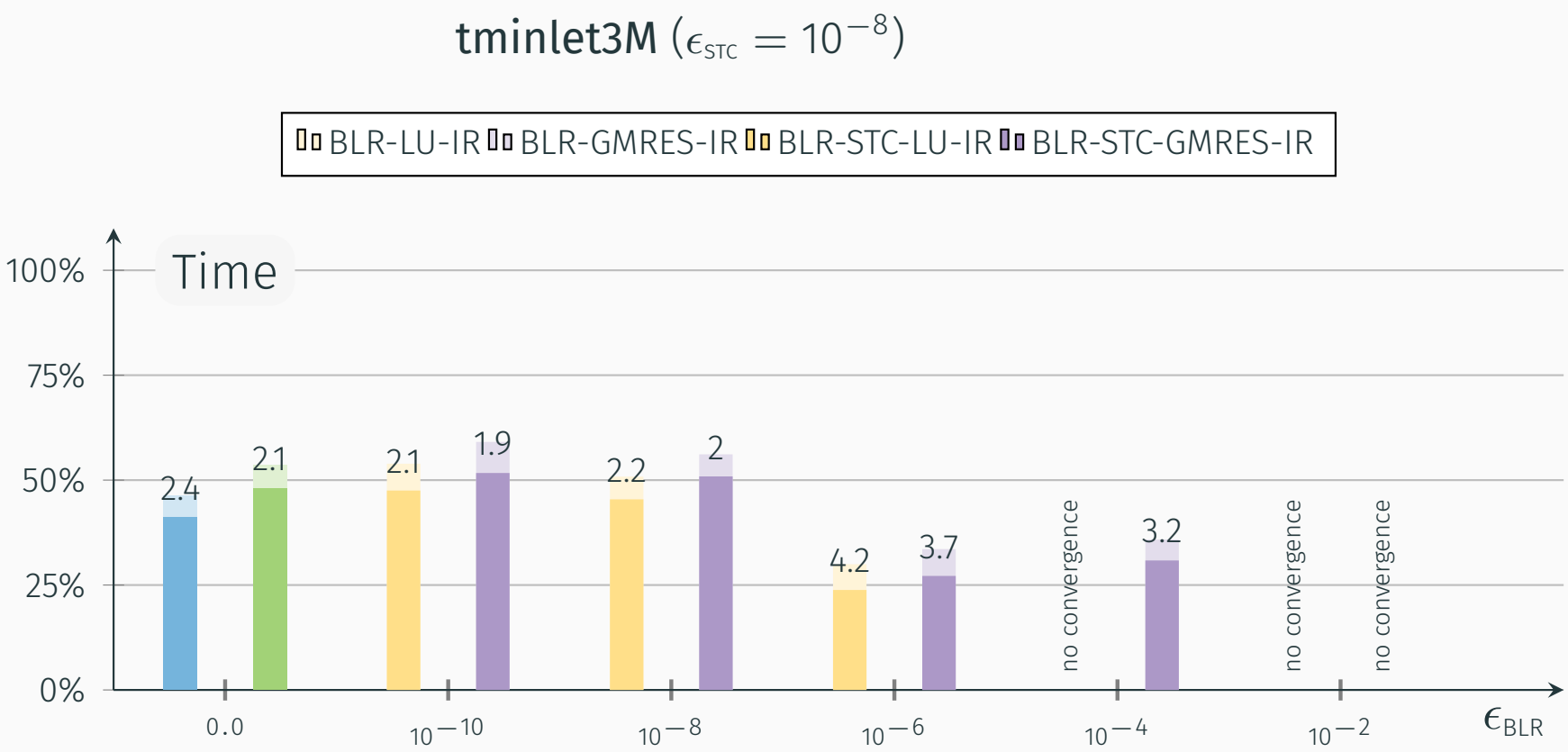
Time and memory performance with BLR w.r.t. DMUMPS



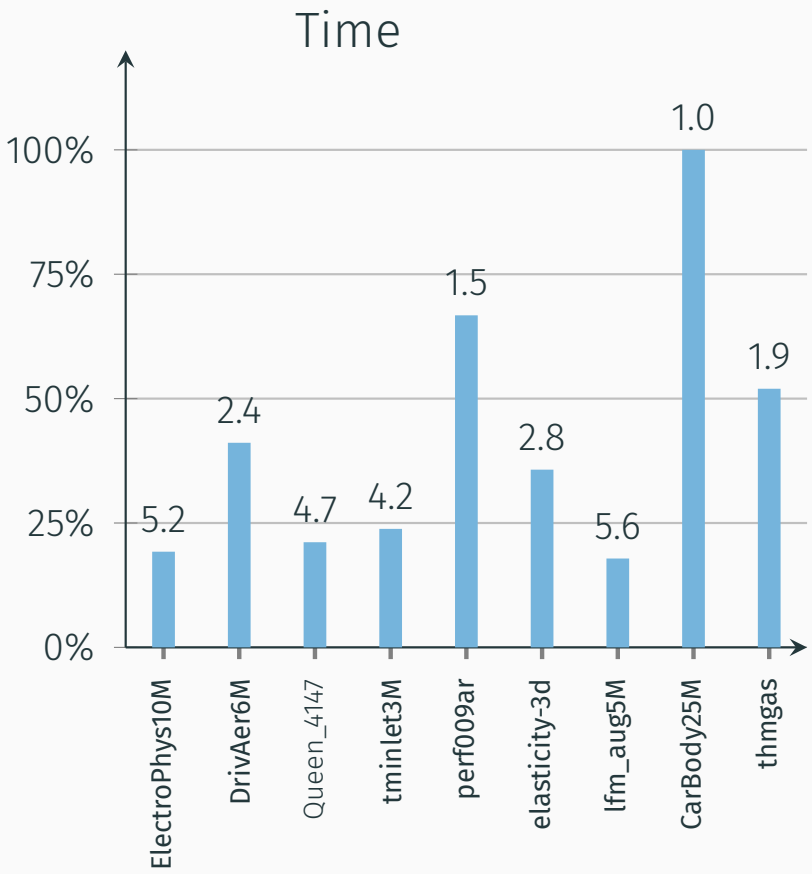
Time performance with BLR + static pivoting w.r.t. DMUMPS

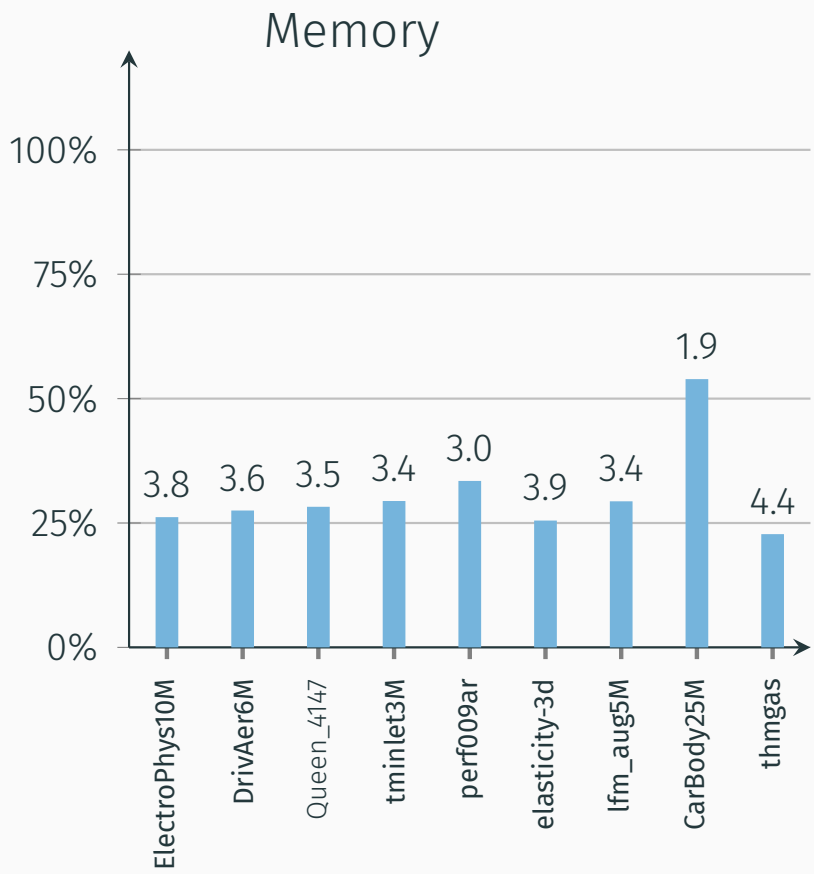


Time performance with BLR + static pivoting w.r.t. DMUMPS



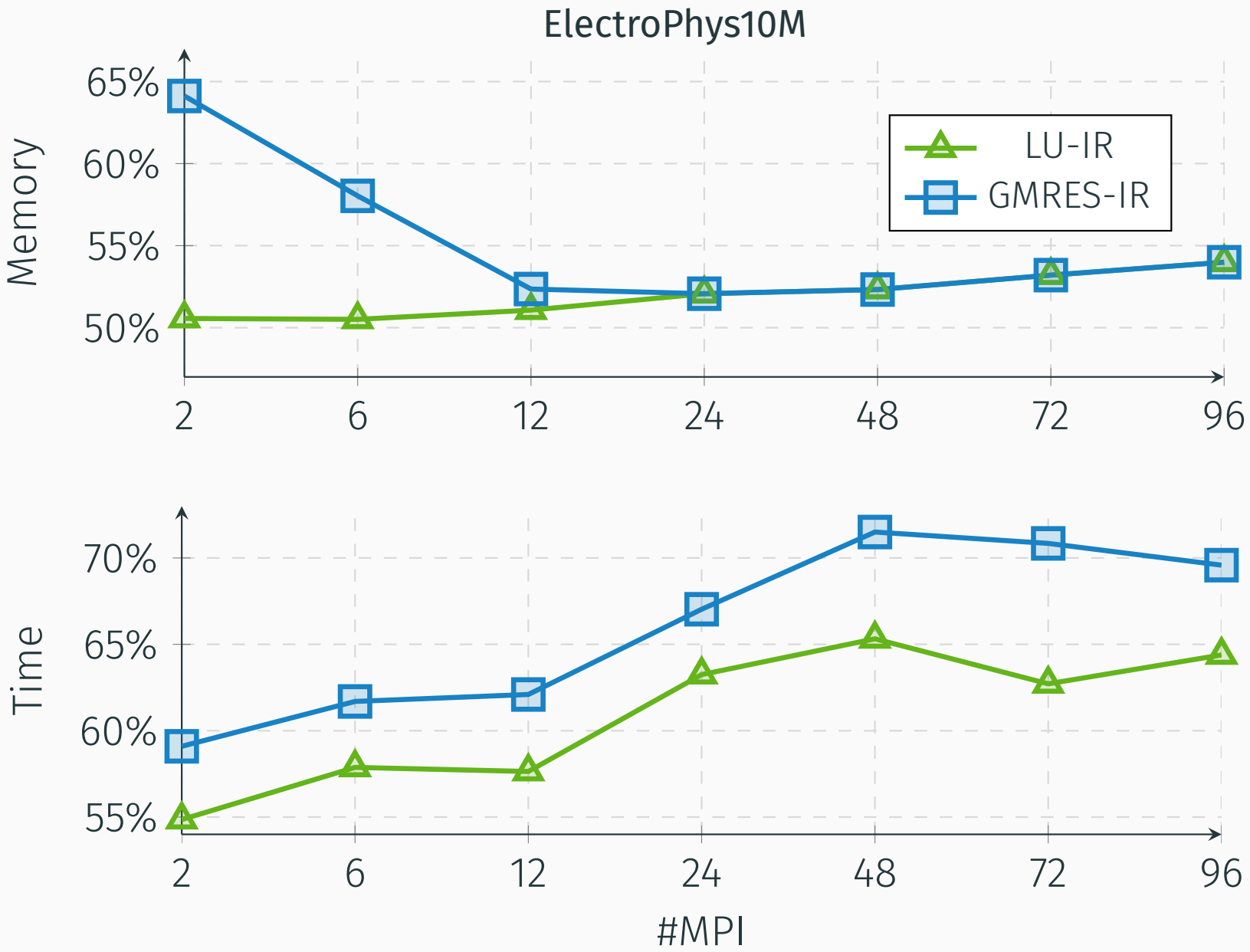
Gather it all : Best time and memory w.r.t. DMUMPS





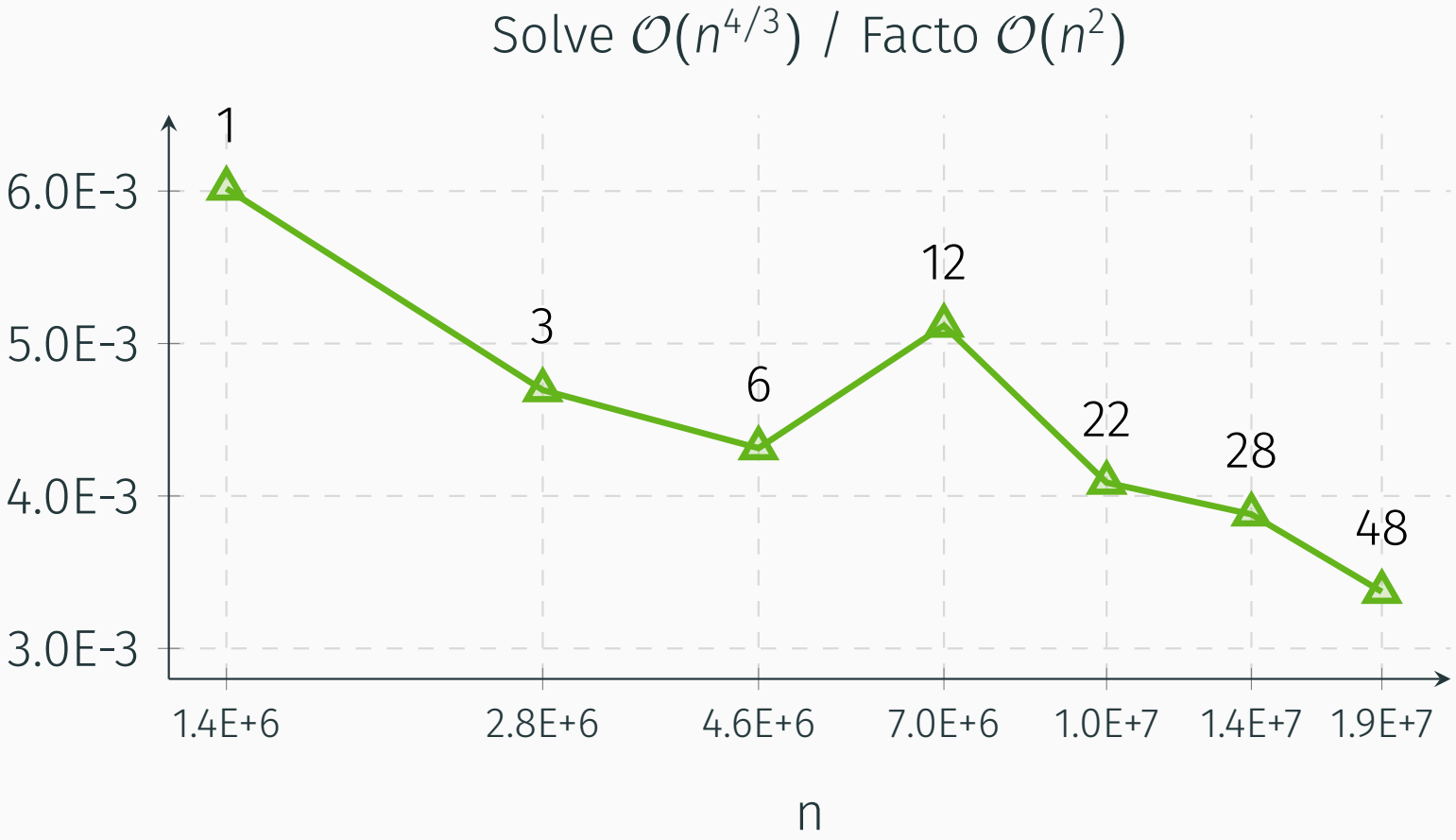
$\Rightarrow$ Up to 5.6$\times$ faster and 4.4$\times$ less memory with the same accuracy on the solution than DMUMPS!

Strong scaling (6OMP/MPI) : Time and mem w.r.t. DMUMPS



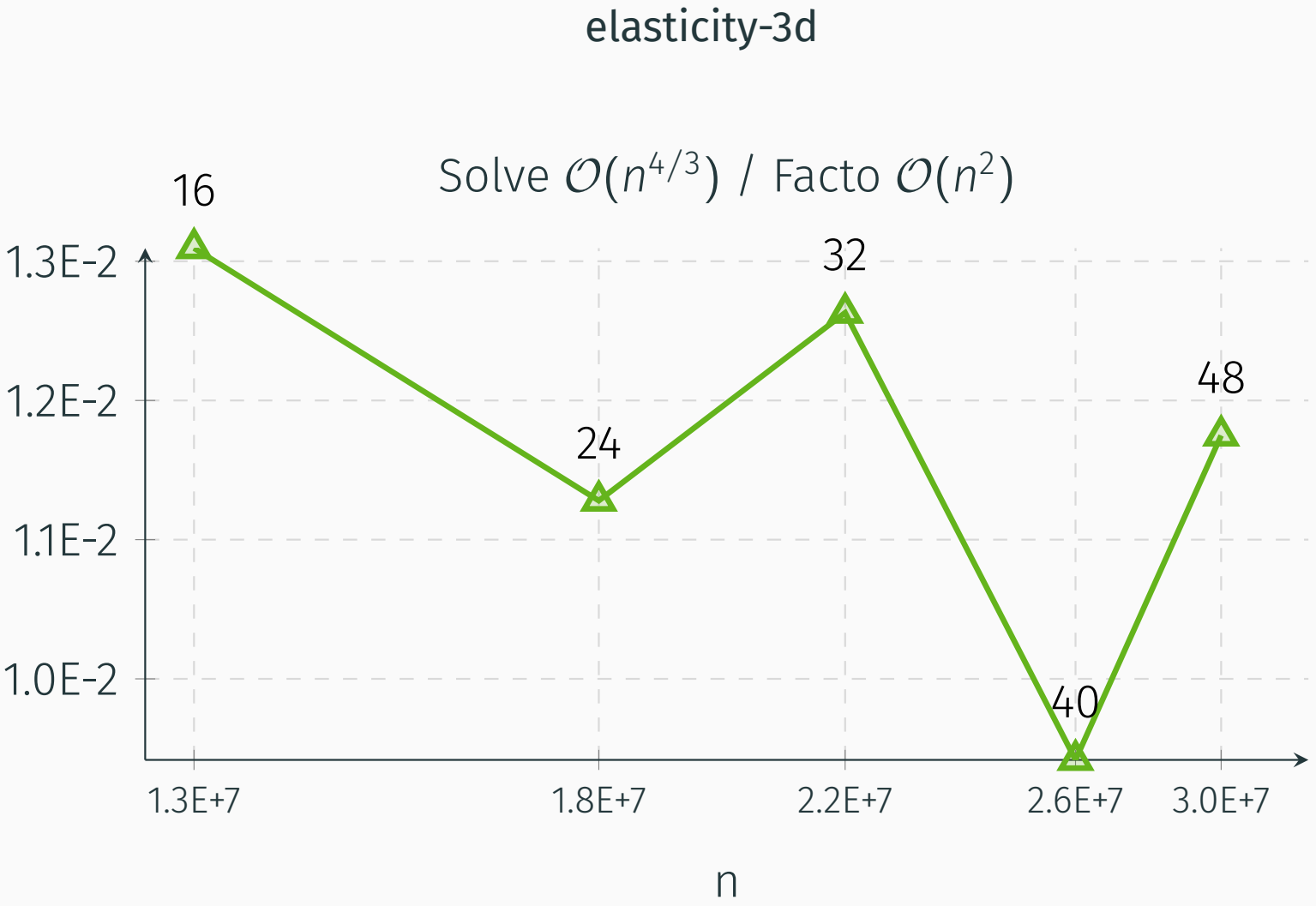
Weak scaling (18OMP/MPI) : Memory/#MPI = cst

Helmholtz



$\Rightarrow$ Less costly refinement steps as n increases.

Weak scaling (18OMP/MPI) : Memory/#MPI = cst



$\Rightarrow$ Constant cost of the refinement steps as n increases.

Conclusion

Summary

Goal : Solve $Ax = b$ with a sparse direct solver ideally with the least memory and computational time.

Nowadays solutions : Combination of numerical approximations and low precisions.

Drawback : Loss of accuracy on the solution.

Proposal : Use iterative refinement to provide an accurate solution for sparse direct solver at low cost [3].

We wish to thank our Industrial partners and the EoCoE project for providing access to their matrices.

3. Amestoy, Buttari, Higham, L'Excellent, Mary, and Vieublé, *Combining sparse approximate factorizations with mixed precision iterative refinement*, 2022