Harnessing inexactness in scientific computing

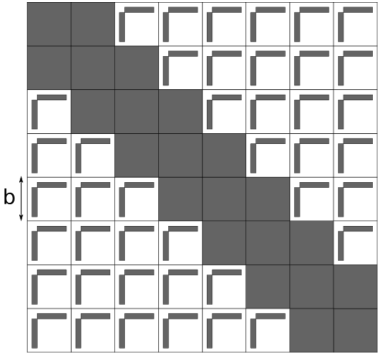
Lecture 14: block low-rank matrices

Theo Mary (CNRS)
theo.mary@lip6.fr
https://perso.lip6.fr/Theo.Mary/

Elisa Riccietti (ENS Lyon)
elisa.riccietti@ens-lyon.fr
https://perso.ens-lyon.fr/elisa.riccietti/

M2 course at ENS Lyon, 2024–2025

Slides available on course webpage



Introduction

Applications

Complexity

High performance implementation

Mixed precision

Multilevel BLR

Two exercises

Introduction

Applications

Complexity

High performance implementation

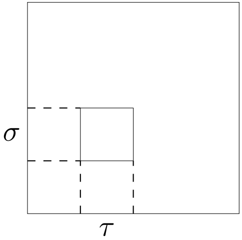
Mixed precision

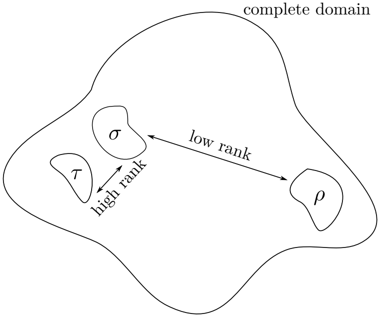
Multilevel BLR

Two exercises

Data sparse matrices

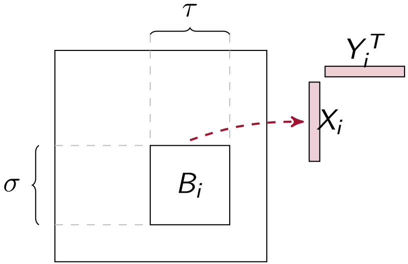
Data sparse matrices are not sparse but can be well approximated by "sparse data":



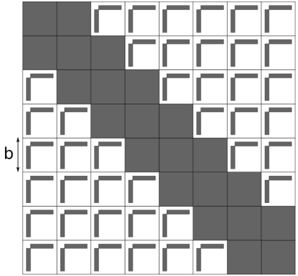


A block B represents the interaction
between two subdomains σ and τ.
Large distance $\Leftrightarrow$ low numerical rank

Block low-rank (BLR) matrices

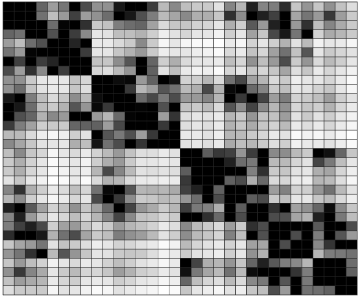


$\|B_i - X_i Y_i^T\| \leq \varepsilon \|A\|$



Block Low Rank

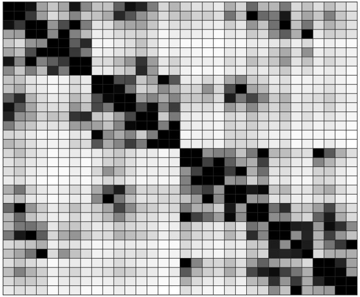
Block low-rank (BLR) matrices



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

- Diagonal blocks are full rank
- Off-diagonal ones are stored in low rank form if their ε-rank is small enough
- $\varepsilon = 10^{-15} \rightarrow 50\%$ entries kept

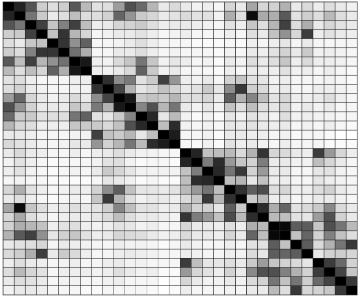
Block low-rank (BLR) matrices



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

- Diagonal blocks are full rank
- Off-diagonal ones are stored in low rank form if their ε-rank is small enough
- $\varepsilon = 10^{-15} \rightarrow 50\%$ entries kept
- $\varepsilon = 10^{-12} \rightarrow 36\%$ entries kept

Block low-rank (BLR) matrices

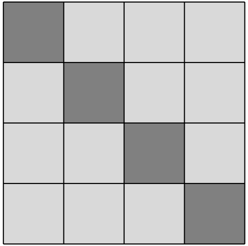


Example of a BLR matrix (Schur complement of a 64^3 Poisson problem with block size 128)

- Diagonal blocks are full rank
- Off-diagonal ones are stored in low rank form if their ε-rank is small enough
- $\varepsilon = 10^{-15} \rightarrow$ 50% entries kept
- $\varepsilon = 10^{-12} \rightarrow$ 36% entries kept
- $\varepsilon = 10^{-9} \rightarrow$ 23% entries kept

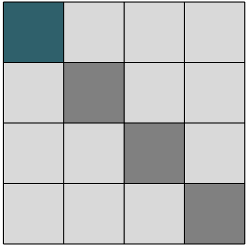
BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant:



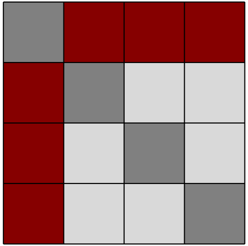
BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor,



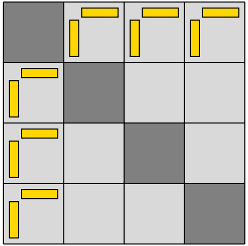
BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve,



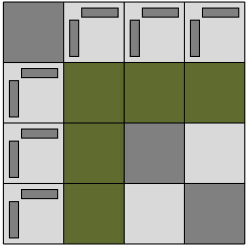
BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress,



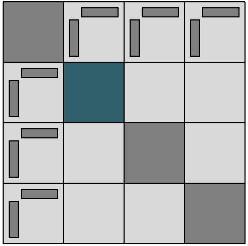
BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update



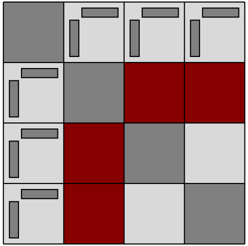
BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update



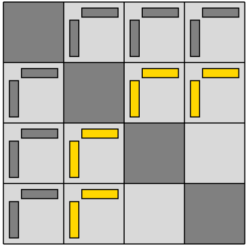
BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update



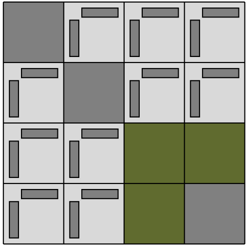
BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update



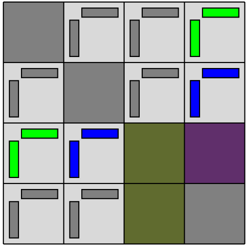
BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update



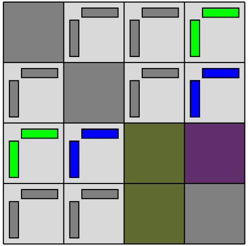
BLR LU factorization: FSCU variant

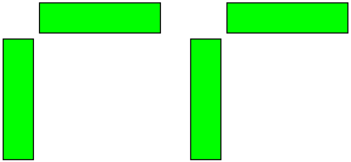
- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update



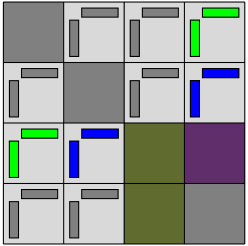
BLR LU factorization: FSCU variant

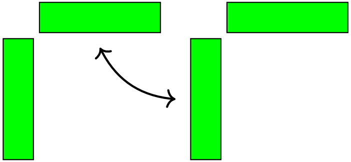
- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update





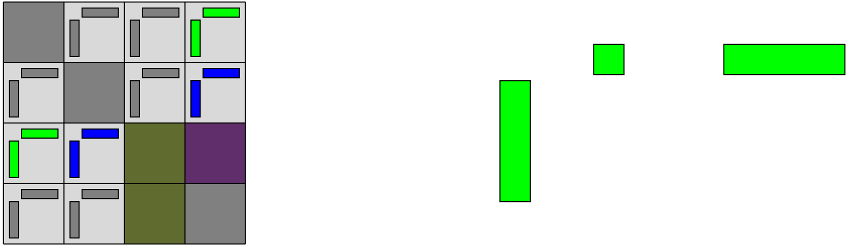
- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update





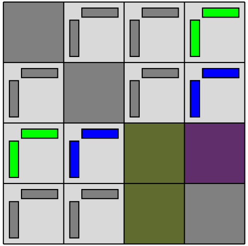
BLR LU factorization: FSCU variant

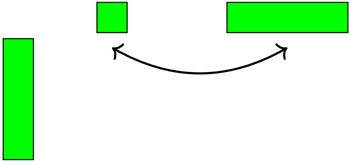
- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update



BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update



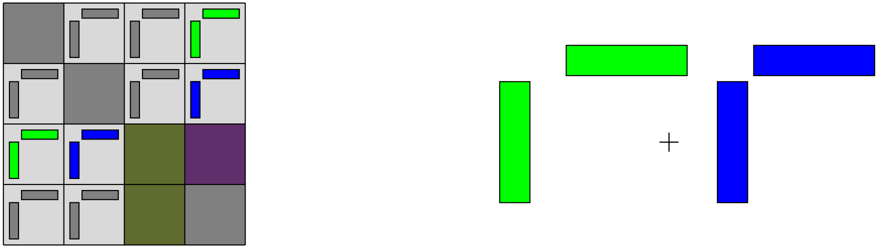


BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update

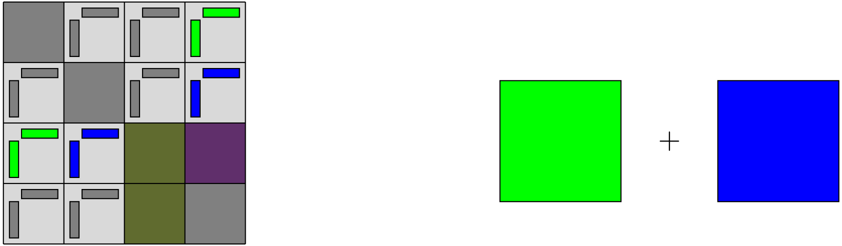
BLR LU factorization: FSCU variant

- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update



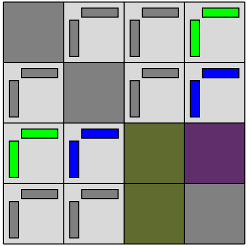
BLR LU factorization: FSCU variant

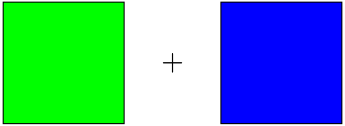
- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update



BLR LU factorization: FSCU variant

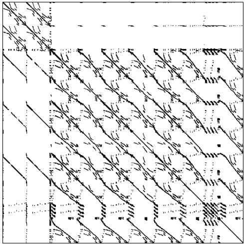
- How to adapt block LU factorization to exploit BLR structure?
- FSCU variant: Factor, Solve, Compress, Update



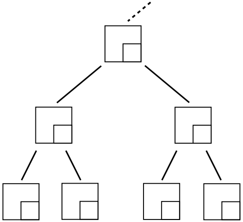


- Computed LU factors satisfy $\widehat{L}\widehat{U} = A + \Delta A$, $\|\Delta A\| \le c_n u \||\widehat{L}||\widehat{U}|\| + \varepsilon \|A\|$

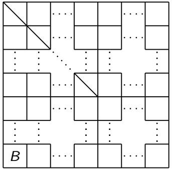
  Higham and M. (2021)



$\Longrightarrow$

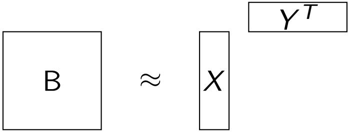


Elimination tree



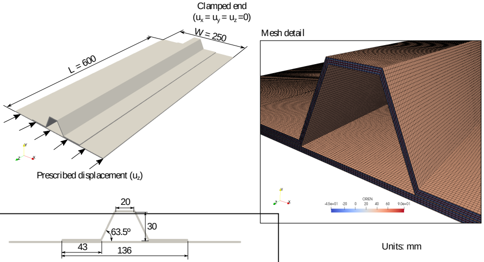
BLR clustering to build 2D block structure

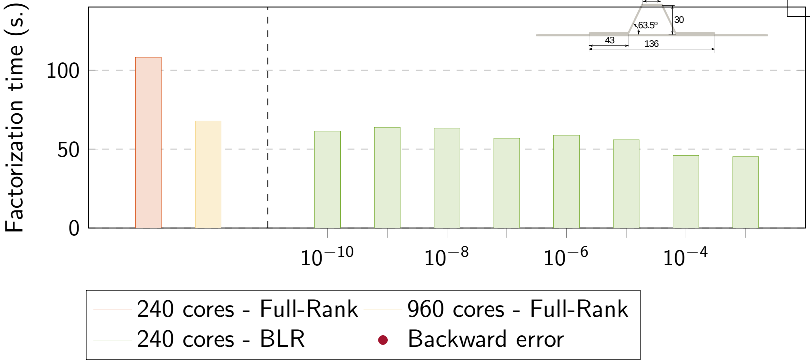
$\Longrightarrow$



SVD or RRQR $\Rightarrow$ X and Y s. t. $\|B - XY^T\| \leq \varepsilon$

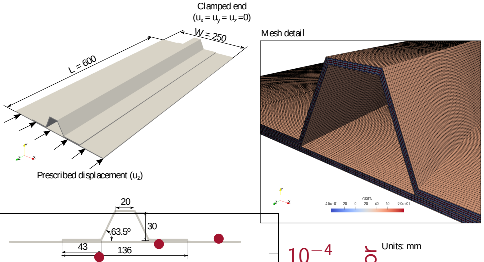
- structure subject to compression load
- matrix of order 10M

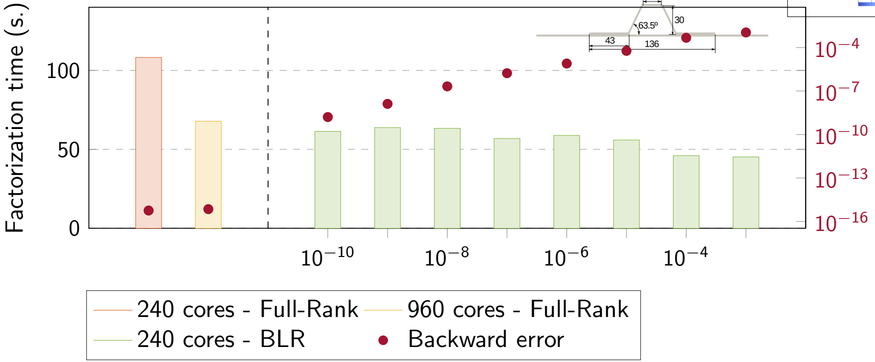




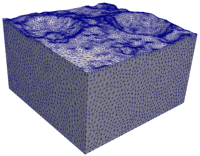
Structural mechanics: wind turbines (ALYA code)

- structure subject to compression load
- matrix of order 10M
- BLR backward error is in very good accordance with ε





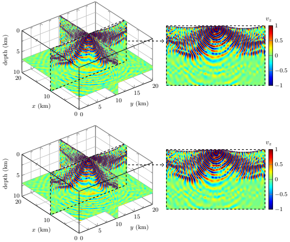
Surface topography



3D mesh (220k cells, $20{\times}20{\times}10$ km^3)

- surface topography: small cells
- below: larger cells and high order

Solution v_z at 3 Hz using Full-Rank (top) or BLR $\varepsilon{=}10^{-5}$ (bottom)



Matrix size (elastic equations, $f{=}3$ Hz, $3 \leq p \leq 6$): N=25M, NZ=5 859M

BLR compression:

	Full-Rank	BLR ($\varepsilon{=}10^{-5}$)
flops	1.1×10^{16}	3.6×10^{15} (34%)

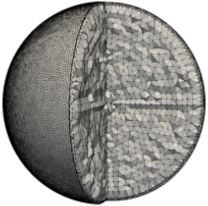
Time compression:

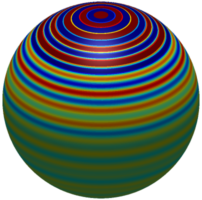
(HLRS cluster Stuttgart, using 40 nodes, 80 MPI, 32 threads/MPI)

Time (seconds)	Analysis	Facto.	Solve
MUMPS Full-Rank	11.7	2894	8.3
MUMPS BLR ($\varepsilon = 10^{-5}$)	13.2	1798	6.0

Helioseismology

Mesh and solution at 2 mHz for r=0.99. Cells near surface are very small (extreme properties)





Experiment with entire solar ball, r_{max}=1.0008, f=1 mHz

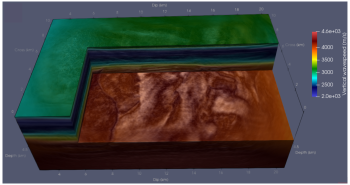
- required 40 nodes, 1 MPI per node (256 GB, 24 threads per node)
- BLR solver used, with $\varepsilon = 10^{-7}$
- Full-Rank estimate (12M, p=2-4): 3×10^{16} flops, memory $>$ **9 TiB**

#cells	p	#dofs	BLR flops (% of FR)	ana	fac	sol	Mem
12M	2-4	374M	4.7×10^{14} (2.0%)	500 s.	773 s.	20 s.	5.3 TiB
12M	3-4	381M	5.8×10^{14} (1.7%)	506 s.	999 s.	20 s.	5.3 TiB

- 50x BLR compression

Full-Waveform Inversion

- Adastra MUMPS4FWI project led by WIND team
- Application: Gorgon Model, reservoir 23km x 11km x 6.5km, grid size 15m, Helmholtz equation, 25-Hz
- Complex matrix, 531 Million dofs, storage(A)=220 GBytes;
- FR cost: flops for one LU factorization= 2.6×10^{18};
  Estimated storage for LU factors= 73 TBytes



(25-Hz Gorgon FWI velocity model)

FR (Full-Rank); BLR with $\varepsilon = 10^{-5}$; 48 000 cores (500 MPI $\times$ 96 threads/MPI)

FR: fp32; Mixed precision BLR: 3 precisions (32bits, 24bits, 16bits) for storage

LU size (TBytes)			Flops		Time BLR + Mixed (sec)			Scaled Resid.
FR	BLR	+mixed	FR	BLR+mixed	Analysis	Facto	Solve	BLR+mixed
73	34	26	2.6×10^{18}	0.5×10^{18}	446	5500	27	7×10^{-4}

in practice: hundreds to thousands of Solve steps (sparse right hand sides (sources))

Structural mechanics: pump of nuclear plant

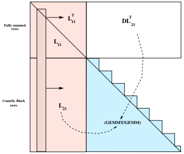
- Matrix from aero-acoustics, complex unsymmetric

N=4.1M, NZ=353M, flops(LU)= 2.6×10^{14} $\boxed{\text{pivots: 4 delayed, 415 off-diag}}$

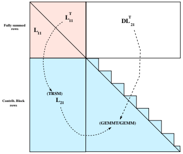
8 MPIx18 threads	Full-Rank		BLR ($\varepsilon = 10^{-8}$)	
Pivoting	std	off	std	off
Factor. time (s.)	273	241	164	114
Scaled residual	7×10^{-13}	1×10^{-11}	6×10^{-8}	2×10^{-7}

Pivoting off $\rightarrow$ improved BLR performance, residual still OK

std



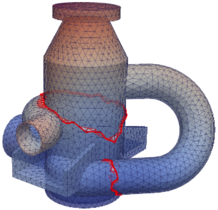
off



BLAS2&3

large BLAS 3

Structural mechanics: pump of nuclear plant

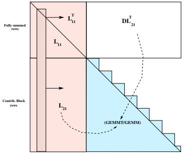


- Matrix from structural mechanics, real symmetric: pump (code_aster, EDF)

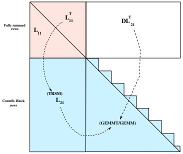
  N=5.4M, NZ=208M, flops(LDL^T)= 1.8×10^{13} | pivots: 79k delayed, 37k 2×2, 74k negative |

1MPIx18 threads	Full-Rank		BLR ($\varepsilon = 10^{-9}$)	
Pivoting	std	off	std	off
Factor. time (s.)	59.5	KO	43.9	KO
Scaled residual	1.9×10^{-15}		1.1×10^{-10}	

std

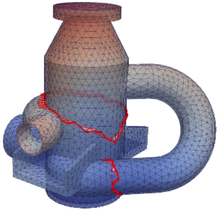


off



BLAS2&3

large BLAS 3



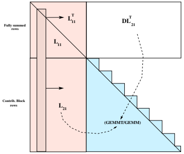
Relaxed pivoting: based on Duff and Pralet

- Matrix from structural mechanics, real symmetric: pump (code_aster, EDF)

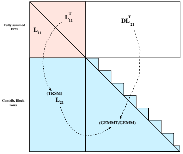
N=5.4M, NZ=208M, flops(LDL^T)= 1.8×10^{13} pivots: 79k delayed, 37k 2×2, 74k negative

1MPIx18 threads	Full-Rank			BLR ($\varepsilon = 10^{-9}$)		
Pivoting	std	off	relaxed	std	off	relaxed
Factor. time (s.)	59.5	KO	47.7	43.9	KO	31.4
Scaled residual	1.9×10^{-15}		1.9×10^{-15}	1.1×10^{-10}		1.3×10^{-10}

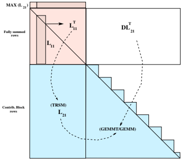
std



off



relaxed



BLAS2&3

large BLAS 3

Structural mechanics: pump of nuclear plant

- Matrix from aero-acoustics, complex unsymmetric

  N=4.1M, NZ=353M, flops(LU)= 2.6×10^{14} | pivots: 4 delayed, 415 off-diag |

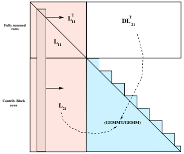
8 MPIx18 threads	Full-Rank			BLR ($\varepsilon = 10^{-8}$)		
Pivoting	std	off	relaxed	std	off	relaxed
Factor. time (s.)	273	241	**241**	164	114	**127**
Scaled residual	$7\text{x}10^{-13}$	$1\text{x}10^{-11}$	$1\text{x}10^{-12}$	$6\text{x}10^{-8}$	$2\text{x}10^{-7}$	$1\text{x}10^{-7}$

- Matrix from structural mechanics, real symmetric: pump (code_aster, EDF)

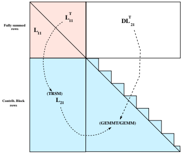
  N=5.4M, NZ=208M, flops(LDL^T)= 1.8×10^{13} | pivots: 79k delayed, 37k 2×2, 74k negative |

1MPIx18 threads	Full-Rank			BLR ($\varepsilon = 10^{-9}$)		
Pivoting	std	off	relaxed	std	off	relaxed
Factor. time (s.)	59.5	KO	47.7	43.9	KO	31.4
Scaled residual	$1.9\text{x}10^{-15}$		$1.9\text{x}10^{-15}$	$1.1\text{x}10^{-10}$		$1.3\text{x}10^{-10}$

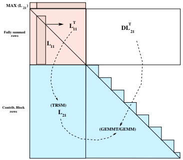
std



off



relaxed

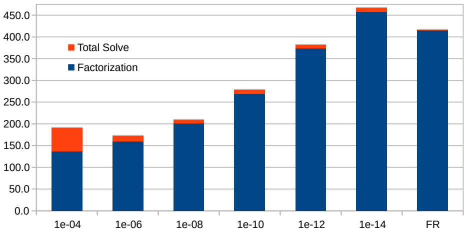


BLAS2&3

large BLAS 3

BLR as a preconditioner

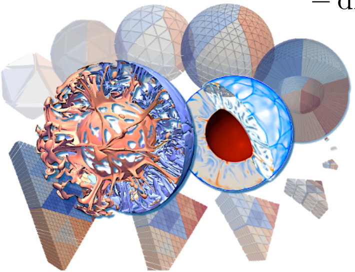
- Application in structural mechanics (ANSYS): highly nonlinear materials, 3D solid elements, matrix `Rubber RVE`
  N=3M, NZ=70M, flops(LDL^T)$= 9.8 \times 10^{14}$
- Conjugate Gradient preconditioned BLR LU:
  - Convergence criterion is scaled residual $\leq 10^{-10}$, time for analysis $\approx$ 45sec
  - CG with simple block-Jacobi preconditioner did not converge
  - Target computer: cluster with 28-core nodes based on Xeon E5-2690 v4-14 cores, 512GB



Solver(ε)	Size of L	Total (nbiter)
CG-BLR(10^{-4})	30GB	192(92)
CG-BLR(10^{-6})	51GB	173(13)
CG-BLR(10^{-8})	78GB	210 (4)
CG-BLR(10^{-10})	109GB	279 (3)
CG-BLR(10^{-12})	136GB	383 (1)
CG-BLR(10^{-14})	153GB	467 (1)
Full-Rank (FR)	157GB	417(0)

BLR in a multigrid solver

Mantle convection simulation:



$$
\begin{aligned}
-\operatorname{div}\left(\frac{\nu}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^\top)\right) + \nabla \mathbf{p} &= \mathbf{f} \quad \text{in } \Omega, \\
\operatorname{div}(\mathbf{u}) &= 0 \quad \text{in } \Omega, \\
\mathbf{u} &= \mathbf{g} \quad \text{on } \partial\Omega,
\end{aligned}
$$

- HHG (Hierarchical Hybrid Grids) geometric multigrid solver
- 6 levels with V-cycle and Uzawa smoother

→ compare BLR vs. PMINRES as coarse grid solvers

Buttari, Huber, Leleux, M., Ruede, Wohlmuth

BLR in a multigrid solver

n-fine	n-coarse
$1.21 \cdot 10^{11}$	$1.94 \cdot 10^{6}$

Time on the fine grid: 1200 s. (43200 cores, HLRS cluster Stuttgart)

Time on the coarse grid:

PMINRES	162.5 s.	$\Rightarrow$ not negligible
MUMPS Full-Rank	184.7 s.	
MUMPS BLR, $\varepsilon = 10^{-5}$	91.4 s.	
MUMPS BLR, $\varepsilon = 10^{-5}$ + single precision	83.6 s.	$\Rightarrow$ $2\times$ speedup

- PMINRES converges slowly $\Rightarrow$ coarse grid solution not negligible
- $2\times$ speedup on the coarse grid $\Rightarrow$ $\sim 6\%$ overall gain

Cost analysis of the involved steps

Let us consider two blocks A and B of size $b \times b$ and of rank bounded by r.

step	type	operation	cost
Factor	FR	$A \leftarrow LU$	$\mathcal{O}(b^3)$
Solve	FR-FR	$B \leftarrow BU^{-1}$	$\mathcal{O}(b^3)$
Compress	LR	$A \leftarrow \widetilde{A}$	$\mathcal{O}(b^2 r)$
Update	FR-FR	$C \leftarrow AB$	$\mathcal{O}(b^3)$
Update	LR-FR	$C \leftarrow \widetilde{A}B$	$\mathcal{O}(b^2 r)$
Update	FR-LR	$C \leftarrow A\widetilde{B}$	$\mathcal{O}(b^2 r)$
Update	LR-LR	$C \leftarrow \widetilde{A}\widetilde{B}$	$\mathcal{O}(b^2 r)$

This is not enough to compute the complexity $\Rightarrow$ we need to bound the number of FR blocks!

Admissible block partitions

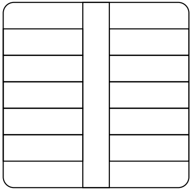
Admissibility of a block

An block $\sigma \times \tau$ is admissible if $\mathrm{dist}(\sigma,\tau) \geq \eta \max(\mathrm{diam}(\sigma), \mathrm{diam}(\tau))$.

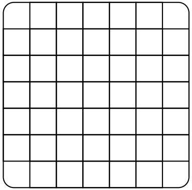
Admissibility of a block partition

A block partion $\mathcal{P}$ is admissible if there exists $q = O(1)$ such that

$$\begin{cases} \#\{\sigma, \quad \sigma \times \tau \in \mathcal{P} \text{ is not admissible}\} \leq q \\ \#\{\tau, \quad \sigma \times \tau \in \mathcal{P} \text{ is not admissible}\} \leq q \end{cases}$$



Non-Admissible



Admissible

Admissible block partitions

Key result

For any matrix, an admissible $\mathcal{P}$ can be built geometrically

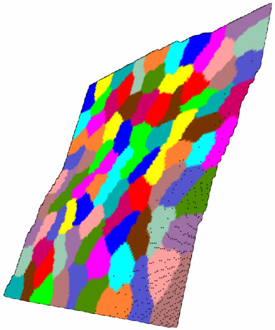
Amestoy, Buttari, L'Excellent, M. (2017)

Admissible block partitions

Key result

For any matrix, an admissible $\mathcal{P}$ can be built geometrically

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



What about algebraic partitions?
For sparse problems, use the adjacency graph!

Root separator of a 128^3 Poisson problem clustered with SCOTCH via k-means Weisbecker (2015)

Memory complexity of the dense BLR factorization

We consider a dense $m \times m$ matrix with $m = pb$

The memory complexity to store the matrix can be computed as
$\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$

Memory complexity of the dense BLR factorization

We consider a dense $m \times m$ matrix with $m = pb$

The memory complexity to store the matrix can be computed as
$\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$

- Step 1: we have

$$\mathcal{M}_{FR}(b, p) = \mathcal{O}(pb^2)$$
$$\mathcal{M}_{LR}(b, p, r) = \mathcal{O}(p^2 br)$$

Memory complexity of the dense BLR factorization

We consider a dense $m \times m$ matrix with $m = pb$

The memory complexity to store the matrix can be computed as
$\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$

- Step 1: we have

$$\mathcal{M}_{FR}(b, p) = \mathcal{O}(pb^2)$$
$$\mathcal{M}_{LR}(b, p, r) = \mathcal{O}(p^2 br)$$

- Step 2: assuming $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$, we have

$$\mathcal{M}_{total}(m, x, \alpha) = \mathcal{O}(m^{1+x} + m^{2-x+\alpha})$$

Memory complexity of the dense BLR factorization

We consider a dense $m \times m$ matrix with $m = pb$

The memory complexity to store the matrix can be computed as
$\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$

- Step 1: we have

$$\mathcal{M}_{FR}(b, p) = \mathcal{O}(pb^2)$$
$$\mathcal{M}_{LR}(b, p, r) = \mathcal{O}(p^2 br)$$

- Step 2: assuming $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$, we have

$$\mathcal{M}_{total}(m, x, \alpha) = \mathcal{O}(m^{1+x} + m^{2-x+\alpha})$$

- Step 3: the optimal block size is $b^* = m^{(1+\alpha)/2}$ and the resulting optimal complexity is

$$\mathcal{M}_{opt}(m, r) = \mathcal{M}_{total}(m, x^*, \alpha) = \mathcal{O}(m^{3/2} r^{1/2})$$

Flop complexity of the dense BLR factorization

We consider a dense $m \times m$ matrix with $m = pb$

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	
Compress	LR	$\mathcal{O}(b^2 r)$	$\mathcal{O}(p^2)$	
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	
Update	LR-FR	$\mathcal{O}(b^2 r)$	$\mathcal{O}(p^2)$	
Update	LR-LR	$\mathcal{O}(b^2 r)$	$\mathcal{O}(p^3)$	

Flop complexity of the dense BLR factorization

We consider a dense $m \times m$ matrix with $m = pb$

step	type	cost	number	$\mathcal{C}_{step}(b,p,r)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$
Update	LR-LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$

- Step 1: compute $\mathcal{C}_{step}(b,p,r) = \text{cost} \times \text{number}$

Flop complexity of the dense BLR factorization

We consider a dense $m \times m$ matrix with $m = pb$

step	type	cost	number	$\mathcal{C}_{step}(b,p,r)$	$\mathcal{C}_{step}(m,x,\alpha)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$	$\mathcal{O}(m^{2+x})$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	LR-LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$	$\mathcal{O}(m^{3-x+\alpha})$

- Step 1: compute $\mathcal{C}_{step}(b,p,r) = \text{cost} \times \text{number}$
- Step 2: compute $\mathcal{C}_{step}(m,x,\alpha)$ with $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$.

Flop complexity of the dense BLR factorization

We consider a dense $m \times m$ matrix with $m = pb$

step	type	cost	number	$\mathcal{C}_{step}(b,p,r)$	$\mathcal{C}_{step}(m,x,\alpha)$
`Factor`	`FR`	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
`Solve`	`FR-FR`	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$	$\mathcal{O}(m^{2+x})$
`Compress`	`LR`	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
`Update`	`FR-FR`	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
`Update`	`LR-FR`	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
`Update`	`LR-LR`	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$	$\mathcal{O}(m^{3-x+\alpha})$

- Step 1: compute $\mathcal{C}_{step}(b,p,r) = \text{cost} \times \text{number}$
- Step 2: compute $\mathcal{C}_{step}(m,x,\alpha)$ with $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$.
- Step 3: compute the total complexity (sum of all steps)

$$\mathcal{C}_{total}(m,x,\alpha) = \mathcal{O}(m^{3-x+\alpha} + m^{2+x})$$

Flop complexity of the dense BLR factorization

We consider a dense $m \times m$ matrix with $m = pb$

step	type	cost	number	$\mathcal{C}_{step}(b,p,r)$	$\mathcal{C}_{step}(m,x,\alpha)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$	$\mathcal{O}(m^{2+x})$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	LR-LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$	$\mathcal{O}(m^{3-x+\alpha})$

- Step 1: compute $\mathcal{C}_{step}(b,p,r) =$ cost $\times$ number
- Step 2: compute $\mathcal{C}_{step}(m,x,\alpha)$ with $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$.
- Step 3: compute the total complexity (sum of all steps)

$$\mathcal{C}_{total}(m,x,\alpha) = \mathcal{O}(m^{3-x+\alpha} + m^{2+x})$$

- Step 4: the optimal block size is $b^* = m^{(1+\alpha)/2}$ and the resulting optimal complexity is $\mathcal{C}_{opt}(m,r) = \mathcal{C}_{total}(m,x^*,\alpha) = \mathcal{O}(m^{5/2}r^{1/2})$

Complexity of the sparse multifrontal BLR factorization

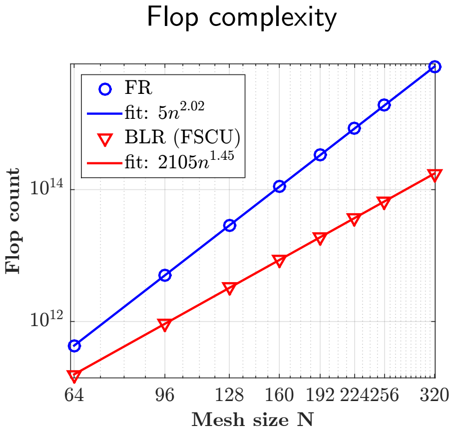
For a dense complexity $\mathcal{C}_{opt}(m, r)$, the sparse complexity is computed as

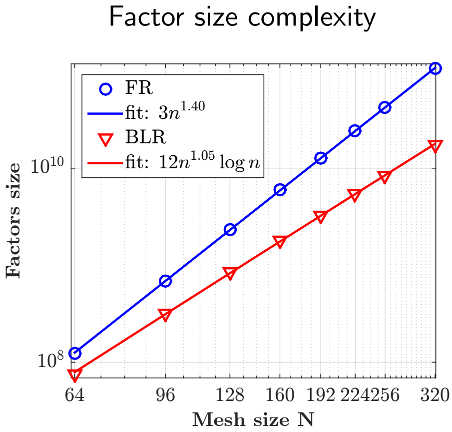
$$N \times N \text{ grid:} \quad \mathcal{C}_{mf} = \sum_{\ell=0}^{\log_2 N} 2^{2\ell} \mathcal{C}_{opt}\left(\frac{N}{2^\ell}\right),$$

$$N \times N \times N \text{ grid:} \quad \mathcal{C}_{mf} = \sum_{\ell=0}^{\log_4 N^2} 2^{3\ell} \mathcal{C}_{opt}\left(\frac{N^2}{2^{2\ell}}\right).$$

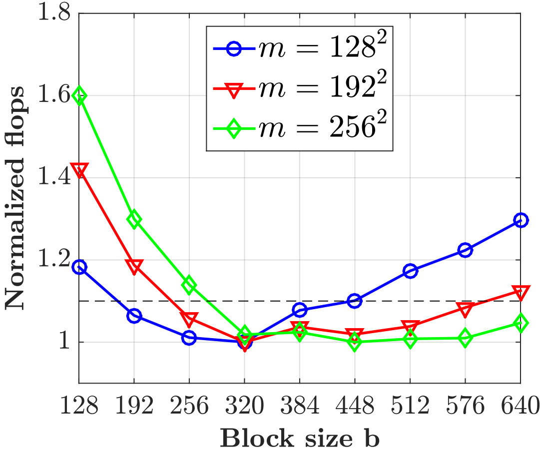
	operations (OPC)	factor size (NNZ)
	$N \times N$ grid	
FR	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2 \log N)$
BLR	$\mathcal{O}(N^{5/2} r^{1/2})$	$\mathcal{O}(N^2)$
	$N \times N \times N$ grid	
FR	$\mathcal{O}(N^6)$	$\mathcal{O}(N^4)$
BLR	$\mathcal{O}(N^5 r^{1/2})$	$\mathcal{O}(N^3 \max(r^{1/2}, \log N))$

Experimental complexity on Poisson problem





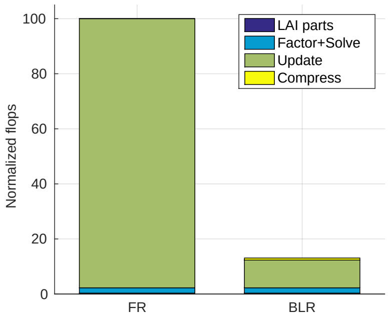
Influence of the block size b on the complexity



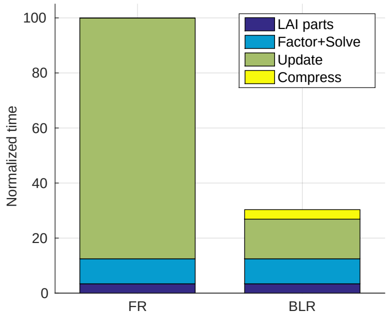
Analysis on the root node (of size $m = N^2$):

- large range of acceptable block sizes around the optimal b^* $\Rightarrow$ flexibility to tune block size for performance
- that range increases with the size of the matrix $\Rightarrow$ necessity to have variable block sizes

Sequential result



Normalized Flops

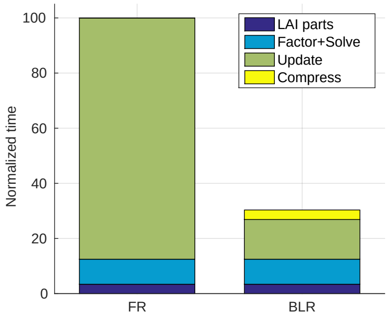


Normalized Time

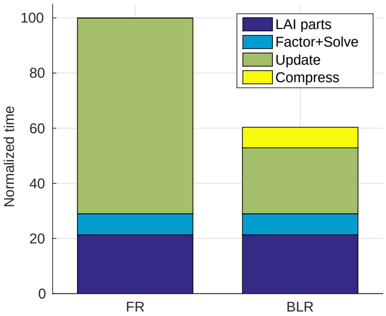
7.7 gain in flops only translated to a **3.3** gain in time: why?

- lower granularity of the Update
- higher relative weight of the FR parts
- inefficient Compress

Multithreaded result on 24 threads



Normalized Time (Seq.)

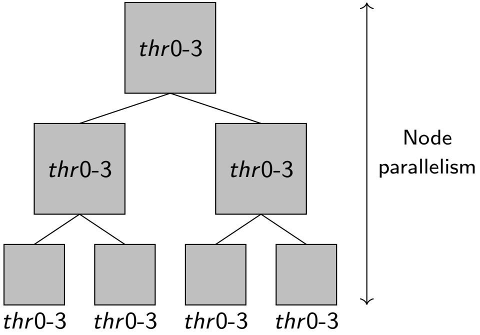


Normalized Time (MT)

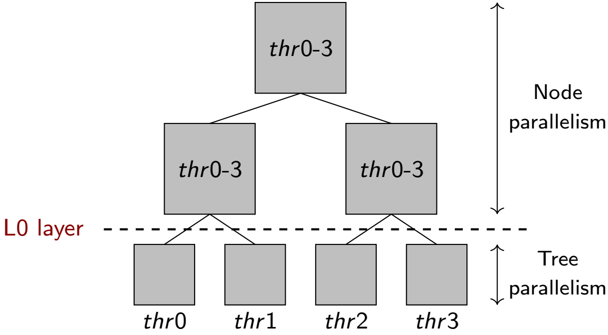
3.3 gain in sequential becomes **1.7** in multithreaded: why?

- LAI parts have become critical
- Update and Compress are memory-bound

Exploiting tree-based multithreading in MF solvers

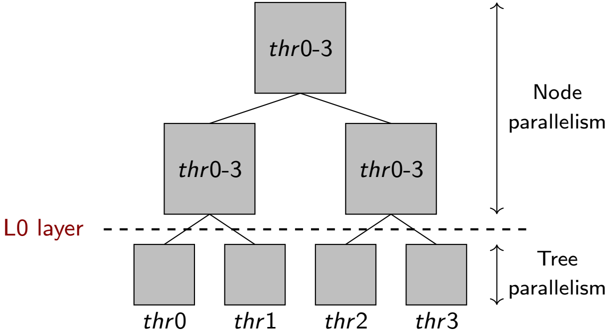


Exploiting tree-based multithreading in MF solvers



L'Excellent and Sid-Lakhdar (2014)

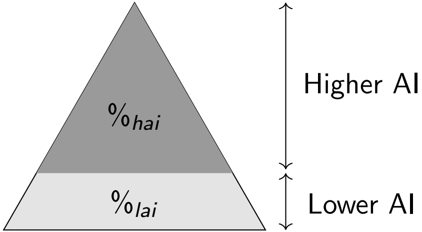
Exploiting tree-based multithreading in MF solvers



📄 L'Excellent and Sid-Lakhdar (2014)

$\Rightarrow$ how big an impact can tree-based multithreading make?

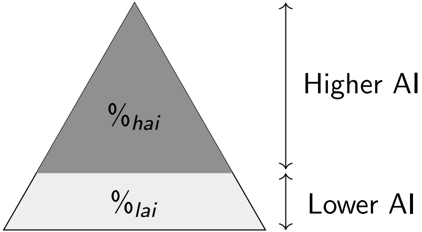
Impact of tree-based multithreading on BLR (24 threads)



	node only		node + tree	
	time	$\%_{lai}$	time	$\%_{lai}$
FR	509	21%		
BLR				

- In FR, top of the tree is dominant

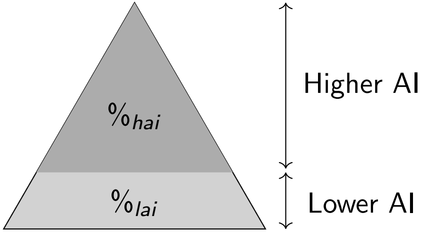
Impact of tree-based multithreading on BLR (24 threads)



	node only		node + tree	
	time	$\%_{lai}$	time	$\%_{lai}$
FR BLR	509	21%	424	13%

- In FR, top of the tree is dominant ⇒ tree MT brings little gain

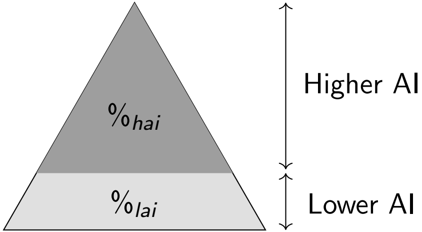
Impact of tree-based multithreading on BLR (24 threads)



	node only		node + tree	
	time	$\%_{lai}$	time	$\%_{lai}$
FR	509	21%	424	13%
BLR	307	35%		

- In FR, top of the tree is dominant $\Rightarrow$ tree MT brings little gain
- In BLR, bottom of the tree compresses less, becomes important

Impact of tree-based multithreading on BLR (24 threads)

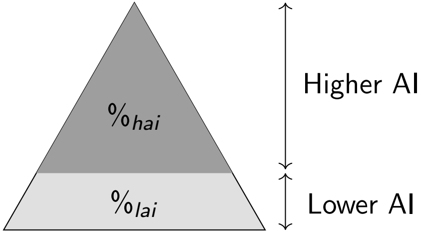


	node only		node + tree	
	time	$\%_{lai}$	time	$\%_{lai}$
FR	509	21%	424	13%
BLR	307	35%	221	24%

- In FR, top of the tree is dominant $\Rightarrow$ tree MT brings little gain
- In BLR, bottom of the tree compresses less, becomes important

$\Rightarrow$ **1.7** gain becomes **1.9** thanks to tree-based multithreading

Impact of tree-based multithreading on BLR (24 threads)



	node only		node + tree	
	time	$\%_{lai}$	time	$\%_{lai}$
FR	509	21%	424	13%
BLR	307	35%	221	24%

- In FR, top of the tree is dominant $\Rightarrow$ tree MT brings little gain
- In BLR, bottom of the tree compresses less, becomes important

$\Rightarrow$ **1.7** gain becomes **1.9** thanks to tree-based multithreading

Theoretical speedup

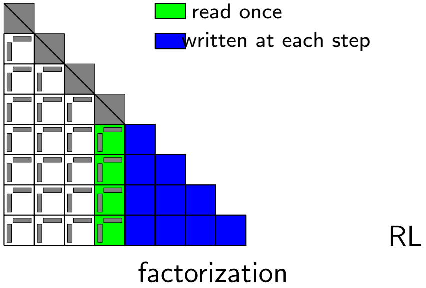
		tree only	node only	node + tree
$N \times N \times N$ grid	FR	$\mathcal{O}(1)$	$\mathcal{O}(N^3)$	$\mathcal{O}(N^4)$
	BLR	$\mathcal{O}(1)$	$\mathcal{O}(N^2 r^{1/2})$	$\mathcal{O}(N^3 r^{1/2})$

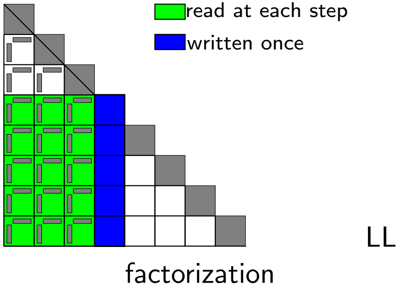
Right-looking Vs. Left-looking analysis (24 threads)

	FR time		BLR time	
	RL	LL	RL	LL
Update	338	336	110	67
Total	424	421	221	175

Right-looking Vs. Left-looking analysis (24 threads)

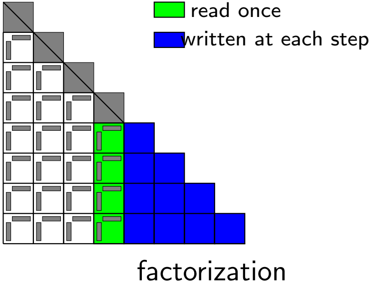
	FR time		BLR time	
	RL	LL	RL	LL
Update	338	336	110	67
Total	424	421	221	175



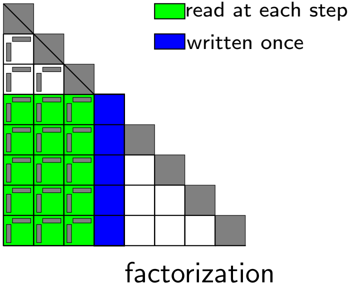


Right-looking Vs. Left-looking analysis (24 threads)

	FR time		BLR time	
	RL	LL	RL	LL
Update	338	336	110	67
Total	424	421	221	175



RL

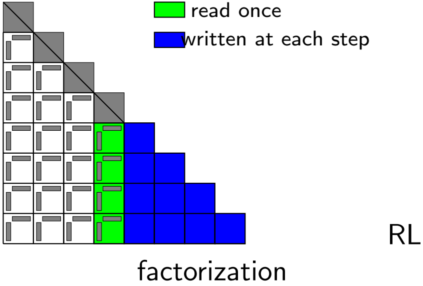


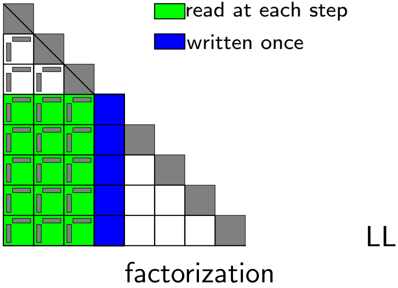
LL

⇒ Lower volume of memory transfers in LL (more critical in MT)

Right-looking Vs. Left-looking analysis (24 threads)

	FR time		BLR time	
	RL	LL	RL	LL
Update	338	336	110	67
Total	424	421	221	175

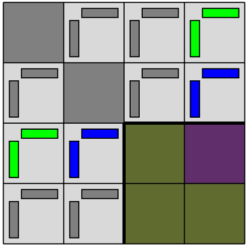


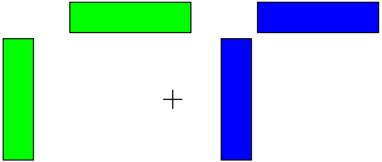


⇒ Lower volume of memory transfers in LL (more critical in MT)

Update is now less memory-bound: **1.9** gain becomes **2.4** in LL

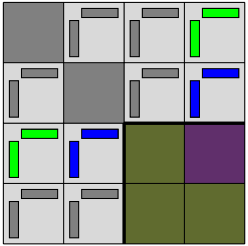
LUAR variant: accumulation and recompression

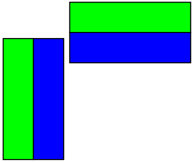




- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR

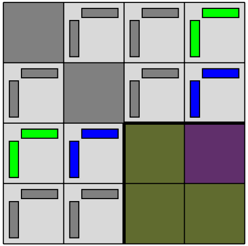
LUAR variant: accumulation and recompression

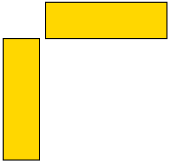




- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations

LUAR variant: accumulation and recompression

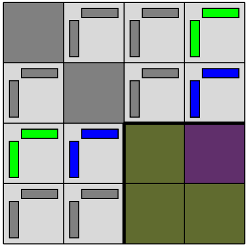


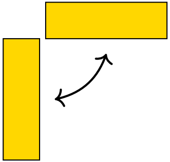


- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential recompression $\Rightarrow$ asymptotic complexity reduction?

    $\Rightarrow$ Designed and compared several recompression strategies

LUAR variant: accumulation and recompression

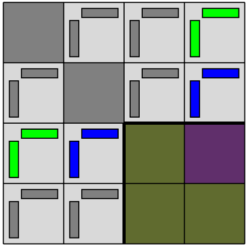


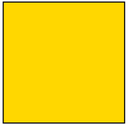


- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential recompression ⇒ asymptotic complexity reduction?

    ⇒ Designed and compared several recompression strategies

LUAR variant: accumulation and recompression



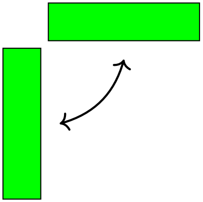


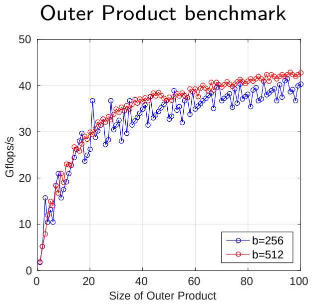
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential recompression $\Rightarrow$ asymptotic complexity reduction?

    $\Rightarrow$ Designed and compared several recompression strategies

Performance of Outer Product with LUA(R) (24 threads)

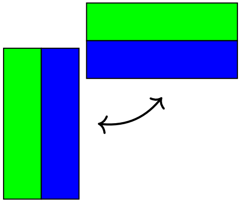
		LL	LUA	LUAR*
average size of Outer Product		16.5		
flops ($\times 10^{12}$)	Outer Product	3.8		
	Total	10.2		
time (s)	Outer Product	21		
	Total	175		

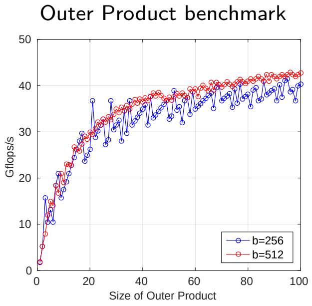




Performance of Outer Product with LUA(R) (24 threads)

		LL	LUA	LUAR*
average size of Outer Product		16.5	61.0	
flops ($\times 10^{12}$)	Outer Product	3.8	3.8	
	Total	10.2	10.2	
time (s)	Outer Product	21	14	
	Total	175	167	

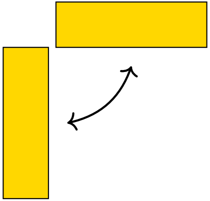




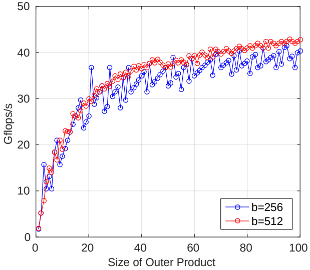
Performance of Outer Product with LUA(R) (24 threads)

		LL	LUA	LUAR*
average size of Outer Product		16.5	61.0	32.8
flops ($\times 10^{12}$)	Outer Product	3.8	3.8	1.6
	Total	10.2	10.2	8.1
time (s)	Outer Product	21	14	6
	Total	175	167	160

* All metrics include the Recompression overhead



Outer Product benchmark



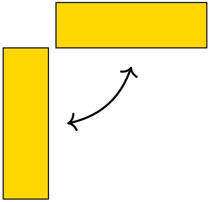
Performance of Outer Product with LUA(R) (24 threads)

		LL	LUA	LUAR*
average size of Outer Product		16.5	61.0	32.8
flops ($\times 10^{12}$)	Outer Product	3.8	3.8	1.6
	Total	10.2	10.2	8.1
time (s)	Outer Product	21	14	6
	Total	175	167	160

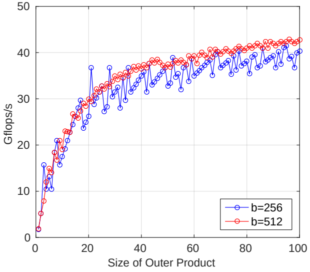
* All metrics include the Recompression overhead

Higher granularity and lower flops in Update:

$\Rightarrow$ **2.4** gain becomes **2.6**



Outer Product benchmark



Impact of machine properties on BLR: roofline model

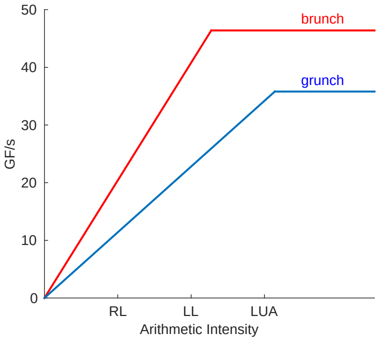
	specs		time (s) for BLR factorization		
	peak (GF/s)	bw (GB/s)	RL	LL	LUA
grunch (28 threads)	37	57	248	228	196
brunch (24 threads)	46	102	221	175	167

Impact of machine properties on BLR: roofline model

	specs		time (s) for BLR factorization		
	peak (GF/s)	bw (GB/s)	RL	LL	LUA
grunch (28 threads)	37	57	248	228	196
brunch (24 threads)	46	102	221	175	167

Arithmetic Intensity in BLR:

- LL > RL (lower volume of memory transfers)
- LUA > LL (higher granularities ⇒ more efficient cache use)

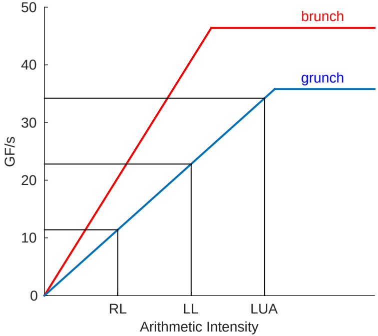


Impact of machine properties on BLR: roofline model

	specs		time (s) for BLR factorization		
	peak (GF/s)	bw (GB/s)	RL	LL	LUA
grunch (28 threads)	37	57	248	228	196
brunch (24 threads)	46	102	221	175	167

Arithmetic Intensity in BLR:

- LL > RL (lower volume of memory transfers)
- LUA > LL (higher granularities ⇒ more efficient cache use)

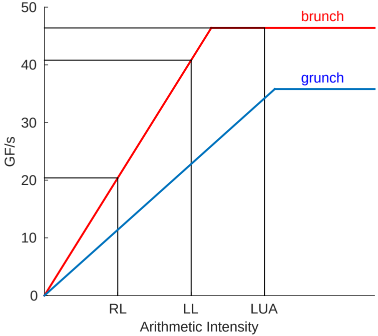


Impact of machine properties on BLR: roofline model

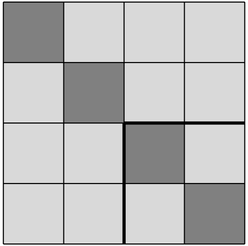
	specs		time (s) for BLR factorization		
	peak (GF/s)	bw (GB/s)	RL	LL	LUA
grunch (28 threads)	37	57	248	228	196
brunch (24 threads)	46	102	221	175	167

Arithmetic Intensity in BLR:

- LL > RL (lower volume of memory transfers)
- LUA > LL (higher granularities ⇒ more efficient cache use)



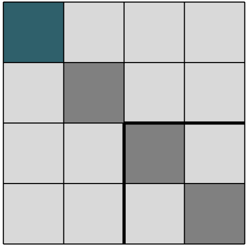
FCSU variant: compress before solve



- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential recompression $\Rightarrow$ asymptotic complexity reduction?

    $\Rightarrow$ Designed and compared several recompression strategies
- FCSU(+LUAR)

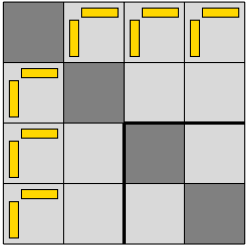
FCSU variant: compress before solve



- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential recompression ⇒ asymptotic complexity reduction?

    ⇒ Designed and compared several recompression strategies
- FCSU(+LUAR)

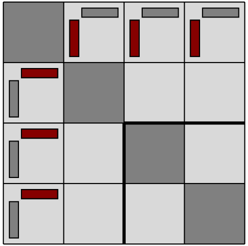
FCSU variant: compress before solve



- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential recompression ⇒ asymptotic complexity reduction?

    ⇒ Designed and compared several recompression strategies
- FCSU(+LUAR)
  - Compress performed before Solve

FCSU variant: compress before solve

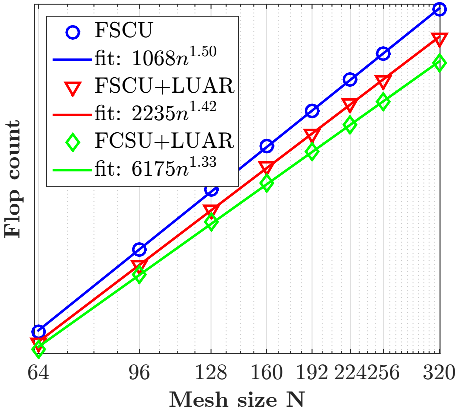


- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential recompression ⇒ asymptotic complexity reduction?
    ⇒ Designed and compared several recompression strategies
- FCSU(+LUAR)
  - Compress performed before Solve
  - Low-rank Solve ⇒ asymptotic complexity reduction?
  - On previous matrix: 160 → 111s ⇒ **2.6** gain becomes **3.7**

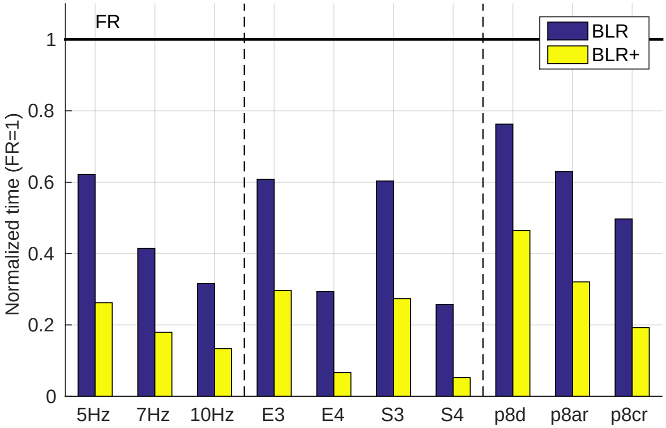
FCSU+LUAR improves asymptotic complexity! (see slide 57 for a proof ☺)

	FSCU	→ FCSU+LUAR
dense	$\mathcal{O}(m^{5/2} r^{1/2})$	$\to \mathcal{O}(m^2 r)$
sparse (3D)	$\mathcal{O}(N^5 r^{1/2})$	$\to \mathcal{O}(N^4 r)$

Experimental complexity



Multicore performance results (24 threads)



- "BLR": FSCU, right-looking, node only multithreading
- "BLR+": FCSU+LUAR, left-looking, node+tree multithreading

  Amestoy, Buttari, L'Excellent, M. (2019)

Combining (data) sparsity and low precisions

Two approaches: coarse and fine grain mixed precision

- **Coarse grain mixed precision:** Run baseline algorithm in low precision, refine the result to high precision
  - ☺ Simple and efficient, can rely on optimized libraries
  - ☹ Extra work for the refinement, not always guaranteed to work
- **Fine grain mixed precision:** Adapt the precision of each instruction/operation to achieve a given accuracy target
  - ☺ Optimal use of low precision, with guaranteed target accuracy
  - ☹ Much more intrusive, may be less efficient

Iterative refinement with BLR LU (coarse grain)

Error analysis: replace $\mathbf{u_f}$ by $\mathbf{u_f} + \varepsilon$ in the convergence conditions

Example on tminlet3M matrix
fp64 LU reference: time $\to$ 295.5 memory $\to$ 241.1

	time (s)		memory (GB)	
	LU-IR	GMRES-IR	LU-IR	GMRES-IR
FR	136.2	157.9	121.0	169.9

Iterative refinement with BLR LU (coarse grain)

Error analysis: replace $\mathbf{u_f}$ by $\mathbf{u_f} + \varepsilon$ in the convergence conditions

Example on tminlet3M matrix
fp64 LU reference: time $\rightarrow$ 295.5 memory $\rightarrow$ 241.1

	time (s)		memory (GB)	
	LU-IR	GMRES-IR	LU-IR	GMRES-IR
FR	136.2	157.9	121.0	169.9
$\varepsilon = 10^{-8}$	149.7	165.3	114.0	161.9

Iterative refinement with BLR LU (coarse grain)

Error analysis: replace $\mathbf{u_f}$ by $\mathbf{u_f} + \varepsilon$ in the convergence conditions

Example on tminlet3M matrix
fp64 LU reference: time $\rightarrow$ 295.5 memory $\rightarrow$ 241.1

	time (s)		memory (GB)	
	LU-IR	GMRES-IR	LU-IR	GMRES-IR
FR	136.2	157.9	121.0	169.9
$\varepsilon = 10^{-8}$	149.7	165.3	114.0	161.9
$\varepsilon = 10^{-6}$	**88.3**	98.8	82.4	93.8

Iterative refinement with BLR LU (coarse grain)

Error analysis: replace $\mathbf{u_f}$ by $\mathbf{u_f} + \varepsilon$ in the convergence conditions

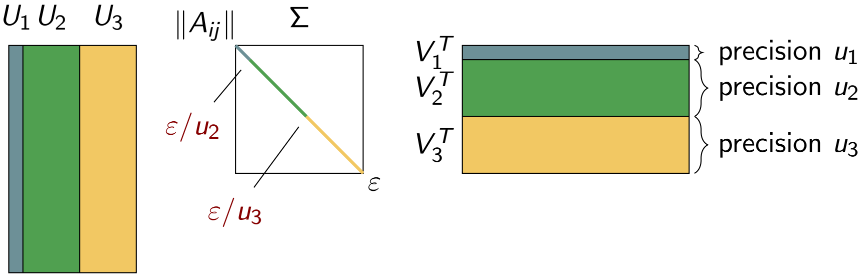
Example on tminlet3M matrix
fp64 LU reference: time $\rightarrow$ 295.5 memory $\rightarrow$ 241.1

	time (s)		memory (GB)	
	LU-IR	GMRES-IR	LU-IR	GMRES-IR
FR	136.2	157.9	121.0	169.9
$\varepsilon = 10^{-8}$	149.7	165.3	114.0	161.9
$\varepsilon = 10^{-6}$	**88.3**	98.8	82.4	93.8
$\varepsilon = 10^{-4}$	—	105.6	—	**70.9**

- GMRES-IR allows to push BLR further!

  Amestoy, Buttari, Higham, L'Excellent, M., Vieublé (2023)

Adaptive precision low rank compression

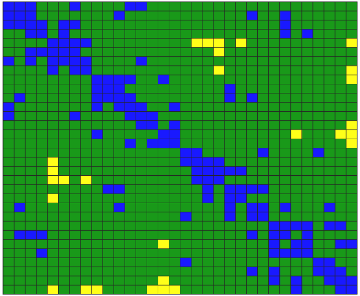


- **Adaptive precision compression:** partition U and V into q groups of decreasing precisions $u_1 \le \varepsilon < u_2 < \ldots < u_q$
- With p precisions and a partitioning such that $\|\Sigma_k\| \le \varepsilon\|A\|/u_k$, $\|A_{ij} - \widehat{U}_\varepsilon \Sigma_\varepsilon \widehat{V}_\varepsilon\| \lesssim (2p-1)\varepsilon\|A\|$
- If $\|A_{ij}\|/\|A\| \ll 1$, Σ_1 may be empty!

Adaptive precision BLR matrices

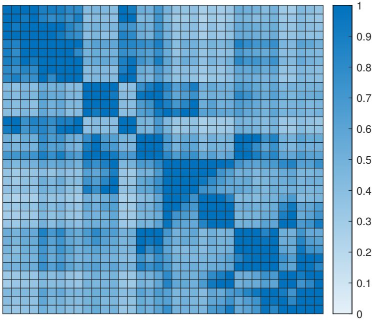
- If $\|A_{ij}\| \leq \varepsilon \|A\| / u_{\text{low}}$, block can be stored in precision u_{low}

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

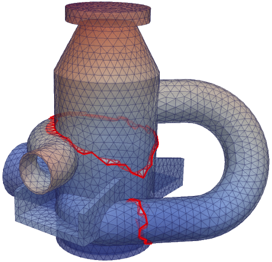


- **fp64**
- **fp32**
- **fp16**

Adaptive precision BLR compression



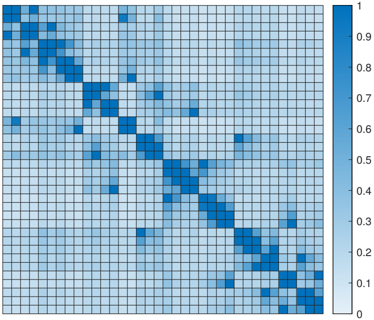
Normalized storage cost of each block



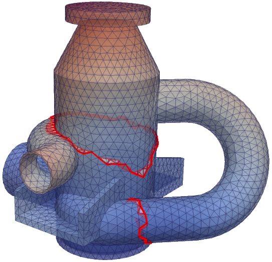
Matrix perf009d
(RIS pump from EDF)

100% entries in fp64

Adaptive precision BLR compression



Normalized storage cost of each block



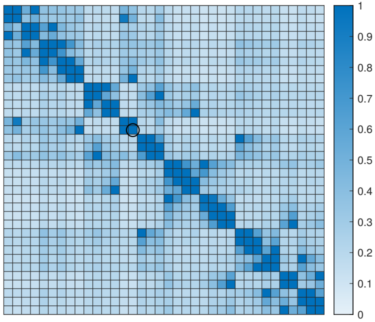
Matrix perf009d
(RIS pump from EDF)

100% entries in fp64

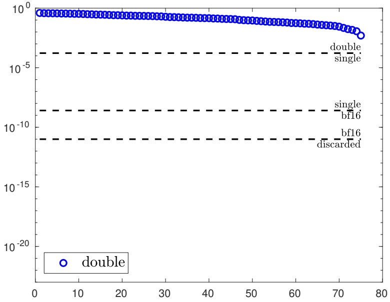
$$\rightarrow \begin{cases} 13\% \text{ in fp64} \\ 53\% \text{ in fp32} \\ 33\% \text{ in bfloat16} \end{cases}$$

$\Rightarrow 2\times$ storage reduction

Adaptive precision BLR compression



Normalized storage cost of each block



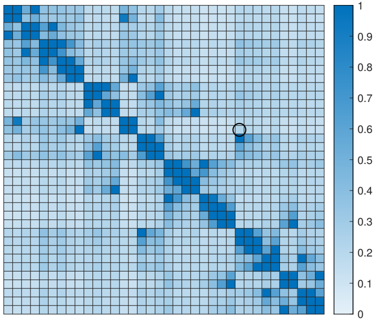
block (15,15)

100% entries in fp64

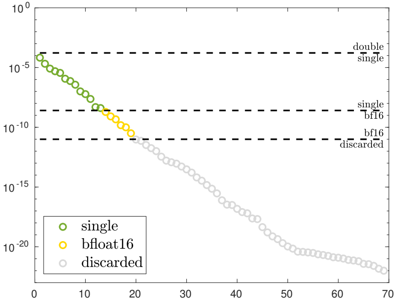
$$\rightarrow \begin{cases} 13\% \text{ in fp64} \\ 53\% \text{ in fp32} \\ 33\% \text{ in bfloat16} \end{cases}$$

$\Rightarrow 2\times$ storage reduction

Adaptive precision BLR compression



Normalized storage cost of each block



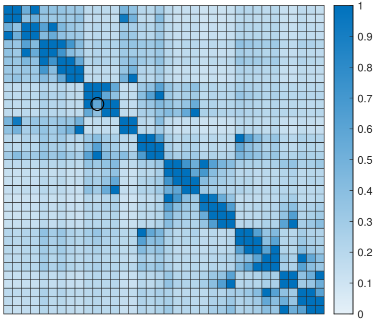
block (15,27)

100% entries in fp64

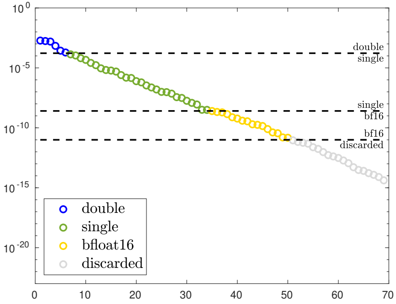
$$\to \begin{cases} 13\% \text{ in fp64} \\ 53\% \text{ in fp32} \\ 33\% \text{ in bfloat16} \end{cases}$$

$\Rightarrow$ $2\times$ storage reduction

Adaptive precision BLR compression



Normalized storage cost of each block



block (12,11)

100% entries in fp64

$$\rightarrow \begin{cases} 13\% \text{ in fp64} \\ 53\% \text{ in fp32} \\ 33\% \text{ in bfloat16} \end{cases}$$

$\Rightarrow$ 2$\times$ storage reduction

Low-rank admissibility condition
$k(m+n) \leq mn$ becomes

$$(\omega_1 k_1 + \omega_2 k_2 + \ldots + \omega_p k_p)(m+n) \leq mn$$

in mixed precision !

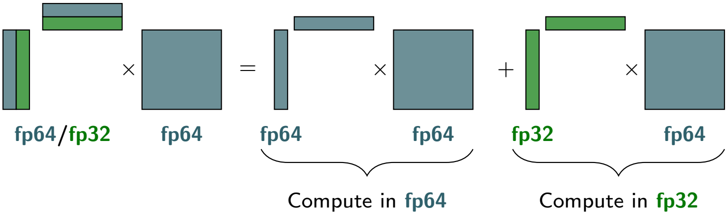
Adaptive precision BLR LU factorization

Stability of LU factorization: $\widehat{L}\widehat{U} = A + \Delta A$

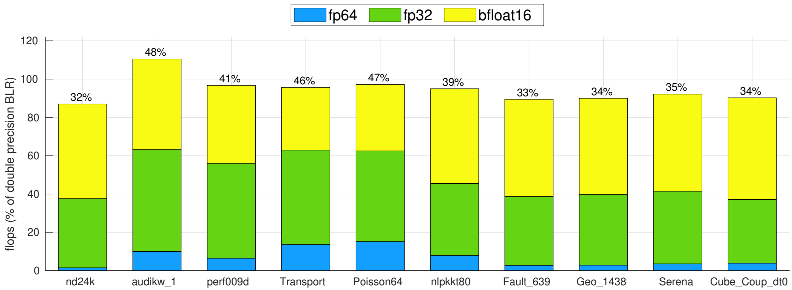
- **Standard LU** : $\|\Delta A\| \lesssim c_n u_1 \|A\|$
- **BLR LU** : $\|\Delta A\| \lesssim c'_n(\varepsilon + u_1)\|A\|$
- **Adaptive precision BLR LU** : $\|\Delta A\| \lesssim c''_n(\varepsilon + u_1)\|A\|$

  Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2022)

Example of kernel: LR $\times$ matrix multiplication:



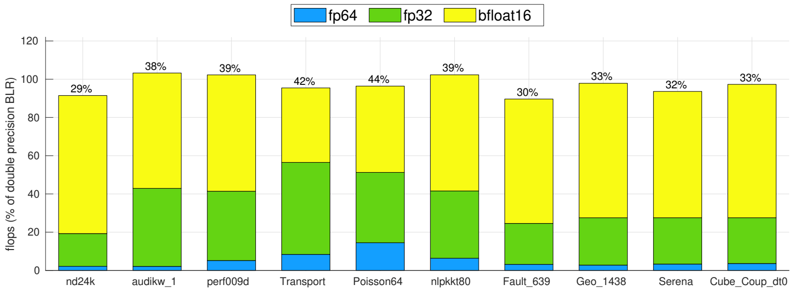
Adaptive precision BLR LU factorization



$$\varepsilon = 10^{-12}$$

Top of the bars: cost w.r.t. fp64 BLR, assuming
1 flop(fp64) = 2 flops(fp32) = 4 flops(bfloat16)

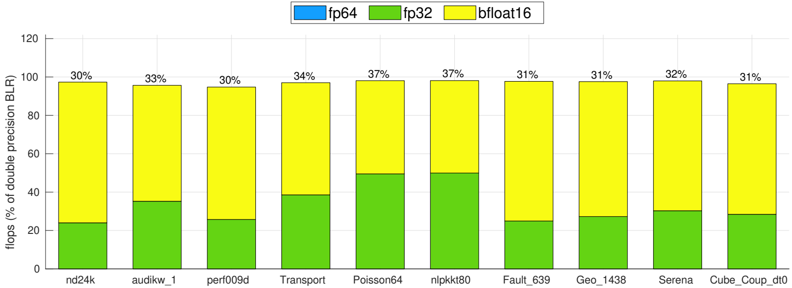
Adaptive precision BLR LU factorization



$\varepsilon = 10^{-9}$

Top of the bars: cost w.r.t. fp64 BLR, assuming
1 flop(fp64) = 2 flops(fp32) = 4 flops(bfloat16)

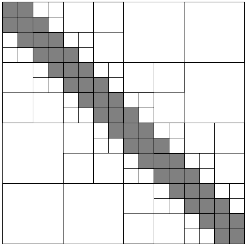
Adaptive precision BLR LU factorization



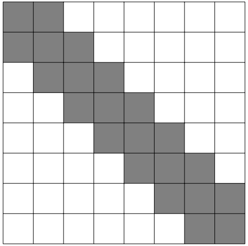
$$\varepsilon = 10^{-6}$$

Top of the bars: cost w.r.t. fp64 BLR, assuming
1 flop(fp64) = 2 flops(fp32) = 4 flops(bfloat16)

$\mathcal{H}$ and BLR matrices



$\mathcal{H}$ matrix Hackbusch (2015)



BLR matrix

- Theoretical complexity can be as low as $O(n)$
- Complex, hierarchical structure

- Theoretical complexity can be as low as $O(n^{4/3})$
- Simpler structure

BLR makes easier to preserve the numerical features of a direct solver and compromises well complexity, accuracy and performance

Nested dissection complexity formulas

$$\textbf{2D:} \quad \mathcal{C}_{sparse} = \sum_{\ell=0}^{\log N} 4^{\ell} \mathcal{C}_{dense}\left(\frac{N}{2^{\ell}}\right)$$

$$\textbf{3D:} \quad \mathcal{C}_{sparse} = \sum_{\ell=0}^{\log N} 8^{\ell} \mathcal{C}_{dense}\left(\frac{N^2}{4^{\ell}}\right)$$

Nested dissection complexity formulas

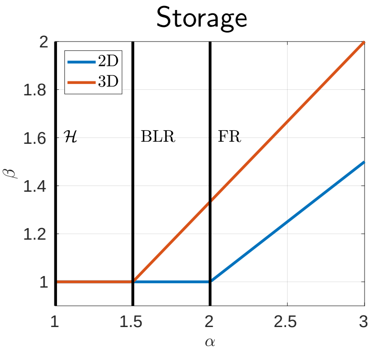
$$\textbf{2D:}\quad \mathcal{C}_{sparse} = \sum_{\ell=0}^{\log N} 4^{\ell}\,\mathcal{C}_{dense}\left(\frac{N}{2^{\ell}}\right) \quad \to \text{common ratio } 2^{2-\alpha}$$

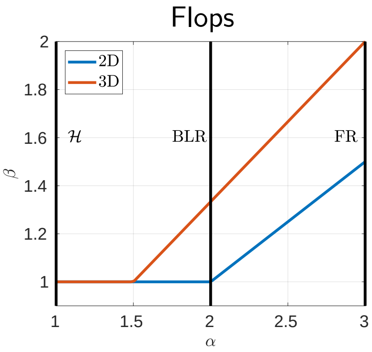
$$\textbf{3D:}\quad \mathcal{C}_{sparse} = \sum_{\ell=0}^{\log N} 8^{\ell}\,\mathcal{C}_{dense}\left(\frac{N^2}{4^{\ell}}\right) \quad \to \text{common ratio } 2^{3-2\alpha}$$

Assume $\mathcal{C}_{dense} = O(m^{\alpha})$. Then:

2D		3D	
	$\mathcal{C}_{sparse}(n)$		$\mathcal{C}_{sparse}(n)$
$\alpha > 2$	$O(n^{\alpha/2})$	$\alpha > 1.5$	$O(n^{2\alpha/3})$
$\alpha = 2$	$O(n \log n)$	$\alpha = 1.5$	$O(n \log n)$
$\alpha < 2$	$O(n)$	$\alpha < 1.5$	$O(n)$

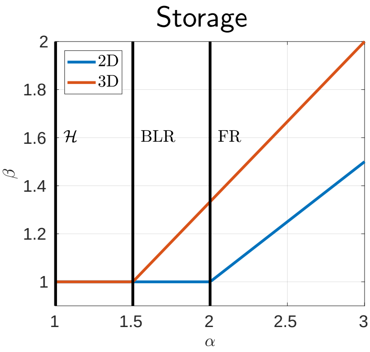
$$\mathcal{C}_{dense} = O(m^{\alpha}) \Rightarrow \mathcal{C}_{sparse} = O(n^{\beta})$$

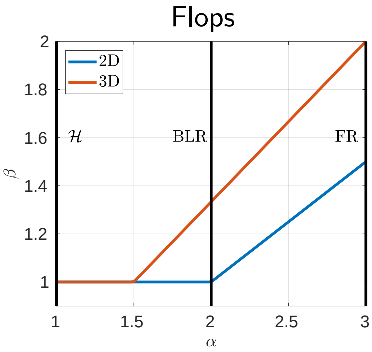




Bridging the gap between flat and hierarchical formats

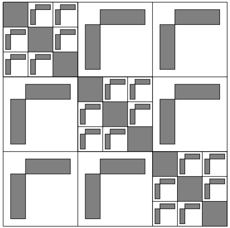
$$\mathcal{C}_{dense} = O(m^{\alpha}) \Rightarrow \mathcal{C}_{sparse} = O(n^{\beta})$$





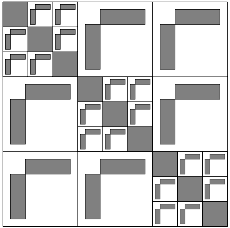
Key motivation: $\mathcal{C}_{dense} < O(m^2)$ (2D) or $O(m^{1.5})$ (3D) is enough to get $O(n)$ sparse complexity!

Complexity of the two-level BLR format



$$
\begin{aligned}
Storage &= cost_{LR} * nb_{LR} + cost_{BLR} * nb_{BLR} \\
&= O(br) * O((\frac{m}{b})^2) + O(b^{3/2} r^{1/2}) * O(\frac{m}{b}) \\
&= O(m^2 r / b + m(br)^{1/2}) \\
&= \mathbf{O(m^{4/3} r^{2/3})} \text{ for } b = (m^2 r)^{1/3}
\end{aligned}
$$

Complexity of the two-level BLR format



$$Storage = cost_{LR} * nb_{LR} + cost_{BLR} * nb_{BLR}$$
$$= O(br) * O((\frac{m}{b})^2) + O(b^{3/2}r^{1/2}) * O(\frac{m}{b})$$
$$= O(m^2 r/b + m(br)^{1/2})$$
$$= \mathbf{O(m^{4/3}r^{2/3})} \text{ for } b = (m^2 r)^{1/3}$$

Similarly, we can prove:

$$FlopLU = \mathbf{O(m^{5/3}r^{4/3})} \text{ for } b = (m^2 r)^{1/3}$$

		FR	BLR	2-BLR	...	$\mathcal{H}$
storage	dense	$O(m^2)$	$O(m^{1.5})$	$O(m^{1.33})$	...	$O(m \log m)$
	sparse	$O(n^{1.33})$	$O(n \log n)$	$O(n)$	...	$O(n)$
flop LU	dense	$O(m^3)$	$O(m^2)$	$O(m^{1.66})$	...	$O(m \log^3 m)$
	sparse	$O(n^2)$	$O(n^{1.33})$	$O(n^{1.11})$	...	$O(n)$

Multilevel BLR complexity

Main result

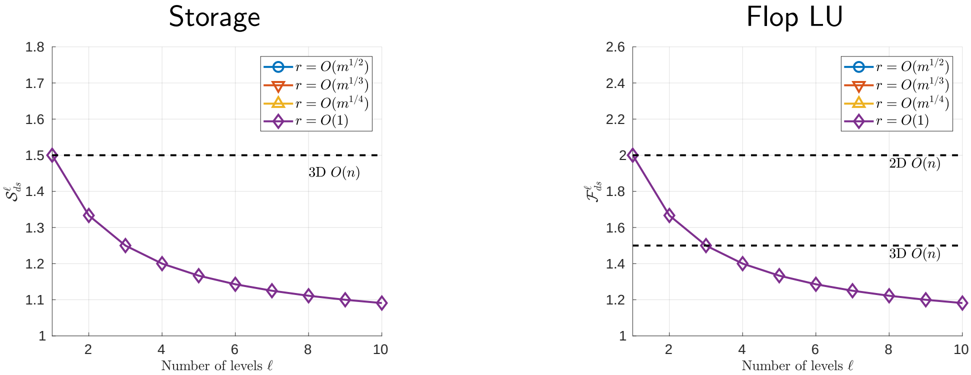
For $b = m^{\ell/(\ell+1)} r^{1/(\ell+1)}$, the $\ell-$level complexities are:

$$Storage = \mathbf{O(m^{(\ell+2)/(\ell+1)} r^{\ell/(\ell+1)})}$$
$$FlopLU = \mathbf{O(m^{(\ell+3)/(\ell+1)} r^{2\ell/(\ell+1)})}$$

Proof: by induction. Amestoy, Buttari, L'Excellent, M. (2019)

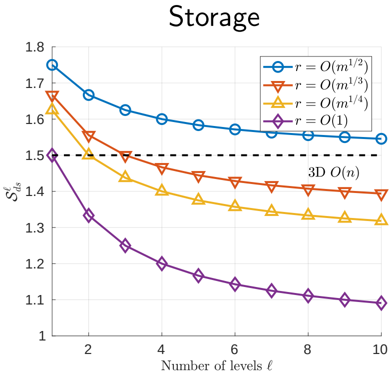
- Simple way to finely control the desired complexity
- Block size $b \propto O(m^{1-1/(\ell+1)}) \ll O(m)$
  $\Rightarrow$ larger blocks that can be efficiently processed in shared-memory

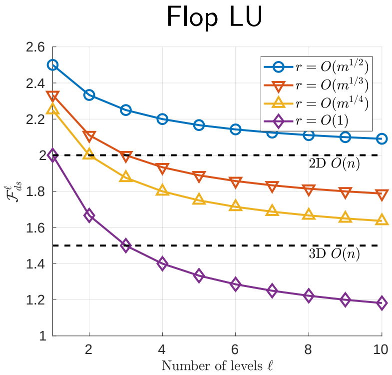
Influence of the number of levels ℓ



- If $r = O(1)$, can achieve $O(n)$ storage complexity with only two levels and $O(n \log n)$ flop complexity with three levels

Influence of the number of levels ℓ

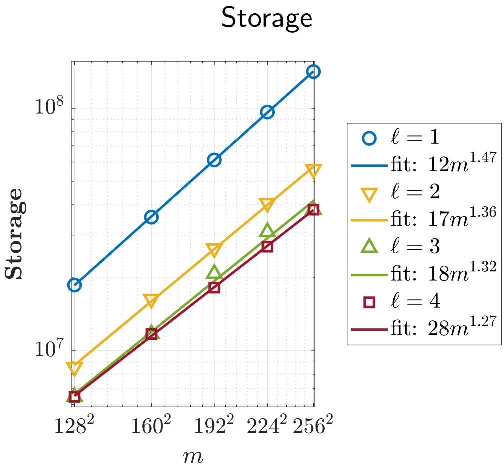


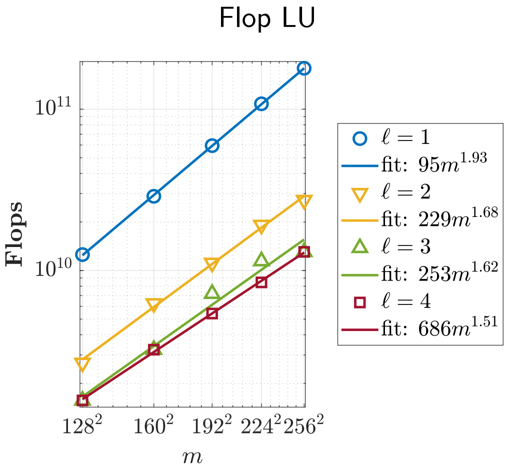


- If $r = O(1)$, can achieve $O(n)$ storage complexity with only two levels and $O(n \log n)$ flop complexity with three levels

- For higher ranks, improvement rate rapidly decreases:
the first few levels achieve most of the asymptotic gain

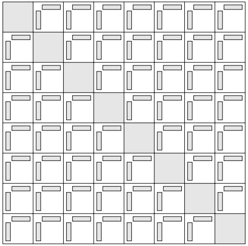
Experimental MBLR complexity (Poisson)



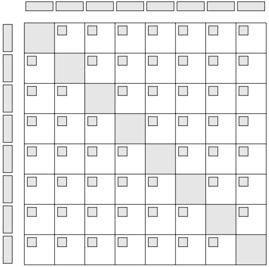


- Experimental complexity in relatively good agreement with theoretical one
- Asymptotic gain decreases with levels

The BLR^2 format

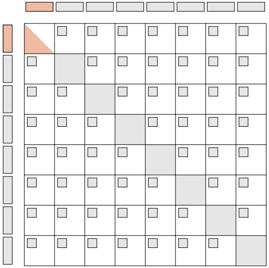


The BLR^2 format



- Common basis of size $s \geq r$ for all the blocks in a row/column

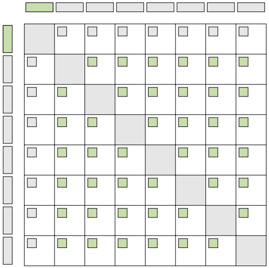
The BLR^2 format



- Solve: only involves the common pivotal row and column bases

- Common basis of size $s \geq r$ for all the blocks in a row/column

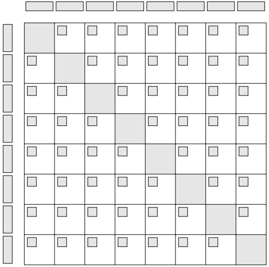
The BLR^2 format



- Solve: only involves the common pivotal row and column bases
- Update: only involves the common pivotal row and column bases and coupling matrices

- Common basis of size $s \geq r$ for all the blocks in a row/column

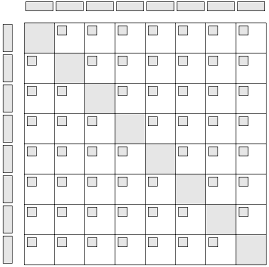
The BLR^2 format



- Solve: only involves the common pivotal row and column bases
- Update: only involves the common pivotal row and column bases and coupling matrices

- Common basis of size $s \geq r$ for all the blocks in a row/column
- If $s = \mathcal{O}(1)$ storage complexity is $\mathcal{O}(n)$ and flops complexity is $\mathcal{O}(n^{1.2})$

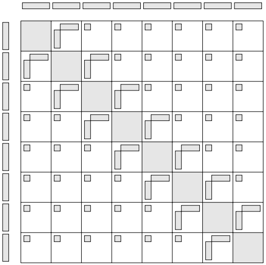
The BLR^2 format



- Solve: only involves the common pivotal row and column bases
- Update: only involves the common pivotal row and column bases and coupling matrices

- Common basis of size $s \geq r$ for all the blocks in a row/column
- If $s = \mathcal{O}(1)$ storage complexity is $\mathcal{O}(n)$ and flops complexity is $\mathcal{O}(n^{1.2})$
- In practice s is usually larger

The BLR^2 format



- Solve: only involves the common pivotal row and column bases
- Update: only involves the common pivotal row and column bases and coupling matrices

- Common basis of size $s \geq r$ for all the blocks in a row/column
- If $s = \mathcal{O}(1)$ storage complexity is $\mathcal{O}(n)$ and flops complexity is $\mathcal{O}(n^{1.2})$
- In practice s is usually larger
- Keep high-rank blocks off the common basis $\rightarrow$ substantial storage gains but complex factorization
- Ashcraft, Buttari, M. (2021) (one of the possible research papers to be read for the evaluation)

Pen & paper exercise (4 points)

- Goal: prove the reduced $\mathcal{O}(m^2 r)$ complexity of the FCSU+LUAR variant given on slide 37
- The accumulated updates always recompress to rank r no matter how many updates have been accumulated
- Suggested steps:
  - What is the cost of the FR-LR solve operation? How many times do we perform it?
  - What is the cost of the Recompress operation? How many times do we perform it?
  - Add these two new operations in the table on slide 23. What else changes in the table?
  - Recompute the optimal complexity as previously.

Practical exercise (4 points)

- Goal: implement BLR LU factorization and assess storage, flops and time gains w.r.t. matrix size and ε accuracy
- Already provided:
  - FR factorization (`FR_factorization.m`)
  - Basic kernels and utilitary routines
  - Main test launcher (`test.m`)
  - BLR factorization template, initialized identical to FR factorization (`BLR_factorization.m`) $\Rightarrow$ this is the file you need to modify
- 3 test matrices given, root separator of a 3D Poisson problem of dimensions $N \times N \times N$. Can change value of N (30, 50, or 70) in `test.m`
- Choice of BLR LU variant left up to you (vanilla FSCU is certainly the simplest)