Block Low-Rank Matrices:

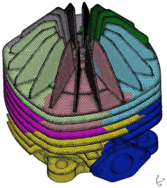
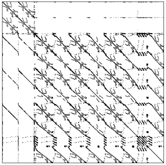
Main Results and Recent Advances

Theo Mary
University of Manchester, School of Mathematics

Grenoble, 5 July 2018

M\cr NA
Manchester Numerical Analysis

Context

 →  → 

Linear system $Ax = b$

Often a keystone in scientific computing applications (discretization of PDEs, step of an optimization method, ...)

Matrix sparsity

A sparse matrix is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Large-scale systems

Increasingly faster computers available, need to efficiently make use of them

Iterative vs direct methods

Iterative methods

Build sequence x_k converging towards x

- ☺ Computational cost: $\mathcal{O}(n)$ operations/iteration and memory
- ☹ Convergence is application-dependent

Direct methods

Factorize $A = LU$ and solve $LUx = b$

- ☺ Numerically reliable
- ☹ Computational cost: $\mathcal{O}(n^2)$ operations, $\mathcal{O}(n^{4/3})$ memory
  Practical example on a 1000^3 27-point Helmholtz problem:
  15 ExaFlops and 209 TeraBytes for factors!

Iterative vs direct methods

Iterative methods

Build sequence x_k converging towards x

- ☺ Computational cost: $\mathcal{O}(n)$ operations/iteration and memory
- ☹ Convergence is application-dependent

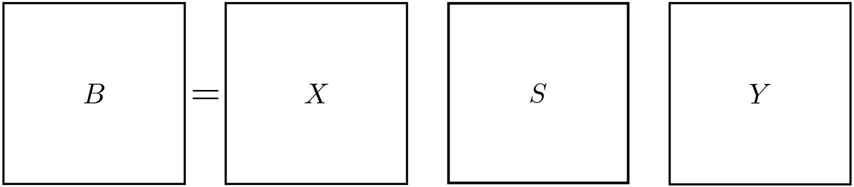
Direct methods

Factorize $A = LU$ and solve $LUx = b$

- ☺ Numerically reliable
- ☹ Computational cost: $\mathcal{O}(n^2)$ operations, $\mathcal{O}(n^{4/3})$ memory
  Practical example on a 1000^3 27-point Helmholtz problem:
  15 ExaFlops and 209 TeraBytes for factors!

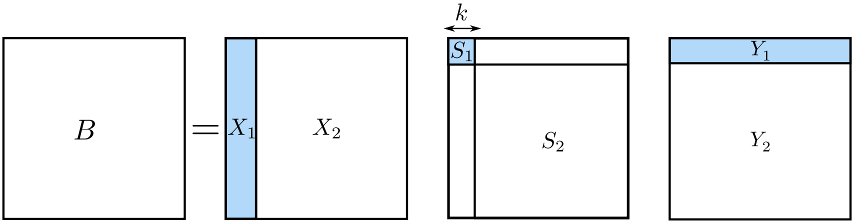
Our objective:
reduce the cost of sparse direct solvers ...
...while maintaining their numerical reliability

Take a dense matrix B of size $b \times b$ and compute its SVD $B = XSY$:



Low-rank matrices

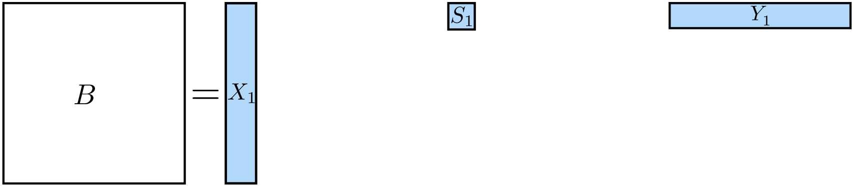
Take a dense matrix B of size $b \times b$ and compute its SVD $B = XSY$:



$k = \min\{k \le b; \sigma_{k+1} \le \varepsilon\}$ is the numerical rank at accuracy ε

Low-rank matrices

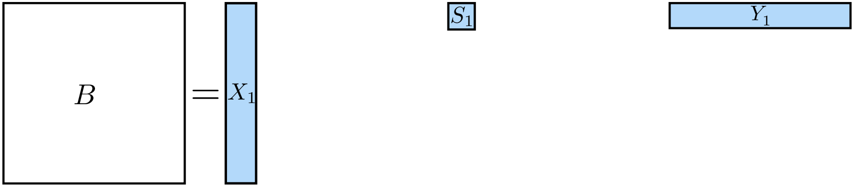
Take a dense matrix B of size $b \times b$ and compute its SVD $B = XSY$:



$k = \min\{k \le b; \sigma_{k+1} \le \varepsilon\}$ is the numerical rank at accuracy ε

$\tilde{B} = X_1 S_1 Y_1$ is a low-rank approximation to B: $\|B - \tilde{B}\|_2 \le \varepsilon$

Take a dense matrix B of size $b \times b$ and compute its SVD $B = XSY$:

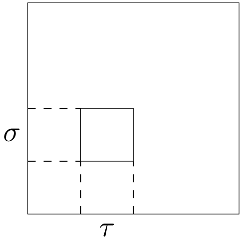


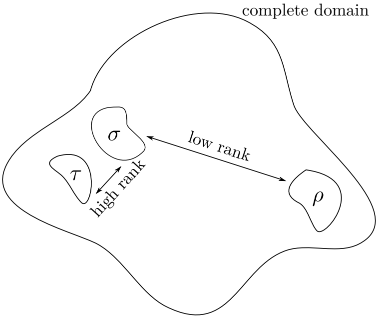
$k = \min\{k \le b; \sigma_{k+1} \le \varepsilon\}$ is the numerical rank at accuracy ε

$\tilde{B} = X_1 S_1 Y_1$ is a low-rank approximation to B: $\|B - \tilde{B}\|_2 \le \varepsilon$

Storage savings: $b^2/2bk = b/2k$
Similar flops savings when used in most linear algebra kernels

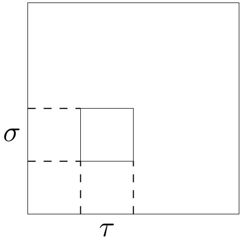
Most matrices are not low-rank in general but in some applications they exhibit low-rank blocks

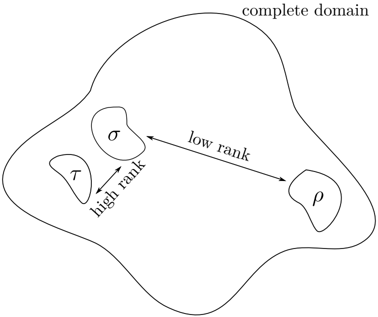




A block B represents the interaction between two subdomains σ and τ.
Small diameter and far away $\Rightarrow$ low numerical rank.

Most matrices are not low-rank in general but in some applications they exhibit low-rank blocks

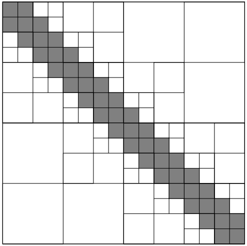




A block B represents the interaction between two subdomains σ and τ.
Small diameter and far away $\Rightarrow$ low numerical rank.

How to choose a good block partitioning of the matrix?

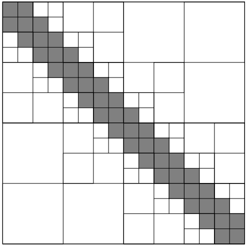
$\mathcal{H}$ and BLR matrices



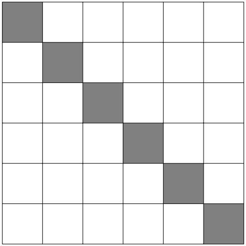
$\mathcal{H}$-matrix

- Nearly linear complexity
- Complex, hierarchical structure

$\mathcal{H}$ and BLR matrices



$\mathcal{H}$-matrix

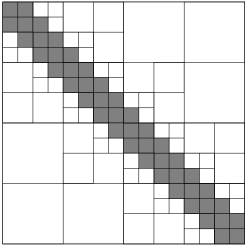


BLR matrix

- Nearly linear complexity
- Complex, hierarchical structure

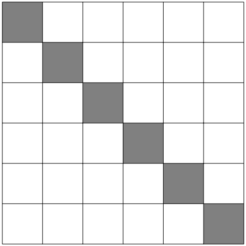
- Superlinear complexity
- Simple, flat structure

$\mathcal{H}$ and BLR matrices



$\mathcal{H}$-matrix

- Nearly linear complexity
- Complex, hierarchical structure



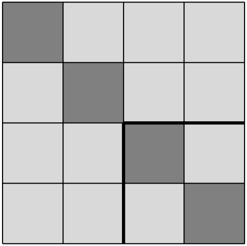
BLR matrix

- Superlinear complexity
- Simple, flat structure

BLR is a comprise between complexity and performance:

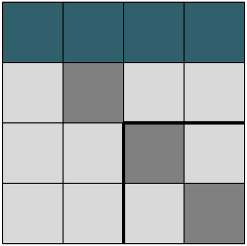
- Small blocks $\Rightarrow$ can fit on single shared-memory node
- No global order between blocks $\Rightarrow$ flexible data distribution
- Easy to handle numerical pivoting

Standard BLR factorization: FSCU



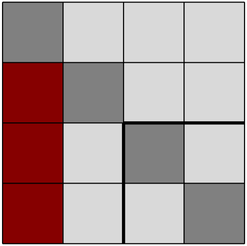
- FSCU

Standard BLR factorization: FSCU



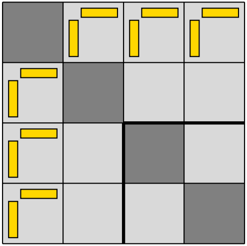
- FSCU (Factor,
- Easy to handle numerical pivoting, a critical feature often lacking in other low-rank solvers

Standard BLR factorization: FSCU



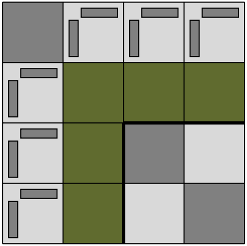
- FSCU (Factor, Solve,
- Easy to handle numerical pivoting, a critical feature often lacking in other low-rank solvers

Standard BLR factorization: FSCU



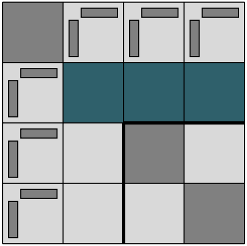
- FSCU (Factor, Solve, Compress,
- Easy to handle numerical pivoting, a critical feature often lacking in other low-rank solvers

Standard BLR factorization: FSCU



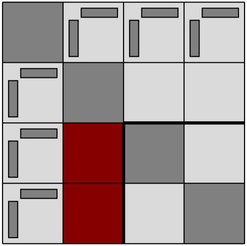
- FSCU (Factor, Solve, Compress, Update)
- Easy to handle numerical pivoting, a critical feature often lacking in other low-rank solvers

Standard BLR factorization: FSCU



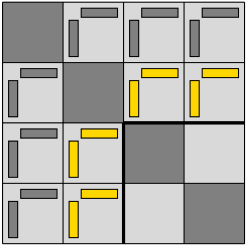
- FSCU (Factor, Solve, Compress, Update)
- Easy to handle numerical pivoting, a critical feature often lacking in other low-rank solvers

Standard BLR factorization: FSCU



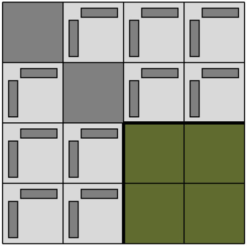
- FSCU (Factor, Solve, Compress, Update)
- Easy to handle numerical pivoting, a critical feature often lacking in other low-rank solvers

Standard BLR factorization: FSCU



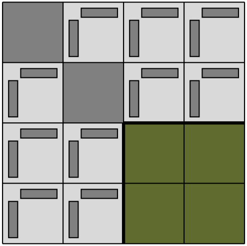
- FSCU (Factor, Solve, Compress, Update)
- Easy to handle numerical pivoting, a critical feature often lacking in other low-rank solvers

Standard BLR factorization: FSCU



- FSCU (Factor, Solve, Compress, Update)
- Easy to handle numerical pivoting, a critical feature often lacking in other low-rank solvers

Standard BLR factorization: FSCU



- FSCU (Factor, Solve, Compress, Update)
- Easy to handle numerical pivoting, a critical feature often lacking in other low-rank solvers
- Potential of this variant was studied in

Amestoy, Ashcraft, Boiteau, Buttari, L'Excellent, and Weisbecker, *Improving Multifrontal Methods by Means of Block Low-Rank Representations*, SIAM J. Sci. Comput. (2015).

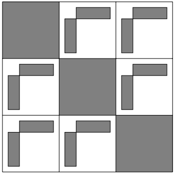
Outline

1. **Complexity**
   $\Rightarrow$ Joint work with P. Amestoy, A. Buttari, J.-Y. L'Excellent
2. **Parallelism**
   $\Rightarrow$ Joint work with PA, AB, JYL
3. **Comparison with HSS**
   $\Rightarrow$ Joint work with PA, AB, JYL, P. Ghysels, X. S. Li, F.-H. Rouet
4. **Multilevel BLR Matrices**
   $\Rightarrow$ Joint work with PA, AB, JYL
5. **Error Analysis**
   $\Rightarrow$ Joint work with N. Higham
6. **Fast BLR Matrix Arithmetic**
   $\Rightarrow$ Ongoing work

Complexity

Computing the BLR complexity

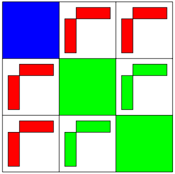
Assume all off-diagonal blocks are low-rank. Then:



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

Computing the BLR complexity

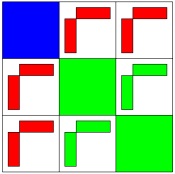
Assume all off-diagonal blocks are low-rank. Then:



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

$$
\begin{aligned}
FlopLU &= cost_{getrf} * nb_{getrf} + cost_{trsm} * nb_{trsm} + cost_{gemm} * nb_{gemm} \\
&= O(b^3) * O(\frac{m}{b}) + O(b^2 r) * O((\frac{m}{b})^2) + O(br^2) * O((\frac{m}{b})^3) \\
&= O(mb^2 + m^2 r + m^3 r^2 / b^2) \\
&= \mathbf{O(m^2 r)} \text{ for } b = (mr)^{1/2}
\end{aligned}
$$

Assume all off-diagonal blocks are low-rank. Then:



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

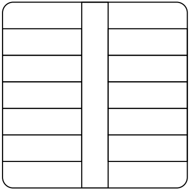
$$\begin{aligned} FlopLU &= cost_{getrf} * nb_{getrf} + cost_{trsm} * nb_{trsm} + cost_{gemm} * nb_{gemm} \\ &= O(b^3) * O(\frac{m}{b}) + O(b^2 r) * O((\frac{m}{b})^2) + O(br^2) * O((\frac{m}{b})^3) \\ &= O(mb^2 + m^2 r + m^3 r^2/b^2) \\ &= \mathbf{O(m^2 r)} \text{ for } b = (mr)^{1/2} \end{aligned}$$

Result holds if a **constant** number of off-diag. blocks is full-rank.
⇒ how to ensure this condition holds?

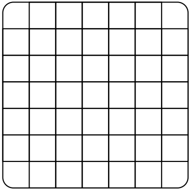
BLR admissibility condition

BLR-admissibility condition of a partition $\mathcal{P}$

$$\mathcal{P} \text{ is admissible } \Leftrightarrow \begin{cases} \#\{\sigma, \quad \sigma \times \tau \in \mathcal{P} \text{ is full-rank}\} \leq q \\ \#\{\tau, \quad \sigma \times \tau \in \mathcal{P} \text{ is full-rank}\} \leq q \end{cases}$$



Non-Admissible

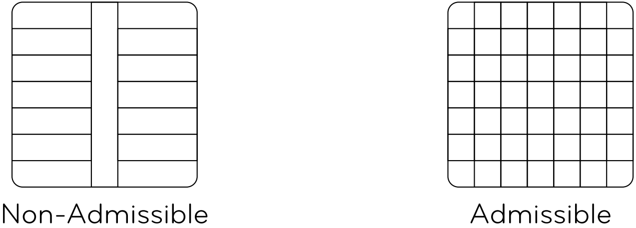


Admissible

BLR admissibility condition

BLR-admissibility condition of a partition $\mathcal{P}$

$$\mathcal{P} \text{ is admissible } \Leftrightarrow \begin{cases} \#\{\sigma, \quad \sigma \times \tau \in \mathcal{P} \text{ is full-rank}\} \leq q \\ \#\{\tau, \quad \sigma \times \tau \in \mathcal{P} \text{ is full-rank}\} \leq q \end{cases}$$

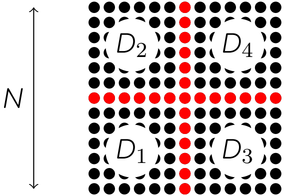


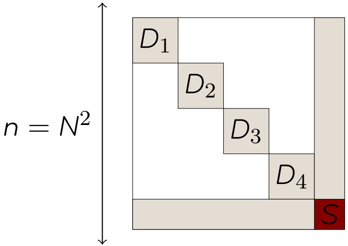
Main result

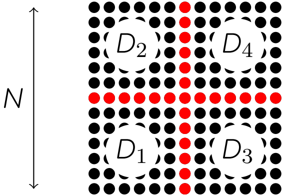
For any matrix, we can build an admissible $\mathcal{P}$ for $q = \mathcal{O}(1)$, s.t. the maximal rank of the admissible blocks of A is $r = \mathcal{O}\left(r_{max}^{\mathcal{H}}\right)$

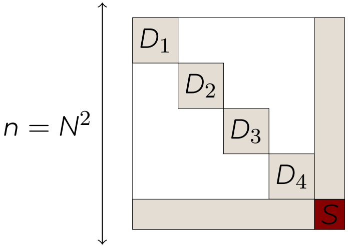
Amestoy, Buttari, L'Excellent, and Mary, *On the Complexity of the Block Low-Rank Multifrontal Factorization*, SIAM J. Sci. Comput. (2017).

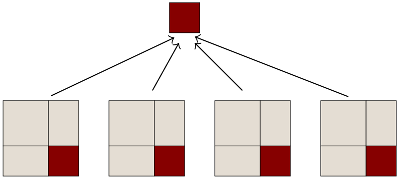
From dense to sparse: nested dissection











Proceed recursively to compute separator tree

Factorizing a sparse matrix amounts to factorizing a sequence of dense matrices

$\Rightarrow$

sparse complexity is directly derived from dense one

Nested dissection complexity formulas

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

Nested dissection complexity formulas

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

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

2D: $\mathcal{C}_{sparse} = \sum_{\ell=0}^{\log N} 4^{\ell} \mathcal{C}_{dense}(\frac{N}{2^{\ell}})$ → common ratio $2^{2-\alpha}$

3D: $\mathcal{C}_{sparse} = \sum_{\ell=0}^{\log N} 8^{\ell} \mathcal{C}_{dense}(\frac{N^2}{4^{\ell}})$ → common ratio $2^{3-2\alpha}$

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

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

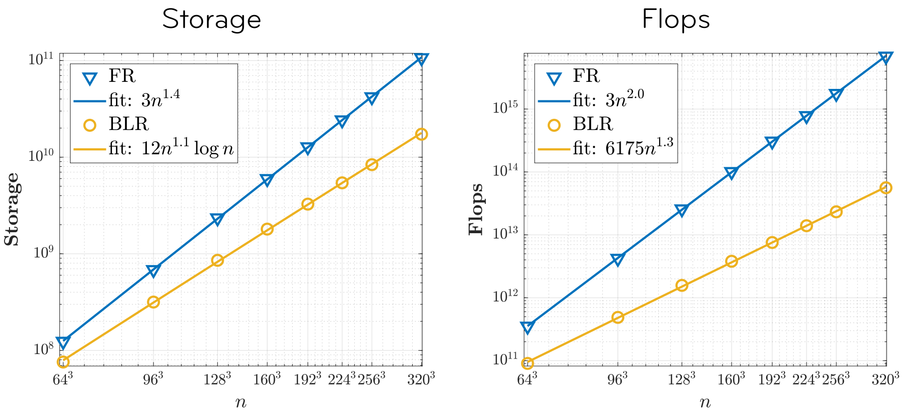
Complexity of the BLR factorization

		storage	flops
dense	FR	$O(m^2)$	$O(m^3)$
	BLR	$O(m^{3/2})$	$O(m^2)$
sparse 2D	FR	$O(n \log n)$	$O(n^{3/2})$
	BLR	$O(n)$	$O(n \log n)$
sparse 3D	FR	$O(n^{4/3})$	$O(n^2)$
	BLR	$O(n \log n)$	$O(n^{4/3})$

(assuming $r = O(1)$)

- Significant asymptotic complexity reduction compared to FR
- Almost optimal for sparse 2D problems!!
- Still superlinear in 3D

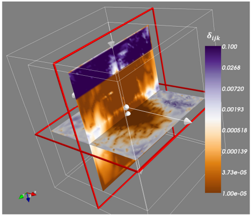
Experimental complexity fit: Poisson ($\varepsilon = 10^{-10}$)



- Good agreement with theoretical complexity:
  - Storage: $O(n \log n) \to O(n^{1.1} \log n)$
  - Flops: $O(n^{4/3}) \to O(n^{1.3})$

Parallelism

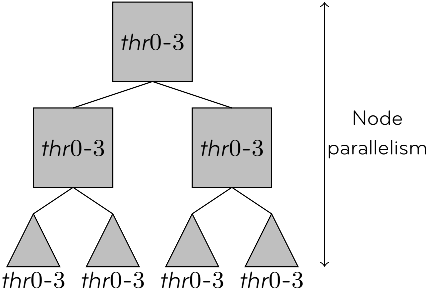
Matrix S3
Double complex (z) symmetric
Electromagnetics application (CSEM)
3.3 millions unknowns
Required accuracy: $\varepsilon = 10^{-7}$



	flops ($\times 10^{12}$)	time (1 core)	time (24 cores)
FR	78.0	7390	509
BLR	10.2	2242	309
ratio	7.7	3.3	1.7

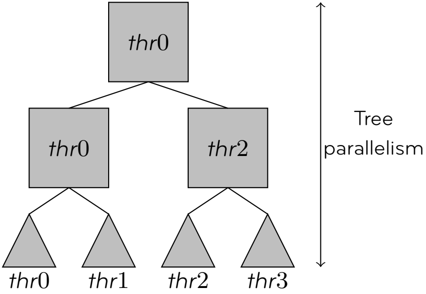
7.7 gain in flops only translated to a **1.7** gain in time:
Can we do better?

Exploiting tree-based multithreading in MF solvers

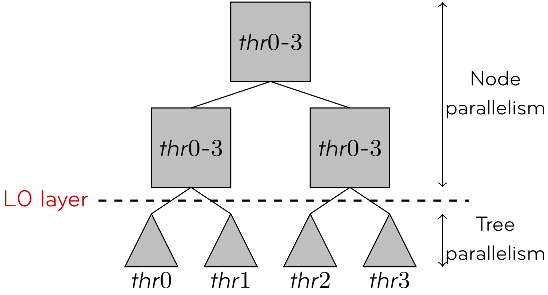


- Node parallelism approach based on OpenMP loops

Exploiting tree-based multithreading in MF solvers



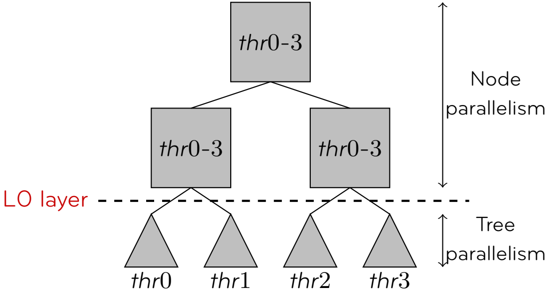
- Node parallelism approach based on OpenMP loops



- Node parallelism approach based on OpenMP loops
- Node+tree parallelism approach based on Sid-Lakhdar's PhD

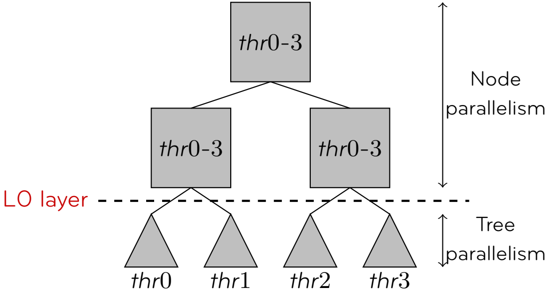
L'Excellent and Sid-Lakhdar, *A study of shared-memory parallelism in a multifrontal solver*, Parallel Computing (2014).

Exploiting tree-based multithreading in MF solvers



- Node parallelism approach based on OpenMP loops
- Node+tree parallelism approach based on Sid-Lakhdar's PhD
  - L'Excellent and Sid-Lakhdar, *A study of shared-memory parallelism in a multifrontal solver*, Parallel Computing (2014).
- In FR, top of the tree is dominant ⇒ tree MT brings little gain

Exploiting tree-based multithreading in MF solvers

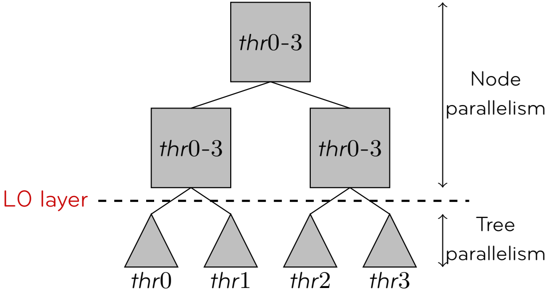


- Node parallelism approach based on OpenMP loops
- Node+tree parallelism approach based on Sid-Lakhdar's PhD

  L'Excellent and Sid-Lakhdar, *A study of shared-memory parallelism in a multifrontal solver*, Parallel Computing (2014).

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

Exploiting tree-based multithreading in MF solvers



- Node parallelism approach based on OpenMP loops
- Node+tree parallelism approach based on Sid-Lakhdar's PhD

  L'Excellent and Sid-Lakhdar, *A study of shared-memory parallelism in a multifrontal solver*, Parallel Computing (2014).

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

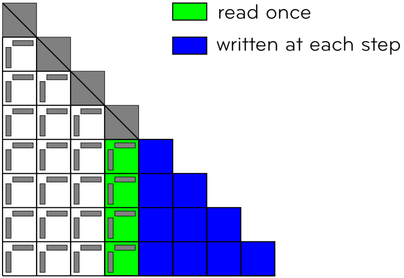
⇒ **1.7** gain becomes **1.9** thanks to tree-based multithreading

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

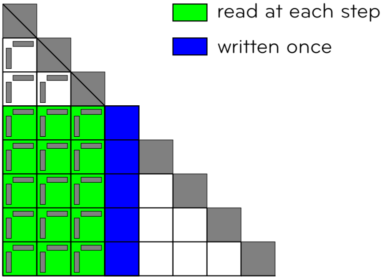
	FR time		BLR time	
	RL	LL	RL	LL
Update	338	336	110	67
Total	424	421	221	175

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



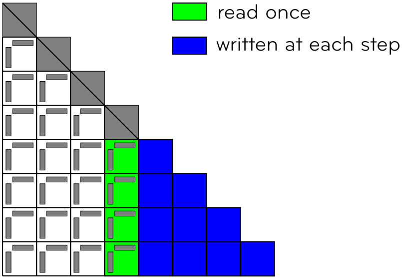
RL factorization



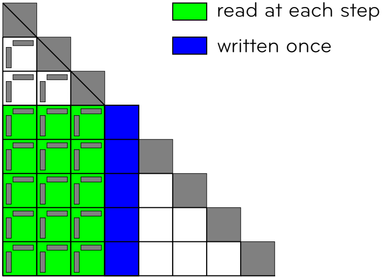
LL factorization

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 factorization

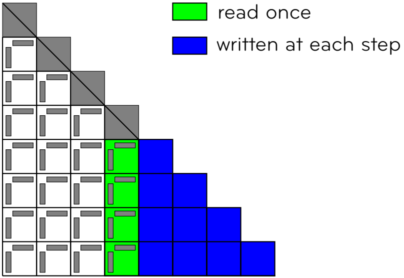


LL factorization

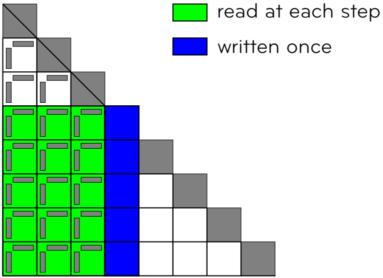
⇒ Lower volume of memory transfers in LL (more critical in MT)

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



RL factorization

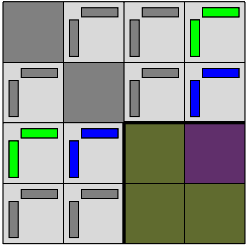


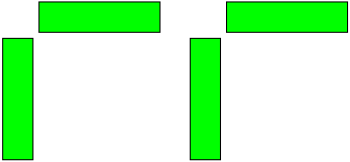
LL factorization

⇒ 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

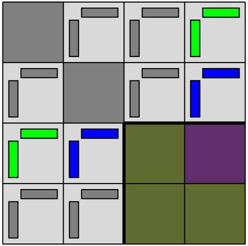
LUAR variant: accumulation and recompression

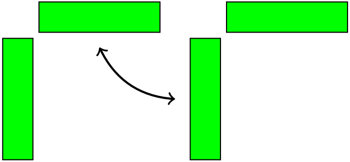




- FSCU (Factor, Solve, Compress, Update)

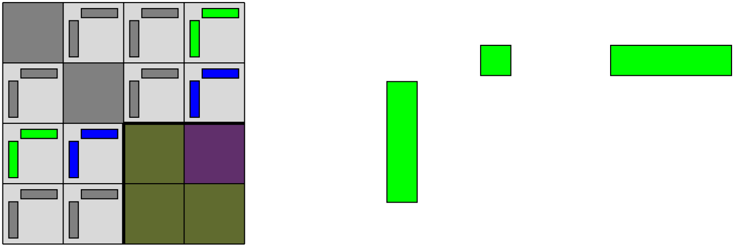
		FSCU
flops ($\times 10^{12}$)	Outer Product	3.8
	Total	10.2
time (s)	Outer Product	21
	Total	175





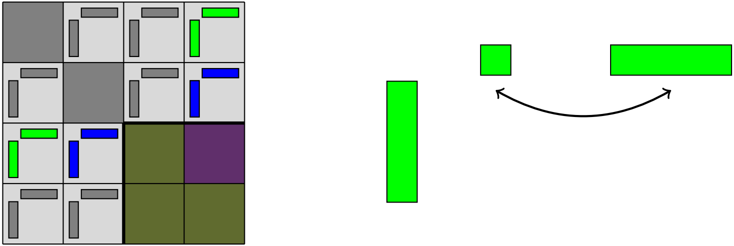
- FSCU (Factor, Solve, Compress, Update)

		FSCU
flops ($\times 10^{12}$)	Outer Product	3.8
	Total	10.2
time (s)	Outer Product	21
	Total	175



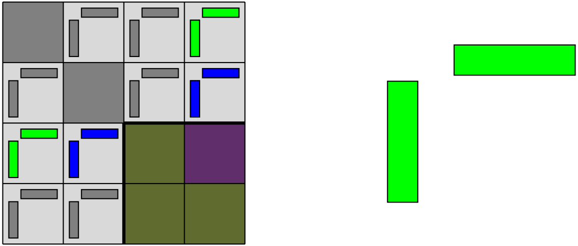
- FSCU (Factor, Solve, Compress, Update)

		FSCU
flops ($\times 10^{12}$)	Outer Product	3.8
	Total	10.2
time (s)	Outer Product	21
	Total	175



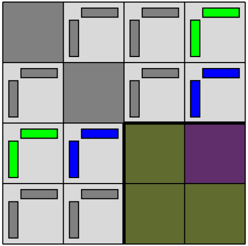
- FSCU (Factor, Solve, Compress, Update)

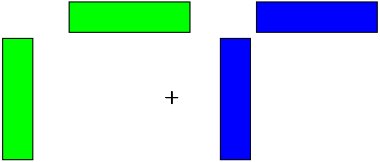
		FSCU
flops ($\times 10^{12}$)	Outer Product	3.8
	Total	10.2
time (s)	Outer Product	21
	Total	175



- FSCU (Factor, Solve, Compress, Update)

		FSCU
flops ($\times 10^{12}$)	Outer Product	3.8
	Total	10.2
time (s)	Outer Product	21
	Total	175

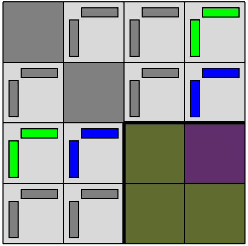


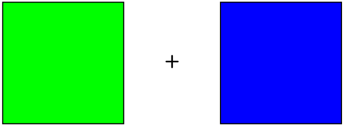


- FSCU (Factor, Solve, Compress, Update)

		FSCU
flops ($\times 10^{12}$)	Outer Product	3.8
	Total	10.2
time (s)	Outer Product	21
	Total	175

LUAR variant: accumulation and recompression

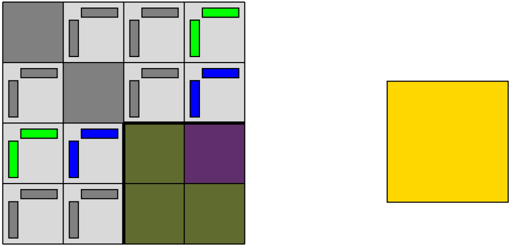




- FSCU (Factor, Solve, Compress, Update)

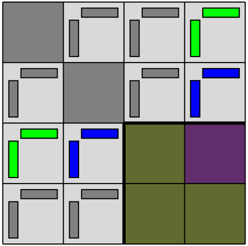
		FSCU
flops ($\times 10^{12}$)	Outer Product	3.8
	Total	10.2
time (s)	Outer Product	21
	Total	175

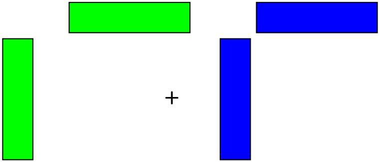
LUAR variant: accumulation and recompression



- FSCU (Factor, Solve, Compress, Update)

		FSCU
flops ($\times 10^{12}$)	Outer Product	3.8
	Total	10.2
time (s)	Outer Product	21
	Total	175

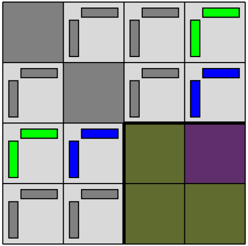


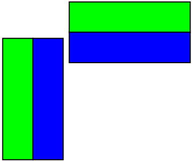


- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR

		FSCU
flops ($\times 10^{12}$)	Outer Product	3.8
	Total	10.2
time (s)	Outer Product	21
	Total	175

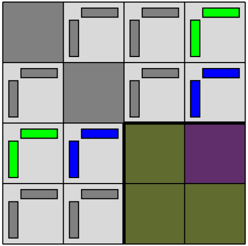
LUAR variant: accumulation and recompression

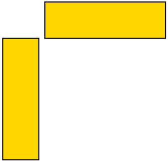




- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations

		FSCU	+LUA
flops ($\times 10^{12}$)	Outer Product	3.8	3.8
	Total	10.2	10.2
time (s)	Outer Product	21	14
	Total	175	167

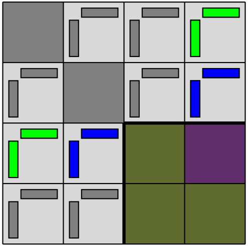


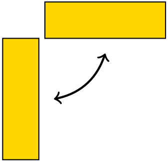


- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential recompression

		FSCU	+LUA
flops ($\times 10^{12}$)	Outer Product	3.8	3.8
	Total	10.2	10.2
time (s)	Outer Product	21	14
	Total	175	167

LUAR variant: accumulation and recompression

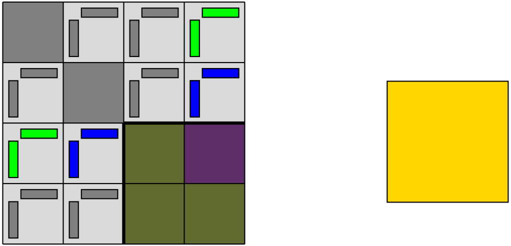




- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential recompression

		FSCU	+LUA	+LUAR
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

LUAR variant: accumulation and recompression

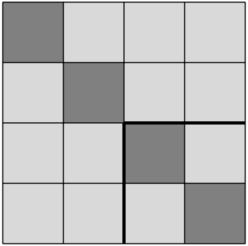


- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential recompression

		FSCU	+LUA	+LUAR
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

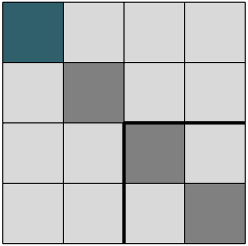
$\Rightarrow$ **2.4** gain becomes **2.6**

FCSU variant: compress before solve



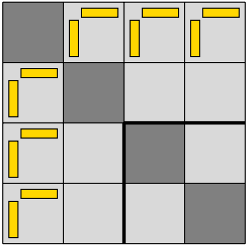
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
- FCSU(+LUAR)

FCSU variant: compress before solve



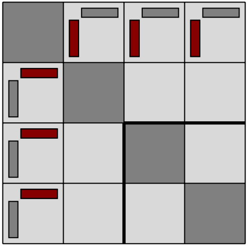
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
- FCSU(+LUAR)
  - Restricted pivoting

FCSU variant: compress before solve



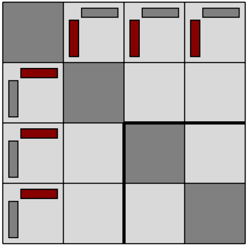
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
- FCSU(+LUAR)
  - Restricted pivoting

FCSU variant: compress before solve



- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
- FCSU(+LUAR)
  - Restricted pivoting
  - Low-rank Solve ⇒ flop reduction

FCSU variant: compress before solve

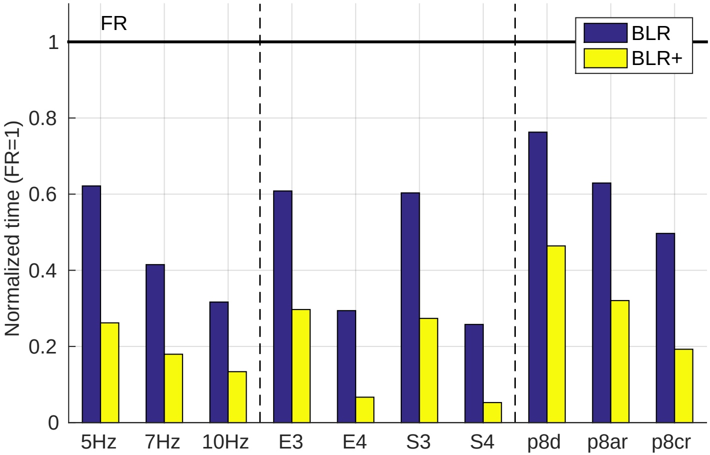


- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
- FCSU(+LUAR)
  - Restricted pivoting
  - Low-rank Solve ⇒ flop reduction

2.6 gain becomes **3.7**

	flops (TF)	time (s)	residual
FSCU	8.1	160	1.5e-09
FCSU	4.0	111	2.7e-09

Multicore performance results (24 threads)



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

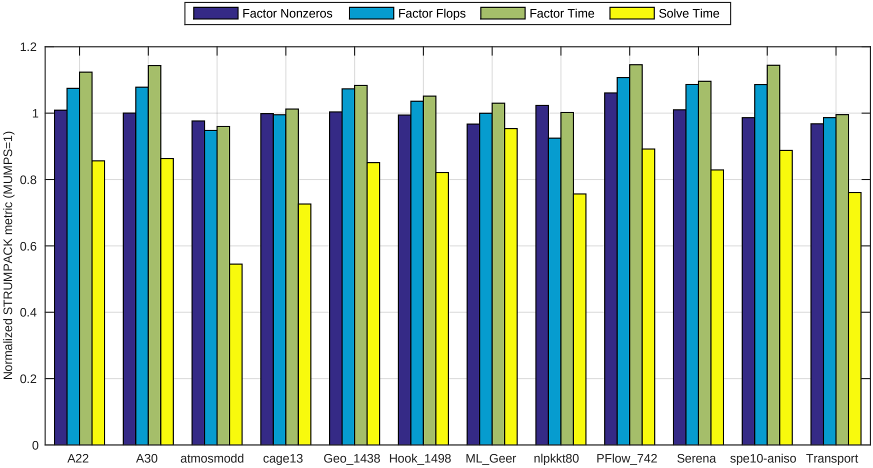
Amestoy, Buttari, L'Excellent, and Mary, *Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures*, ACM Trans. Math. Soft. (2018).

Comparison with HSS Matrices

Experimental Setting

- Experiments are done on the cori supercomputer of NERSC
- We compare
  - the MUMPS solver based on BLR
  - the STRUMPACK solver (LBNL) based on HSS
- Test problems come from several real-life applications: Seismic (5Hz), Electromagnetism (S3), Structural (perf008d, Geo_1438, Hook_1498, ML_Geer, Serena, Transport), CFD (atmosmodd, PFlow_742), MHD (A22, A30), Optimization (nlpkkt80), and Graph (cage13)
- We test 7 tolerance values (from 9e-1 to 1e-6) and FR, and compare the time for factorization + solve with:
  - 1 step of iterative refinement in FR
  - GMRES iterative solver in LR with required accuracy of 10^{-6} and restart of 30

Full-Rank solvers comparison

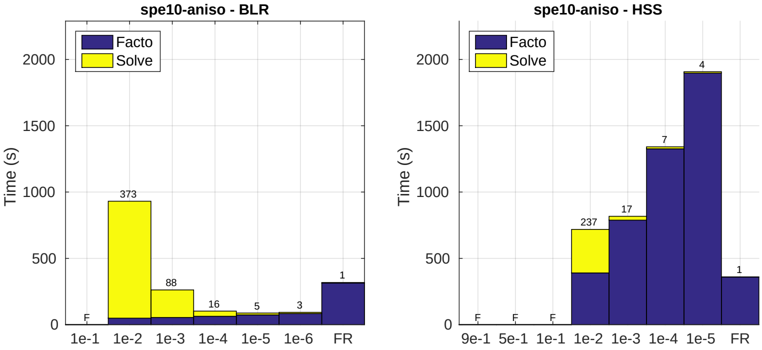


$\Rightarrow$ very similar FR performance

Preconditioner vs direct solver mode

Optimal tolerance choice

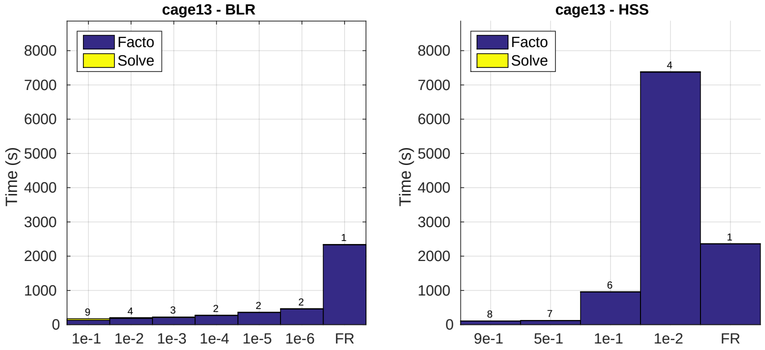
	BLR	HSS
A22	1e-5	FR
A30	1e-4	FR
atmosmodd	1e-4	9e-1
cage13	1e-1	9e-1
Geo_1438	1e-4	FR
Hook_1498	1e-5	FR
ML_Geer	1e-6	FR
nlpkkt80	1e-5	5e-1
PFlow_742	1e-6	FR
Serena	1e-4	1e-1
spe10-aniso	1e-5	FR
Transport	1e-5	FR



spe10-aniso matrix

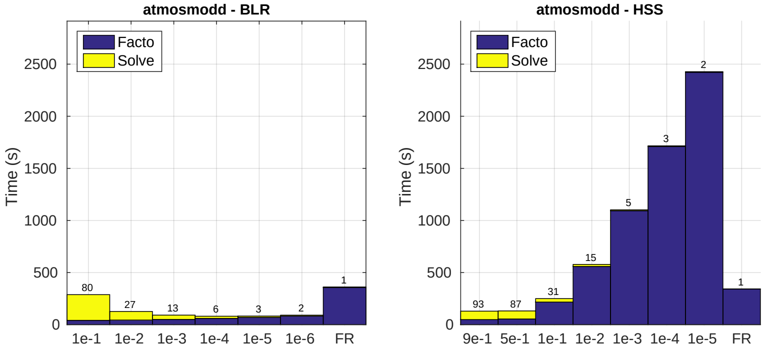
- No convergence except for low tolerances ⇒ direct solver mode is needed
- BLR is better suited as HSS rank is too high

When preconditioning works well...



cage13 matrix

- Fast convergence even for high tolerance ⇒ preconditioner mode is better suited
- As the size grows, HSS will gain the upper hand



atmosmodd matrix

- Find compromise between accuracy and compression
- In general, BLR favors direct solver while HSS favors preconditioner mode

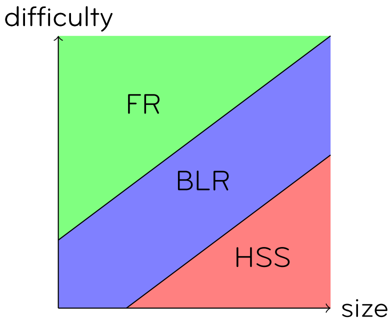
⇒ Performance comparison will depend on numerical difficulty and size of the problem

Preconditioner vs direct solver mode

Optimal tolerance choice

	BLR	HSS
A22	1e-5	FR
A30	1e-4	FR
atmosmodd	1e-4	9e-1
cage13	1e-1	9e-1
Geo_1438	1e-4	FR
Hook_1498	1e-5	FR
ML_Geer	1e-6	FR
nlpkkt80	1e-5	5e-1
PFlow_742	1e-6	FR
Serena	1e-4	1e-1
spe10-aniso	1e-5	FR
Transport	1e-5	FR

These results seem to suggest the following trend:



Ongoing work on BLR preconditioners

N. J. Higham and T. Mary, *A New Preconditioner that Exploits Low-Rank Approximations to Factorization Error*, MIMS EPrint 2018.10.

BLR threshold $= 10^{-2}$, iterate until converged to accuracy 10^{-9}

Recent work with N. Higham to

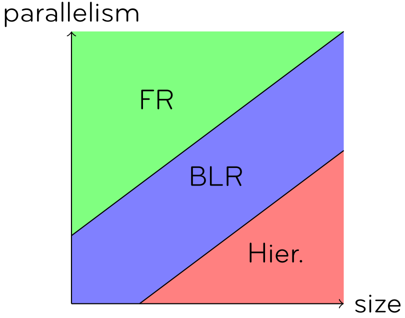
improve factorization-based preconditioners

Matrix	n	Standard		Improved	
		Iter.	Time	Iter.	Time
audikw_1	1.0M	691	1163	331	625
Bump_2911	2.9M	–	–	284	1708
Emilia_923	0.9M	174	304	136	267
Fault_639	0.6M	–	–	294	345
Ga41As41H72	0.3M	–	–	135	143
Hook_1498	1.5M	417	902	356	808
Si87H76	0.2M	–	–	131	116

Good potential to improve low-precision, low-memory BLR solvers

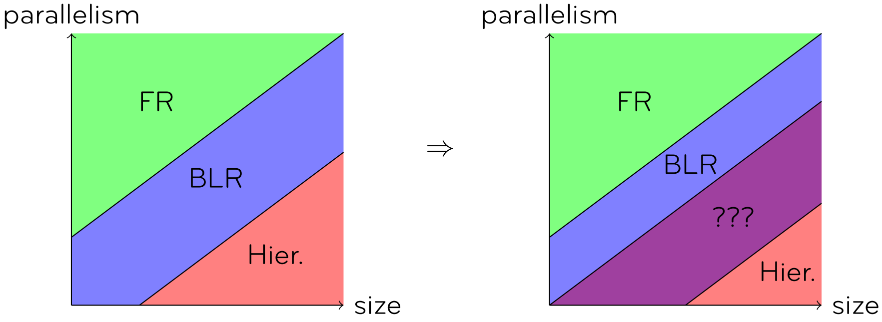
The MBLR Format

Compromise between complexity and parallelism



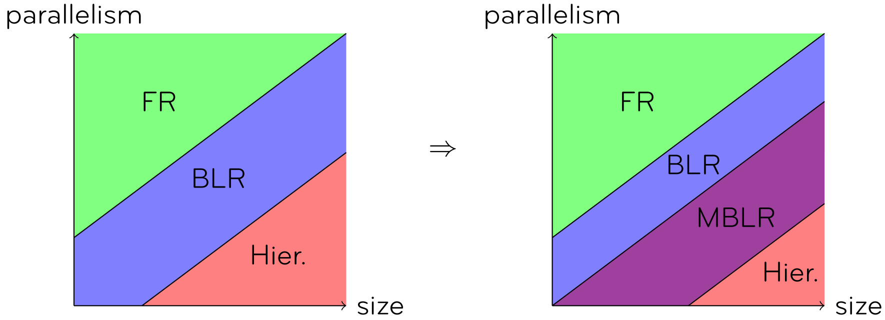
BLR is a compromise between complexity and performance

Compromise between complexity and parallelism



BLR is a compromise between complexity and performance

Can we find an even better compromise?



BLR is a compromise between complexity and performance

Can we find an even better compromise?

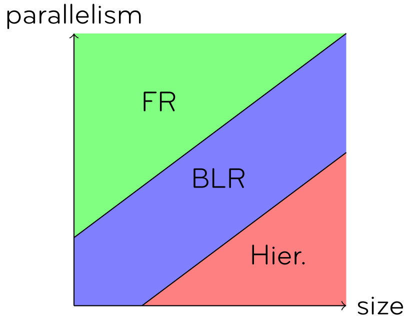
Multilevel BLR (MBLR)

Fixed number of levels ℓ

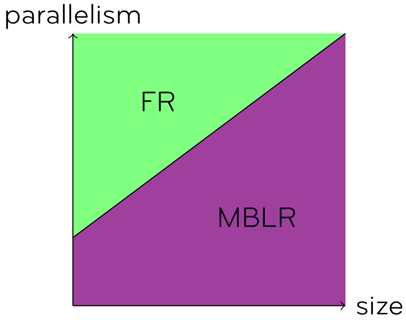
BLR ($\ell = 1$)	$\ell = 2$	$\ell = 3$	$\cdots$	Hier. ($\ell = \infty$)

parallelism ⟷ complexity

Compromise between complexity and parallelism



⇒



BLR is a compromise between complexity and performance

Can we find an even better compromise?

Multilevel BLR (MBLR): one format to englobe them all?

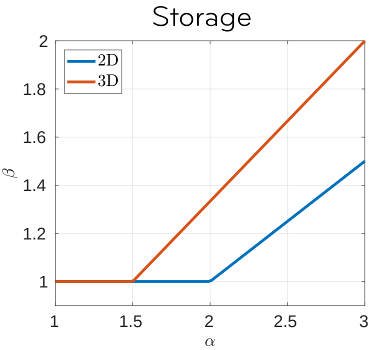
Fixed number of levels ℓ

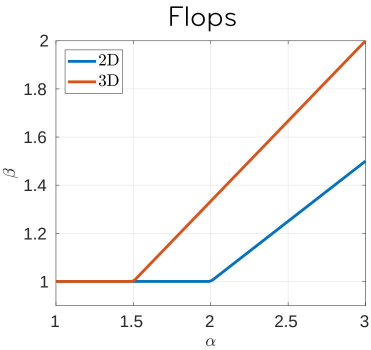
BLR ($\ell = 1$)	$\ell = 2$	$\ell = 3$	$\cdots$	Hier. ($\ell = \infty$)

parallelism ⟷ complexity

Bridging the gap between flat and hierarchical formats

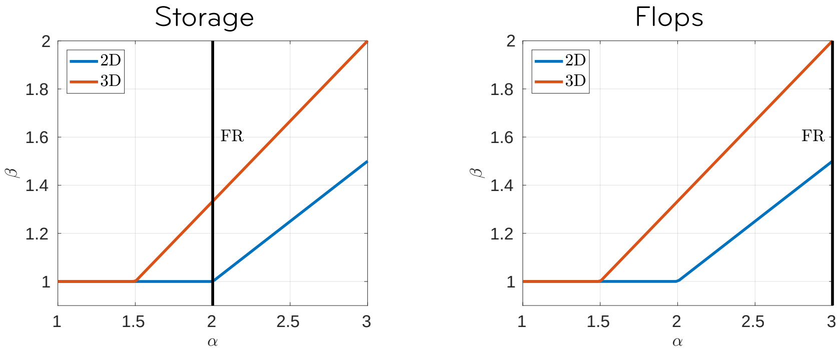
$$\mathcal{C}_{dense} = O(m^{\alpha}) \Rightarrow \mathcal{C}_{sparse} = O(n^{\beta})$$





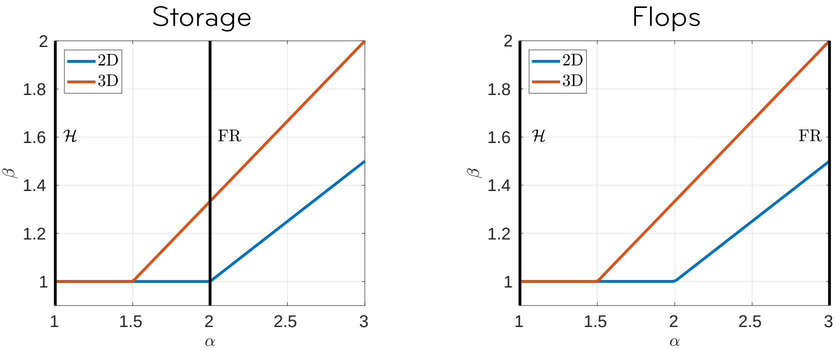
Bridging the gap between flat and hierarchical formats

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



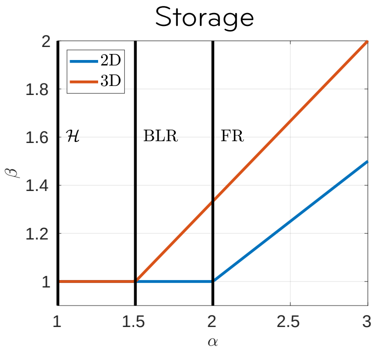
Bridging the gap between flat and hierarchical formats

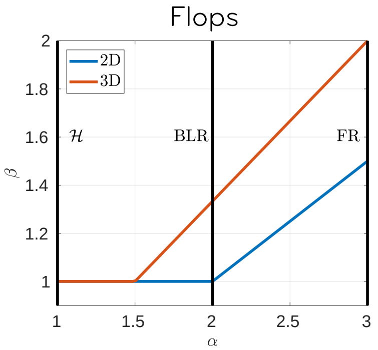
$$\mathcal{C}_{dense} = O(m^{\alpha}) \Rightarrow \mathcal{C}_{sparse} = O(n^{\beta})$$



Bridging the gap between flat and hierarchical formats

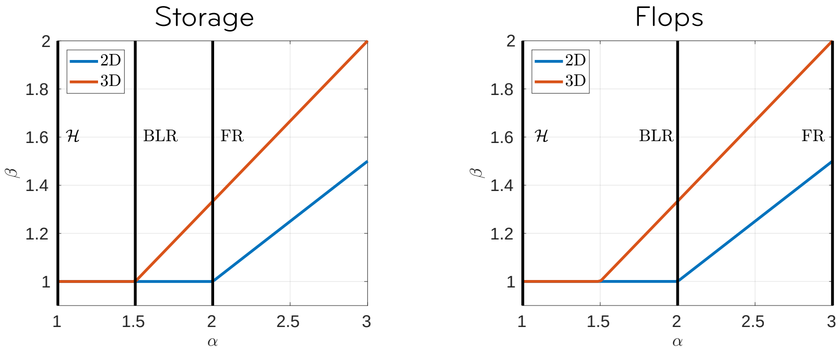
$$\mathcal{C}_{dense} = O(m^{\alpha}) \Rightarrow \mathcal{C}_{sparse} = O(n^{\beta})$$





Bridging the gap between flat and hierarchical formats

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



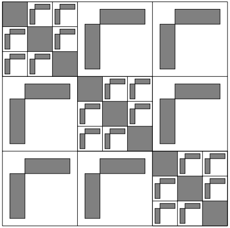
Key motivation: $\mathcal{C}_{dense} < O(m^2)$ (2D) or $O(m^{3/2})$ (3D) is enough to get $O(n)$ sparse complexity!

- 2D flop and 3D storage complexity: just a little improvement needed
- 3D flop complexity: still a large gap between BLR and $\mathcal{H}$

We propose a multilevel BLR format to bridge the gap

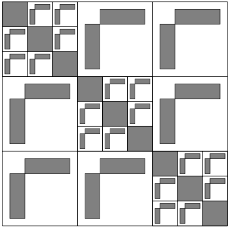
Complexity of the two-level BLR format

Assume all off-diagonal blocks are low-rank. Then:



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

Assume all off-diagonal blocks are low-rank. Then:



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

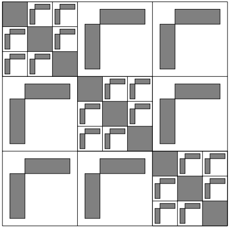
Similarly, we can prove:

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

Result holds if a **constant** number of off-diag. blocks is BLR.

Complexity of the two-level BLR format

Assume all off-diagonal blocks are low-rank. Then:



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

Similarly, we can prove:

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

Result holds if a **constant** number of off-diag. blocks is BLR.

		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

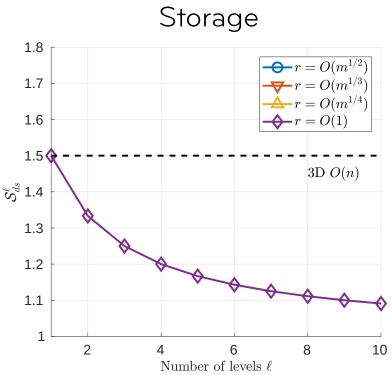
For $b = m^{\ell/(\ell+1)} r^{1/(\ell+1)}$, the ℓ−level complexities are:

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

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

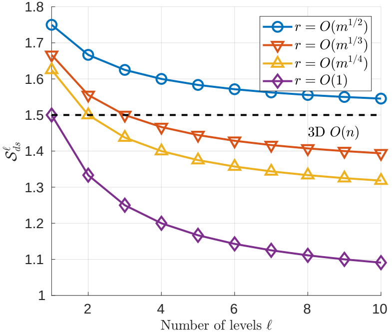
Amestoy, Buttari, L'Excellent, and Mary, *Bridging the gap between flat and hierarchical low-rank matrix formats: the multilevel BLR format*, submitted (2018).

- Simple way to finely control the desired complexity
- Block size $b \propto O(m^{\ell/(\ell+1)}) \ll O(m)$
  $\Rightarrow$ may be efficiently processed in shared-memory
- Number of blocks per row/column $\propto O(m^{1/(\ell+1)}) \gg O(1)$
  $\Rightarrow$ flexibility to distribute data in parallel

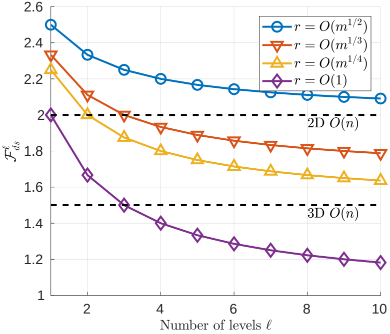


- If $r = O(1)$, can achieve $O(n)$ storage complexity with only two levels and $O(n \log n)$ flop complexity with three levels

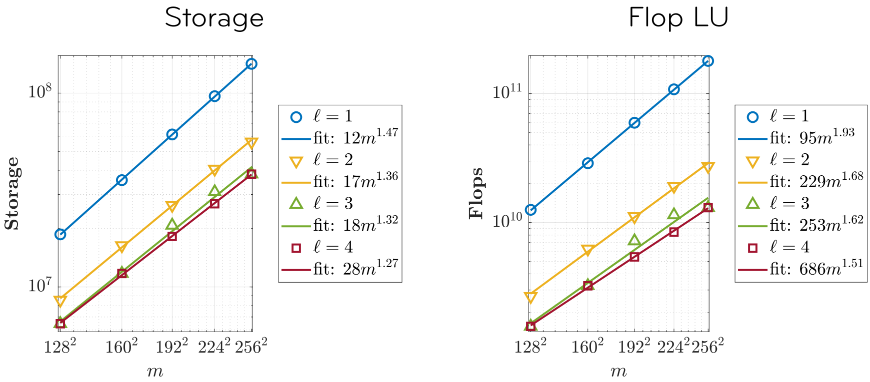
Storage



Flop LU



- 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, optimal sparse complexity is not attainable with constant ℓ **but** improvement rate is rapidly decreasing: **the first few levels achieve most of the asymptotic gain**



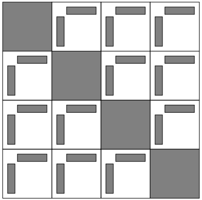
- Experimental complexity in relatively good agreement with theoretical one
- Asymptotic gain decreases with levels

Error analysis

BLR builds an approximate factorization $\mathbf{A}_\varepsilon = \mathbf{L}_\varepsilon \mathbf{U}_\varepsilon$

The BLR threshold ε is controlled by the user

BUT the user does not know how to choose ε!



Each off-diagonal block B is approximated by a low-rank matrix $\widetilde{B}$ such that $\|B - \widetilde{B}\| \le \varepsilon$

$\|A - L_\varepsilon U_\varepsilon\| \ne \varepsilon$ because of error propagation

$\Rightarrow$ **What is the overall accuracy $\|A - L_\varepsilon U_\varepsilon\|$?**

- Can we prove that $\|A - L_\varepsilon U_\varepsilon\| = O(\varepsilon)$?
- What is the error growth, i.e., how does the error depend on the matrix size m?
- How do the different variants (FCSU, LUAR, etc.) compare?
- Should we use an absolute threshold ($\|B - \widetilde{B}\| \le \varepsilon$) or a relative one ($\|B - \widetilde{B}\| \le \varepsilon \|B\|$)?

Main result

Theorem

The FSCU factorization of a matrix of order m with block size b and absolute threshold ε produces an error equal to

$$\|A - L_\varepsilon U_\varepsilon\| = \sqrt{\frac{m}{b}}\varepsilon\|L\|\|U\| + O(u\varepsilon).$$

- $\|L\|\|U\| \le \rho_m\|A\|$ where ρ_m is the growth factor; with partial pivoting, ρ_m is typically small $\Rightarrow$ BLR factorization is stable!
- Error growth behaves as $\sqrt{m/b} = O(m^{1/4}) \Rightarrow$ very slow growth!
- Factorization variants only change the $O(u\varepsilon)$ term $\Rightarrow$ no significant difference!
- $\sqrt{m/b}$ term can be dropped using relative threshold, but compression rate is also lower

Experimental results

matrix	$\varepsilon = 10^{-4}$		$\varepsilon = 10^{-8}$		$\varepsilon = 10^{-12}$	
	error	bound	error	bound	error	bound
pwtk	$7.7e{-}05$	$3.4e{-}04$	$7.3e{-}09$	$3.4e{-}08$	$5.1e{-}13$	$3.4e{-}12$
cfd2	$2.3e{-}04$	$2.7e{-}04$	$2.3e{-}08$	$2.7e{-}08$	$1.9e{-}12$	$2.7e{-}12$
2cubes_sphere	$9.3e{-}05$	$1.3e{-}04$	$9.9e{-}09$	$1.3e{-}08$	$1.2e{-}12$	$1.3e{-}12$
af_shell3	$1.4e{-}04$	$2.0e{-}04$	$1.7e{-}08$	$2.0e{-}08$	$1.7e{-}12$	$2.0e{-}12$
audikw_1	$2.8e{-}04$	$4.3e{-}04$	$1.6e{-}08$	$4.3e{-}08$	$1.2e{-}12$	$4.3e{-}12$
cfd2	$2.3e{-}04$	$2.7e{-}04$	$2.3e{-}08$	$2.7e{-}08$	$1.9e{-}12$	$2.7e{-}12$
Dubcova3	$2.0e{-}04$	$1.5e{-}04$	$2.3e{-}08$	$1.5e{-}08$	$2.4e{-}12$	$1.5e{-}12$
Fault_639	$1.6e{-}05$	$2.4e{-}03$	$3.3e{-}09$	$2.4e{-}07$	$6.6e{-}13$	$2.4e{-}11$
hood	$1.6e{-}05$	$8.5e{-}04$	$1.7e{-}09$	$8.5e{-}08$	$1.6e{-}13$	$8.5e{-}12$
nasasrb	$8.7e{-}05$	$5.3e{-}04$	$5.4e{-}09$	$5.3e{-}08$	$5.7e{-}13$	$5.3e{-}12$
nd24k	$1.1e{-}04$	$6.8e{-}04$	$1.5e{-}08$	$6.8e{-}08$	$1.1e{-}12$	$6.8e{-}12$
oilpan	$5.7e{-}06$	$2.8e{-}03$	$1.2e{-}09$	$2.8e{-}07$	$5.3e{-}14$	$2.8e{-}11$
pwtk	$7.7e{-}05$	$3.4e{-}04$	$7.3e{-}09$	$3.4e{-}08$	$5.1e{-}13$	$3.4e{-}12$
shallow_water1	$9.3e{-}07$	$1.1e{-}04$	$3.4e{-}09$	$1.1e{-}08$	$6.2e{-}14$	$1.1e{-}12$
ship_003	$5.4e{-}05$	$3.2e{-}04$	$6.0e{-}09$	$3.2e{-}08$	$6.1e{-}13$	$3.2e{-}12$
thermomech_dM	$5.5e{-}06$	$1.1e{-}04$	$1.6e{-}09$	$1.1e{-}08$	$3.7e{-}14$	$1.1e{-}12$
x104	$2.0e{-}05$	$1.1e{-}03$	$2.6e{-}09$	$1.1e{-}07$	$2.1e{-}13$	$1.1e{-}11$

Measured error matches bound

Open questions

- Choice of scaling strategy
- Error analysis of BLR solution phase and its use in conjunction of iterative refinement
- Pivoting strategies for the BLR factorization
- Error analysis of multilevel BLR factorization
- Probabilistic error analysis: in the standard LU case, the deterministic bound

  $$|A - LU| \le \gamma_n |L||U| = O(nu)|L||U|$$

  is known to be pessimistic. In recent work, we have shown that

  $$|A - LU| \le \widetilde{\gamma}_n |L||U| = O(\sqrt{n}u)|L||U|$$

  holds with high probability assuming rounding errors are random. Can we apply this to BLR factorizations?

Fast BLR Matrix Arithmetic

Context and objective

- Standard $O(m^3)$ matrix multiplication algorithm is not optimal: $O(m^\omega)$ can be achieved, with $2 \le \omega \le \omega_0 = \log_2 7 \approx 2.81$.

Context and objective

- Standard $O(m^3)$ matrix multiplication algorithm is not optimal: $O(m^\omega)$ can be achieved, with $2 \le \omega \le \omega_0 = \log_2 7 \approx 2.81$.
- Reminder: given a $O(m^\omega)$ matrix multiplication algorithm, the LU factorization has the same complexity

Context and objective

- Standard $O(m^3)$ matrix multiplication algorithm is not optimal: $O(m^\omega)$ can be achieved, with $2 \leq \omega \leq \omega_0 = \log_2 7 \approx 2.81$.
- Reminder: given a $O(m^\omega)$ matrix multiplication algorithm, the LU factorization has the same complexity
- Example: Strassen's algorithm achieves $O(m^{\omega_0})$ complexity

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\
M_2 &= (A_{21} + A_{22})B_{11}, \\
M_3 &= A_{11}(B_{12} - B_{22}), \\
M_4 &= A_{22}(B_{21} - B_{11}), \\
M_5 &= (A_{11} + A_{12})B_{22}, \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}),
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7, \\
C_{12} &= M_3 + M_5, \\
C_{21} &= M_2 + M_4, \\
C_{22} &= M_1 - M_2 + M_3 + M_6.
\end{aligned}$$

Context and objective

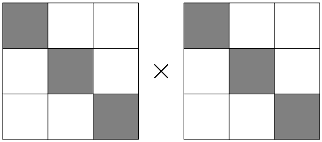
- Standard $O(m^3)$ matrix multiplication algorithm is not optimal: $O(m^\omega)$ can be achieved, with $2 \le \omega \le \omega_0 = \log_2 7 \approx 2.81$.
- Reminder: given a $O(m^\omega)$ matrix multiplication algorithm, the LU factorization has the same complexity
- Example: Strassen's algorithm achieves $O(m^{\omega_0})$ complexity

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\ M_2 &= (A_{21} + A_{22})B_{11}, \\ M_3 &= A_{11}(B_{12} - B_{22}), \\ M_4 &= A_{22}(B_{21} - B_{11}), \\ M_5 &= (A_{11} + A_{12})B_{22}, \\ M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\ M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}), \end{aligned} \quad \Rightarrow \quad \begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7, \\ C_{12} &= M_3 + M_5, \\ C_{21} &= M_2 + M_4, \\ C_{22} &= M_1 - M_2 + M_3 + M_6. \end{aligned}$$

- **Question: can we use fast matrix arithmetic to improve the $O(m^2 r)$ BLR complexity?**

- We model a BLR matrix A as $A = S_A + E_A$, where S_A consists of the FR blocks and E_A of the LR ones
- Then, $AB = (S_A + E_A)(S_B + E_B) = S_A S_B + S_A E_B + S_B E_A + E_A E_B$
- $S_A S_B$ product: $O(p)$ FR-FR products

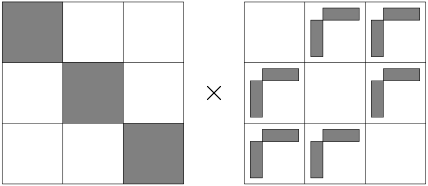


$O(pb^3) \rightarrow O(pb^\omega) \Rightarrow$ good enough

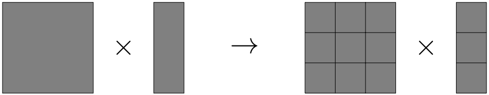
- Not so straighforward for the other three products!

First naive approach

$S_A E_B$ product: $O(p^2)$ FR-LR products



Problem: fast matrix multiplication works on square matrices



$$O(p^2) \times O(b^2 r) \to O(p^2) \times O(b^2 r^{\omega-1}) = O(m^2 r^{\omega-1})$$

$\Rightarrow$ **no asymptotic reduction in m, only in r**

Since $m \gg r$, this is not a satisfying result $\Rightarrow$ can we do better?

Second approach based on accumulation

$$O(p^2) \times O(b^2 r) \rightarrow O(p) \times O(b^2 pr)$$

$$\rightarrow O(p) \times O(\max((pr)^{\omega-2} b^2, prb^{\omega-1}))$$

$\Rightarrow$ find new optimal b that equilibrates cost$(S_A E_B)$ and cost$(E_A E_B)$

Theorem

With this approach, the complexity of the BLR factorization becomes

$$O(m^{(3\omega-1)/(\omega+1)} r^{(\omega-1)^2/(\omega+1)}).$$

$\Rightarrow \approx O(m^{1.95} r^{0.86})$ for $\omega = \omega_0$ and $O(m^{5/3} r^{1/3})$ for $\omega = 2$

$\Rightarrow$ **asymptotic gain in m... but still not optimal**

(lower bound is given by size$(A) = O(m^{3/2} r^{1/2})$)

Third approach based on Strassen's algorithm

- Key idea: use Strassen's algorithm on the entire BLR matrix

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\ &\vdots \\ M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}), \end{aligned} \quad \Rightarrow \quad \begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7, \\ &\vdots \\ C_{22} &= M_1 - M_2 + M_3 + M_6. \end{aligned}$$

$\Rightarrow$ Requires the stronger assumption that each M_i is BLR

Theorem

With this approach, the complexity of the BLR factorization becomes

$$O(m^{(\omega\omega_0-1)/(\omega+\omega_0-2)} r^{(\omega-1)^2/(\omega+\omega_0-2)}).$$

$\Rightarrow \approx O(m^{1.90} r^{0.90})$ for $\omega = \omega_0$ and $\approx O(m^{1.64} r^{0.36})$ for $\omega = 2$

Can we generalize this result to algorithms other than Strassen's?

Replacing ω_0 by $\omega \rightarrow O(m^{(\omega+1)/2} r^{(\omega-1)/2})$ achieves lower bound for $\omega = 2$

Conclusion

Summary

Main results

- BLR dense factorization achieves $O(m^2 r)$ complexity
- We must rethink our algorithms to convert this theoretical reduction into actual time gains
- Good compromise between complexity and performance compared to hierarchical formats

Recent advances

- Multilevel extension can achieve an even better compromise
- Error analysis provides both theoretical guarantees and new insights
- Ongoing work on fast BLR matrix arithmetic

Slides and papers available here

http://personalpages.manchester.ac.uk/staff/theo.mary/