Performance and scalability of the Block Low-Rank multifrontal factorization

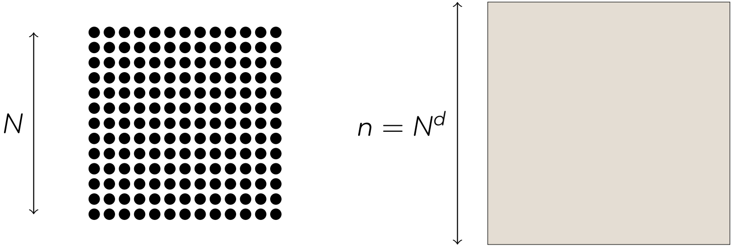
P. Amestoy*,[1] A. Buttari*,[2] J.-Y. L'Excellent†,[3] T. Mary*,[4]

*Université de Toulouse †ENS Lyon
[1] INPT-IRIT [2] CNRS-IRIT [3] INRIA-LIP [4] UPS-IRIT

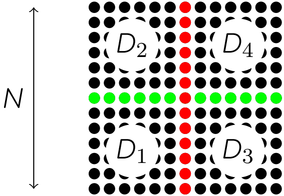
PhD Days '16, Toulouse Sept. 27

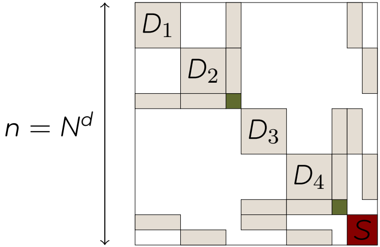
Introduction

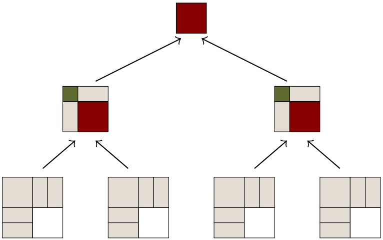
Multifrontal (Duff '83) with Nested Dissection (George '73)



Multifrontal (Duff '83) with Nested Dissection (George '73)



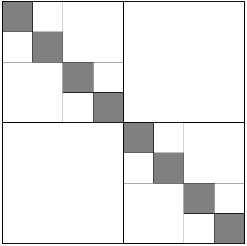




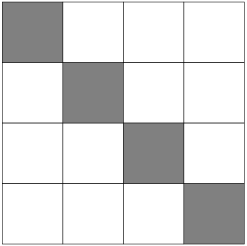
3D problem cost $\propto$

→ Flops:$O(n^2)$, mem:$O(n^{4/3})$

$\mathcal{H}$ and BLR matrices

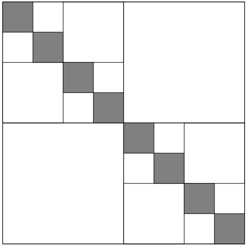


$\mathcal{H}$-matrix

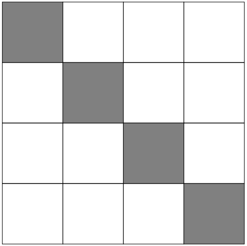


BLR matrix

$\mathcal{H}$ and BLR matrices



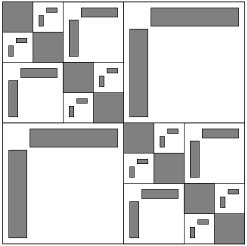
$\mathcal{H}$-matrix



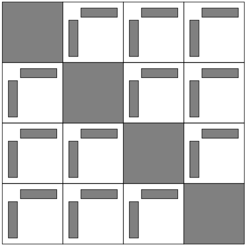
BLR matrix

A block B represents the interaction between two subdomains. If they have a small diameter and are far away their interaction is weak $\Rightarrow$ rank is low.

$\mathcal{H}$ and BLR matrices



$\mathcal{H}$-matrix



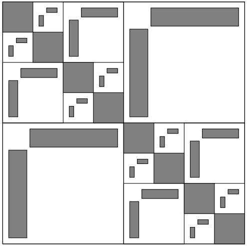
BLR matrix

A block B represents the interaction between two subdomains. If they have a small diameter and are far away their interaction is weak $\Rightarrow$ rank is low.

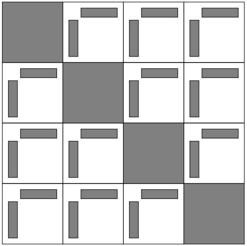
$$\tilde{B} = XY^T \text{ such that } \operatorname{rank}(\tilde{B}) = k_\varepsilon \text{ and } \|B - \tilde{B}\| \leq \varepsilon$$

If $k_\varepsilon \ll \text{size}(B) \Rightarrow$ memory and flops can be reduced with a controlled loss of accuracy ($\leq \varepsilon$)

$\mathcal{H}$ and BLR matrices



$\mathcal{H}$-matrix

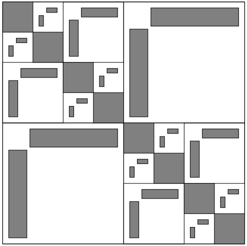


BLR matrix

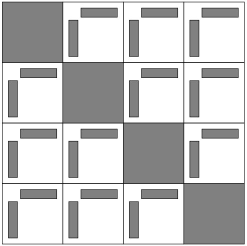
- Theoretical complexity can be as low as $O(n)$
- Complex, hierarchical structure

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

$\mathcal{H}$ and BLR matrices



$\mathcal{H}$-matrix



BLR matrix

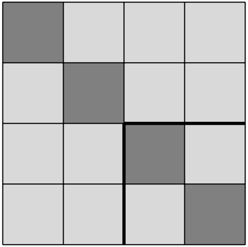
- Theoretical complexity can be as low as $O(n)$
- Complex, hierarchical structure

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

$\Rightarrow$ Our hope is to find a good comprise between theoretical complexity and performance/usability

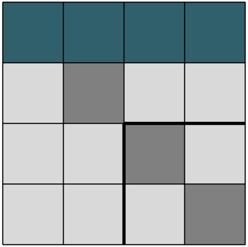
Variants of the BLR factorization

Variants of the BLR LU factorization



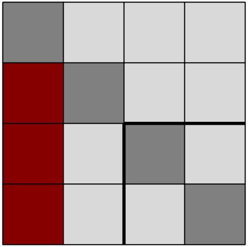
- FSCU

Variants of the BLR LU factorization



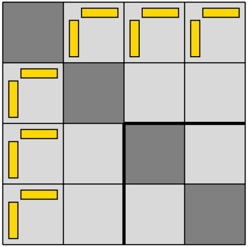
- FSCU (Factor,

Variants of the BLR LU factorization



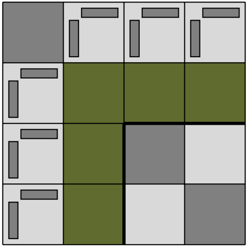
- FSCU (Factor, Solve,

Variants of the BLR LU factorization



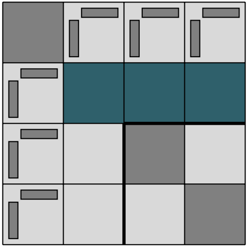
- FSCU (Factor, Solve, Compress,

Variants of the BLR LU factorization



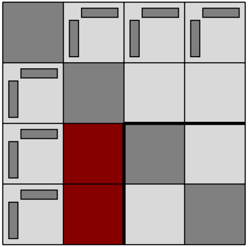
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



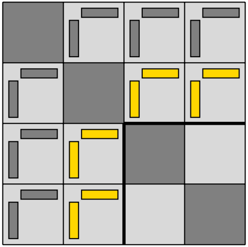
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



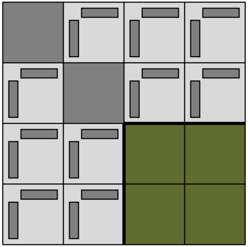
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



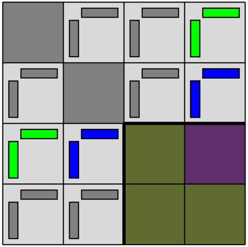
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



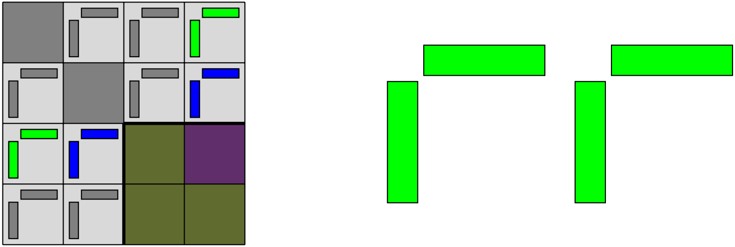
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



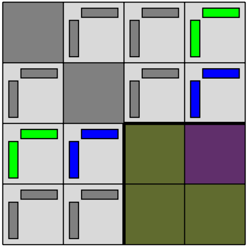
- FSCU (Factor, Solve, Compress, Update)

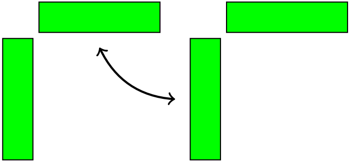
Variants of the BLR LU factorization



- FSCU (Factor, Solve, Compress, Update)

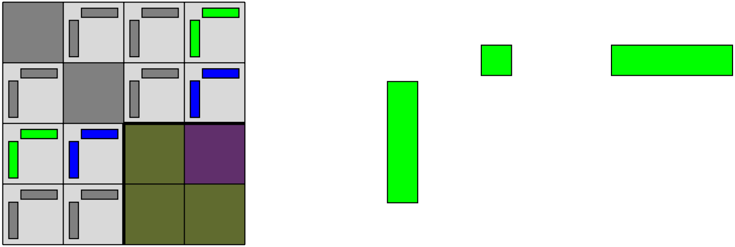
Variants of the BLR LU factorization





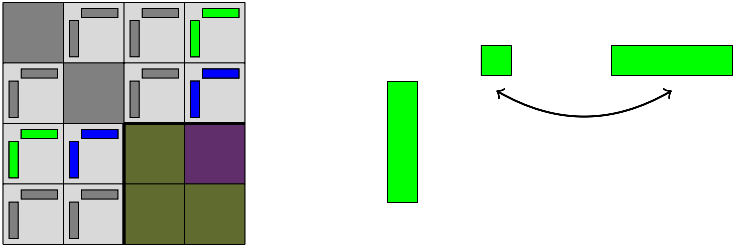
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



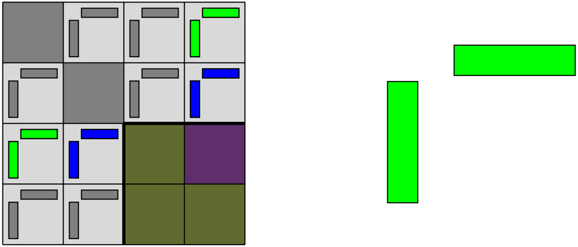
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



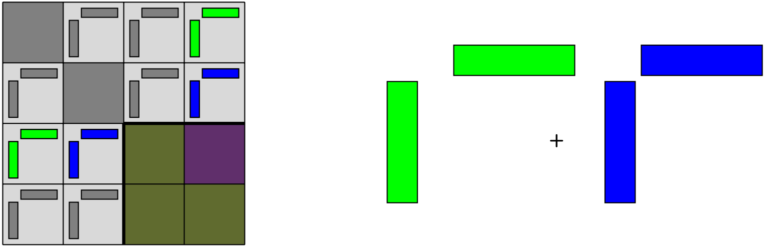
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



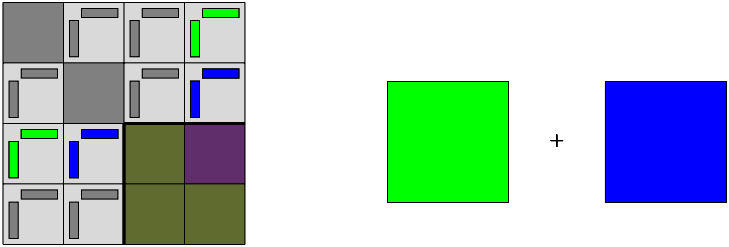
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



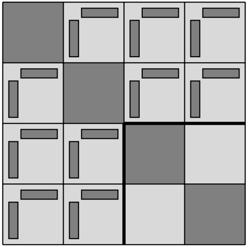
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



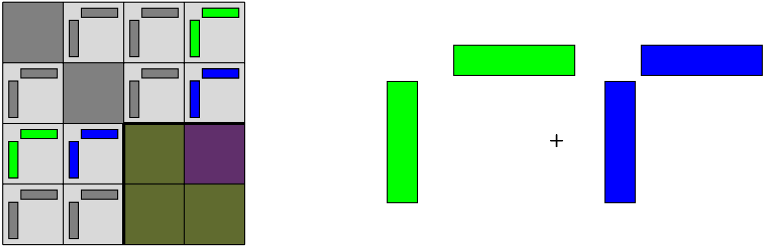
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



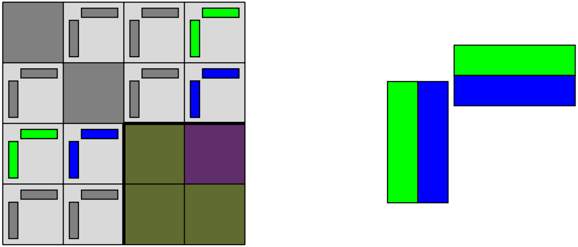
- FSCU (Factor, Solve, Compress, Update)

Variants of the BLR LU factorization



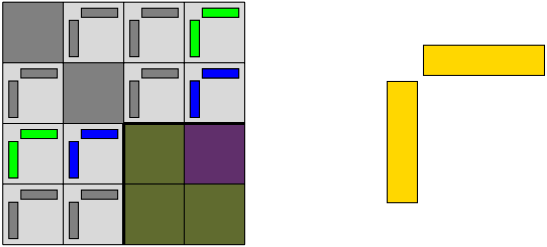
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR

Variants of the BLR LU factorization



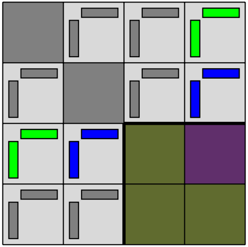
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
    - Better granularity in Update operations

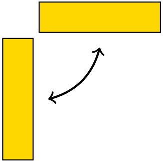
Variants of the BLR LU factorization



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

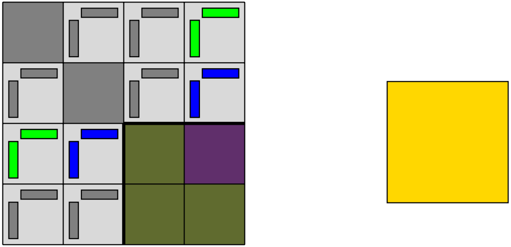
Variants of the BLR LU factorization





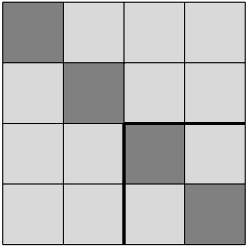
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential for recompression ⇒ flop reduction

Variants of the BLR LU factorization



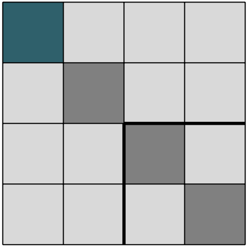
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential for recompression ⇒ flop reduction

Variants of the BLR LU factorization



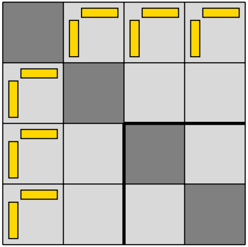
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential for recompression ⇒ flop reduction
- FCSU(+LUAR)

Variants of the BLR LU factorization



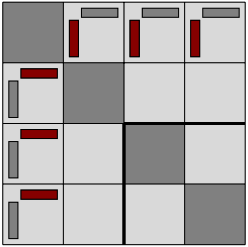
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential for recompression ⇒ flop reduction
- FCSU(+LUAR)
  - Restricted pivoting, e.g. to diagonal blocks

Variants of the BLR LU factorization



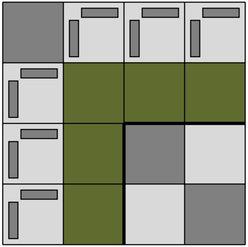
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential for recompression ⇒ flop reduction
- FCSU(+LUAR)
  - Restricted pivoting, e.g. to diagonal blocks

Variants of the BLR LU factorization



- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential for recompression ⇒ flop reduction
- FCSU(+LUAR)
  - Restricted pivoting, e.g. to diagonal blocks
  - Low-rank Solve ⇒ flop reduction
  - Better BLAS-3/BLAS-2 ratio in Solve operations

Variants of the BLR LU factorization



- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
  - Better granularity in Update operations
  - Potential for recompression ⇒ flop reduction
- FCSU(+LUAR)
  - Restricted pivoting, e.g. to diagonal blocks
  - Low-rank Solve ⇒ flop reduction
  - Better BLAS-3/BLAS-2 ratio in Solve operations

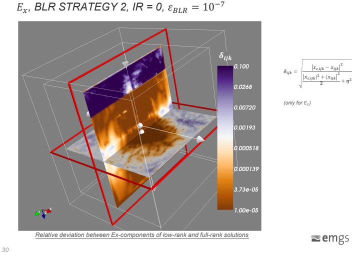
Experimental results

Experimental Setting: Machines

1. Distributed memory experiments are done on the `eos` supercomputer at the CALMIP center of Toulouse (grant 2014-P0989):
    - Two Intel(r) 10-cores Ivy Bridge @ 2,8 GHz
    - Peak per core is 22.4 GF/s
    - 64 GB memory per node
    - Infiniband FDR interconnect

2. Shared memory experiments are done on `grunch` at the LIP laboratory of Lyon:
    - Two Intel(r) 14-cores Haswell @ 2,3 GHz
    - Peak per core is 36.8 GF/s
    - Total memory is 768 GB

Experimental Setting: Matrices (1/3)



E_x, BLR STRATEGY 2, IR = 0, $\varepsilon_{BLR} = 10^{-7}$

$$\delta_{ijk} = \sqrt{\frac{|x_{\varepsilon,ijk} - x_{ijk}|^2}{\frac{|x_{\varepsilon,ijk}|^2 + |x_{ijk}|^2}{2} + \eta^2}}$$

(only for E_x)

Relative deviation between Ex-components of low-rank and full-rank solutions

emgs

30

3D Electromagnetic Modeling

Maxwell equation

Double complex (z) arithmetic

Symmetric LDL^T factorization

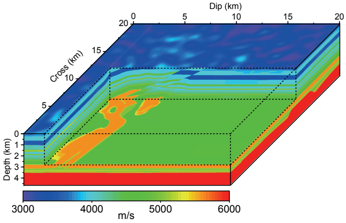
Required accuracy: $\varepsilon = 10^{-7}$

Credits: EMGS

matrix	n	nnz	flops	storage
S3	3.3M	43M	78 TF	189 GB
S4	21M	266M	2.5 PF	2.1 TB
D4	30M	384M	3.6 PF	3.0 TB

Full-Rank statistics

Experimental Setting: Matrices (2/3)



3D Seismic Modeling

Helmholtz equation

Single complex (c) arithmetic

Unsymmetric *LU* factorization

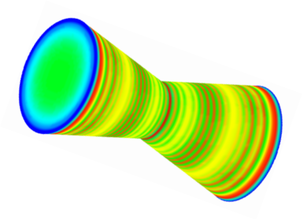
Required accuracy: $\varepsilon = 10^{-3}$

Credits: SEISCOPE

matrix	n	nnz	flops	storage
7Hz	7M	177M	410 TF	211 GB
10Hz	17M	446M	2600 TF	722 GB

Full-Rank statistics

Experimental Setting: Matrices (3/3)



3D Structural Mechanics

Double real (d) arithmetic

Symmetric LDL^T factorization

Required accuracy: $\varepsilon = 10^{-9}$

Credits: Code_Aster (EDF)

matrix	n	nnz	flops	storage
perf008ar	4M	159M	378 TF	148 GB

Full-Rank statistics

Performance on 900 cores

Low-rank threshold ε is set according to the application's target

matrix	MUMPS-(Full-Rank)			BLR	
	time	sp-up*	$\%_{peak}$	ε	time
10Hz	1017s	257	26%	10^{-3}	280s
S4	1538s	371	32%	10^{-7}	412s
D4	2221s	373	33%	10^{-7}	515s

*estimated speedup on 90×10 cores

- good speedup and $\%_{peak}$ on 900 cores ⇒ good FR reference
- BLR improves performance by a substantial factor of order 4

Performance on 900 cores

Low-rank threshold ε is set according to the application's target

matrix	MUMPS-(Full-Rank)			BLR	
	time	sp-up*	$\%_{peak}$	ε	time
10Hz	1017s	257	26%	10^{-3}	280s
S4	1538s	371	32%	10^{-7}	412s
D4	2221s	373	33%	10^{-7}	515s

*estimated speedup on 90×10 cores

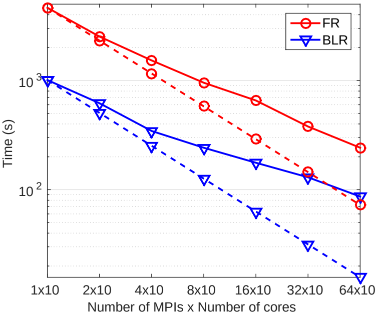
- good speedup and $\%_{peak}$ on 900 cores ⇒ good FR reference
- BLR improves performance by a substantial factor of order 4

⇒ *but does BLR scale as well as FR?*

Scalability of the BLR factorization (distributed)

MPI+OpenMP parallelism (10 threads/MPI process, 1 MPI/node)

7Hz matrix (extracted from MUMPS-SEISCOPE research work submitted to *Geophysics*)



- each time the number of processes doubles, speedup of ~ 1.6 for FR and ~ 1.5 for BLR

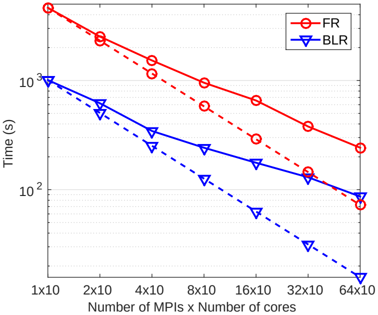
⇒ both FR and BLR scale reasonably well

⇒ ability to maintain gain due to BLR when the number of processes grows

Scalability of the BLR factorization (distributed)

MPI+OpenMP parallelism (10 threads/MPI process, 1 MPI/node)

7Hz matrix (extracted from MUMPS-SEISCOPE research work submitted to *Geophysics*)



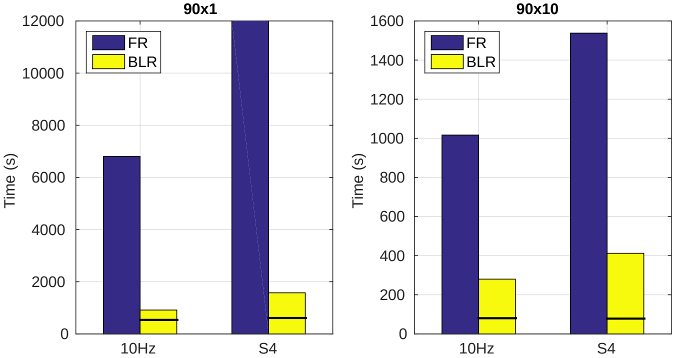
- each time the number of processes doubles, speedup of ~ 1.6 for FR and ~ 1.5 for BLR

⇒ both FR and BLR scale reasonably well

⇒ ability to maintain gain due to BLR when the number of processes grows

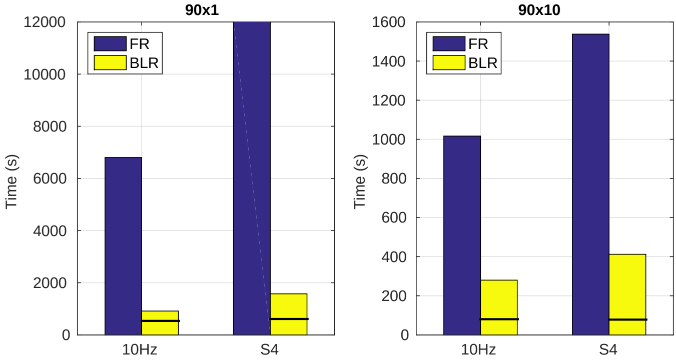
⇒ *so, we are happy?*

Gain due to BLR: impact of multithreading



- gain in flops (black line) does not fully translate into gain in time
- multithreaded efficiency lower in LR than in FR

Gain due to BLR: impact of multithreading



- gain in flops (black line) does not fully translate into gain in time
- multithreaded efficiency lower in LR than in FR

⇒ *improve efficiency of operations and multithreading with variants*

Right Looking Vs. Left-Looking (shared)

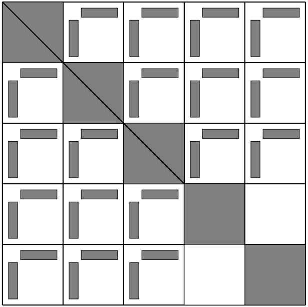
Focus on the Update step (which includes the Decompress)

		1 thread		28 threads	
		RL	LL	RL	LL
S3	FR			468s	526s
	BLR	847s	763s	112s	89s
perf008ar	FR			663s	766s
	BLR	2174s	2005s	236s	161s

Right Looking Vs. Left-Looking (shared)

Focus on the Update step (which includes the Decompress)

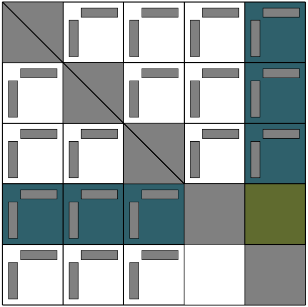
		1 thread		28 threads	
		RL	LL	RL	LL
S3	FR			468s	526s
	BLR	847s	763s	112s	89s
perf008ar	FR			663s	766s
	BLR	2174s	2005s	236s	161s



Right Looking Vs. Left-Looking (shared)

Focus on the Update step (which includes the Decompress)

		1 thread		28 threads	
		RL	LL	RL	LL
S3	FR			468s	526s
	BLR	847s	763s	112s	89s
perf008ar	FR			663s	766s
	BLR	2174s	2005s	236s	161s

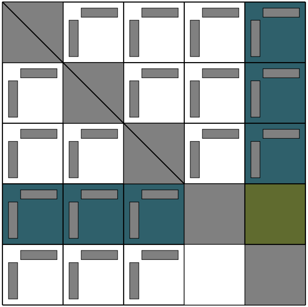


- in RL: FR (green) block is accessed many times; LR (blue) blocks are accessed once
- in LL: FR (green) block is accessed once; LR (blue) blocks are accessed many times

Right Looking Vs. Left-Looking (shared)

Focus on the Update step (which includes the Decompress)

		1 thread		28 threads	
		RL	LL	RL	LL
S3	FR			468s	526s
	BLR	847s	763s	112s	89s
perf008ar	FR			663s	766s
	BLR	2174s	2005s	236s	161s



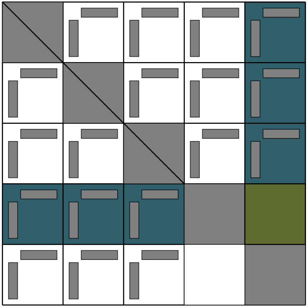
- in RL: FR (green) block is accessed many times; LR (blue) blocks are accessed once
- in LL: FR (green) block is accessed once; LR (blue) blocks are accessed many times

⇒ lower volume of memory transfers (more critical in multithreaded)

Right Looking Vs. Left-Looking (shared)

Focus on the Update step (which includes the Decompress)

		1 thread		28 threads	
		RL	LL	RL	LL
S3	FR			468s	526s
	BLR	847s	763s	112s	89s
perf008ar	FR			663s	766s
	BLR	2174s	2005s	236s	161s

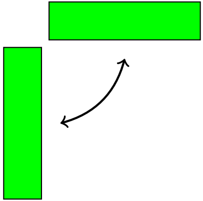


- in RL: FR (green) block is accessed many times; LR (blue) blocks are accessed once
- in LL: FR (green) block is accessed once; LR (blue) blocks are accessed many times

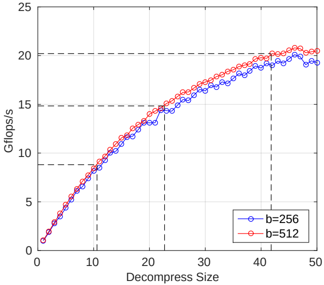
⇒ lower volume of memory transfers (more critical in multithreaded)

⇒ *the Decompress part remains the bottleneck of the Update*

Performance of Update step with LUA(R) (shared, 28 threads)



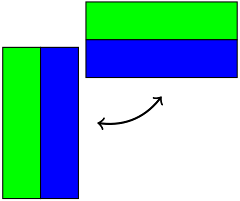
Double precision (d) performance benchmark of Decompress



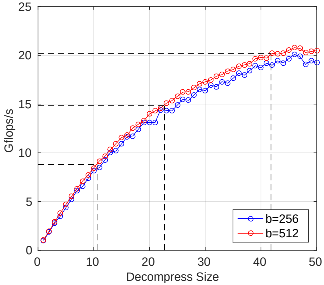
	S3			perf008ar		
	LL	LUA	LUAR*	LL	LUA	LUAR*
Flops in Update ($\times 10^{12}$)	4.0	4.0	2.9	44	44	33
Avg. decompress size	10.6	41.8	22.7	23.3	89.7	48.1
Time in Update	89s	59s	64s	161s	123s	119s

* All metrics include the Recompression overhead

Performance of Update step with LUA(R) (shared, 28 threads)



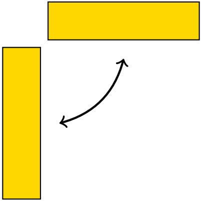
Double precision (d) performance benchmark of Decompress



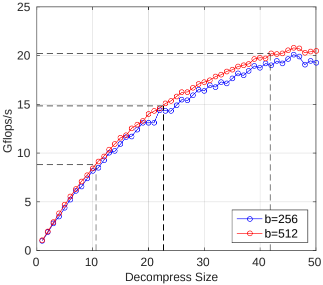
	S3			perf008ar		
	LL	LUA	LUAR*	LL	LUA	LUAR*
Flops in Update ($\times 10^{12}$)	4.0	4.0	2.9	44	44	33
Avg. decompress size	10.6	41.8	22.7	23.3	89.7	48.1
Time in Update	89s	59s	64s	161s	123s	119s

* All metrics include the Recompression overhead

Performance of Update step with LUA(R) (shared, 28 threads)



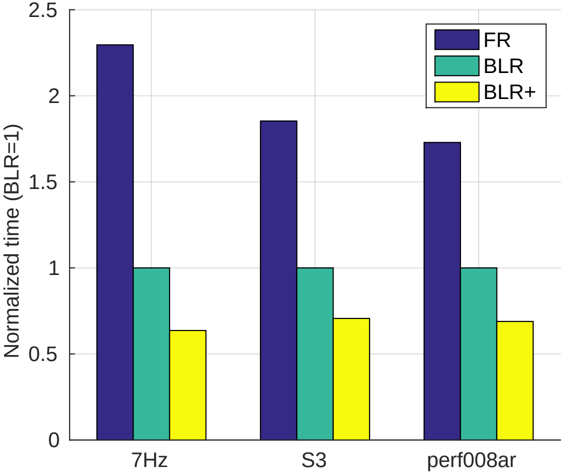
Double precision (d) performance benchmark of Decompress



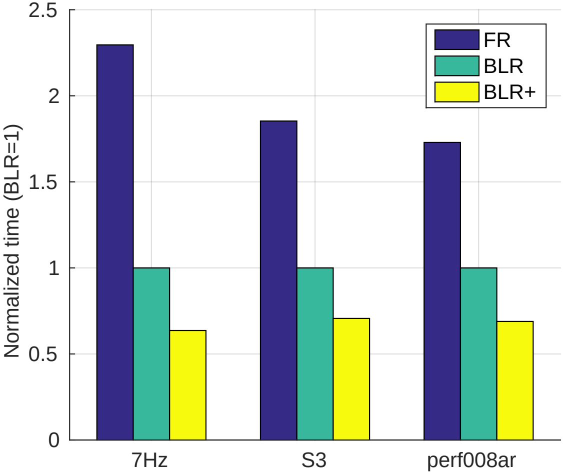
	S3			perf008ar		
	LL	LUA	LUAR*	LL	LUA	LUAR*
Flops in Update ($\times 10^{12}$)	4.0	4.0	2.9	44	44	33
Avg. decompress size	10.6	41.8	22.7	23.3	89.7	48.1
Time in Update	89s	59s	64s	161s	123s	119s

* All metrics include the Recompression overhead

Performance of BLR+ (FCSU+LL+LUA)

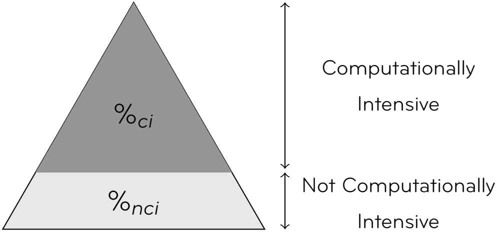


Performance of BLR+ (FCSU+LL+LUA)



⇒ *is there still room for improvement?*

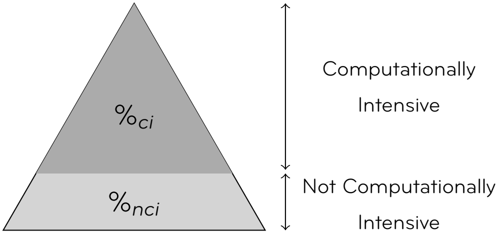
Relative weight of bottom fronts in FR/BLR



	28 threads		
	time	$\%_{nci}$	
FR	585s	18%	

S3 matrix

Relative weight of bottom fronts in FR/BLR



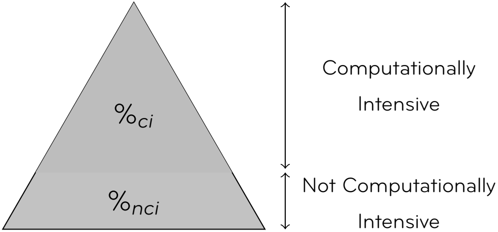
Computationally Intensive

Not Computationally Intensive

	28 threads	
	time	$\%_{nci}$
FR	585s	18%
BLR	315s	34%

S3 matrix

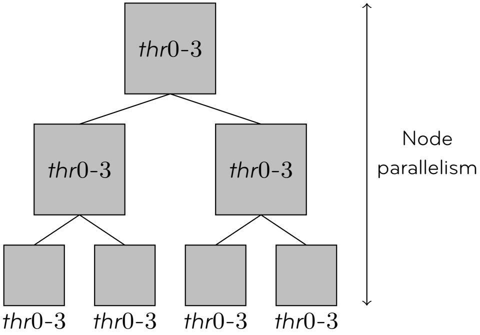
Relative weight of bottom fronts in FR/BLR



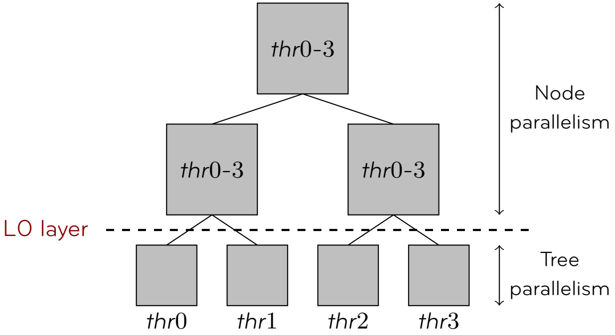
	28 threads	
	time	$\%_{nci}$
FR	585s	18%
BLR	315s	34%
BLR+	223s	48%

S3 matrix

Exploiting tree-based multithreading in MF solvers

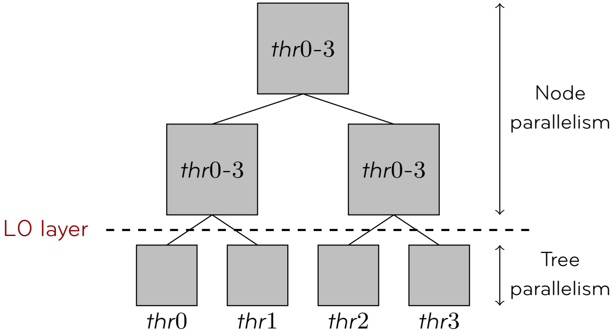


Exploiting tree-based multithreading in MF solvers



- Work based on W. M. Sid-Lakhdar's PhD thesis
  - L0 layer computed with a variant of the Geist-Ng algorithm
  - NUMA-aware implementation
  - use of Idle Core Recycling technique (variant of work-stealing)

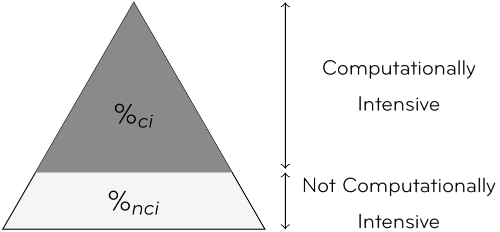
Exploiting tree-based multithreading in MF solvers



- Work based on W. M. Sid-Lakhdar's PhD thesis
  - L0 layer computed with a variant of the Geist-Ng algorithm
  - NUMA-aware implementation
  - use of Idle Core Recycling technique (variant of work-stealing)

⇒ *how big an impact can tree-based multithreading make?*

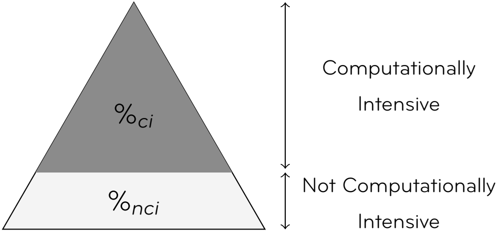
Impact of tree-based multithreading on BLR/BLR+



	28 threads		28 threads + tree MT	
	time	$\%_{nci}$	time	$\%_{nci}$
FR	585s	18%	519s	8%
BLR	315s	34%		
BLR+	223s	48%		

S3 matrix

Impact of tree-based multithreading on BLR/BLR+



	28 threads		28 threads + tree MT	
	time	$\%_{nci}$	time	$\%_{nci}$
FR	585s	18%	519s	8%
BLR	315s	34%	239s	10%
BLR+	223s	48%	136s	9%

S3 matrix

Conclusion and perspectives

Performance results on real-life problems

- Standard BLR variant (FSCU) achieves speedups of order 4 on 900 cores w.r.t. FR
- Scalability of BLR factorization is comparable to FR one
- But flop reduction is not fully translated into performance gain, especially with multithreading
- Improved BLR variants (BLR+) possess better properties (efficiency, granularity, volume of communications, number of operations)
- Tree-based multithreading becomes critical in BLR, especially BLR+
- Combination of tree MT and BLR+ leads to speedups of order 3 on 28 threads w.r.t. standard BLR

Perspectives

- Implementation and performance analysis of the BLR variants in distributed memory (MPI+OpenMP parallelism)
- Efficient strategies to recompress LR updates
- Pivoting strategies compatible with the BLR variants
- Influence of the BLR variants on the accuracy of the factorization

Acknowledgements

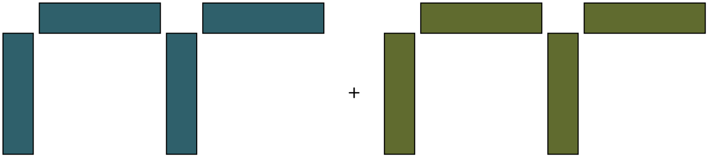
- CALMIP and LIP for providing access to the machines
- EMGS, SEISCOPE and EDF for providing the test matrices
- LSTC members for scientific discussions

Thanks!

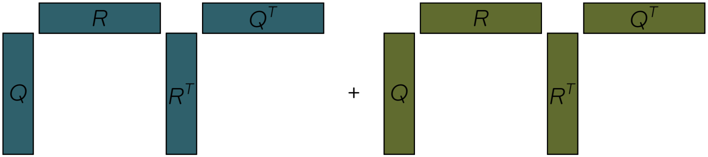
Questions?

Backup Slides

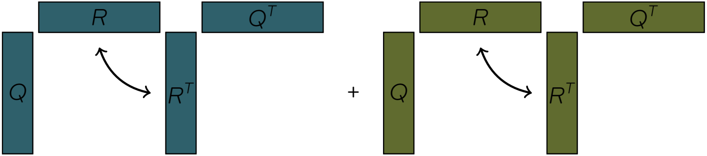
Accumulator recompression



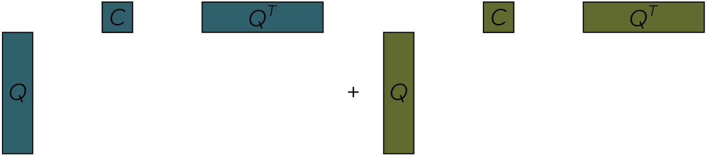
Accumulator recompression



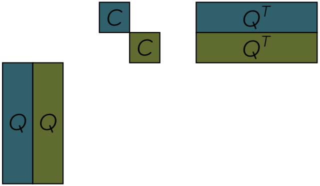
Accumulator recompression



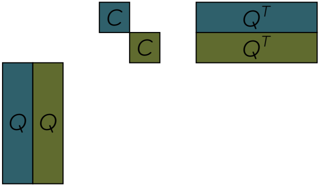
Accumulator recompression



Accumulator recompression



Accumulator recompression



- Weight recompression on $\{C_i\}_i$

  ⇒ With absolute threshold ε, each C_i can be compressed separately

- Redundancy recompression on $\{Q_i\}_i$

  ⇒ Bigger recompression overhead, when is it worth it?