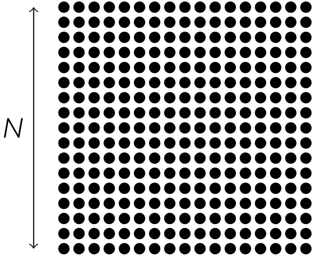
Improving multifrontal solvers by means of Block Low-Rank approximations

The MUMPS team
INP-IRIT, INRIA-LIP, Université de Bordeaux, CNRS-IRIT

LSTC Workshop, Livermore 20/03/2015

The Multifrontal method

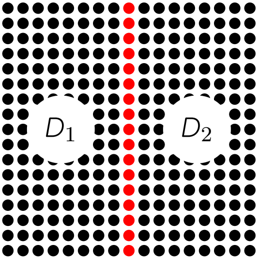
MF (Duff'83) ND (George'73)



2D problem cost $\propto$

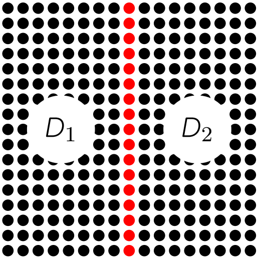
Flops:$\mathcal{O}(N^6)$, mem:$\mathcal{O}(N^4)$

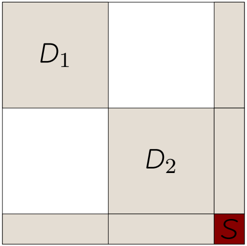
MF (Duff'83) ND (George'73)



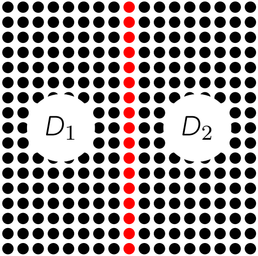
2D problem cost $\propto$

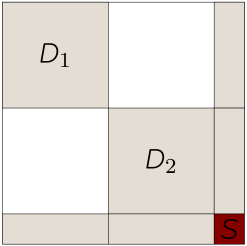
Flops:$\mathcal{O}(N^6)$, mem:$\mathcal{O}(N^4)$





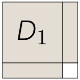
2D problem cost $\propto$
Flops:$\mathcal{O}(N^6)$, mem:$\mathcal{O}(N^4)$



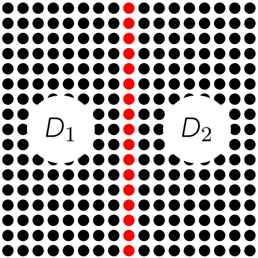


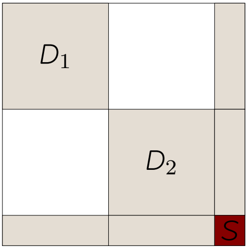
2D problem cost $\propto$

Flops:$\mathcal{O}(N^6)$, mem:$\mathcal{O}(N^4)$



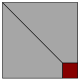
MF (Duff'83) ND (George'73)



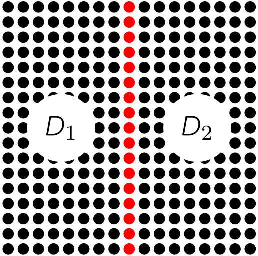


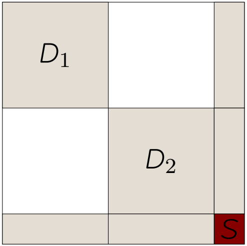
2D problem cost $\propto$

Flops:$\mathcal{O}(N^6)$, mem:$\mathcal{O}(N^4)$

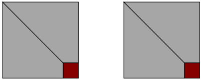


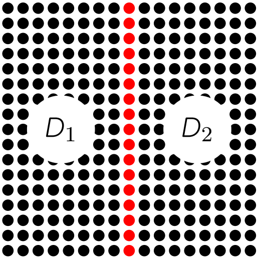
MF (Duff'83) ND (George'73)

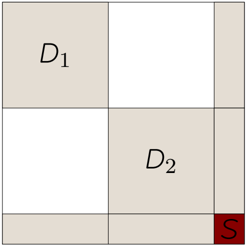




2D problem cost $\propto$
Flops:$\mathcal{O}(N^6)$, mem:$\mathcal{O}(N^4)$

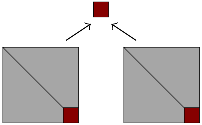


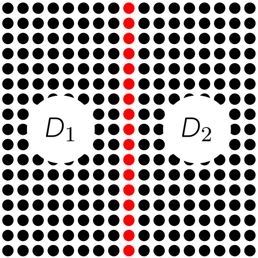


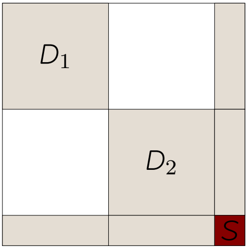


2D problem cost $\propto$

Flops:$\mathcal{O}(N^6)$, mem:$\mathcal{O}(N^4)$



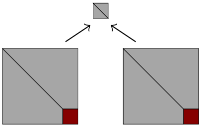


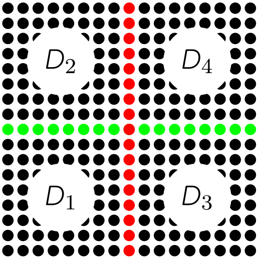


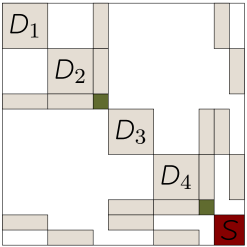
2D problem cost $\propto$

Flops:$\mathcal{O}(N^6)$, mem:$\mathcal{O}(N^4)$

$\rightarrow$ Flops:$\mathcal{O}(N^6/8)$, mem:$\mathcal{O}(N^4/2)$





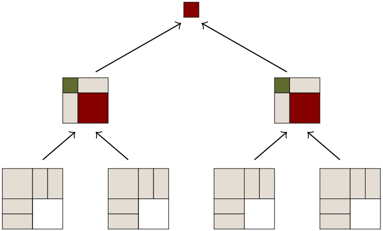


2D problem cost $\propto$

Flops:$\mathcal{O}(N^6)$, mem:$\mathcal{O}(N^4)$

$\rightarrow$ Flops:$\mathcal{O}(N^6/8)$, mem:$\mathcal{O}(N^4/2)$

$\rightarrow$ Flops:$\mathcal{O}(N^3)$, mem:$\mathcal{O}(N^2 log(N))$

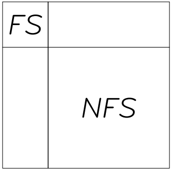


3D problem cost $\propto$

$\rightarrow$ Flops:$\mathcal{O}(N^6)$, mem:$\mathcal{O}(N^4)$

Important things to remember about MF:

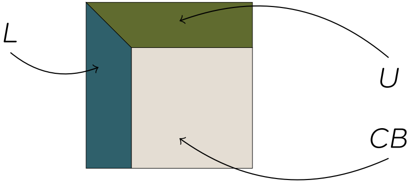
- the elimination tree can be traversed in any topological order
- two sources of parallelism:
  1. Tree: concurrent processing for nodes in different branches
  2. Node: parallel processing for big nodes
- many small nodes at the bottom, few but large on top
- two types of variables in each front: Fully Summed (FS) and Non-FS
- delayed pivoting is a necessary evil.



The Multifrontal method

Important things to remember about MF:

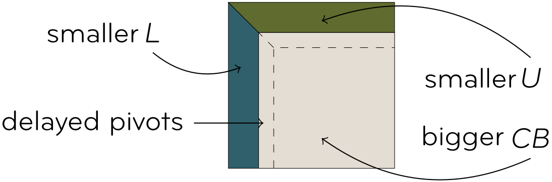
- the elimination tree can be traversed in any topological order
- two sources of parallelism:
  1. Tree: concurrent processing for nodes in different branches
  2. Node: parallel processing for big nodes
- many small nodes at the bottom, few but large on top
- two types of variables in each front: Fully Summed (FS) and Non-FS
- delayed pivoting is a necessary evil.



The Multifrontal method

Important things to remember about MF:

- the elimination tree can be traversed in any topological order
- two sources of parallelism:
  1. Tree: concurrent processing for nodes in different branches
  2. Node: parallel processing for big nodes
- many small nodes at the bottom, few but large on top
- two types of variables in each front: Fully Summed (FS) and Non-FS
- delayed pivoting is a necessary evil. If a pivot does not match a stability criterion, its elimination is postponed to the parent front



The Multifrontal method

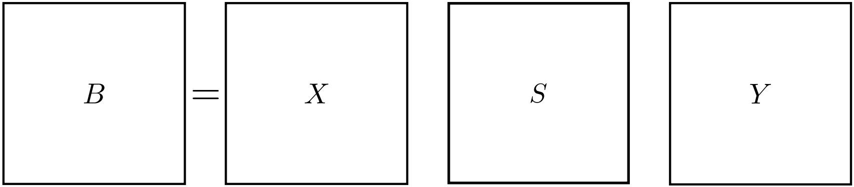
Advantages over iterative solvers:

- easy to use (push button $\rightarrow$ get answer)
- numerically robust
- do one factorization and multiple bw/fw substitutions
- direct solvers are Swiss army knives:
  - solve system
  - compute Schur complement
  - compute rank/null-space
  - compute (selected entries of) the inverse matrix
  - ...
- can be used to precondition iterative solvers

All these features come at the price of high memory and CPU consumption. Low-rank approximations can help.

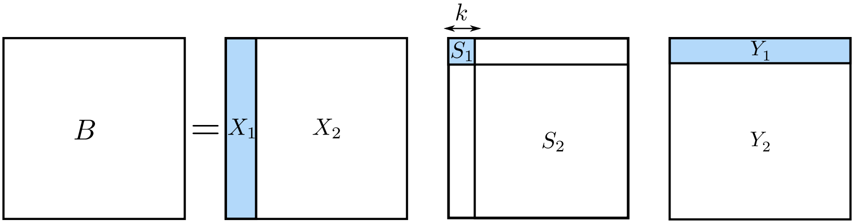
Low-Rank property

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



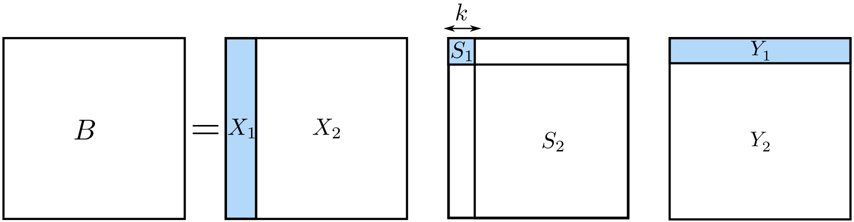
Low-rank matrices

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



$$B = X_1 S_1 Y_1 + X_2 S_2 Y_2 \quad \text{with} \quad S_1(k,k) = \sigma_k > \varepsilon,\ S_2(1,1) = \sigma_{k+1} \leq \varepsilon$$

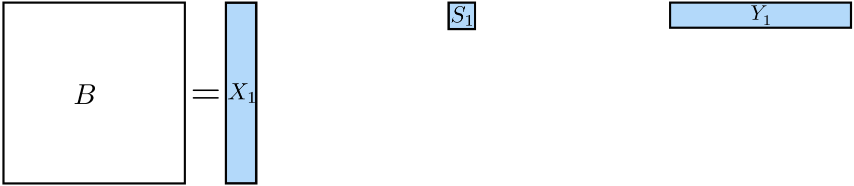
Take a dense matrix B of size $n \times n$ and compute its SVD $B = XSY$:



$B = X_1 S_1 Y_1 + X_2 S_2 Y_2 \quad \text{with} \quad S_1(k,k) = \sigma_k > \varepsilon,\ S_2(1,1) = \sigma_{k+1} \leq \varepsilon$

If $\tilde{B} = X_1 S_1 Y_1 \quad \text{then} \quad \|B - \tilde{B}\|_2 = \|X_2 S_2 Y_2\|_2 = \sigma_{k+1} \leq \varepsilon$

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



$B = X_1 S_1 Y_1 + X_2 S_2 Y_2$ with $S_1(k,k) = \sigma_k > \varepsilon,\ S_2(1,1) = \sigma_{k+1} \leq \varepsilon$

If $\tilde{B} = X_1 S_1 Y_1$ then $\|B - \tilde{B}\|_2 = \|X_2 S_2 Y_2\|_2 = \sigma_{k+1} \leq \varepsilon$

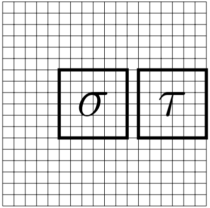
If the singular values of B decay very fast (e.g. exponentially) then $k \ll n$ even for very small ε (e.g. 10^{-14}) $\Rightarrow$ memory and CPU consumption can be reduced considerably with a controlled loss of accuracy ($\leq \varepsilon$) if $\tilde{B}$ is used instead of B

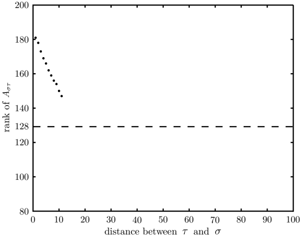
Can we exploit low-rankness in frontal matrices?

Frontal matrices are usually not low-rank but in many applications they exhibit low-rank blocks.
A block represents the interaction between two subdomains σ and τ. If they have a small diameter and are far away the interaction is weak $\Rightarrow$ rank is low

Can we exploit low-rankness in frontal matrices?

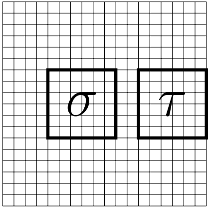
Frontal matrices are usually not low-rank but in many applications they exhibit low-rank blocks.
A block represents the interaction between two subdomains σ and τ. If they have a small diameter and are far away the interaction is weak $\Rightarrow$ rank is low

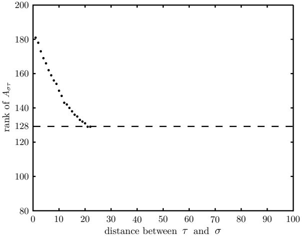




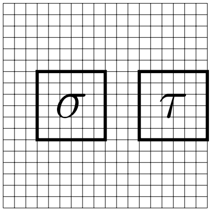
Can we exploit low-rankness in frontal matrices?

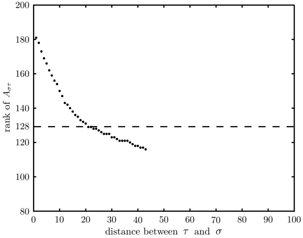
Frontal matrices are usually not low-rank but in many applications they exhibit low-rank blocks.
A block represents the interaction between two subdomains σ and τ. If they have a small diameter and are far away the interaction is weak $\Rightarrow$ rank is low





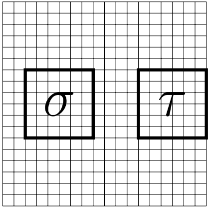
Frontal matrices are usually not low-rank but in many applications they exhibit low-rank blocks.
A block represents the interaction between two subdomains σ and τ. If they have a small diameter and are far away the interaction is weak $\Rightarrow$ rank is low

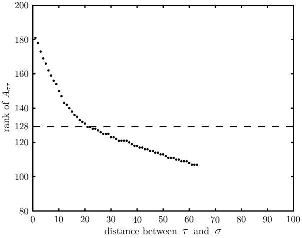




Can we exploit low-rankness in frontal matrices?

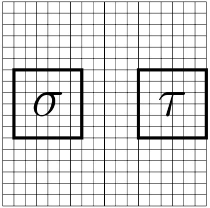
Frontal matrices are usually not low-rank but in many applications they exhibit low-rank blocks.
A block represents the interaction between two subdomains σ and τ. If they have a small diameter and are far away the interaction is weak $\Rightarrow$ rank is low

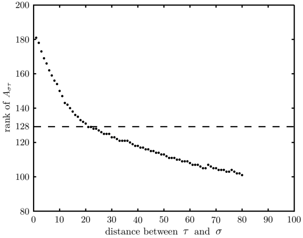




Frontal matrices are usually not low-rank but in many applications they exhibit low-rank blocks.

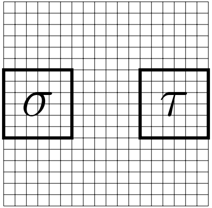
A block represents the interaction between two subdomains σ and τ. If they have a small diameter and are far away the interaction is weak $\Rightarrow$ rank is low

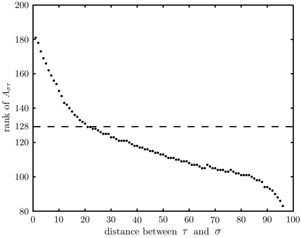




Can we exploit low-rankness in frontal matrices?

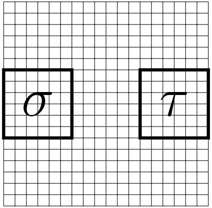
Frontal matrices are usually not low-rank but in many applications they exhibit low-rank blocks.
A block represents the interaction between two subdomains σ and τ. If they have a small diameter and are far away the interaction is weak $\Rightarrow$ rank is low

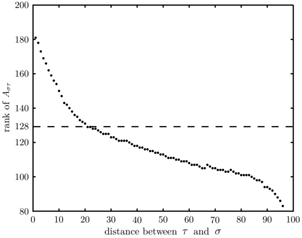




Can we exploit low-rankness in frontal matrices?

Frontal matrices are usually not low-rank but in many applications they exhibit low-rank blocks.
A block represents the interaction between two subdomains σ and τ. If they have a small diameter and are far away the interaction is weak $\Rightarrow$ rank is low

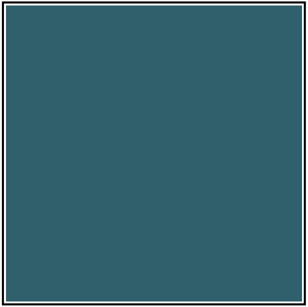




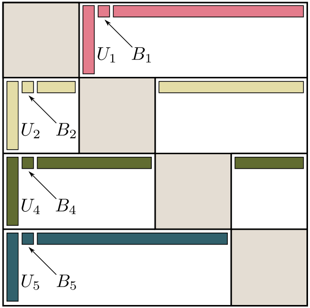
1. compute a clustering of your domain (mesh)
2. permute the matrix accordingly
3. enjoy low-rankness

Low-rank formats

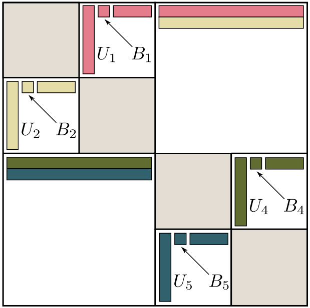
Once the blocking is defined, several low-rank formats are possible.



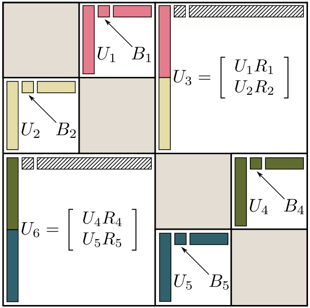
Once the blocking is defined, several low-rank formats are possible. One is Hierarchically Semi-Separable (HSS)



Once the blocking is defined, several low-rank formats are possible. One is Hierarchically Semi-Separable (HSS)

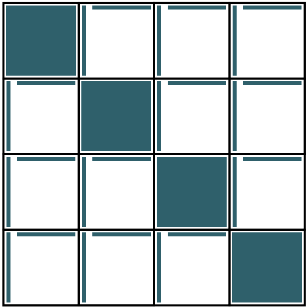


Once the blocking is defined, several low-rank formats are possible. One is Hierarchically Semi-Separable (HSS)



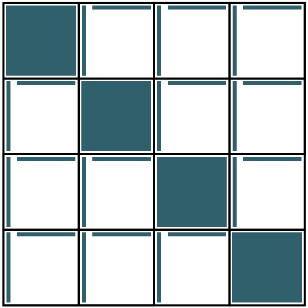
- Leads to very low complexity (fact. is $\sim O(n)$, with a big constant).
- Complex, hierarchical structure.
- Relatively inefficient and expensive SVD/RRQR...(very T&S blocks), unless randomization is used.
- Parallelism is difficult to exploit.

Once the blocking is defined, several low-rank formats are possible. Another one (ours) is Block Low-Rank



- Very simple structure (very little logic to handle).
- Cheap SVD/RRQR.
- Completely parallel.
- Complexity is an open question under investigation.

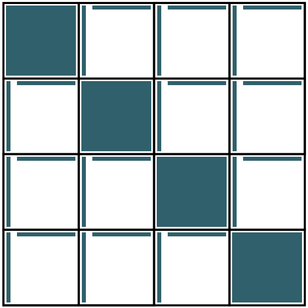
Once the blocking is defined, several low-rank formats are possible. Another one (ours) is Block Low-Rank



- Very simple structure (very little logic to handle).
- Cheap SVD/RRQR.
- Completely parallel.
- Complexity is an open question under investigation.

We believe Block Low-Rank (BLR) aims at a good compromise between complexity and performance/usability.

Once the blocking is defined, several low-rank formats are possible. Another one (ours) is Block Low-Rank



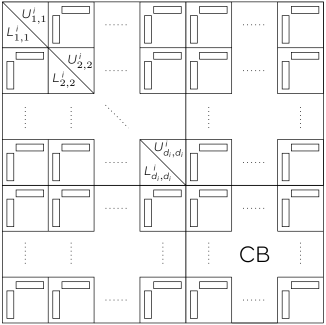
- Very simple structure (very little logic to handle).
- Cheap SVD/RRQR.
- Completely parallel.
- Complexity is an open question under investigation.

We believe Block Low-Rank (BLR) aims at a good compromise between complexity and performance/usability.
Visiting to start joint work on HSS vs BLR comparison.

Clustering

Principle

We aim at a clustering which is such that each frontal matrix has a maximum of low-rank blocks:

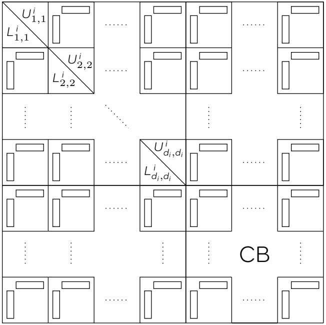


All the variables in a front belong to a separator

- FS: to the separator associated with the front
- NFS: to separator associated with ancestors

Principle

We aim at a clustering which is such that each frontal matrix has a maximum of low-rank blocks:



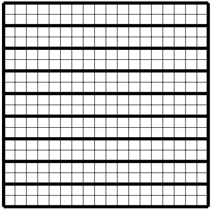
All the variables in a front belong to a separator

- FS: to the separator associated with the front
- NFS: to separator associated with ancestors

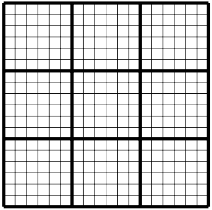
loop over the separators at the analysis phase and compute a clustering for the associated variable

Clustering

If the geometry of the domain, and of the separators is known, the task would be relatively simple



large diameters
small distances



small diameters
large distances

- maximize the relative distance between clusters
- minimize their diameter...
- but not too much to achieve an acceptable BLAS efficiency

In MUMPS we don't have the luxury of knowing the geometry because we only know the matrix, i.e., we are in a purely algebraic context.

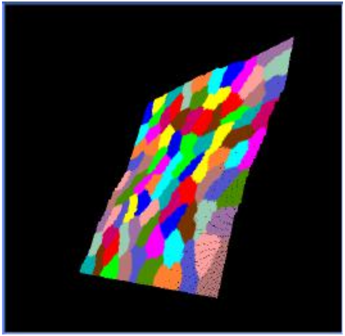
$\rightarrow$ use the adjacency graph instead of the domain geometry

For all the separators

- extract the adjacency graph
- extend it with halo
- pass it to a partitioning tool

End for

SCOTCH-partitioned SCOTCH separator on a cubic domain of size $N = 128$ $\rightarrow$

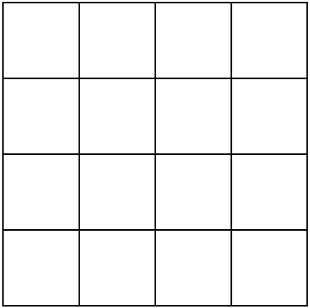


Factorization

BLR LU factorization

task	operation type	full-rank	low-rank
Factor (F)	$B = LU^T$	$(2/3)b^3$	$(2/3)b^3$
Solve (S)	$B = X(YL^{-1})$	b^3	rb^2
Compress (C)	$B = XY$	---	rb^2
Update (U)	$B = B - X_1(Y_1X_2)Y_2$	$2b^3$	rb^2

(b=block size, r=rank)



_GETRF

_TRSM

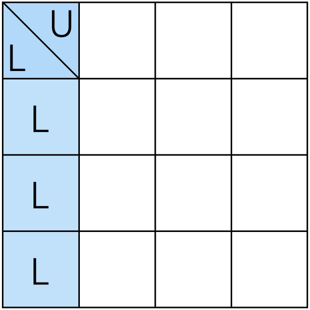
_GEQP3/_GESVD

_GEMM

BLR LU factorization

task	operation type	full-rank	low-rank
Factor (F)	$B = LU^T$	$(2/3)b^3$	$(2/3)b^3$
Solve (S)	$B = X(YL^{-1})$	b^3	rb^2
Compress (C)	$B = XY$	---	rb^2
Update (U)	$B = B - X_1(Y_1X_2)Y_2$	$2b^3$	rb^2

(b=block size, r=rank)

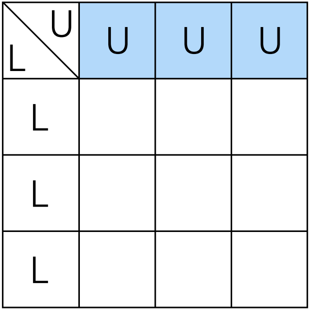


- ▶ _GETRF
- _TRSM
- _GEQP3/_GESVD
- _GEMM

BLR LU factorization

task	operation type	full-rank	low-rank
Factor (F)	$B = LU^T$	$(2/3)b^3$	$(2/3)b^3$
Solve (S)	$B = X(YL^{-1})$	b^3	rb^2
Compress (C)	$B = XY$	---	rb^2
Update (U)	$B = B - X_1(Y_1X_2)Y_2$	$2b^3$	rb^2

(b=block size, r=rank)

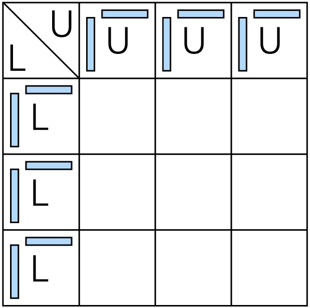


- `_GETRF`
- ▶ `_TRSM`
- `_GEQP3/_GESVD`
- `_GEMM`

BLR LU factorization

task	operation type	full-rank	low-rank
Factor (F)	$B = LU^T$	$(2/3)b^3$	$(2/3)b^3$
Solve (S)	$B = X(YL^{-1})$	b^3	rb^2
Compress (C)	$B = XY$	---	rb^2
Update (U)	$B = B - X_1(Y_1X_2)Y_2$	$2b^3$	rb^2

(b=block size, r=rank)

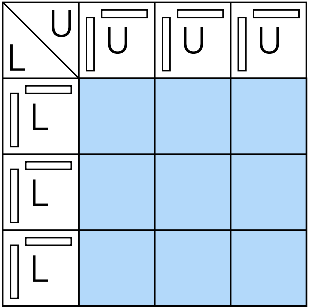


- _GETRF
- _TRSM
- ▶ _GEQP3/_GESVD
- _GEMM

BLR LU factorization

task	operation type	full-rank	low-rank
Factor (F)	$B = LU^T$	$(2/3)b^3$	$(2/3)b^3$
Solve (S)	$B = X(YL^{-1})$	b^3	rb^2
Compress (C)	$B = XY$	---	rb^2
Update (U)	$B = B - X_1(Y_1X_2)Y_2$	$2b^3$	rb^2

(b=block size, r=rank)



_GETRF

_TRSM

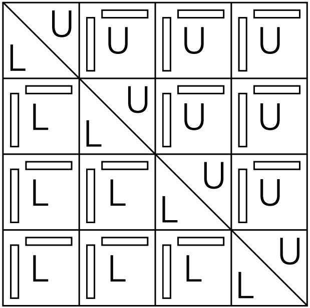
_GEQP3/_GESVD

▶ _GEMM

BLR LU factorization

task	operation type	full-rank	low-rank
Factor (F)	$B = LU^T$	$(2/3)b^3$	$(2/3)b^3$
Solve (S)	$B = X(YL^{-1})$	b^3	rb^2
Compress (C)	$B = XY$	---	rb^2
Update (U)	$B = B - X_1(Y_1X_2)Y_2$	$2b^3$	rb^2

(b=block size, r=rank)



_GETRF

_TRSM

_GEQP3/_GESVD

_GEMM

BLR LU factorization

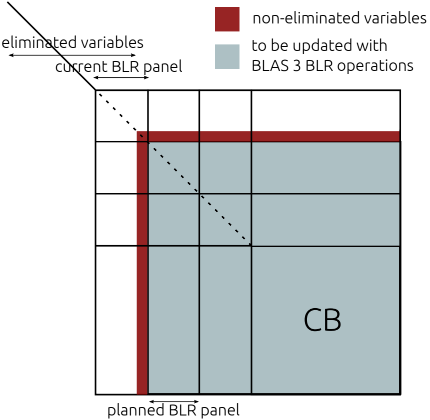
Depending on when and how the compression is done, different variants are possible with different theoretical complexity:

	operations		memory	
	2D	3D	2D	3D
FR	$O(n^{\frac{3}{2}})$	$O(n^2)$	$O(n \log n)$	$O(n^{\frac{4}{3}})$
BLR FSCU	$O(n^{\frac{5}{4}})$	$O(n^{\frac{5}{3}})$	$O(n)$	$O(n \log n)$
BLR FCSU	$O(n^{\frac{7}{6}})$	$O(n^{\frac{14}{9}})$	$O(n)$	$O(n \log n)$
BLR FSCU+LUA	$O(n^{\frac{7}{6}})$	$O(n^{\frac{14}{9}})$	$O(n)$	$O(n \log n)$
BLR FCSU+LUA	$O(n \log n)$	$O(n^{\frac{4}{3}})$	$O(n)$	$O(n \log n)$
HSS	$O(n \log n)$	$O(n^{\frac{4}{3}})$	$O(n)$	$O(n)$

If updates are accumulated and applied at once (LUA), a further reduction can be achieved which leads to the same theoretical complexity as HSS.

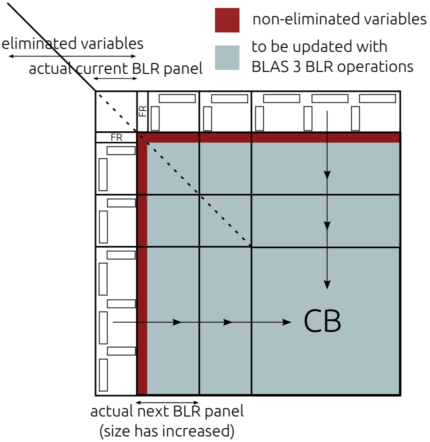
This is work in progress and still not 100% validated (neither theoretically nor experimentally)

Threshold partial pivoting with BLR



Pivots are delayed panelwise and eventually to the parent node

Threshold partial pivoting with BLR



Pivots are delayed panelwise and eventually to the parent node

Experimental results

Experimental MF complexity

Setting:

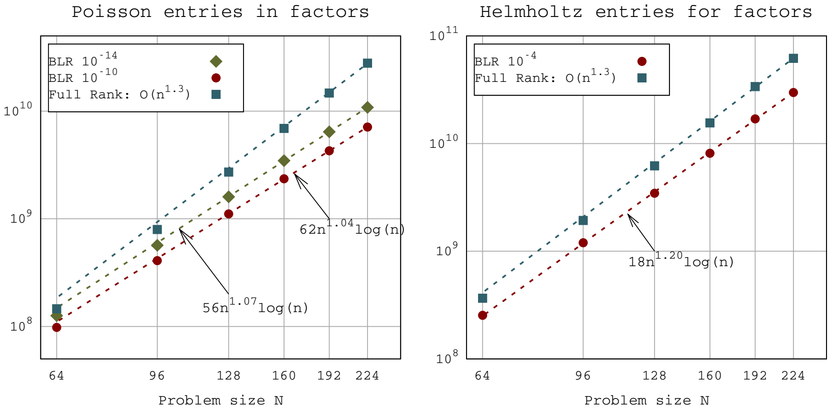
1. `Poisson`: N^3 grid with a 7-point stencil with $u = 1$ on the boundary $\partial\Omega$

$$\Delta u = f$$

2. `Helmholtz`: N^3 grid with a 27-point stencil, ω is the angular frequency, $v(x)$ is the seismic velocity field, and $u(x, \omega)$ is the time-harmonic wavefield solution to the forcing term $s(x, \omega)$.

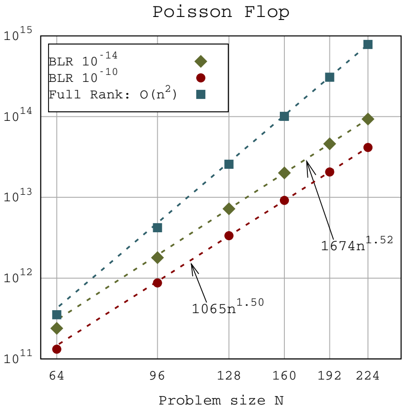
$$\left(-\Delta - \frac{\omega^2}{v(x)^2}\right) u(x, \omega) = s(x, \omega)$$

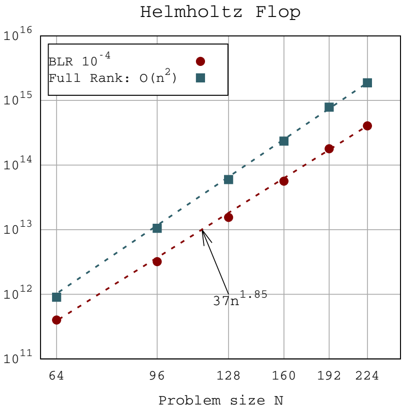
Experimental MF complexity: entries in factor



- ε only plays a role in the constant factor
- good agreement with theory for Poisson but not with Helmholtz (under investigation)
- for Poisson a factor ~ 3 gain with almost no loss of accuracy

Experimental MF complexity: operations





- ε only plays a role in the constant factor
- good agreement with theory for Poisson but not with Helmholtz (under investigation)
- for Poisson a factor ~ 9 gain with almost no loss of accuracy

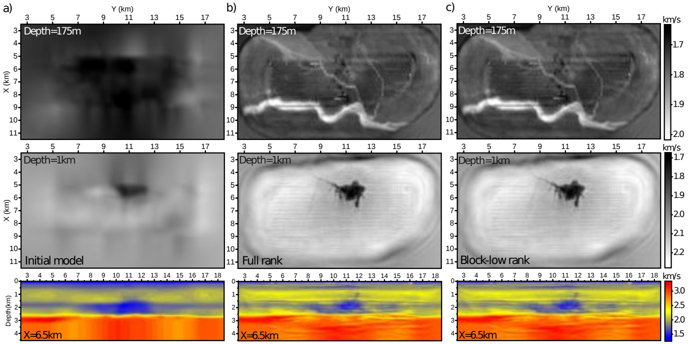
Application to frequency-domain seismic modeling

- Credits: SEISCOPE project
- Seismic modeling in the frequency domain through Full Waveform Inversion
- Helmholtz equation

Freq.	n	nnz	factors	flops	time	cores
5Hz	3M	70M	2.5GB	6.5E+13	80s	240
7Hz	7M	177M	6.4GB	4.1E+14	323s	320
10Hz	17M	446M	10.5GB	2.6E+15	1117s	680

Full-rank statistics

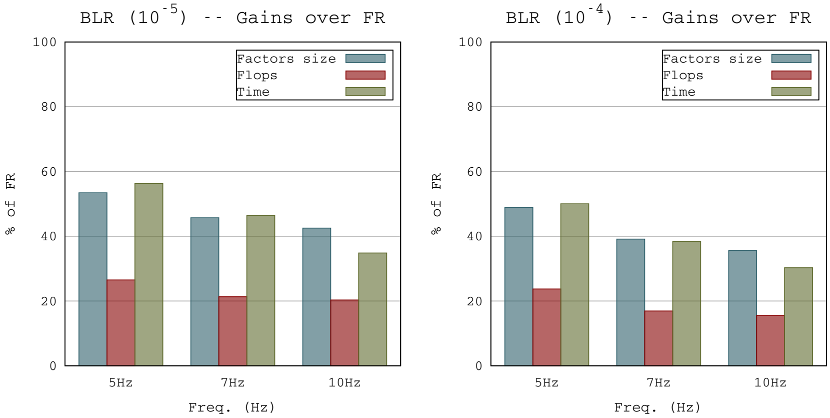
Application to frequency-domain seismic modeling



$7Hz$ problem with single-precision on 320 cores:

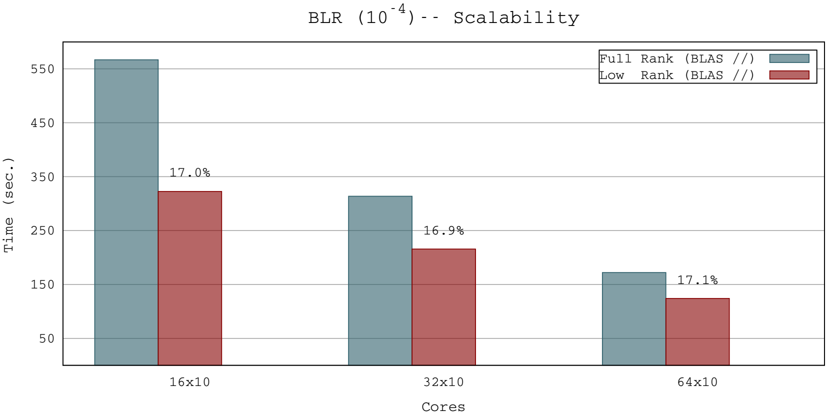
- each row is a different section of the domain
- first column: initial model obtained with traveltime tomography
- second column: FWI solution computed with FR-MUMPS
- third column: FWI solution computed with BLR-MUMPS ($\varepsilon = 10^{-5}$)

Application to frequency-domain seismic modeling



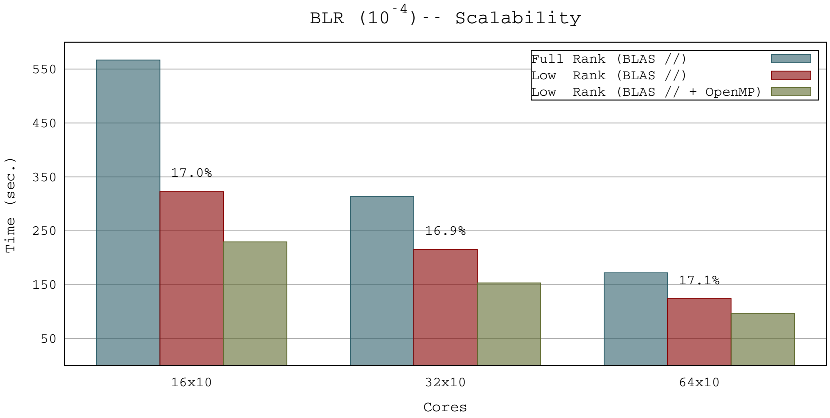
Gains in execution time do not match those in Flops because of the weaker efficiency of BLAS kernels due to the small granularity.
Must tune the block size to achieve the best compromise between compression and efficiency of operations

Application to frequency-domain seismic modeling



Due to the small size of blocks, multithreaded BLAS is inefficient.

Application to frequency-domain seismic modeling



Due to the small size of blocks, multithreaded BLAS is inefficient.
We have added OpenMP directives to exploit multicores on BLR computations

Application to Electromagnetism

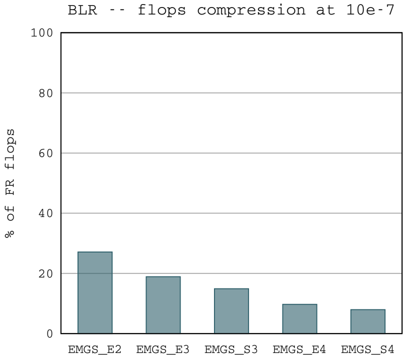
Matrices from EMGS (Norway). All matrices are complex and solved in double-precision

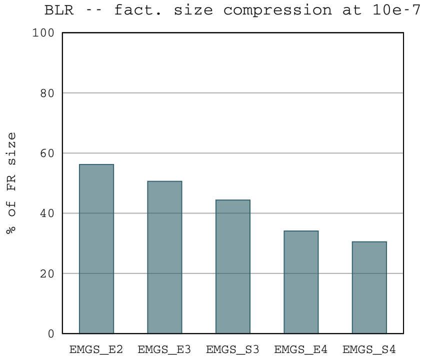
Mat.	n	nnz	factors	flops
EMGS_E2	0.9 M	12M	16GB	6.1e+12
EMGS_E3	2.9 M	37M	76GB	5.6e+13
EMGS_S3	3.3 M	43M	92GB	7.5e+13
EMGS_E4	17.4 M	226M	897GB	2.1e+15
EMGS_S4	20.6 M	266M	1122GB	3.0e+15

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 and 64 GB memory
- Peak per core is 22.4 GFlop/s
- Infiniband interconnect

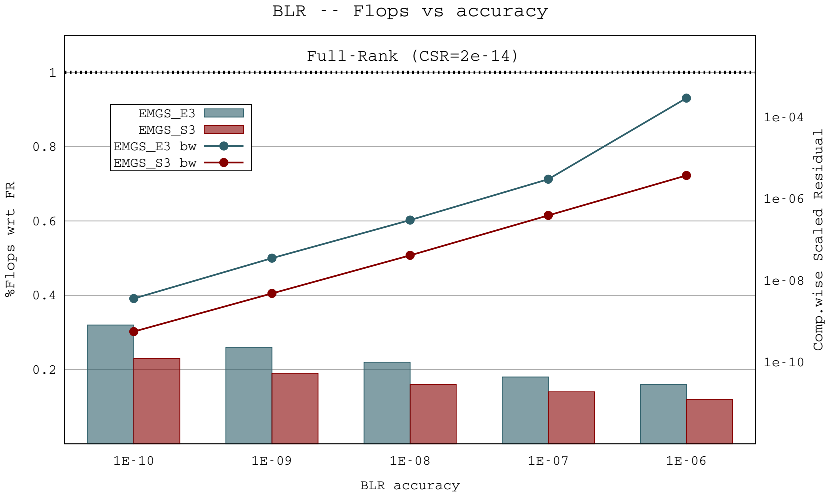
Application to Electromagnetism





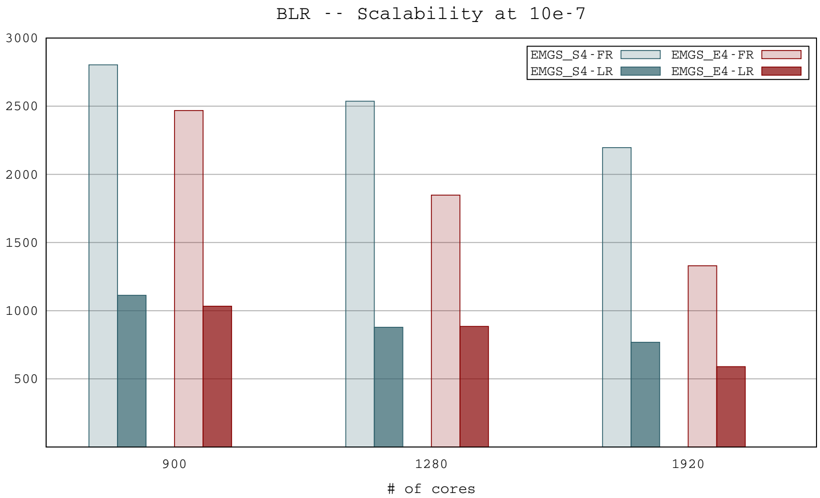
- Gains increase with the size of the problem
- Global memory is reduced more than just factors
- Compression overhead is included

Application to Electromagnetism



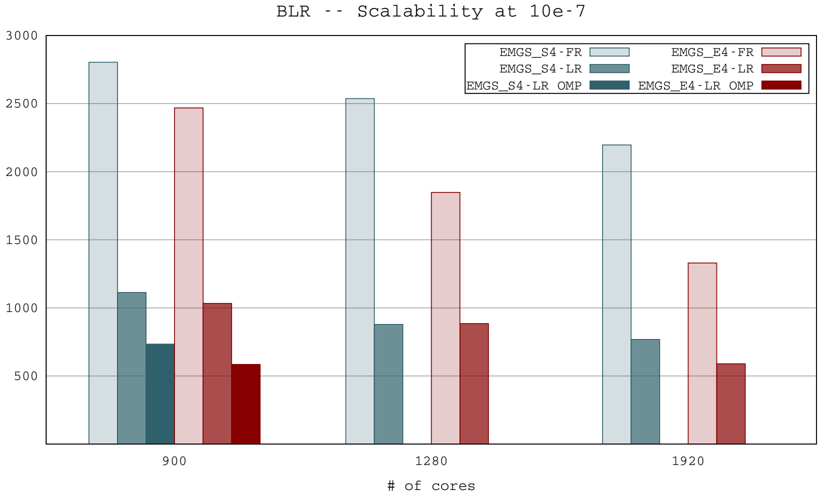
- compression improves, accuracy deteriorates as ε increases
- good agreement between ε and solution accuracy

Application to Electromagnetism



- smaller BLAS granularity (lower seq. and m.threaded speed)
- a factor ~ 2.5 out of ~ 10

Application to Electromagnetism



- smaller BLAS granularity (lower seq. and m.threaded speed)
- a factor ~ 4.2 out of ~ 10 thanks to OpenMP

Thanks!
Questions?