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
1INPT-IRIT  2CNRS-IRIT  3INRIA-LIP  4UPS-IRIT

PMAA’16, Bordeaux July 6-8
Introduction
Multifrontal (Duff ‘83) with Nested Dissection (George ’73)

\[ n = N^d \]

3/25
Multifrontal (Duff ’83) with Nested Dissection (George ’73)

\[ n = N^d \]

3D problem cost \( \propto \)

\( \rightarrow \) Flops: \( O(n^2) \), mem: \( O(n^{4/3}) \)
$\mathcal{H}$ 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.
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.

$$\tilde{B} = XY^T$$ such that $\text{rank}(\tilde{B}) = k_\varepsilon$ 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 ($\leq \varepsilon$)
**H** and BLR matrices

- **H**-matrix
  - Very low theoretical complexity
  - Complex, hierarchical structure

- **BLR** matrix
  - Simple structure
  - Theoretical complexity can be as low as the non-fully structured **H** case

PMAA'16, Bordeaux July 6-8
Very low theoretical complexity
Complex, hierarchical structure

⇒ Our hope is to find a good compromise between theoretical complexity and performance/usability
Variants of the BLR factorization
Variants of the BLR LU factorization

- FSCU

**FSCU**

(Factor, Solve, Compress, Update)

- Better granularity in Update operations
- Potential for recompression (flop reduction)

PMAA’16, Bordeaux July 6-8
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, Update)

- Better granularity in Update operations
  - Potential for recompression and flop reduction
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)

Better granularity in Update operations
Potential for recompression
flop reduction
Variants of the BLR LU factorization

- FSCU (Factor, Solve, Compress, Update)

- Better granularity in Update operations
- Potential for recompression (flop reduction)

PMAA'16, Bordeaux July 6-8
Variants of the BLR LU factorization

- **FSCU** (Factor, Solve, Compress, Update)

- Better granularity in Update operations
  - Potential for recompression
  - flop reduction
Variants of the BLR LU factorization

- FSCU (Factor, Solve, Compress, Update)

Better granularity in Update operations
Potential for recompression
Flop reduction
Variants of the BLR LU factorization

- FSCU (Factor, Solve, Compress, Update)
Variants of the BLR LU factorization

- FSCU (Factor, Solve, Compress, Update)

Better granularity in Update operations
Potential for recompression
flop reduction
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 \(\Rightarrow\) 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
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
3D Electromagnetic Modeling

Maxwell equation

Double complex (z) arithmetic

Symmetric $LDL^T$ factorization

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

Credits: EMGS

<table>
<thead>
<tr>
<th>matrix</th>
<th>n</th>
<th>nnz</th>
<th>flops</th>
<th>storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3</td>
<td>3.3M</td>
<td>43M</td>
<td>78 TF</td>
<td>189 GB</td>
</tr>
<tr>
<td>S4</td>
<td>21M</td>
<td>266M</td>
<td>2.5 PF</td>
<td>2.1 TB</td>
</tr>
<tr>
<td>D4</td>
<td>30M</td>
<td>384M</td>
<td>3.6 PF</td>
<td>3.0 TB</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>matrix</th>
<th>n</th>
<th>nnz</th>
<th>flops</th>
<th>storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>7Hz</td>
<td>7M</td>
<td>177M</td>
<td>410 TF</td>
<td>211 GB</td>
</tr>
<tr>
<td>10Hz</td>
<td>17M</td>
<td>446M</td>
<td>2600 TF</td>
<td>722 GB</td>
</tr>
</tbody>
</table>

Full-Rank statistics
3D Structural Mechanics
Double real (d) arithmetic
Symmetric $LDL^T$ factorization
Required accuracy: $\varepsilon = 10^{-9}$
Credits: Code_Aster (EDF)

<table>
<thead>
<tr>
<th>matrix</th>
<th>n</th>
<th>nnz</th>
<th>flops</th>
<th>storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>perf008ar</td>
<td>4M</td>
<td>159M</td>
<td>378 TF</td>
<td>148 GB</td>
</tr>
</tbody>
</table>

Full-Rank statistics
Low-rank threshold $\varepsilon$ is set according to the application's target.

<table>
<thead>
<tr>
<th>matrix</th>
<th>MUMPS-(Full-Rank)</th>
<th>BLR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>sp-up*</td>
</tr>
<tr>
<td>10Hz</td>
<td>1017s</td>
<td>257</td>
</tr>
<tr>
<td>S4</td>
<td>1538s</td>
<td>371</td>
</tr>
<tr>
<td>D4</td>
<td>2221s</td>
<td>373</td>
</tr>
</tbody>
</table>

*estimated speedup on $90 \times 10$ cores

- good speedup and $%_{peak}$ on 900 cores $\Rightarrow$ good FR reference
- BLR improves performance by a substantial factor of order 4
Low-rank threshold $\varepsilon$ is set according to the application's target

<table>
<thead>
<tr>
<th>matrix</th>
<th>MUMPS-(Full-Rank)</th>
<th>BLR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>sp-up*</td>
<td>%$_{peak}$</td>
</tr>
<tr>
<td>10Hz</td>
<td>1017s</td>
<td>257</td>
<td>26%</td>
</tr>
<tr>
<td>S4</td>
<td>1538s</td>
<td>371</td>
<td>32%</td>
</tr>
<tr>
<td>D4</td>
<td>2221s</td>
<td>373</td>
<td>33%</td>
</tr>
</tbody>
</table>

*estimated speedup on 90 x 10 cores

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

$\Rightarrow$ 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 $\sim 1.6$ for FR and $\sim 1.5$ for BLR

$\Rightarrow$ both FR and BLR scale reasonably well

$\Rightarrow$ ability to maintain gain due to BLR when the number of processes grows
each time the number of processes doubles, speedup of $\sim 1.6$ for FR and $\sim 1.5$ 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 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
### Focus on the Update step (which includes the Decompress)

<table>
<thead>
<tr>
<th></th>
<th>1 thread RL</th>
<th>LL</th>
<th>28 threads RL</th>
<th>LL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S3</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR</td>
<td></td>
<td></td>
<td>468s</td>
<td>526s</td>
</tr>
<tr>
<td>BLR</td>
<td>847s</td>
<td>763s</td>
<td>112s</td>
<td>89s</td>
</tr>
<tr>
<td><strong>perf008ar</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR</td>
<td></td>
<td></td>
<td>663s</td>
<td>766s</td>
</tr>
<tr>
<td>BLR</td>
<td>2174s</td>
<td>2005s</td>
<td>236s</td>
<td>161s</td>
</tr>
</tbody>
</table>

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.
Focus on the Update step (which includes the Decompress)

<table>
<thead>
<tr>
<th></th>
<th>1 thread</th>
<th></th>
<th>28 threads</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RL</td>
<td>LL</td>
<td>RL</td>
<td>LL</td>
</tr>
<tr>
<td>S3</td>
<td>FR</td>
<td>847s</td>
<td>468s</td>
<td>112s</td>
</tr>
<tr>
<td></td>
<td>BLR</td>
<td>763s</td>
<td>526s</td>
<td>89s</td>
</tr>
<tr>
<td>perf008ar</td>
<td>FR</td>
<td>2174s</td>
<td>663s</td>
<td>236s</td>
</tr>
<tr>
<td></td>
<td>BLR</td>
<td>2005s</td>
<td>766s</td>
<td>161s</td>
</tr>
</tbody>
</table>
### Focus on the Update step (which includes the Decompress)

<table>
<thead>
<tr>
<th></th>
<th>1 thread</th>
<th></th>
<th>28 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RL</td>
<td>LL</td>
<td>RL</td>
</tr>
<tr>
<td>S3</td>
<td>FR</td>
<td>847s</td>
<td>763s</td>
</tr>
<tr>
<td></td>
<td>BLR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>perf008ar</td>
<td>FR</td>
<td>2174s</td>
<td>2005s</td>
</tr>
<tr>
<td></td>
<td>BLR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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.
Focus on the Update step (which includes the Decompress)

<table>
<thead>
<tr>
<th></th>
<th>1 thread RL</th>
<th>28 threads RL</th>
<th>28 threads LL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S3</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR</td>
<td>468s</td>
<td>112s</td>
<td>89s</td>
</tr>
<tr>
<td>BLR</td>
<td>847s</td>
<td>763s</td>
<td>526s</td>
</tr>
<tr>
<td><strong>perf008ar</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR</td>
<td>663s</td>
<td>236s</td>
<td>161s</td>
</tr>
<tr>
<td>BLR</td>
<td>2174s</td>
<td>2005s</td>
<td>766s</td>
</tr>
</tbody>
</table>

- 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)
## Focus on the Update step (which includes the Decompress)

<table>
<thead>
<tr>
<th></th>
<th>1 thread</th>
<th>28 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RL</td>
<td>LL</td>
</tr>
<tr>
<td>S3</td>
<td>FR</td>
<td>847s</td>
</tr>
<tr>
<td></td>
<td>BLR</td>
<td>763s</td>
</tr>
<tr>
<td>perf008ar</td>
<td>FR</td>
<td>2174s</td>
</tr>
<tr>
<td></td>
<td>BLR</td>
<td>2005s</td>
</tr>
</tbody>
</table>

- 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

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

* All metrics include the Recompression overhead
Performance of Update step with LUA(R) (shared, 28 threads)

Double precision (d) performance benchmark of Decompress

<table>
<thead>
<tr>
<th></th>
<th>LL</th>
<th>LUA</th>
<th>LUAR*</th>
<th></th>
<th>LL</th>
<th>LUA</th>
<th>LUAR*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flops in Update (\times 10^{12})</td>
<td>4.0</td>
<td>4.0</td>
<td>2.9</td>
<td>perf008ar</td>
<td>44</td>
<td>44</td>
<td>33</td>
</tr>
<tr>
<td>Avg. decompress size</td>
<td>10.6</td>
<td>41.8</td>
<td>22.7</td>
<td></td>
<td>23.3</td>
<td>89.7</td>
<td>48.1</td>
</tr>
<tr>
<td>Time in Update</td>
<td>89s</td>
<td>59s</td>
<td>64s</td>
<td></td>
<td>161s</td>
<td>123s</td>
<td>119s</td>
</tr>
</tbody>
</table>

* All metrics include the Recompression overhead
### Double precision (d) performance benchmark of Decompress

<table>
<thead>
<tr>
<th>Decompress Size</th>
<th>Gflops/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>50</td>
<td>25</td>
</tr>
</tbody>
</table>

- **b=256**
- **b=512**

### Table: Performance Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>LL</th>
<th>LUA</th>
<th>LUAR*</th>
<th>perf008ar LL</th>
<th>perf008ar LUA</th>
<th>perf008ar LUAR*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flops in Update ($\times 10^{12}$)</td>
<td>4.0</td>
<td>4.0</td>
<td>2.9</td>
<td>44</td>
<td>44</td>
<td>33</td>
</tr>
<tr>
<td>Avg. decompress size</td>
<td>10.6</td>
<td>41.8</td>
<td>22.7</td>
<td>23.3</td>
<td>89.7</td>
<td>48.1</td>
</tr>
<tr>
<td>Time in Update</td>
<td>89s</td>
<td>59s</td>
<td>64s</td>
<td>161s</td>
<td>123s</td>
<td>119s</td>
</tr>
</tbody>
</table>

* All metrics include the Recompression overhead
Performance of BLR+ (FCSU+LL+LUA)

- 7Hz
- S3
- perf008ar

Normalized time (BLR=1)

- FR
- BLR
- BLR+

Is there still room for improvement?
Performance of BLR+ (FCSU+LL+LUA)

Is there still room for improvement?
Relative weight of bottom fronts in FR/BLR

<table>
<thead>
<tr>
<th></th>
<th>28 threads</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>%(_{nci})</td>
<td></td>
</tr>
<tr>
<td>FR</td>
<td>585s</td>
<td>18%</td>
</tr>
</tbody>
</table>

S3 matrix
Relative weight of bottom fronts in FR/BLR

<table>
<thead>
<tr>
<th>28 threads</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>%nci</td>
</tr>
<tr>
<td>FR</td>
<td>585s</td>
</tr>
<tr>
<td>BLR</td>
<td>315s</td>
</tr>
</tbody>
</table>

S3 matrix
### Relative weight of bottom fronts in FR/BLR

<table>
<thead>
<tr>
<th></th>
<th>28 threads</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>%$_{nci}$</td>
</tr>
<tr>
<td>FR</td>
<td>585s</td>
<td>18%</td>
</tr>
<tr>
<td>BLR</td>
<td>315s</td>
<td>34%</td>
</tr>
<tr>
<td>BLR+</td>
<td>223s</td>
<td>48%</td>
</tr>
</tbody>
</table>

S3 matrix
Exploiting tree-based multithreading in MF solvers

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?
Exploiting tree-based multithreading in MF solvers

- Work based on W. M. Sid-Lakhdar's PhD thesis
  - LO 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
  - LO 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+

<table>
<thead>
<tr>
<th></th>
<th>28 threads</th>
<th>28 threads + tree MT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>%nci</td>
</tr>
<tr>
<td>FR</td>
<td>585s</td>
<td>18%</td>
</tr>
<tr>
<td>BLR</td>
<td>315s</td>
<td>34%</td>
</tr>
<tr>
<td>BLR+</td>
<td>223s</td>
<td>48%</td>
</tr>
</tbody>
</table>

S3 matrix
Impact of tree-based multithreading on BLR/BLR+

<table>
<thead>
<tr>
<th></th>
<th>28 threads</th>
<th>28 threads + tree MT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>%nci</td>
</tr>
<tr>
<td>FR</td>
<td>585s</td>
<td>18%</td>
</tr>
<tr>
<td>BLR</td>
<td>315s</td>
<td>34%</td>
</tr>
<tr>
<td>BLR+</td>
<td>223s</td>
<td>48%</td>
</tr>
</tbody>
</table>

S3 matrix

PMAA'16, Bordeaux July 6-8
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

Weight recompression on $f(C_i)$ with absolute threshold $Q_i$. Each $C_i$ can be compressed separately.

Redundancy recompression on $f(Q_i)$. Bigger recompression overhead. When is it worth it?
Accumulator recompression

Weight recompression on $f$ (with absolute threshold), each $C_i$ can be compressed separately.

Redundancy recompression on $f$ (bigger recompression overhead, when is it worth it?)
Accumulator recompression

Weight recompression on $Q(RQ^T)$, with absolute threshold $C_i$, each can be compressed separately.

Redundancy recompression on $Q(RQ^T)$, bigger recompression overhead, when is it worth it?
Accumulator recompression

Weight recompression on \( f \) (with absolute threshold)

Redundancy recompression on \( f \)

Bigger recompression overhead, when is it worth it?
Accumulator recompression

Weight recompression on $f(C_i)$

With absolute threshold $\mathcal{T}$, each $C_i$ can be compressed separately

Redundancy recompression on $f(Q_i)$

Bigger recompression overhead, when is it worth it?
Accumulator recompression

- Weight recompression on \( \{C_i\}_i \)
  \( \Rightarrow \) With absolute threshold \( \varepsilon \), each \( C_i \) can be compressed separately

- Redundancy recompression on \( \{Q_i\}_i \)
  \( \Rightarrow \) Bigger recompression overhead, when is it worth it?