Sparse Supernodal Solver exploiting Low-Rankness Property

September 6th, 2017 - Sparse Days

Grégoire Pichon\textsuperscript{a}, Eric Darve\textsuperscript{b}, Mathieu Faverge\textsuperscript{a}, Pierre Ramet\textsuperscript{a}, Jean Roman\textsuperscript{a}

\textsuperscript{a}Inria, Bordeaux INP, CNRS, University of Bordeaux
\textsuperscript{b}Stanford University
Introduction

Current sparse direct solver for 3D problems

- $\Theta(n^2)$ time complexity
- $\Theta(n^{4/3})$ memory complexity
- BLAS 3 operations

Block Low-Rank solver

- Large blocks are compressed into a low-rank form
- Operations keep untouched the global behaviour of PASTIX
- *Minimal Memory* strategy allows to save memory
- *Just-In-Time* strategy allows to reduce time-to-solution

Objective: build an algebraic low-rank solver following the supernodal approach of PASTIX, with block-data structures
Symbolic Factorization

General approach

1. Build a partition with the nested dissection process
2. Compress information on data blocks
3. Compute the block elimination tree using the block quotient graph

Adjacency graph \((G)\).

Quotient graph \((G^*/P) = (G/P)^*\).

Elimination tree \((T)\).

Factorized matrix \((L)\).
Numerical Factorization

Algorithm to eliminate the column block $k$:
1. **Factorize** the diagonal block (POTRF/GETRF)
2. **Solve** off-diagonal blocks in the current column (TRSM)
3. **Update** the trailing matrix with the column’s contribution (GEMM)

Parallelism:
- Right-Looking
- Left-Looking
Large off-diagonal are low-rank, in the form $uv^t$
Some Related Works

Block Low-Rank solvers

- BLR-MUMPS introduced an approach focused on reducing the time to solution which is close to our *Just-In-Time* strategy
- LSTC developed compression/recompression techniques in a dense context

Other approaches for sparse

- Strumpack HSS by Ghysels et al. utilizes randomized sampling to perform an efficient extend-add
- HODLR by Darve et al. uses pre-selection of rows and columns (BDLR)
- $\mathcal{H}$-LU by Hackbusch et al. does not fully exploit the symbolic factorization
Block-Low-Rank Algorithm

Approach
- Large supernodes are partitioned into a set of smaller supernodes
- Large off-diagonal blocks are represented as low-rank blocks

Operations
- Diagonal blocks are dense
- TRSM are performed on low-rank off-diagonal blocks
- GEMM are performed between low-rank off-diagonal blocks. It creates contributions to dense or low-rank blocks: this is the extend-add problem

Compression techniques
- SVD, RRQR for now
- Possible extension to any algebraic method: ACA, randomized techniques...
**Strategy Just-In-Time**

**Compress $L$**

1. Eliminate each column block
   1.1 Factorize the dense diagonal block
      *Compress off-diagonal blocks belonging to the supernode*
   1.2 Apply a TRSM on LR blocks (cheaper)
   1.3 LR update on dense matrices (LR2GE extend-add)

2. Solve triangular systems with low-rank blocks
Strategy *Just-In-Time*

**Compress \( L \)**

1. Eliminate each column block
   1.1 Factorize the dense diagonal block
      Compress off-diagonal blocks belonging to the supernode
   1.2 Apply a TRSM on LR blocks (cheaper)
   1.3 LR update on dense matrices (*LR2GE* extend-add)

2. Solve triangular systems with low-rank blocks
Strategy *Just-In-Time*

**Compress $L$**

1. Eliminate each column block
   1.1 Factorize the dense diagonal block
   Compress off-diagonal blocks belonging to the supernode
   1.2 Apply a TRSM on LR blocks (cheaper)
   1.3 LR update on dense matrices ($LR2GE$ extend-add)

2. Solve triangular systems with low-rank blocks
Strategy *Just-In-Time*

Compress $L$

1. Eliminate each column block
   1.1 Factorize the dense diagonal block
      Compress off-diagonal blocks belonging to the supernode
   1.2 Apply a TRSM on LR blocks (cheaper)
   1.3 LR update on dense matrices ($LR2GE$ extend-add)

2. Solve triangular systems with low-rank blocks
Strategy *Just-In-Time*

**Compress \( L \)**

1. Eliminate each column block
   1.1 Factorize the dense diagonal block
   Compress off-diagonal blocks belonging to the supernode
   1.2 Apply a TRSM on LR blocks (cheaper)
   1.3 LR update on dense matrices (*LR2GE* extend-add)

2. Solve triangular systems with low-rank blocks

---

*Compression*  
*GETRF* (Facto)  
*TRSM* (Solve)  
*LR2LR* (Update)  
*LR2GE* (Update)
Strategy *Minimal Memory*

**Compress \( A \)**

1. Compress large off-diagonal blocks in \( A \) (exploiting sparsity)
2. Eliminate each column block
   - 2.1 Factorize the dense diagonal block
   - 2.2 Apply a TRSM on LR blocks (cheaper)
   - 2.3 LR update on LR matrices (*LR2LR* extend-add)
3. Solve triangular systems with LR blocks

![Diagram](image_url)
Strategy *Minimal Memory*

Compress $A$

1. Compress large off-diagonal blocks in $A$ (exploiting sparsity)
2. Eliminate each column block
   2.1 Factorize the dense diagonal block
   2.2 Apply a TRSM on LR blocks (cheaper)
   2.3 LR update on LR matrices ($LR2LR$ extend-add)
3. Solve triangular systems with LR blocks
Strategy *Minimal Memory*

Compress $A$

1. Compress large off-diagonal blocks in $A$ (exploiting sparsity)
2. Eliminate each column block
   - 2.1 Factorize the dense diagonal block
   - 2.2 Apply a TRSM on LR blocks (cheaper)
   - 2.3 LR update on LR matrices ($LR2LR$ extend-add)
3. Solve triangular systems with LR blocks
Strategy *Minimal Memory*

**Compress $A$**

1. Compress large off-diagonal blocks in $A$ (exploiting sparsity)
2. Eliminate each column block
   2.1 Factorize the dense diagonal block
   2.2 Apply a TRSM on LR blocks (cheaper)
   2.3 LR update on LR matrices (*LR2LR* extend-add)
3. Solve triangular systems with LR blocks
Comparison of both strategies

Memory consumption

- *Minimal Memory* strategy really saves memory
- *Just-In-Time* strategy reduces the size of $L'$ factors, but supernodes are allocated dense at the beginning: no gain in pure *right-looking*

Low-Rank extend-add

- *Minimal Memory* strategy requires expensive extend-add algorithms to update (recompress) low-rank structures with the $LR2LR$ kernel
- *Just-In-Time* strategy continues to apply dense update at a smaller cost through the $LR2GE$ kernel
Focus on the **LR2LR** kernel

- Update of $C$ with contribution from blocs $A$ and $B$
- The low-rank matrix $u_{AB}v_{AB}^t$ is added to $u_Cv_C^t$
- $u_{AB}v_{AB}^t = (u_A(v_A^tv_B))u_B^t$ or $u_{AB}v_{AB}^t = u_A((v_A^tv_B)u_B^t)$
- Eventually, recompression of $(v_A^tv_B)$
**LR2LR kernel using SVD**

A low-rank structure $u_Cv_C^t$ receives a low-rank contribution $u_{AB}v_{AB}^t$.

**Algorithm**

\[
A = u_Cv_C^t + u_{AB}v_{AB}^t = ([u_C, u_{AB}]) \times ([v_C, v_{AB}])^t
\]

- QR: $[u_C, u_{AB}] = Q_1 R_1 \Theta(m(r_C + r_{AB})^2)$
- QR: $[v_C, v_{AB}] = Q_2 R_2 \Theta(n(r_C + r_{AB})^2)$
- SVD: $R_1 R_2^t = u\sigma v^T \Theta((r_C + r_{AB})^3)$

\[
A = (Q_1u\sigma) \times (Q_2v)^t
\]

Optimizations using RRQR to reduce complexity to $\Theta(n(r_C + r_{AB})r_C^*)$. 

\[
\begin{bmatrix}
0 \\
u_C \\
u_{AB} \\
0
\end{bmatrix}
\]

\[
\begin{bmatrix}
\frac{m_C}{r_C} & \frac{r_{AB}}{r_C} \\
\frac{r_{C}}{r_{AB}} & \frac{m_C}{r_{AB}}
\end{bmatrix}
\]

G. Pichon – Sparse Supernodal Solver exploiting Low-Rankness Property

12/23
Experimental setup

Machine: 2 INTEL Xeon E5 – 2680v3 at 2.50 GHz
- 128 GB
- 24 threads

3D Matrices from The SuiteSparse Matrix Collection
- *Audi*: structural problem (943 695 dofs)
- *Atmosmodj*: atmospheric model (1 270 432 dofs)
- *Geo1438*: geomechanical model of earth (1 437 960 dofs)
- *Hook*: model of a steel hook (1 498 023 dofs)
- *Serena*: gas reservoir simulation (1 391 349 dofs)
- + Laplacian: Poisson problem (7-points stencil)

Parallelism is obtained following PASTIX static scheduling for multi-threaded architectures
Parameters

Entry parameters

- Tolerance $\tau$: absolute parameter (normalized for each block)
- Compression method: SVD or RRQR
- Compression strategy: *Minimal Memory* or *Just-In-Time*
- Blocking sizes: between 128 and 256 in following experiments

Strategy *Minimal Memory*

- Blocks are compressed at the beginning
- Each contribution implies a recompression

Strategy *Just-In-Time*

- Blocks are compressed just before a supernode is eliminated
- Those blocks are never uncompressed
Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>SVD</td>
</tr>
<tr>
<td>Factorization time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>49.53</td>
<td>418.5</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>0.9635</td>
<td>1.000</td>
<td>1.003</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>15.80</td>
<td>6.970</td>
<td>6.526</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR product</td>
<td>-</td>
<td>64.10</td>
<td>91.15</td>
</tr>
<tr>
<td>LR addition</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dense udpate (GEMM)</td>
<td>418.7</td>
<td>47.94</td>
<td>47.03</td>
</tr>
<tr>
<td>Total</td>
<td>436</td>
<td>169</td>
<td>564</td>
</tr>
<tr>
<td>Solve time (s)</td>
<td>2.43</td>
<td>1.54</td>
<td>1.8</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>15.9</td>
<td>7.4</td>
<td>6.86</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>15.9</td>
<td>15.9</td>
<td>15.9</td>
</tr>
</tbody>
</table>
Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>SVD</td>
</tr>
<tr>
<td>Factorization time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>49.53</td>
<td>418.5</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>0.9635</td>
<td>1.000</td>
<td>1.003</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>15.80</td>
<td>6.970</td>
<td>6.526</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR product</td>
<td>-</td>
<td>64.10</td>
<td>91.15</td>
</tr>
<tr>
<td>LR addition</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dense udpate (GEMM)</td>
<td>418.7</td>
<td>47.94</td>
<td>47.03</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>436</td>
<td>169</td>
<td>564</td>
</tr>
<tr>
<td>Solve time (s)</td>
<td>2.43</td>
<td>1.54</td>
<td>1.8</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>15.9</td>
<td>7.4</td>
<td>6.86</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>15.9</td>
<td>15.9</td>
<td>15.9</td>
</tr>
</tbody>
</table>
Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>SVD</td>
</tr>
<tr>
<td>Factorization time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>49.53</td>
<td>418.5</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>0.9635</td>
<td>1.000</td>
<td>1.003</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>15.80</td>
<td>6.970</td>
<td>6.526</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR product</td>
<td>-</td>
<td>64.10</td>
<td>91.15</td>
</tr>
<tr>
<td>LR addition</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dense update (GEMM)</td>
<td>418.7</td>
<td>47.94</td>
<td>47.03</td>
</tr>
<tr>
<td>Total</td>
<td>436</td>
<td>169</td>
<td>564</td>
</tr>
<tr>
<td>Solve time (s)</td>
<td>2.43</td>
<td>1.54</td>
<td>1.8</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>15.9</td>
<td>7.4</td>
<td>6.86</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>15.9</td>
<td>15.9</td>
<td>15.9</td>
</tr>
</tbody>
</table>
Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>SVD</td>
</tr>
<tr>
<td>Factorization time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>49.53</td>
<td>418.5</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>0.9635</td>
<td>1.000</td>
<td>1.003</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>15.80</td>
<td>6.970</td>
<td>6.526</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR product</td>
<td>-</td>
<td>64.10</td>
<td>91.15</td>
</tr>
<tr>
<td>LR addition</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dense update (GEMM)</td>
<td>418.7</td>
<td>47.94</td>
<td>47.03</td>
</tr>
<tr>
<td>Total</td>
<td>436</td>
<td>169</td>
<td>564</td>
</tr>
<tr>
<td>Solve time (s)</td>
<td>2.43</td>
<td>1.54</td>
<td>1.8</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>15.9</td>
<td>7.4</td>
<td>6.86</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>15.9</td>
<td>15.9</td>
<td>15.9</td>
</tr>
</tbody>
</table>
Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>SVD</td>
</tr>
<tr>
<td><strong>Factorization time (s)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>49.53</td>
<td>418.5</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>0.9635</td>
<td>1.000</td>
<td>1.003</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>15.80</td>
<td>6.970</td>
<td>6.526</td>
</tr>
<tr>
<td><strong>Update</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR product</td>
<td>-</td>
<td>64.10</td>
<td>91.15</td>
</tr>
<tr>
<td>LR addition</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dense update (GEMM)</td>
<td>418.7</td>
<td>47.94</td>
<td>47.03</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>436</td>
<td>169</td>
<td>564</td>
</tr>
<tr>
<td><strong>Solve time (s)</strong></td>
<td>2.43</td>
<td>1.54</td>
<td>1.8</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>15.9</td>
<td>7.4</td>
<td>6.86</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>15.9</td>
<td>15.9</td>
<td>15.9</td>
</tr>
</tbody>
</table>
## Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>SVD</td>
</tr>
<tr>
<td>Factorization time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>49.53</td>
<td>418.5</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>0.9635</td>
<td>1.000</td>
<td>1.003</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>15.80</td>
<td>6.970</td>
<td>6.526</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR product</td>
<td>-</td>
<td>64.10</td>
<td>91.15</td>
</tr>
<tr>
<td>LR addition</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dense update (GEMM)</td>
<td>418.7</td>
<td>47.94</td>
<td>47.03</td>
</tr>
<tr>
<td>Total</td>
<td>436</td>
<td>169</td>
<td>564</td>
</tr>
<tr>
<td>Solve time (s)</td>
<td>2.43</td>
<td>1.54</td>
<td>1.8</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>15.9</td>
<td>7.4</td>
<td>6.86</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>15.9</td>
<td>15.9</td>
<td>15.9</td>
</tr>
</tbody>
</table>
Behaviour of RRQR/Minimal Memory

Performance

Memory footprint
Performance of RRQR/Just-In-Time

![Bar chart showing the performance of RRQR/Just-In-Time for different problems and parameter values. The x-axis represents different problems: lap120, atmosmodj, audi, Geo1438, Hook, and Serena. The y-axis represents the time ratio of BLR to PaStiX. The chart includes three parameter values: $\tau = 10^{-4}$, $\tau = 10^{-8}$, and $\tau = 10^{-12}$.](image-url)
Convergence of RRQR/Minimal Memory

![Graph showing the convergence of RRQR/Minimal Memory](image)

- Backward error: $\|Ax - b\|_2 / \|b\|_2$
- Numbers on the x-axis represent the number of refinement iterations.
- τ = 10^{-4}
- τ = 10^{-8}

Legend:
- audi
- Geo1438
- Hook
- Serena
- lap120
- atmosmodj
Scaling on Laplacians: Memory Consumption

RRQR/Minimal Memory

Factors size, full rank
Factors size, \( \tau = 10^{-4} \)
Factors size, \( \tau = 10^{-8} \)
Factors size, \( \tau = 10^{-12} \)
Ongoing work with E. Esnard and M. Predari

- Single interaction between $S$ and $S_A \cup S_B$ in the original graph
- Two large interaction blocks $P_{A_1} \cup P_{B_1}$ and $P_{A_2} \cup P_{B_2}$ during the factorization

- Double interaction: between $S$ and $S_A$ and between $S$ and $S_B$ in the original graph
- Three “large” interaction blocks during the factorization
Example on a $200^2$ Laplacian

Scotch ordering

Aligned ordering
Conclusion

Block Low-Rank solver

- Allows to see how to use the symbolic structure and the extend-add issues in a supernodal context
- The version presented follows the parallelism of PASTIX
- On-going work to implement efficiently this approach over the PARSEC runtime system

Ordering strategies

- Study of a nested dissection designed to increase compressibility
- Add pre-selection to isolate data which is not compressible, and thus reduce the overhead of trying to compress uncompressed blocks
PASTIX 6.0.0alpha is now available!
http://gitlab.inria.fr/solverstack/pastix

- Support shared memory with different schedulers:
  - sequential
  - static scheduler
  - PaRSEC runtime system
- Low-rank support
- Cholesky and LU factorizations

Thank you.