Sparse supernodal solver with low-rank compression for solving the frequency-domain Maxwell equations discretized by a high order HDG method

November 20th, 2017 - JJC

G. Pichon, E. Darve, M. Faverge, S. Lanteri, P. Ramet, J. Roman

Inria, Bordeaux INP, CNRS, University of Bordeaux, Stanford University, Inria, Université Nice - Sophia Antipolis, CNRS
## Introduction

### PaStiX: Sparse Direct Solver
- Originally designed for distributed heterogeneous architectures
- Recent works introduced low-rank compression to reduce the burden on time and memory complexities

### HORSE: HDG solver
- Solve a reduced system on faces instead of elements
- Lead to a sparse system smaller than the one using CG or DG
- Domain-Decomposition is then used
- Use a sparse direct solver on each subdomain

Objective for this talk: evaluate the impact of using low-rank compression to enhance subdomains’ factorization in HORSE.
Outline

1. Background on Sparse Direct Solvers
2. Block Low-Rank Operations
3. Numerical Experiments
4. Case study: PASTIX inside a HDG method
1

Background on Sparse Direct Solvers
PaStiX: a sparse direct solver

Current sparse direct solvers for 3D problems
- $\Theta(n^2)$ time complexity
- $\Theta(n^{4/3})$ memory complexity
- BLAS 3 operations: efficient operations

Block Low-Rank solver
- In many applications, the machine precision is not required
- **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
Sparse Direct Solvers approach

Problem: solve $Ax = b$

- Cholesky: factorize $A = LL^t$ (symmetric pattern $(A + A^t)$ for $LU$)
- Solve $Ly = b$
- Solve $L^tx = y$

PaStiX steps

1. Order unknowns to minimize the fill-in
2. Compute a symbolic factorization to build $L$ structure
3. Factorize the matrix in place on $L$ structure
4. Solve the system with forward and backward triangular solves
Ordering with Nested Dissection

Entry graph $G = (V, E, \sigma_p)$
$V$: vertices, $E$: edges, $\sigma_p$: unknowns permutation

Algorithm to compute $\sigma_p$

1. Partition $V = A \cup B \cup C$
2. Order $C$ with larger numbers: $V_A < V_C$ and $V_B < V_C$
3. Apply the process recursively on $A$ and $B$

Three-levels of nested dissection on a regular cube.

Metis and Scotch are mostly used to perform this partitioning.
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)
Block Low-Rank Operations
Low-Rank Compression

\[ M = uv^T \]

\( M \in \mathbb{R}^{n \times n}; u, v \in \mathbb{R}^{n \times r} \)

Memory Consumption reduced: \(2nr\) instead of \(n^2\)

Several methods can be used: SVD, RRQR, BDLR, ACA...

The objective is to compute \( \hat{A} \) such that \( ||A - \hat{A}||_2 < \tau \)
Block Low-Rank Compression for a dense matrix

Some related works for dense matrices

- LSTC
- Airbus (hmat-oss)
- CEA
Block Low-Rank Compression – Symbolic Factorization

Large off-diagonal are low-rank, in the form $uv^t$
Some Related Works

Sparse Block Low-Rank solver: BLR-MUMPS
- They introduced an approach to reduce the time-to-solution
- The main difference is the difference between multifrontal and supernodal

Some other approaches for sparse
- Strumpack HSS by Ghysels et al. uses 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
- HODLR/HSS like method by Chadwick et al. uses estimation of ranks and the geometry of the problem
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
   1.2 Compress off-diagonal blocks belonging to the supernode
   1.3 Apply a TRSM on LR blocks (cheaper)
   1.4 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
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 (\textbf{LR2LR} extend-add)
3. Solve triangular systems with LR blocks

G. Pichon – \textsc{PaStiX}: a Sparse Supernodal Solver
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 ($\text{LR2LR}$ 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
Focus on the LR2LR kernel: build the $u_{AB}v_{AB}^t$ contribution

- Update of $C$ with contribution from blocks $A$ and $B$
- The low-rank matrix $u_{AB}v_{AB}^t$ is added to $u_Cv_C^t$
- $u_{AB}v_{AB}^t = (u_A(v_A^tv_B^t))u_B^t$ or $u_{AB}v_{AB}^t = u_A((v_A^tv_B^t)u_B^t)$
- Eventually, recompression of $(v_A^tv_B)$
Focus on the LR2LR kernel: add the contribution into $u_C v_C^t$

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

Algorithm

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

- 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: $R_1 R_2^t = u\sigma v^t \Theta((r_C + r_{AB})^3)$

$$A = (Q_1 u\sigma) \times (Q_2 v)^t$$

Optimizations using RRQR to reduce complexity to $\Theta(n(r_C + r_{AB})r_C^*)$
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
3

Numerical Experiments
### Experimental setup

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

**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 with 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
- Compression sizes: 128 for width and 20 for height

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>Complete</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>RRQR</td>
</tr>
<tr>
<td>Factorization time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>3.4e+01</td>
<td>5.6e+00</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>7.2e-01</td>
<td>7.3e-01</td>
<td>7.6e-01</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>1.7e+01</td>
<td>7.4e+00</td>
<td>7.9e+00</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formation of contribution</td>
<td>-</td>
<td>-</td>
<td>4.2e+01</td>
</tr>
<tr>
<td>Addition of contribution</td>
<td>-</td>
<td>-</td>
<td>7.3e+02</td>
</tr>
<tr>
<td>Dense update (GEMM)</td>
<td>4.6e+02</td>
<td>9.7e+01</td>
<td>2.4e+01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4.7e+02</td>
<td>1.4e+02</td>
<td>8.1e+02</td>
</tr>
<tr>
<td>Solve time (s)</td>
<td>6.3e+00</td>
<td>3.0e+00</td>
<td>3.2e+00</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>16.3</td>
<td>7.49</td>
<td>7.31</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>16.3</td>
<td>16.3</td>
<td>7.31</td>
</tr>
</tbody>
</table>
Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>RRQR</td>
</tr>
<tr>
<td>Factorization time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>3.4e+01</td>
<td>5.6e+00</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>7.2e-01</td>
<td>7.3e-01</td>
<td>7.6e-01</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>1.7e+01</td>
<td>7.4e+00</td>
<td>7.9e+00</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formation of contribution</td>
<td>-</td>
<td>-</td>
<td>4.2e+01</td>
</tr>
<tr>
<td>Addition of contribution</td>
<td>-</td>
<td>-</td>
<td>7.3e+02</td>
</tr>
<tr>
<td>Dense update (GEMM)</td>
<td>4.6e+02</td>
<td>9.7e+01</td>
<td>2.4e+01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4.7e+02</td>
<td>1.4e+02</td>
<td>8.1e+02</td>
</tr>
<tr>
<td>Solve time (s)</td>
<td>6.3e+00</td>
<td>3.0e+00</td>
<td>3.2e+00</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>16.3</td>
<td>7.49</td>
<td>7.31</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>16.3</td>
<td>16.3</td>
<td>7.31</td>
</tr>
</tbody>
</table>
## Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>RRQR</td>
</tr>
<tr>
<td>Factorization time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>3.4e+01</td>
<td>5.6e+00</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>7.2e-01</td>
<td>7.3e-01</td>
<td>7.6e-01</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>1.7e+01</td>
<td>7.4e+00</td>
<td>7.9e+00</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formation of contribution</td>
<td>-</td>
<td>-</td>
<td>4.2e+01</td>
</tr>
<tr>
<td>Addition of contribution</td>
<td>-</td>
<td>-</td>
<td>7.3e+02</td>
</tr>
<tr>
<td>Dense udpate (GEMM)</td>
<td>4.6e+02</td>
<td>9.7e+01</td>
<td>2.4e+01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4.7e+02</td>
<td>1.4e+02</td>
<td>8.1e+02</td>
</tr>
<tr>
<td>Solve time (s)</td>
<td>6.3e+00</td>
<td>3.0e+00</td>
<td>3.2e+00</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>16.3</td>
<td>7.49</td>
<td>7.31</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>16.3</td>
<td>16.3</td>
<td>7.31</td>
</tr>
</tbody>
</table>
### Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>RRQR</td>
</tr>
<tr>
<td><strong>Factorization time (s)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>3.4e+01</td>
<td>5.6e+00</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>7.2e-01</td>
<td>7.3e-01</td>
<td>7.6e-01</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>1.7e+01</td>
<td>7.4e+00</td>
<td>7.9e+00</td>
</tr>
<tr>
<td><strong>Update</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formation of contribution</td>
<td>-</td>
<td>-</td>
<td>4.2e+01</td>
</tr>
<tr>
<td>Addition of contribution</td>
<td>-</td>
<td>-</td>
<td>7.3e+02</td>
</tr>
<tr>
<td>Dense update (GEMM)</td>
<td>4.6e+02</td>
<td>9.7e+01</td>
<td>2.4e+01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4.7e+02</td>
<td>1.4e+02</td>
<td>8.1e+02</td>
</tr>
<tr>
<td><strong>Solve time (s)</strong></td>
<td>6.3e+00</td>
<td>3.0e+00</td>
<td>3.2e+00</td>
</tr>
<tr>
<td><strong>Factors final size (GB)</strong></td>
<td>16.3</td>
<td>7.49</td>
<td>7.31</td>
</tr>
<tr>
<td><strong>Memory peak (GB)</strong></td>
<td>16.3</td>
<td>16.3</td>
<td>7.31</td>
</tr>
</tbody>
</table>
Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>RRQR</td>
</tr>
<tr>
<td>Factorization time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>3.4e+01</td>
<td>5.6+00</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>7.2e-01</td>
<td>7.3e-01</td>
<td>7.6e-01</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>1.7e+01</td>
<td>7.4e+00</td>
<td>7.9e+00</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formation of contribution</td>
<td>-</td>
<td>-</td>
<td>4.2e+01</td>
</tr>
<tr>
<td>Addition of contribution</td>
<td>-</td>
<td>-</td>
<td>7.3e+02</td>
</tr>
<tr>
<td>Dense update (GEMM)</td>
<td>4.6e+02</td>
<td>9.7e+01</td>
<td>2.4e+01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4.7e+02</td>
<td>1.4e+02</td>
<td>8.1e+02</td>
</tr>
<tr>
<td>Solve time (s)</td>
<td>6.3e+00</td>
<td>3.0e+00</td>
<td>3.2e+00</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>16.3</td>
<td>7.49</td>
<td>7.31</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>16.3</td>
<td>16.3</td>
<td>7.31</td>
</tr>
</tbody>
</table>
Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>RRQR</td>
</tr>
<tr>
<td>Factorization time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td></td>
<td>3.4e+01</td>
<td>5.6+00</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>7.2e-01</td>
<td>7.3e-01</td>
<td>7.6e-01</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>1.7e+01</td>
<td>7.4e+00</td>
<td>7.9e+00</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formation of contribution</td>
<td></td>
<td></td>
<td>4.2e+01</td>
</tr>
<tr>
<td>Addition of contribution</td>
<td></td>
<td></td>
<td>7.3e+02</td>
</tr>
<tr>
<td>Dense update (GEMM)</td>
<td>4.6e+02</td>
<td>9.7e+01</td>
<td>2.4e+01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4.7e+02</td>
<td>1.4e+02</td>
<td>8.1e+02</td>
</tr>
<tr>
<td>Solve time (s)</td>
<td>6.3e+00</td>
<td>3.0e+00</td>
<td>3.2e+00</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>16.3</td>
<td>7.49</td>
<td>7.31</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>16.3</td>
<td>16.3</td>
<td>7.31</td>
</tr>
</tbody>
</table>
## Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>RRQR</td>
</tr>
<tr>
<td><strong>Factorization time (s)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>-</td>
<td>3.4e+01</td>
<td>5.6e+00</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>7.2e-01</td>
<td>7.3e-01</td>
<td>7.6e-01</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>1.7e+01</td>
<td>7.4e+00</td>
<td>7.9e+00</td>
</tr>
<tr>
<td><strong>Update</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formation of contribution</td>
<td>-</td>
<td>-</td>
<td>4.2e+01</td>
</tr>
<tr>
<td>Addition of contribution</td>
<td>-</td>
<td>-</td>
<td>7.3e+02</td>
</tr>
<tr>
<td>Dense update (GEMM)</td>
<td>4.6e+02</td>
<td>9.7e+01</td>
<td>2.4e+01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4.7e+02</td>
<td>1.4e+02</td>
<td>8.1e+02</td>
</tr>
<tr>
<td><strong>Solve time (s)</strong></td>
<td>6.3e+00</td>
<td>3.0e+00</td>
<td>3.2e+00</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>16.3</td>
<td>7.49</td>
<td>7.31</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>16.3</td>
<td>16.3</td>
<td>7.31</td>
</tr>
</tbody>
</table>
Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Just-In-Time</th>
<th>Minimal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RRQR</td>
<td>RRQR</td>
</tr>
<tr>
<td>Factorization time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td></td>
<td>3.4e+01</td>
<td>5.6e+00</td>
</tr>
<tr>
<td>Block factorization (GETRF)</td>
<td>7.2e-01</td>
<td>7.3e-01</td>
<td>7.6e-01</td>
</tr>
<tr>
<td>Panel solve (TRSM)</td>
<td>1.7e+01</td>
<td>7.4e+00</td>
<td>7.9e+00</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formation of contribution</td>
<td></td>
<td>-</td>
<td>4.2e+01</td>
</tr>
<tr>
<td>Addition of contribution</td>
<td></td>
<td>-</td>
<td>7.3e+02</td>
</tr>
<tr>
<td>Dense update (GEMM)</td>
<td>4.6e+02</td>
<td>9.7e+01</td>
<td>2.4e+01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4.7e+02</td>
<td>1.4e+02</td>
<td>8.1e+02</td>
</tr>
<tr>
<td>Solve time (s)</td>
<td>6.3e+00</td>
<td>3.0e+00</td>
<td>3.2e+00</td>
</tr>
<tr>
<td>Factors final size (GB)</td>
<td>16.3</td>
<td>7.49</td>
<td>7.31</td>
</tr>
<tr>
<td>Memory peak (GB)</td>
<td>16.3</td>
<td>16.3</td>
<td>7.31</td>
</tr>
</tbody>
</table>
Behaviour of RRQR/Minimal Memory

Performance

Memory footprint
Performance of RRQR/Just-In-Time

\[ \tau = 10^{-4} \quad \tau = 10^{-8} \quad \tau = 10^{-12} \]

- Time BLR / Time PaStiX

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Time BLR / Time PaStiX</th>
<th>( \tau = 10^{-4} )</th>
<th>( \tau = 10^{-8} )</th>
<th>( \tau = 10^{-12} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>lap120</td>
<td>1.0e-05</td>
<td>6.1e-10</td>
<td>2.3e-14</td>
<td></td>
</tr>
<tr>
<td>atmosmodj</td>
<td>1.1e-05</td>
<td>6.5e-10</td>
<td>1.6e-14</td>
<td></td>
</tr>
<tr>
<td>audi</td>
<td>2.6e-04</td>
<td>2.4e-10</td>
<td>2.1e-15</td>
<td></td>
</tr>
<tr>
<td>Geo1438</td>
<td>6.9e-05</td>
<td>1.4e-08</td>
<td>9.1e-13</td>
<td></td>
</tr>
<tr>
<td>Hook</td>
<td>4.2e-04</td>
<td>3.7e-10</td>
<td>7.4e-15</td>
<td></td>
</tr>
<tr>
<td>Serena</td>
<td>5.7e-05</td>
<td>1.9e-08</td>
<td>1.8e-12</td>
<td></td>
</tr>
</tbody>
</table>
Convergence of RRQR/Minimal Memory

![Graph showing the convergence of RRQR/Minimal Memory with backward error and number of refinement iterations.]

- **Backward error:** $\|Ax - b\|_2$
- $\|b\|_2$
- $\tau = 10^{-4}$
- $\tau = 10^{-8}$

Legend:
- audi
- Geo1438
- Hook
- Serena
- lap120
- atmosmodj
Scaling on Laplacians: size of the factors using RRQR/Minimal Memory

Size of the factors (GB)

Size of the factors, complete factorization
Size of the factors, low-rank tolerance=10^{-8}
Size of the factors, low-rank tolerance=10^{-4}
4

Case study: PASTIX inside a HDG method
HORSE

Method

- Introduce an hybrid variable to solve a system smaller than the one obtained through DG
- Perform Domain-Decomposition to solve this system
- Use a direct solver for subdomains

Solve 3D frequency-domain Maxwell’s equations

- $i\omega \varepsilon_r E - \text{curl} H = -J$, in $\Omega$
- $i\omega \mu_r H + \text{curl} E = 0$, in $\Omega$
- $n \times E = 0$, on $\Gamma_m$
- $n \times E + n \times (n \times H) = n \times E^{\text{inc}} + n \times (n \times H^{\text{inc}})$, on $\Gamma_a$
The challenge with DG for frequency-domain problems

Classical DG formulation

- Naturally adapted to heterogeneous media and discontinuous solutions
- Can easily deal with unstructured, possibly non-conforming meshes ($h$-adaptivity)
- High order with compact stencils and non-conforming approximations ($p$-adaptivity)
- Usually rely on polynomial interpolation but can also accommodate alternative functions (e.g. plane waves)
- Well-suited for efficient parallelization
- But leads to larger problems compared to continuous finite element methods
Hybridizable DG method in 3D

Principles of a HDG formulation

- Keep the advantages of classical DG methods
- Introduce an hybrid variable to decouple local problems defined at the element level
- Solve a reduced linear system for the hybrid variable unknowns only

Complexity: number of globally coupled degrees of freedom

- Classical DG method with \( \mathbb{P}_p \) interpolation
  \[
  (p + 1)(p + 2)(p + 3)N_e, \quad N_e \text{ is the } \# \text{ of elements}
  \]
- HDG method with \( \mathbb{P}_p \) interpolation
  \[
  (p + 1)(p + 2)N_f, \quad N_f \text{ is the } \# \text{ of faces}
  \]
- Continuous finite element formulation based on Nedelec’s first family of face/edge elements in a simplex (tetrahedron):
  \[
  \frac{p(p + 2)(p + 3)}{2}N_e
  \]
Hybridizable DG method in 3D

ANR TEC SER project (May 2014 - April 2017)
Funded by DGA (French armament procurement agency)
http://www-sop.inria.fr/nachos/projects/tecser

- Implementation of HDG for arbitrary high order interpolation
- Local definition (element-wise and face-wise) of the interpolation degree
- Extension of the formulation to a non-conforming hybrid
  hexahedral/tetrahedral mesh
- Scalability improvement
  PDE-based Schwartz domain decomposition algorithm
  PaStiX as a local (subdomain) solver
- Coupling with a BEM for an accurate treatment of far field radiation
- HORSE software
  High Order solver for Radar cross Section Evaluation
Hybridizable DG method in 3D

Scattering of a plane wave by a jet

- Unstructured tetrahedral mesh: 1,645,874 elements / 3,521,251 faces
- Frequency: 600 MHz, Wavelength: $\lambda \simeq 0.5$ m, Penalty parameter: $\tau = 1$

<table>
<thead>
<tr>
<th>HDG method</th>
<th># DoF $\Lambda$ field</th>
<th># DoF EM field</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDG-$P_1$</td>
<td>21,127,506</td>
<td>39,500,976</td>
</tr>
<tr>
<td>HDG-$P_2$</td>
<td>42,255,012</td>
<td>98,752,440</td>
</tr>
<tr>
<td>HDG-$P_3$</td>
<td>70,425,020</td>
<td>197,504,880</td>
</tr>
</tbody>
</table>

Contour line of $|E|$ - HDG$-P_1$ to HDG$-P_3$
Practical example

Study case

- Jet with $P_2$ interpolation
- 42 255 012 unknowns for $\Lambda$
- A single factorization is performed on each subdomain
- There may be several solves for iterative refinement between subdomains

Machine

- Occigen: 24-cores nodes with 128 GB
- Each MPI process is hold on a socket with 12 cores
- In next experiments, we use 32, 48 or 64 subdomains
- Subdomains are ordered accordingly to the number of operations for the complete factorization
Factorization on each subdomain – 32 processes with \( \tau = 1e^{-4} \)

![Factorization stats on each subdomain for jet2/deg2](image)

- **Complete factorization**, moy=183.0, med=174.0, max=341.0
- **Minimal Memory**, moy=151.0, med=135.0, max=260.0
- **Just-In-Time**, moy=74.0, med=73.0, max=119.0
Memory Consumption on each subdomain – \(32\) processes with \(\tau = 1e - 4\)

![Factorization stats on each subdomain for jet2/deg2](image)

Complete factorization, moy=33.0, med=33.0, max=42.0

Minimal Memory, moy=22.0, med=21.0, max=25.0

Just-In-Time, moy=20.0, med=20.0, max=22.0
Impact on HORSE

<table>
<thead>
<tr>
<th>Subs</th>
<th>$\tau$</th>
<th>Method</th>
<th>Fact(s)</th>
<th>Nb. of iters</th>
<th>Refinement (s)</th>
<th>HDGM (s)</th>
<th>Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td></td>
<td>Complete-Factorization</td>
<td>526.0</td>
<td>9</td>
<td>254.5</td>
<td>814.0</td>
<td>41.7</td>
</tr>
<tr>
<td></td>
<td>1e-4</td>
<td>Just-In-Time</td>
<td>209.3</td>
<td>9</td>
<td>164.8</td>
<td>405.8</td>
<td>41.7 (22.3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimal-Memory</td>
<td>516.7</td>
<td>15</td>
<td>273.2</td>
<td>822.0</td>
<td>24.5</td>
</tr>
<tr>
<td></td>
<td>1e-8</td>
<td>Just-In-Time</td>
<td>325.2</td>
<td>9</td>
<td>192.9</td>
<td>550.4</td>
<td>41.7 (29.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimal-Memory</td>
<td>600.1</td>
<td>9</td>
<td>193.2</td>
<td>825.8</td>
<td>30.5</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td>Complete-Factorization</td>
<td>237.3</td>
<td>8</td>
<td>118.6</td>
<td>374.6</td>
<td>24.9</td>
</tr>
<tr>
<td></td>
<td>1e-4</td>
<td>Just-In-Time</td>
<td>112.2</td>
<td>9</td>
<td>106.1</td>
<td>237.1</td>
<td>24.9 (14.1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimal-Memory</td>
<td>229.3</td>
<td>13</td>
<td>159.7</td>
<td>407.5</td>
<td>15.3</td>
</tr>
<tr>
<td></td>
<td>1e-8</td>
<td>Just-In-Time</td>
<td>171.5</td>
<td>8</td>
<td>105.8</td>
<td>296.1</td>
<td>24.9 (18.1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimal-Memory</td>
<td>319.4</td>
<td>8</td>
<td>109.8</td>
<td>447.9</td>
<td>18.8</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>Complete-Factorization</td>
<td>179.8</td>
<td>9</td>
<td>104.7</td>
<td>298.3</td>
<td>17.4</td>
</tr>
<tr>
<td></td>
<td>1e-4</td>
<td>Just-In-Time</td>
<td>79.7</td>
<td>10</td>
<td>91.1</td>
<td>184.1</td>
<td>17.4 (10.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimal-Memory</td>
<td>138.1</td>
<td>13</td>
<td>120.7</td>
<td>272.2</td>
<td>11.0</td>
</tr>
<tr>
<td></td>
<td>1e-8</td>
<td>Just-In-Time</td>
<td>124.6</td>
<td>9</td>
<td>90.0</td>
<td>228.2</td>
<td>17.4 (12.9)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimal-Memory</td>
<td>239.2</td>
<td>9</td>
<td>91.5</td>
<td>344.5</td>
<td>13.7</td>
</tr>
</tbody>
</table>
Conclusion

Block Low-Rank solver

- Exploits the symbolic structure
- Demonstrate 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 runtime systems

PaStiX in HORSE

- Low-rank compression is useful to reduce memory footprint and/or time-to-solution
- Future work: study load imbalance between subdomains
- Future work: impact with $P_3/P_4$ interpolation
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.