Hierarchical Sparse Hybrid Solvers for Manycores Platforms

Emmanuel AGULLO  Luc GIRAUD  Abdou GUERMOUCHE
Stojce NAKOV  Jean ROMAN

Third Workshop of the CNPq-Inria HOSCAR Project, September 2\textsuperscript{nd}-5\textsuperscript{th}, 2013

Inria Bordeaux - Sud-Ouest Research Center, France
1. Introduction

2. Hybrid MPI + threads parallelization

3. Task-based parallelization

4. Conclusion and future work
Goal: solving $Ax = b$, where $A$ is sparse

**Usual trades off**

**Direct**
- Robust/accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large 3D problems

**Iterative**
- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster
Motivations

Goal: solving $Ax = b$, where $A$ is sparse

**Usual trades off**

**Direct**
- Robust/accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large 3D problems

**Iterative**
- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster
Introduction

Motivations

Goal: solving $Ax = b$, where $A$ is sparse

Usual trades off

**Direct**
- Robust/accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large 3D problems

**Iterative**
- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster
Goal: solving $Ax = b$, where $A$ is sparse

Usual trades off

**Direct**
- Robust/accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large 3D problems

**Iterative**
- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster
Goal: solving $Ax = b$, where $A$ is sparse

Usual trades off

**Direct**
- Robust/accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large 3D problems

**Iterative**
- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster
Sparse hybrid (direct/iterative) domain decomposition linear solvers

Partitioning the global matrix

★ Global hybrid decomposition:

$$\mathcal{A} = \begin{pmatrix} \mathcal{A}_{II} & \mathcal{A}_{IG} \\ \mathcal{A}_{GI} & \mathcal{A}_{GG} \end{pmatrix}$$

★ Global Schur complement:

$$\mathcal{S} = \mathcal{A}_{GG} - \mathcal{A}_{GI} \mathcal{A}_{II}^{-1} \mathcal{A}_{GI}$$
Sparse hybrid (direct/iterative) domain decomposition linear solvers

Partitioning the global matrix

- **Global hybrid decomposition:**
  \[
  A = \begin{pmatrix} A_{II} & A_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} \end{pmatrix}
  \]

- **Global Schur complement:**
  \[
  S = A_{\Gamma\Gamma} - A_{\Gamma I} A_{II}^{-1} A_{I\Gamma}
  \]
Sparse hybrid (direct/iterative) domain decomposition linear solvers

Partitioning the global matrix

- Global hybrid decomposition:

\[
A = \begin{pmatrix}
A_{II} & A_{IG} \\
A_{GI} & A_{GG}
\end{pmatrix}
\]

- Global Schur complement:

\[
S = A_{GG} - A_{GI}A_{II}^{-1}A_{IG}
\]
Sparse hybrid (direct/iterative) domain decomposition linear solvers

Global preconditioner based solvers
- HIPS (Inria, HiePACS)
- PDSLin (LBNL, USA)
- ShyLU (Sandia NL)

Local preconditioner based solver
- MaPHyS (Inria, HiePACS)
Sparse hybrid (direct/iterative) domain decomposition linear solvers

Global preconditioner based solvers
- HIPS (Inria, HiePACS)
- PDSLin (LBNL, USA)
- ShyLU (Sandia NL)

Local preconditioner based solver
- MaPHyS (Inria, HiePACS)
Method used in MAPHYs

Factorization of local interiors

★ Global matrix:

\[
A = \begin{pmatrix}
A_{II} & A_{I\Gamma} \\
A_{\Gamma I} & A_{\Gamma\Gamma}
\end{pmatrix}
\]

★ Global Schur Complement:

\[
S^{(i)} = A^{(i)}_{\Gamma\Gamma} - A_{\Gamma I_i} A_{I_i I_i}^{-1} A_{I_i \Gamma_i}
\]
Factorization of local interiors

- Local matrix:

\[ A^{(i)} = \begin{pmatrix} A \mathcal{I}_i \mathcal{I}_i & A \mathcal{I}_i \Gamma_i \\ A \Gamma_i \mathcal{I}_i & A^{(i)} \Gamma \Gamma \end{pmatrix} \]

- Local Schur Complement:

\[ S^{(i)} = A^{(i)} \Gamma \Gamma - A \Gamma_i \mathcal{I}_i A^{-1} \mathcal{I}_i \mathcal{I}_i A \Gamma_i \Gamma_i \]
Constructing of the preconditioner

- **Local Schur Complement:**
  \[ S(i) = A_{ΓΓ}^{(i)} - A_{Γi} A_{Ii}^{-1} A_{Ii} Γ_i \]

- **Algebraic Additive Schwarz Preconditioner**
  \[ M = \sum_{i=1}^{N} R_{Γ_i}^T (\bar{S}(i))^{-1} R_{Γ_i} \]
  where \( \bar{S}(i) \) is obtained from \( S(i) \) via neighbor to neighbor communications.

- **Global Schur complement:**
  \[ S = \sum_{i=1}^{N} R_{Γ_i}^T (S(i)) R_{Γ_i} \]
Software used in MAPHYs

- Partitioning the global matrix in several local matrices
  - SCOTCH [Pellegrini and al.]
  - METIS [G. Karypis and V. Kumar]
- Local factorization
  - MUMPS [P. Amestoy and al.] (with Schur option)
  - PaSTiX [P. Ramet and al.] (with Schur option)
- Constructing of the preconditioner
  - MKL library
- Solving the reduced system
  - CG/GMRES/FGMRES [V. Fraysse and L. Giraud] using MKL library for the reduced system
Software used in MAPHYs

★ Partitioning the global matrix in several local matrices
  ▶ SCOTCH [Pellegrini and al.]
  ▶ METIS [G. Karypis and V. Kumar]

★ Local factorization
  ▶ MUMPS [P. Amestoy and al.] (with Schur option)
  ▶ PASTIX [P. Ramet and al.] (with Schur option)

★ Constructing of the preconditioner
  ▶ MKL library

★ Solving the reduced system
  ▶ CG/GMRES/FGMRES [V. Fraysse and L. Giraud] using MKL library for the reduced system
Software used in MAPHYS

- Partitioning the global matrix in several local matrices
  - SCOTCH [Pellegrini and al.]
  - METIS [G. Karypis and V. Kumar]
- Local factorization
  - MUMPS [P. Amestoy and al.] (with Schur option)
  - PASTIX [P. Ramet and al.] (with Schur option)
- Constructing of the preconditioner
  - MKL library
- Solving the reduced system
  - CG/GMRES/FGMRES [V.Fraysse and L.Giraud] using MKL library for the reduced system
Software used in MAPHYS

- Partitioning the global matrix in several local matrices
  - SCOTCH [Pellegrini and al.]
  - METIS [G. Karypis and V. Kumar]
- Local factorization
  - MUMPS [P. Amestoy and al.] (with Schur option)
  - PASTIX [P. Ramet and al.] (with Schur option)
- Constructing of the preconditioner
  - MKL library
- Solving the reduced system
  - CG/GMRES/FGMRES [V.Fraysse and L.Giraud] using MKL library for the reduced system
Software used in MAPHYS

- Partitioning the global matrix in several local matrices
  - SCOTCH [Pellegrini and al.]
  - METIS [G. Karypis and V. Kumar]
- Local factorization
  - MUMPS [P. Amestoy and al.] (with Schur option)
  - PASTIX [P. Ramet and al.] (with Schur option)
- Constructing of the preconditioner
  - MKL library
- Solving the reduced system
  - CG/GMRES/FGMRES [V. Fraysse and L. Giraud] using MKL library for the reduced system
Software used in MAPHYS

- Partitioning the global matrix in several local matrices
  - SCOTCH [Pellegrini and al.]
  - METIS [G. Karypis and V. Kumar]
- Local factorization
  - MUMPS [P. Amestoy and al.] (with Schur option)
  - PASTIX [P. Ramet and al.] (with Schur option)
- Constructing of the preconditioner
  - MKL library
- Solving the reduced system
  - CG/GMRES/FGMRES [V.Fraysse and L.Giraud] using MKL library for the reduced system
Software used in MAPHYS

★ Partitioning the global matrix in several local matrices
  ▶ **SCOTCH** [Pellegrini and al.]
  ▶ **METIS** [G. Karypis and V. Kumar]
★ Local factorization
  ▶ **MUMPS** [P. Amestoy and al.] (with Schur option)
  ▶ **PASTIX** [P. Ramet and al.] (with Schur option)
★ Constructing of the preconditioner
  ▶ **MKL** library
★ Solving the reduced system
  ▶ **CG/GMRES/FGMRES** [V.Fraysse and L.Giraud] using **MKL** library for the reduced system
Outline

1. Introduction

2. Hybrid MPI + threads parallelization

3. Task-based parallelization

4. Conclusion and future work
Different type of parallelization of the sparse hybrid linear solvers

Global preconditioner based solvers
- HIPS
- PDSL\text{IN}
- SHYLU

Local preconditioner based solver
- MAPHY\text{S}
Different type of parallelization of the sparse hybrid linear solvers

Global preconditioner based solvers

- HIPS
- PDSL\_IN
- \textsc{SHYLU}

Local preconditioner based solver

- \textsc{MAPHYs}

Node 1

Node 2
Different type of parallelization of the sparse hybrid linear solvers

**Global preconditioner based solvers**
- HIPS
- PDSL\textsubscript{IN}
- HYLU

**Local preconditioner based solver**
- MaPH\textsubscript{YS}

Node 1 Node 2

Process MPI Domain

Node 1

Node 2
Different type of parallelization of the sparse hybrid linear solvers

Global preconditioner based solvers
- HIPS
- PDSL\textsc{in}
- \textsc{ShyLU}

Local preconditioner based solver
- \textsc{MapPhys}

Node 1
Node 2
Different type of parallelization of the sparse hybrid linear solvers

Global preconditioner based solvers

- HIPS
- PDSLIN
- HYLU

Local preconditioner based solver

- MAPHYs (before)
Different type of parallelization of the sparse hybrid linear solvers

Global preconditioner based solvers
- HIPS
- PDSLIN
- SHYLU

Local preconditioner based solver
- MAPHYS (this study)
Software used in MAPHYS (before)

Partitioners
- SCOTCH
- METIS

Dense direct solver
- MKL library

Sparse direct solvers
- MUMPS
- PaStiX

Iterative Solvers
- CG/GMRES/FGMRES using MKL library
Software used in MAPHYs (this study)

Dense direct solver
- Multi-threaded MKL library

Sparse direct solvers
- MUMPS
- Multi-threaded PASTIX

Iterative Solvers
- CG/GMRES/FGMRES using multi-threaded MKL library
Software used in MAPHYS (this study)

Dense direct solver
- Multi-threaded MKL library

Sparse direct solvers
- MUMPS
- Multi-threaded PASTIX

Iterative Solvers
- CG/GMRES/FGMRES using multi-threaded MKL library

Challenge
- Composability, Performance
Experimental set up

**Hardware** (on each node)
- Two Quad-core Nehalem Intel® Xeon® X5550
- Memory: 24 GB GDDR3
- Double precision

**Matrices**

<table>
<thead>
<tr>
<th></th>
<th>Haltere</th>
<th>Audikw_1</th>
<th>tdr455k</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>1,288,825</td>
<td>943,695</td>
<td>2,738,556</td>
</tr>
<tr>
<td>nnz</td>
<td>10,476,775</td>
<td>39,297,771</td>
<td>112,756,352</td>
</tr>
<tr>
<td>Precision</td>
<td>Complex</td>
<td>Real</td>
<td>Complex</td>
</tr>
<tr>
<td>Description</td>
<td>Electromagnetic problem</td>
<td>3D unstructured structural mechanics</td>
<td>Electromagnetic problem</td>
</tr>
</tbody>
</table>
Hybrid MPI + threads parallelization

Scalability of MAPHYS on multicore nodes

Node 1
Process MPI
Domain

Node 2

Node 3

Node 4

S. Nakov
Hierarchical Sparse Hybrid Solvers for Manycores Platforms
Hybrid MPI + threads parallelization

Scalability of MAPHYS on multicore nodes

Node 1
Node 2
Node 3
Node 4
Scalability of MAPHYs on multicore nodes
Hybrid MPI + threads parallelization

Scalability of MAPHYS on multicore nodes

Node 1
Process MPI
Thread
Domain

Node 2

Node 3

Node 4
Hybrid MPI + threads parallelization

Scalability of MAPHY’S on multicore nodes

Achieved performance with the Audi matrix on four nodes

All computational steps

Factorization step

Preconditioning step

Solve step

Hierarchical Sparse Hybrid Solvers for Manycores Platforms
Hybrid MPI + threads parallelization

Flexibility to exploit entire multicore nodes

Node 1
Node 2
Node 3
Node 4
Hybrid MPI + threads parallelization

Flexibility to exploit entire multicore nodes

- Process MPI
- Thread
- Domain

Node 1
Node 2
Node 3
Node 4

S. NAKOV

Hierarchical Sparse Hybrid Solvers for Manycores Platforms
Hybrid MPI + threads parallelization

Flexibility to exploit entire multicore nodes

Node 1
Node 2
Node 3
Node 4
Hybrid MPI + threads parallelization

Flexibility to exploit entire multicore nodes

Process MPI

Thread

Domain

Node 1

Node 2

Node 3

Node 4
Hybrid MPI + threads parallelization

Flexibility to exploit entire multicore nodes

Achieved performance with the Haltere matrix on two nodes

All computational steps

Memory used per node
Hybrid MPI + threads parallelization

Achieved performance with the tdr455k matrix on sixteen nodes

All computational steps

Memory used per node
Outline

1. Introduction

2. Hybrid MPI + threads parallelization

3. Task-based parallelization

4. Conclusion and future work
Task-based parallelization

Factorization step

Constructing of the preconditioner

Solve
Task-based parallelization

Constructing of the preconditioner

Factorization step

Solve
Approach: Three-layer paradigm

High-level (task-based) algorithm

Runtime (task-based) System

STARPU, PARSEC, QUARK, SuperMatrix ...

Device kernels
Approach: Three-layer paradigm

High-level (task-based) algorithm

Runtime (task-based) System

*STARPU, PARSEC, QUARK, SuperMatrix ...*

Device kernels
Approach: Three-layer paradigm

High-level (task-based) algorithm

Runtime (task-based) System

Device kernels
## Dense linear algebra libraries
- **PLASMA** *(QUARK)*
- **DPLASMA** *(PARSEC)*
- **MAGMA-MORSE** *(STARPU)*
- **FLAME** *(SUPERMATRIX)*

## Sparse linear algebra libraries
- **PaSTiX** *(STARPU and PARSEC)*
- **qr_mumps** *(STARPU and PARSEC)*
Runtime systems in linear algebra

Dense linear algebra libraries
- PLASMA (QUARK)
- DPLASMA (PARSEC)
- MAGMA-MORSE (STARPU)
- FLAME (SUPERMATRIX)

Sparse linear algebra libraries
- PaSTiX (STARPU and PARSEC)
- qr_mumps (STARPU and PARSEC)
Software used in MAPHYs

Dense direct solver
- MAGMA-MORSE library

Sparse direct solvers
- PaStiX

Iterative Solvers
- CG/GMRES/FGMRES
Software used in MAPHYS

Dense direct solver
- MAGMA-MORSE library with STARPU

Sparse direct solvers
- PaStiX with STARPU

Iterative Solvers
- CG/GMRES/FGMRES
Software used in MAPHYS

Dense direct solver
- MAGMA-MORSE library with STARPU

Sparse direct solvers
- PaStiX with STARPU

Iterative Solvers
- CG/GmRES/FgMRES
Tackifying the CG Algorithm

CG Algorithm

1: \( r \leftarrow b \)
2: \( r \leftarrow r - Ax \)
3: \( p \leftarrow r \)
4: \( \delta_{\text{new}} \leftarrow \text{dot}(r, r) \)
5: \( \delta_{\text{old}} \leftarrow \delta_{\text{new}} \)
6: \( \text{while } \frac{\|b - Ax\|}{\|b\|} \leq \text{eps} \text{ do} \)
7: \( q \leftarrow Ap \)
8: \( \alpha \leftarrow \frac{\delta_{\text{new}}}{\text{dot}(p, q)} \)
9: \( x \leftarrow x + \alpha p \)
10: \( r \leftarrow r - \alpha q \)
11: \( \delta_{\text{new}} \leftarrow \text{dot}(r, r) \)
12: \( \beta \leftarrow \frac{\delta_{\text{new}}}{\delta_{\text{old}}} \)
13: \( \delta_{\text{old}} \leftarrow \delta_{\text{new}} \)
14: \( p \leftarrow r + \beta p \)
15: \( \text{end while} \)
Task-based parallelization

Tackifying the CG Algorithm

CG Algorithm

1: \( r \leftarrow b \)
2: \( r \leftarrow r - Ax \)
3: \( p \leftarrow r \)
4: \( \delta_{\text{new}} \leftarrow \text{dot}(r, r) \)
5: \( \delta_{\text{old}} \leftarrow \delta_{\text{new}} \)
6: while \( \frac{\|b - Ax\|}{\|b\|} \leq \text{eps} \) do
7: \( q \leftarrow Ap \)
8: \( \alpha \leftarrow \delta_{\text{new}} / \text{dot}(p, q) \)
9: \( x \leftarrow x + \alpha p \)
10: \( r \leftarrow r - \alpha q \)
11: \( \delta_{\text{new}} \leftarrow \text{dot}(r, r) \)
12: \( \beta \leftarrow \delta_{\text{new}} / \delta_{\text{old}} \)
13: \( \delta_{\text{old}} \leftarrow \delta_{\text{new}} \)
14: \( p \leftarrow r + \beta p \)
15: end while
Task-based parallelization

Tackifying the CG Algorithm

Main CG loop

1: \textbf{while } \frac{\|b-Ax\|}{\|b\|} \leq \text{eps} \text{ do}
2: \quad q \leftarrow Ap
3: \quad \alpha \leftarrow \delta_{\text{new}} / \text{dot}(p, q)
4: \quad x \leftarrow x + \alpha p
5: \quad r \leftarrow r - \alpha q
6: \quad \delta_{\text{new}} \leftarrow \text{dot}(r, r)
7: \quad \beta \leftarrow \delta_{\text{new}} / \delta_{\text{old}}
8: \quad \delta_{\text{old}} \leftarrow \delta_{\text{new}}
9: \quad p \leftarrow r + \beta p
10: \textbf{end while}
Task-based parallelization

Execution of CG on 3 GPUs

1: while $\frac{\|b-Ax\|}{\|b\|} \leq eps$ do
2:     $q \leftarrow Ap$
3:     $\alpha \leftarrow \delta_{\text{new}} / \text{dot}(p, q)$
4:     $x \leftarrow x + \alpha p$
5:     $r \leftarrow r - \alpha q$
6:     $\delta_{\text{new}} \leftarrow \text{dot}(r, r)$
7:     $\beta \leftarrow \delta_{\text{new}} / \delta_{\text{old}}$
8:     $\delta_{\text{old}} \leftarrow \delta_{\text{new}}$
9:     $p \leftarrow r + \beta p$
10: end while

Trace of one iteration of CG with 3 GPUs.

S. NAKOV
Hierarchical Sparse Hybrid Solvers for Manycores Platforms
Task-based parallelization

Execution of CG on 3 GPUs

1: while $\frac{|b-Ax|}{|b|} \leq \text{eps}$ do
2: \quad $q \leftarrow Ap$
3: \quad $\alpha \leftarrow \delta_{\text{new}} / \text{dot}(p, q)$
4: \quad $x \leftarrow x + \alpha p$
5: \quad $r \leftarrow r - \alpha q$
6: \quad $\delta_{\text{new}} \leftarrow \text{dot}(r, r)$
7: \quad $\beta \leftarrow \delta_{\text{new}} / \delta_{\text{old}}$
8: \quad $\delta_{\text{old}} \leftarrow \delta_{\text{new}}$
9: \quad $p \leftarrow r + \beta p$
10: end while

Trace of one iteration of CG with 3 GPUs after optimizing the task-flow.
Task-based parallelization

Performance

Hardware

NVIDIA Tesla M2070

- Memory: 6 GB GDDR5
- Double precision

<table>
<thead>
<tr>
<th>Matrices</th>
<th>11pts-256-256-256</th>
<th>Audikw_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>16,777,216</td>
<td>943,695</td>
</tr>
<tr>
<td>nnz</td>
<td>350,488,576</td>
<td>154,359,999</td>
</tr>
<tr>
<td>Description</td>
<td>3D PDE on a Cartesian grid</td>
<td>3D unstructured structural mechanics</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Basic CG (Gflop/s)</th>
<th>Optimized CG (Gflop/s)</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1GPU</td>
<td>3GPU</td>
<td>Speed-up</td>
</tr>
<tr>
<td>Audikw_1</td>
<td>9.3</td>
<td>12.97</td>
<td>1.39</td>
</tr>
<tr>
<td>11ptd-256-256-256</td>
<td>5.24</td>
<td>7.5</td>
<td>1.43</td>
</tr>
</tbody>
</table>
Task-based parallelization

Data transfers between GPUs

**GPU-CPU-GPU mechanism**

**GPU-GPU mechanism**

Hierarchical Sparse Hybrid Solvers for Manycores Platforms
1: while $\frac{\|b-Ax\|}{\|b\|} \leq eps$ do
2:    $q \leftarrow Ap$
3:    $\alpha \leftarrow \frac{\delta_{\text{new}}}{\text{dot}(p, q)}$
4:    $x \leftarrow x + \alpha p$
5:    $r \leftarrow r - \alpha q$
6:    $\delta_{\text{new}} \leftarrow \text{dot}(r, r)$
7:    $\beta \leftarrow \frac{\delta_{\text{new}}}{\delta_{\text{old}}}$
8:    $\delta_{\text{old}} \leftarrow \delta_{\text{new}}$
9:    $p \leftarrow r + \beta p$
10: end while

GPU-CPU-GPU communication mechanism
Task-based parallelization

Benefit of using runtime systems

1: while $\frac{\|b-Ax\|}{\|b\|} \leq \text{eps}$ do
2: \hspace{1em} $q \leftarrow Ap$
3: \hspace{1em} $\alpha \leftarrow \delta_{\text{new}}/\dot{\text{dot}}(p, q)$
4: \hspace{1em} $x \leftarrow x + \alpha p$
5: \hspace{1em} $r \leftarrow r - \alpha q$
6: \hspace{1em} $\delta_{\text{new}} \leftarrow \text{dot}(r, r)$
7: \hspace{1em} $\beta \leftarrow \delta_{\text{new}}/\delta_{\text{old}}$
8: \hspace{1em} $\delta_{\text{old}} \leftarrow \delta_{\text{new}}$
9: \hspace{1em} $p \leftarrow r + \beta p$
10: end while

GPU-GPU communication mechanism
1. Introduction

2. Hybrid MPI + threads parallelization

3. Task-based parallelization

4. Conclusion and future work
Conclusion and future work

Conclusion

- Two-level parallelism efficiently exploits multicore architectures
- Synchronizations may be a bottleneck
- Investigating task-based parallelization to pipeline all the steps

Future work

- Full task-based solver
- Further experiments in the collaboration with TOTAL