Memory-Aware Scheduling for Sparse Direct Methods

Emmanuel AGULLO, ICL - University of Tennessee
Abdou GUERMOUCHE, LaBRI, Université de Bordeaux
Jean-Yves L’EXCELLENT, LIP - INRIA

May 15, 2009
Solving sparse linear systems

\[ Ax = b \]

\[ \Rightarrow \text{Direct methods: } A = LU \]

Typical matrix: BRGM matrix

- 3.7 \times 10^6 \text{ variables}
- 156 \times 10^6 \text{ non zeros in } A
- 4.5 \times 10^9 \text{ non zeros in } LU
- 26.5 \times 10^{12} \text{ flops}

Hardware paradigm

- Many-core architecture.
- Large global amount of memory.
- Limited memory per core.

Software challenge

- Need for algorithms whose memory usage scales with the number of processors.
- Case study: MUMPS
Solving sparse linear systems

\[ Ax = b \]

⇒ Direct methods: \( A = LU \)

Typical matrix: BRGM matrix

- 3.7 × 10^6 variables
- 156 × 10^6 non zeros in \( A \)
- 4.5 × 10^9 non zeros in \( LU \)
- 26.5 × 10^{12} \text{ flops}

Hardware paradigm

- Many-core architecture.
- Large global amount of memory.
- Limited memory per core.

Software challenge

- Need for algorithms whose memory usage scales with the number of processors.
- Case study: MUMPS
Solving sparse linear systems

\[ Ax = b \]

⇒ Direct methods: \[ A = LU \]

Typical matrix: BRGM matrix

- \[ 3.7 \times 10^6 \] variables
- \[ 156 \times 10^6 \] non zeros in \( A \)
- \[ 4.5 \times 10^9 \] non zeros in \( LU \)
- \[ 26.5 \times 10^{12} \] flops

Hardware paradigm

- Many-core architecture.
- Large global amount of memory.
- Limited memory per core.

Software challenge

→ Need for algorithms whose memory usage scales with the number of processors.

- Case study: MUMPS
Solving sparse linear systems

\[ Ax = b \]

⇒ Direct methods: \( A = LU \)

Typical matrix: BRGM matrix

- \( 3.7 \times 10^6 \) variables
- \( 156 \times 10^6 \) non zeros in \( A \)
- \( 4.5 \times 10^9 \) non zeros in \( LU \)
- \( 26.5 \times 10^{12} \) flops

Hardware paradigm

- Many-core architecture.
- Large global amount of memory.
- Limited memory per core.

Software challenge

→ Need for algorithms whose memory usage scales with the number of processors.

- Case study: MUMPS
Outline

1. Multifrontal method

2. Limits to memory scalability

3. A new memory-aware algorithm

4. Preliminary results

5. Conclusion
Outline

1. Multifrontal method
2. Limits to memory scalability
3. A new memory-aware algorithm
4. Preliminary results
5. Conclusion
The multifrontal method (Duff, Reid’83)

Storage divided into two parts:

- **Factors** *systematically* written to disk;
- **Active Storage** kept in memory.

**Factors**

- **Factors Stack of contribution blocks**
- **Active frontal matrix**

**Active Storage**

**Elimination tree**
The multifrontal method (Duff, Reid’83)

Storage divided into two parts:
- Factors *systematically* written to disk;
- Active Storage kept in memory.

Factors

Active frontal matrix

Stack of contribution blocks

Active Storage

Elimination tree

Factors

Contribution block

Memory-Aware Scheduling for Sparse Direct Methods
The multifrontal method (Duff, Reid’83)

Storage divided into two parts:
- Factors *systematically* written to disk;
- Active Storage kept in memory.

Factors

Active frontal matrix

Stack of contribution blocks

Active Storage

Elimination tree
The multifrontal method (Duff, Reid’83)

Storage divided into two parts:

★ Factors *systematically* written to disk;

★ Active Storage kept in memory.

Elimination tree
The multifrontal method (Duff, Reid’83)

Storage divided into two parts:

- Factors *systematically* written to disk;
- Active Storage kept in memory.
The multifrontal method (Duff, Reid’83)

Storage divided into two parts:
- Factors \textit{systematically} written to disk;
- Active Storage kept in memory.

\begin{itemize}
  \item Factors \textit{systematically} written to disk;
  \item Active Storage kept in memory.
\end{itemize}
The multifrontal method (Duff, Reid’83)

Storage divided into two parts:

- Factors *systematically* written to disk;
- Active Storage kept in memory.

Elimination tree
Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)

Multifrontal method

AGULLO - GUERMOCHE - L'EXCELLENT

Memory-Aware Scheduling for Sparse Direct Methods
Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)
Multifrontal method

Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)
Sequential case results

Figure: Impact of the tree traversal on the memory behavior.

→ Algorithms to find the optimal tree traversal have been proposed.
Sequential case results

Figure: Impact of the tree traversal on the memory behavior.

→ Algorithms to find the optimal tree traversal have been proposed
Definition: **Memory Efficiency on** $p$ processors (or cores)

$$e(p) = \frac{S_{seq}}{p \times S_{max}(p)}, \quad S_{seq}: \text{serial storage}, \quad S_{max}: \text{parallel storage}$$

Results: Memory Efficiency of **MUMPS** (with factors on disk)

<table>
<thead>
<tr>
<th>Number $p$ of processors</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AUDI_KW_1</strong></td>
<td>0.16</td>
<td>0.12</td>
<td>0.13</td>
<td>0.10</td>
</tr>
<tr>
<td><strong>CONESHL_MOD</strong></td>
<td>0.28</td>
<td>0.28</td>
<td>0.22</td>
<td>0.19</td>
</tr>
<tr>
<td><strong>CONV3D64</strong></td>
<td>0.42</td>
<td>0.40</td>
<td>0.41</td>
<td>0.37</td>
</tr>
<tr>
<td><strong>QIMONDA07</strong></td>
<td>0.30</td>
<td>0.18</td>
<td>0.11</td>
<td>-</td>
</tr>
<tr>
<td><strong>ULTRASOUND80</strong></td>
<td>0.32</td>
<td>0.31</td>
<td>0.30</td>
<td>0.26</td>
</tr>
</tbody>
</table>
1. Multifrontal method

2. Limits to memory scalability

3. A new memory-aware algorithm

4. Preliminary results

5. Conclusion
Parallel multifrontal scheme

- **Type 1**: Nodes processed on a single processor
- **Type 2**: Nodes processed with a parallel 1D blocked factorization
- **Type 3**: Parallel 2D cyclic factorization (root node)
Parallel multifrontal scheme

- **Type 1**: Nodes processed on a single processor
- **Type 2**: Nodes processed with a parallel 1D blocked factorization
- **Type 3**: Parallel 2D cyclic factorization (root node)
* Many simultaneous active tasks;
* Large master tasks;
* Large subtrees;
* Proportional mapping.
Many simultaneous active tasks;
Large master tasks;
Large subtrees;
Proportional mapping.
Many simultaneous active tasks;
Large master tasks;
Large subtrees;
Proportional mapping.
Elimination tree:

Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees.

Advantages and drawbacks:
Proportional mapping:

Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees.
Proportional mapping VS postorder traversal (1/2)

Proportional mapping:

Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees.

Advantages and drawbacks

- Tree-level + task-level parallelism;
- Bad memory efficiency.
Proportional mapping VS postorder traversal (1/2)

Proportional mapping:

Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees.

Advantages and drawbacks

- Tree-level + task-level parallelism;
- Bad memory efficiency.
Proportional mapping VS postorder traversal (1/2)

Proportional mapping:

Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees.

Advantages and drawbacks

- Tree-level + task-level parallelism;
- Bad memory efficiency.
Proportional mapping:

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees.

Advantages and drawbacks:
- Tree-level + task-level parallelism;
- Bad memory efficiency.
Elimination tree:

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks:
Proportional mapping VS postorder traversal (2/2)

Postorder traversal:

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks:
- Only task-level parallelism;
- High memory efficiency.
Traversals

- Postorder traversal, node by node;
- All processors on each node.
Limits to memory scalability

Proportional mapping VS postorder traversal (2/2)

Postorder traversal:

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks

- Only task-level parallelism;
- High memory efficiency.
Postorder traversal:

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks:

- Only task-level parallelism;
- High memory efficiency.
Proportional mapping VS postorder traversal (2/2)

Postorder traversal:

Traversal

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks

- Only task-level parallelism;
- High memory efficiency.
Proportional mapping VS postorder traversal (2/2)

Postorder traversal:

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks:
- Only task-level parallelism;
- High memory efficiency.
Postorder traversal:

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks:

- Only task-level parallelism;
- High memory efficiency.
Postorder traversal:

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks:

- Only task-level parallelism;
- High memory efficiency.
Proportional mapping VS postorder traversal (2/2)

Postorder traversal:

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks:
- Only task-level parallelism;
- High memory efficiency.
Outline

1. Multifrontal method

2. Limits to memory scalability

3. A new memory-aware algorithm

4. Preliminary results

5. Conclusion
A new memory-aware algorithm

Memory-aware mapping algorithm

Elimination tree:

Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees if memory allows for it.
A new memory-aware algorithm

Memory-aware mapping algorithm

Memory-aware mapping:

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees if memory allows for it.
Memory-aware mapping:

512

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees if memory allows for it.
A new memory-aware algorithm

Memory-aware mapping algorithm

Memory-aware mapping:

```
256  256
  512  512
  256  256
```

Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees if memory allows for it.
A new memory-aware algorithm

Memory-aware mapping algorithm

Memory-aware mapping:

Advantages

- Robust: guaranteed (if memory $M_0 < \frac{S_{seq}}{p}$).
- Efficient: available memory provides tree-level parallelism.
Outline

1. Multifrontal method
2. Limits to memory scalability
3. A new memory-aware algorithm
4. Preliminary results
5. Conclusion
Solution of large sparse linear systems with:

- Symmetric positive definite matrices;
- General symmetric matrices;
- General unsymmetric matrices.

Implementation

- Distributed Multifrontal Solver (F90, MPI based);
- Dynamic Distributed Scheduling;
- Use of BLAS, BLACS, ScaLAPACK.

Interfaces

Preliminary results

★ Excellent memory scalability:
  ▶ memory efficiency close to 1.
★ Competitive (time) efficiency
  ▶ close to proportional mapping (if enough memory);
  ▶ memory provides tree-level parallelism:

![Diagram showing memory scalability and efficiency](image-url)
Outline

1. Multifrontal method
2. Limits to memory scalability
3. A new memory-aware algorithm
4. Preliminary results
5. Conclusion
Prototype of a *memory-aware* algorithm

- Maximizes the amount of tree-level parallelism with respect to the amount of memory available per processor/core.
- New static mapping implemented, with constraints on dynamic schedulers; experimented within the OOC version of MUMPS.
- Very good memory scalability obtained.

On-going work

- Further tuning and validation.
- Generalization to the in-core case.
- Reinject dynamic information to schedulers.