Memory-Aware Scheduling for Sparse Direct Methods

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

MS31 Parallel Sparse Matrix Computations and Enabling Algorithms
SIAM CSE 2009, Miami, FL, March 2-6, 2009
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

\[ \mathbf{A}x = b \]

⇒ Direct methods: \( \mathbf{A} = LU \)

Typical matrix: BRGM matrix

- 3.7 × 10^6 variables
- 156 × 10^6 non zeros in \( \mathbf{A} \)
- 4.5 × 10^9 non zeros in \( \mathbf{LU} \)
- 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 × 10^6 variables
- 156 × 10^6 non zeros in A
- 4.5 × 10^9 non zeros in LU
- 26.5 × 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: MUMP S
1. MUMPS

2. Limits to memory scalability

3. A new memory-aware algorithm

4. Preliminary results

5. Conclusion
Outline

1. MUMPS

2. Limits to memory scalability

3. A new memory-aware algorithm

4. Preliminary results

5. Conclusion
MUMPS: a MUltifrontal Massively Parallel sparse direct Solver

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
The multifrontal method (Duff, Reid’83)

Storage divided into two parts:

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

Factors

<table>
<thead>
<tr>
<th>Factors</th>
<th>Active frontal matrix</th>
<th>Stack of contribution blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Elimination tree

Active Storage
Storage divided into two parts:

★ Factors \textit{systematically} written to disk;
★ Active Storage kept in memory.
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.

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

Elimination tree

```
Factors

Factors Stack of
contribution
blocks
Active frontal
matrix
Active Storage

Contribution block
```
The multifrontal method (Duff, Reid’83)

Storage divided into two parts:

- Factors *systematically* written to disk;
- Active Storage *kept in memory.*
Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)
Memory behaviour (serial postorder traversal)
Definition: Memory Efficiency on $p$ processors (or cores)

$$e(p) = \frac{S_{seq}}{p \times S_{max}(p)},$$

$S_{seq}$: serial storage, $S_{max}$: parallel storage

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

<table>
<thead>
<tr>
<th>Number of processors $p$</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>
Outline

1. MUMPS

2. Limits to memory scalability

3. A new memory-aware algorithm

4. Preliminary results

5. Conclusion
Limits to memory scalability

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)
Limits to memory scalability

- 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.
Limits to memory scalability

Proportional mapping VS postorder traversal (1/2)

Elimination tree:

```
\[\begin{array}{c}
\text{d=0} \\
\text{d=1} \\
\text{d=2} \\
\text{d=3} \\
\text{d=4} \\
\end{array}\]
```

Mapping

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

Advantages and drawbacks:

- Fine-grain + coarse-grain parallelism;
- Bad memory efficiency.
Proportional mapping VS postorder traversal (1/2)

Proportional mapping:

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

Advantages and drawbacks:
- Fine-grain + coarse-grain parallelism; 
- Bad memory efficiency.
Proportional mapping VS postorder traversal (1/2)

Proportional mapping:

512

Mapping

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

Advantages and drawbacks

- Fine-grain + coarse-grain 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

- Fine-grain + coarse-grain parallelism;
- Bad memory efficiency.
Limits to memory scalability

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

- Fine-grain + coarse-grain parallelism;
- Bad memory efficiency.
Proportional mapping:

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

Advantages and drawbacks:

- Fine-grain + coarse-grain parallelism;
- Bad memory efficiency.
Proportional mapping VS postorder traversal (2/2)

Elimination tree:

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

Advantages and drawbacks:
- Only fine-grain parallelism;
- High memory efficiency.
Proportional mapping VS postorder traversal (2/2)

Postorder traversal:

- d=0
- d=1
- d=2
- d=3
- d=4

Traversal

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

Advantages and drawbacks

/ Only fine-grain 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 fine-grain parallelism;
- High memory efficiency.
Postorder traversal:

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

Advantages and drawbacks:
- Only fine-grain parallelism;
- High memory efficiency.
Postorder traversal:

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

Advantages and drawbacks:

- Only fine-grain 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 fine-grain 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 fine-grain parallelism;
© High memory efficiency.
Postorder traversal:

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

Advantages and drawbacks:

- Only fine-grain parallelism;
- High memory efficiency.
Postorder traversal:

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

Advantages and drawbacks:

- Only fine-grain 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 fine-grain parallelism;
- High memory efficiency.
Outline

1. MUMPS

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:

```
    d=0
    d=1
    d=2
    d=3
    d=4
```

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:

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:

512

\[ d=0 \]
\[ d=1 \]
\[ d=2 \]
\[ d=3 \]
\[ d=4 \]

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:

Mapping

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

Advantages

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

1. MUMPS

2. Limits to memory scalability

3. A new memory-aware algorithm

4. Preliminary results

5. Conclusion
Preliminary results

★ Excellent memory scalability:
  ▶ memory efficiency closed to 1.
★ Competitive (time) efficiency
  ▶ closed to proportional mapping (if enough memory);
  ▶ memory provides coarse-grain parallelism:

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

1. MUMPS

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 coarse-grain 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.