Constructing Memory-minimizing Schedules for Multifrontal Methods

A. Guermouche\textsuperscript{1} \hspace{1cm} J.-Y. L’Excellent\textsuperscript{2}

\textsuperscript{1}INRIA and ENSEEIHT-IRIT
Toulouse, France

\textsuperscript{2}INRIA and LIP-ENS Lyon
Lyon, France

CSC05
Outline

Multifrontal method
   Memory behaviour

Active memory minimization Algorithm (Liu’s Algorithm)
   Limitation of the approach

New multifrontal schedules and algorithms
   Flexible allocation scheme
   A new memory minimization algorithm

Results

Total memory minimization

Conclusion
The multifrontal method (Duff, Reid’83)

Memory divided into two parts:

- Active memory
- Factors

Dependency tree
Outline

Multifrontal method
  Memory behaviour

Active memory minimization Algorithm (Liu’s Algorithm)
  Limitation of the approach

New multifrontal schedules and algorithms
  Flexible allocation scheme
  A new memory minimization algorithm

Results

Total memory minimization

Conclusion
Sequential case: Memory behaviour (1/2)

Figure: Example illustrating the memory behaviour.
Sequential case: Memory behaviour (1/2)

Figure: Example illustrating the memory behaviour.
Sequential case: Memory behaviour (1/2)

Figure: Example illustrating the memory behaviour.
Sequential case: Memory behaviour (1/2)

Figure: Example illustrating the memory behaviour.
Sequential case: Memory behaviour (1/2)

Figure: Example illustrating the memory behaviour.
Sequential case: Memory behaviour (1/2)

Figure: Example illustrating the memory behaviour.
Figure: Example illustrating the memory behaviour.
Sequential case: Memory behaviour (1/2)

Figure: Example illustrating the memory behaviour.
Consider a parent node in the tree:

- $n$ is the number of children.
- $j$ denotes the $j^{th}$ child of the node.
- $cb_j$ is the size of the contribution block of child $j$.
- $m$ is the memory size of the frontal matrix of the parent.
- $A$ (resp. $A_j$) is the amount of active memory needed to process the parent (resp. child $j$).

The assembly step requires a storage:

$$m + \sum_{j=1}^{n} cb_j$$
Consider a parent node in the tree:

- \( n \) is the number of children.
- \( j \) denotes the \( j^{th} \) child of the node.
- \( cb_j \) is the size of the contribution block of child \( j \).
- \( m \) is the memory size of the frontal matrix of the parent.
- \( A \) (resp. \( A_j \)) is the amount of active memory needed to process the parent (resp. child \( j \)).

The storage required to process child \( j \) is:

\[
A_j + \sum_{k=1}^{j-1} cb_k
\]
Consider a parent node in the tree:

- $n$ is the number of children.
- $j$ denotes the $j^{th}$ child of the node.
- $cb_j$ is the size of the contribution block of child $j$.
- $m$ is the memory size of the frontal matrix of the parent.
- $A$ (resp. $A_j$) is the amount of active memory needed to process the parent (resp. child $j$).

$A$ is thus defined by:

$$A = \max(\max_{j=1,n}(A_j + \sum_{k=1}^{j-1} cb_k), m + \sum_{j=1}^{n} cb_j)$$
Impact of the tree traversal

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

→ GOAL: Find the best tree traversal in terms of memory occupation.
Impact of the tree traversal

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

→ **GOAL:** Find the best tree traversal in terms of memory occupation
Liu’s Theorem (Tree pebbling theorem)

The minimum of \( \max_j (x_j + \sum_{i=1}^{j-1} y_i) \) is obtained when the sequence \((x_i, y_i)\) is sorted in decreasing order of \(x_i - y_i\).

Consequence:
An optimal child sequence is obtained by rearranging the children nodes in decreasing order of \(A_i - cb_i\).

Algorithm:
- Bottom-up greedy process.
- Apply Liu’s theorem at each level of the tree.
Outline

Multifrontal method
Memory behaviour

Active memory minimization Algorithm (Liu’s Algorithm)
Limitation of the approach

New multifrontal schedules and algorithms
Flexible allocation scheme
A new memory minimization algorithm

Results

Total memory minimization

Conclusion
Limitation of the Classical scheme

Classical approach.

Flexible scheme.

→ Decoupling the allocation and the computations can improve the memory behaviour
Limitation of the Classical scheme

Classical approach.

Flexible scheme.

→ Decoupling the allocation and the computations can improve the memory behaviour
Outline

Multifrontal method
Memory behaviour

Active memory minimization Algorithm (Liu’s Algorithm)
Limitation of the approach

New multifrontal schedules and algorithms
Flexible allocation scheme
A new memory minimization algorithm

Results

Total memory minimization

Conclusion
Flexible multifrontal scheme

- $p$ is the position of the allocation of the parent.
- $S_1$ is the set of children treated before the allocation of the parent.
- $S_2$ is the set of children treated after the allocation of the parent.

- The memory behaviour inside $S_1$ is similar to the case of the classical multifrontal scheme.
- Inside $S_2$, the order of the children has no impact on the memory behaviour.

$$A^{\text{flex}} = \max \left( \max_{j=1,p} \left( A_j^{\text{flex}} + \sum_{k=1}^{j-1} cb_k \right), \ m + \sum_{k=1}^{p} cb_k, \ m + \max_{j=p+1,n} A_j^{\text{flex}} \right)$$
Flexible multifrontal scheme

- $p$ is the position of the allocation of the parent.
- $S_1$ is the set of children treated before the allocation of the parent.
- $S_2$ is the set of children treated after the allocation of the parent.

- The memory behaviour inside $S_1$ is similar to the case of the classical multifrontal scheme.
- Inside $S_2$, the order of the children has no impact on the memory behaviour.

\[
A^{\text{flex}} = \max \left( \max_{j=1,p} (A_j^{\text{flex}} + \sum_{k=1}^{j-1} cb_k), m + \sum_{k=1}^{p} cb_k, m + \max_{j=p+1,n} A_j^{\text{flex}} \right)
\]
Flexible multifrontal scheme

- $p$ is the position of the allocation of the parent.
- $S_1$ is the set of children treated before the allocation of the parent.
- $S_2$ is the set of children treated after the allocation of the parent.
- The memory behaviour inside $S_1$ is similar to the case of the classical multifrontal scheme.
- Inside $S_2$, the order of the children has no impact on the memory behaviour.

\[
A^{\text{flex}} = \max \left( \max_{j=1,p} (A_j^{\text{flex}} + \sum_{k=1}^{j-1} cb_k), m + \sum_{k=1}^{p} cb_k, m + \max_{j=p+1,n} A_j^{\text{flex}} \right)
\]
Outline

Multifrontal method
Memory behaviour

Active memory minimization Algorithm (Liu’s Algorithm)
Limitation of the approach

New multifrontal schedules and algorithms
Flexible allocation scheme
A new memory minimization algorithm

Results

Total memory minimization

Conclusion
A new memory minimization algorithm

Theorem

An optimal sequence can be obtained by:

- Sorting the children in decreasing order of $A_j^{\text{flex}}$.
- Trying all the possible positions for the allocation of the parent and sorting the children belonging to $S_1$ according to Liu’s Theorem.
- Selecting the configuration that gives the smallest peak.

Algorithm:
Bottom-up greedy process where the theorem is applied at each level of the tree.
Proof

\[ A^{\text{flex}} = \max \left( \max_{j=1,p} (A_j^{\text{flex}} + \sum_{k=1}^{j-1} cb_k), m + \sum_{k=1}^{p} cb_k, m + \max_{j=p+1,n} A_j^{\text{flex}} \right) \]

- Inside \( S_2 \), the order of the children has no impact on the memory behaviour.
- If \( \exists j \in S_1 \mid A_j^{\text{flex}} \leq \max_{i \in S_2} (A_i^{\text{flex}}) \rightarrow j \) can be moved from \( S_1 \) to \( S_2 \) without increasing the peak.

Optimal configuration
Active memory minimization Algorithm

Algorithm:

Set \(S_1 = \{1, \ldots, n\}\), \(S_2 = \emptyset\) and \(p = n\);
Find the schedule providing an optimal \(A^\text{flex}\) value for partition \((S_1, S_2)\);

repeat

Find \(j\) such that \(A^\text{flex}_j = \min_{k \in S_1} A^\text{flex}_k\);
Set \(S_1 = S_1 \setminus \{j\}\), \(S_2 = S_2 \cup \{j\}\), and \(p = p - 1\);
Find the schedule providing an optimal \(A'^\text{flex}\) value for partition \((S_1, S_2)\);

if \(A'^\text{flex} \leq A^\text{flex}\) then

Keep the value of \(p\), and the schedule of children in \(S_1\) and \(S_2\) corresponding to \(A'^\text{flex}\);
Set \(A^\text{flex} = A'^\text{flex}\);

end if

until \(p == 1\) or \(A'^\text{flex} > A^\text{flex}\)
Experimental environment

MUMPS: Multifrontal Parallel Solver for both $LU$ and $LDL^T$.
Reordering techniques: $AMD$, $AMF$, $METIS$, $PORD$.
Test platform: $IBM$ platform at $IDRIS$.
Test problems: Large range of matrices extracted from various collections (Rutherford-Boeing, University of Florida or PARASOL, ...).

Schedules tested:

- Classical multifrontal scheme (parent allocated after all its children).
- Anticipated parent allocation scheme (parent allocated after its first child).
- Flexible parent allocation scheme (parent allocated at an arbitrary position).

Simulation of memory variations for all the schedules during the analysis step.
Experimental environment

**MUMPS:** Multifrontal Parallel Solver for both $LU$ and $LDL^T$.

**Reordering techniques:** AMD, AMF, METIS, PORD.

**Test platform:** IBM platform at IDRIS.

**Test problems:** Large range of matrices extracted from various collections (Rutherford-Boeing, University of Florida or PARASOL, ...).

**Schedules tested:**

- Classical multifrontal scheme (parent allocated after all its children).
- Anticipated parent allocation scheme (parent allocated after its first child).
- Flexible parent allocation scheme (parent allocated at an arbitrary position).

Simulation of memory variations for all the schedules during the analysis step.
Experimental results

Large gains against the classical allocation scheme for matrices 8, 9 and 10.

A. Guermouche, J.-Y. L'Excellent

Memory-minimizing Schedules for Multifrontal Methods
Experimental results

Figure: Active memory ratios.

Large gains against the classical allocation scheme for matrices 8, 9 and 10.

A. Guermouche, J.-Y. L'Excellent

Memory-minimizing Schedules for Multifrontal Methods
Total memory minimization (1/3)

Memory space $T^{\text{flex}}$ needed for the processing of a node in the tree is given by:

$$\mathcal{P}_1 = \max \left( \max_{j=1,p} (T_j^{\text{flex}} + \sum_{k=1}^{j-1} (cb_k + F_k)), m + \sum_{k=1}^{p} (cb_k + F_k) \right)$$

$$\mathcal{P}_2 = \max \left( m + \sum_{k=1}^{p} F_k + \max (T_{j}^{\text{flex}} + \sum_{k=p+1,n}^{j-1} F_k) \right)$$

$$T^{\text{flex}} = \max (\mathcal{P}_1, \mathcal{P}_2).$$

The order in $S_2$ has an impact on the memory occupation.
Total memory minimization (1/3)

Memory space $T^{flex}$ needed for the processing of a node in the tree is given by:

$$P_1 = \max \left( \max_{j=1,p} \left( T_j^{flex} + \sum_{k=1}^{j-1} (c b_k + F_k) \right), \right.$$

$$m + \sum_{k=1}^{p} (c b_k + F_k) \right),$$

$$P_2 = \max \left( m + \sum_{k=1}^{p} F_k + \max_{j=p+1,n} \left( T_j^{flex} + \sum_{k=p+1}^{j-1} F_k \right) \right).$$

$$T^{flex} = \max(P_1, P_2).$$

The order in $S_2$ has an impact on the memory occupation.
Total memory minimization (1/3)

Memory space $T^{\text{flex}}$ needed for the processing of a node in the tree is given by:

\[
\mathcal{P}_1 = \max \left( \max_{j=1, p} \left( T^{\text{flex}}_j + \sum_{k=1}^{j-1} (c_b_k + F_k) \right), m + \sum_{k=1}^{p} (c_b_k + F_k) \right)
\]

\[
\mathcal{P}_2 = \max \left( m + \sum_{k=1}^{p} F_k + \max_{j=p+1, n} \left( T^{\text{flex}}_j + \sum_{k=p+1}^{j-1} F_k \right) \right)
\]

\[
T^{\text{flex}} = \max (\mathcal{P}_1, \mathcal{P}_2).
\]

The order in $S_2$ has an impact on the memory occupation.
Total memory minimization (1/3)

Memory space $T^{flex}$ needed for the processing of a node in the tree is given by:

$$\mathcal{P}_1 = \max\left( \max_{j=1,p} (T_{j}^{flex} + \sum_{k=1}^{j-1} (cb_k + F_k) ), \right. $$

$$\left. m + \sum_{k=1}^{p} (cb_k + F_k) \right)$$

$$\mathcal{P}_2 = \max\left( m + \sum_{k=1}^{p} F_k + \max_{j=p+1,n} (T_{j}^{flex} + \sum_{k=p+1}^{j-1} F_k) \right)$$

$$T^{flex} = \max(\mathcal{P}_1, \mathcal{P}_2).$$

The order in $S_2$ has an impact on the memory occupation.
Total memory minimization (1/3)

Memory space $T^{\text{flex}}$ needed for the processing of a node in the tree is given by:

$$\mathcal{P}_1 = \max\left( \max_{j=1,p} (T_j^{\text{flex}} + \sum_{k=1}^{j-1} (cb_k + F_k)), \right)$$

$$m + \sum_{k=1}^{p} (cb_k + F_k)$$

$$\mathcal{P}_2 = \max\left( m + \sum_{k=1}^{p} F_k + \max_{j=p+1,n} (T_j^{\text{flex}} + \sum_{k=p+1}^{j-1} F_k) \right)$$

$$T^{\text{flex}} = \max(\mathcal{P}_1, \mathcal{P}_2).$$

The order in $S_2$ has an impact on the memory occupation.
Total memory minimization (2/3)

<table>
<thead>
<tr>
<th>Children sequence</th>
<th>$S_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T^\text{flex}_i - (cb_i + F_i)$</td>
<td>$T^\text{flex}_i - F_i$</td>
</tr>
</tbody>
</table>

Total memory minimizing sequences inside $S_1$ and $S_2$.

Property:

\[ \text{let } j_0 \in S_2 \text{ be the child for which the peak is reached inside } S_2. \]

\[ \rightarrow \text{ The total memory peak cannot decrease if } j_0 \text{ remains in } S_2 \text{ for all configurations where } S_1 \subset S'_1. \]
Total memory minimization (2/3)

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Children sequence</td>
<td>$T_{i}^{\text{flex}} - (cb_i + F_i)$</td>
<td>$T_{i}^{\text{flex}} - F_i$</td>
</tr>
</tbody>
</table>

Total memory minimizing sequences inside $S_1$ and $S_2$.

Property:

let $j_0 \in S_2$ be the child for which the peak is reached inside $S_2$.

→ The total memory peak cannot decrease if $j_0$ remains in $S_2$ for all configurations where $S_1 \subset S_1'$. 
Total memory minimization (2/3)

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Children sequence</td>
<td>$T_i^{flex} - (cb_i + F_i)$</td>
<td>$T_i^{flex} - F_i$</td>
</tr>
</tbody>
</table>

Total memory minimizing sequences inside $S_1$ and $S_2$.

**Property:**

Let $j_0 \in S_2$ be the child for which the peak is reached inside $S_2$.

$\rightarrow$ The total memory peak cannot decrease if $j_0$ remains in $S_2$ for all configurations where $S_1 \subset S'_1$. 

A. Guermouche, J.-Y. L'Excellent

Memory-minimizing Schedules for Multifrontal Methods 20/23
Total memory minimization (2/3)

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Children sequence</td>
<td>$T_i^{\text{flex}} - (c b_i + F_i)$</td>
<td>$T_i^{\text{flex}} - F_i$</td>
</tr>
</tbody>
</table>

Total memory minimizing sequences inside $S_1$ and $S_2$.

Property:

let $j_0 \in S_2$ be the child for which the peak is reached inside $S_2$. → The total memory peak cannot decrease if $j_0$ remains in $S_2$ for all configurations where $S_1 \subset S'_1$. 
Algorithm:

Set $S_1 = \emptyset$, $S_2 = \{1, \ldots, n\}$ and $p = 0$;
Sort $S_2$ according in decreasing order of $T_j^{\text{flex}} - F_j$; Compute $T^{\text{flex}} = P_2$;

repeat

Find $j_0$ such that the peak in $P_2$ is obtained for $j_0$;
Set $S_1 = S_1 \cup \{j_0\}$, $S_2 = S_2 \setminus \{j_0\}$, and $p = p + 1$;
(Remark: $j_0$ is inserted at the position in $S_1$ so that the order inside this set is decreasing in terms of $T_j^{\text{flex}} - (cb_j + F_j)$.)

Compute $P_1$, $P_2$, and $T'^{\text{flex}} = \max(P_1, P_2)$;

if $T'^{\text{flex}} \leq T^{\text{flex}}$ then

Keep the values of $p$, $S_1$ and $S_2$ and set $T^{\text{flex}} = T'^{\text{flex}}$;

end if

until $p = n$ or $P_1 \geq P_2$
Experimental results

Figure: Total memory ratios.
Conclusion and Future work

- Flexible multifrontal scheme and corresponding memory minimization algorithms proposed.
  - Active memory and total memory cases considered.
  - In-place assembly of the last contribution block also considered.

Future work:

- Real-life implementation (modification of the factorization).
- Pivoting management (how to deal with pivoting).
- Extension to the parallel case:
  - Add fictive nodes to assemble the distributed contribution blocks?
  - Preallocate parent nodes?
- Out-of-core context:
  - Design I/O volume minimization algorithms using the flexible multifrontal scheme (find a trade-off between the size of memory and the I/O volume).