Constructing Memory-minimizing Schedules for Multifrontal Methods

A. Guermouche[1] J.-Y. L'Excellent[2]

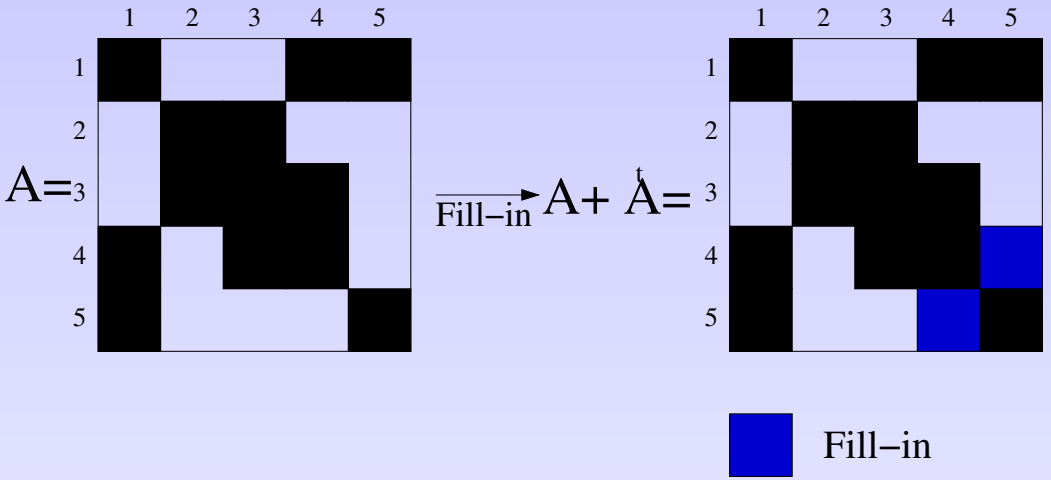
[1]INRIA and ENSEEIHT-IRIT
Toulouse, France

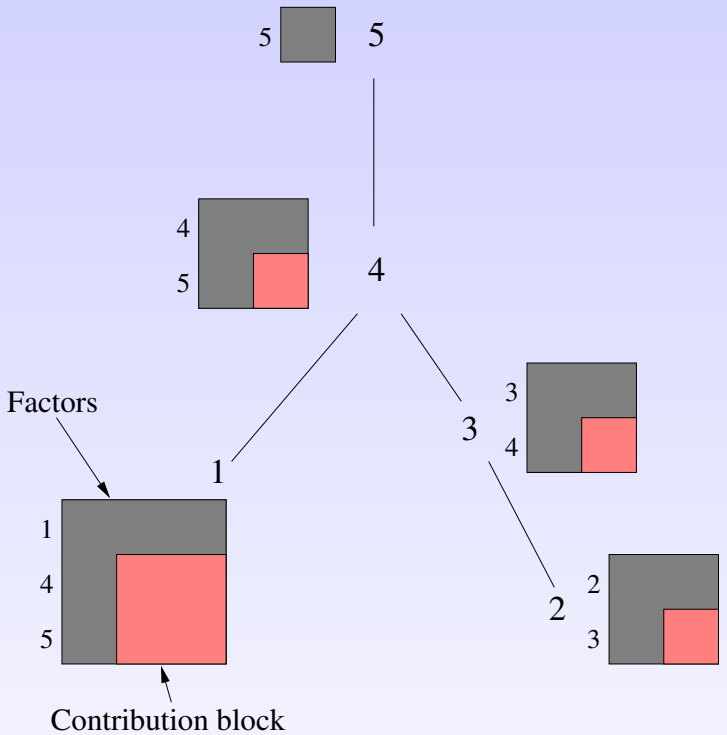
[2]INRIA and LIP-ENS Lyon
Lyon, France

CSC05

Outline

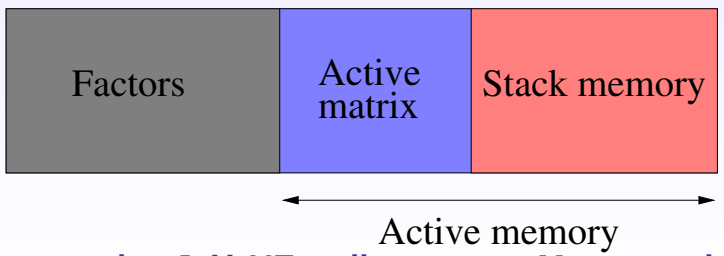
The multifrontal method (Duff, Reid'83)





Memory divided into two parts:

- Active memory
- Factors



Dependency tree

Outline

Sequential case: Memory behaviour (1/2)

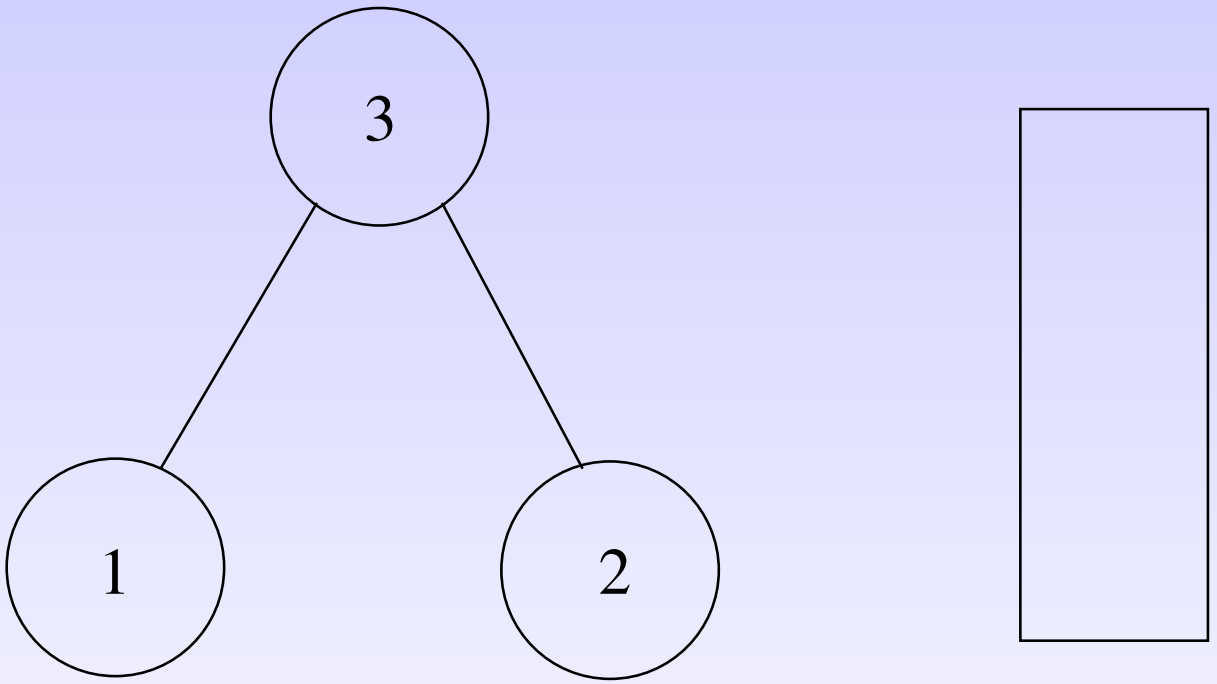


Figure: Example illustrating the memory behaviour.

Sequential case: Memory behaviour (1/2)

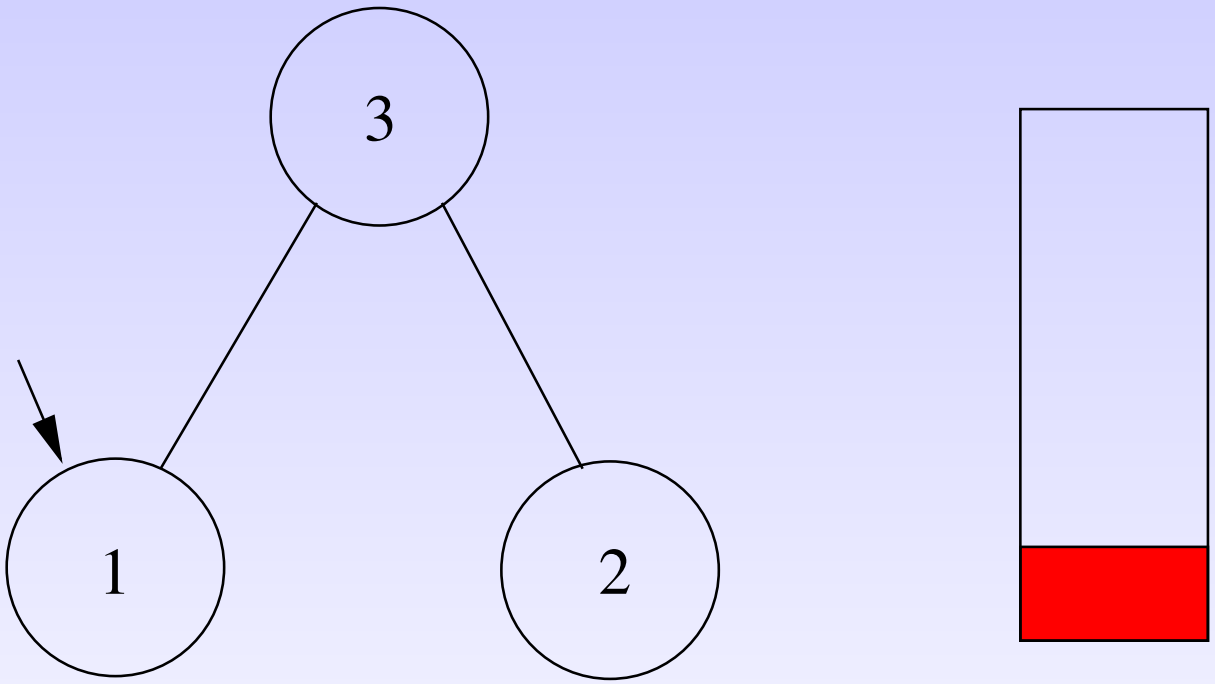


Figure: Example illustrating the memory behaviour.

Sequential case: Memory behaviour (1/2)

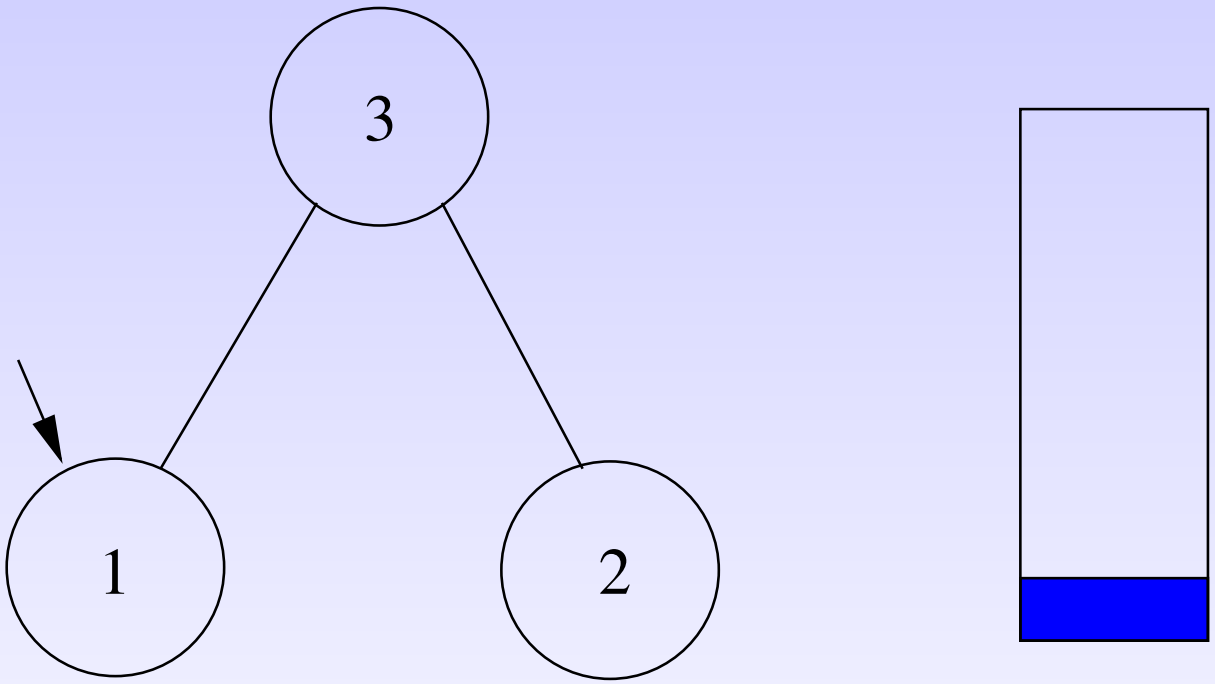


Figure: Example illustrating the memory behaviour.

Sequential case: Memory behaviour (1/2)

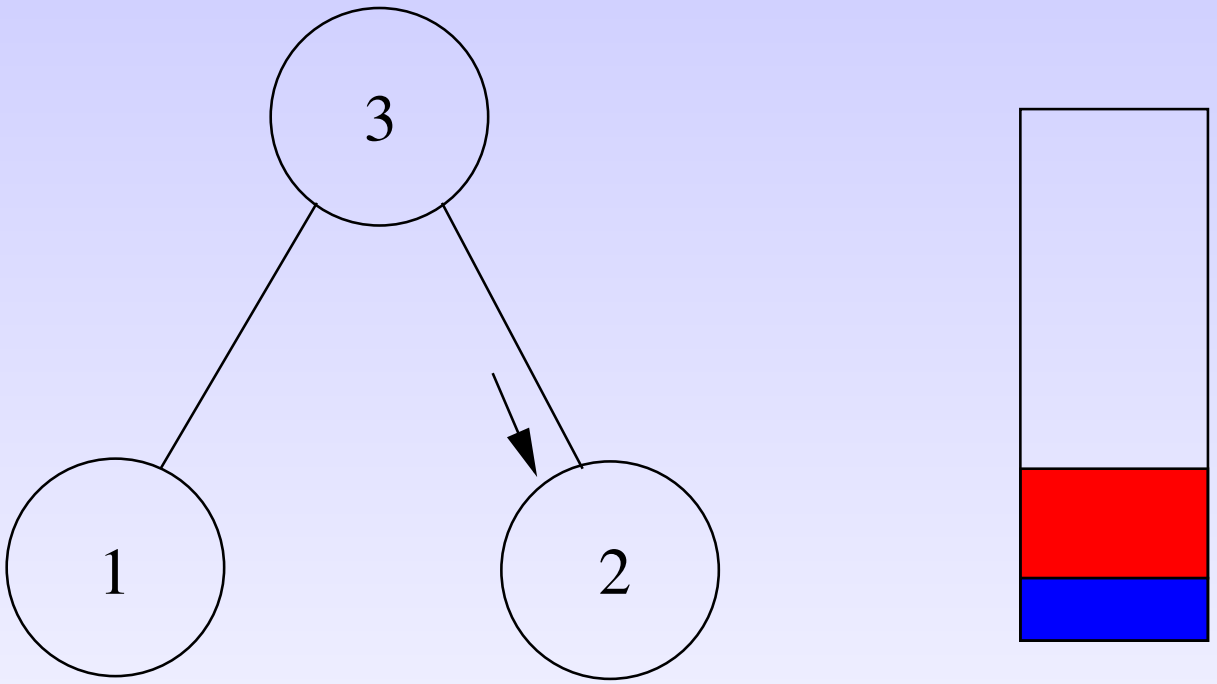


Figure: Example illustrating the memory behaviour.

Sequential case: Memory behaviour (1/2)

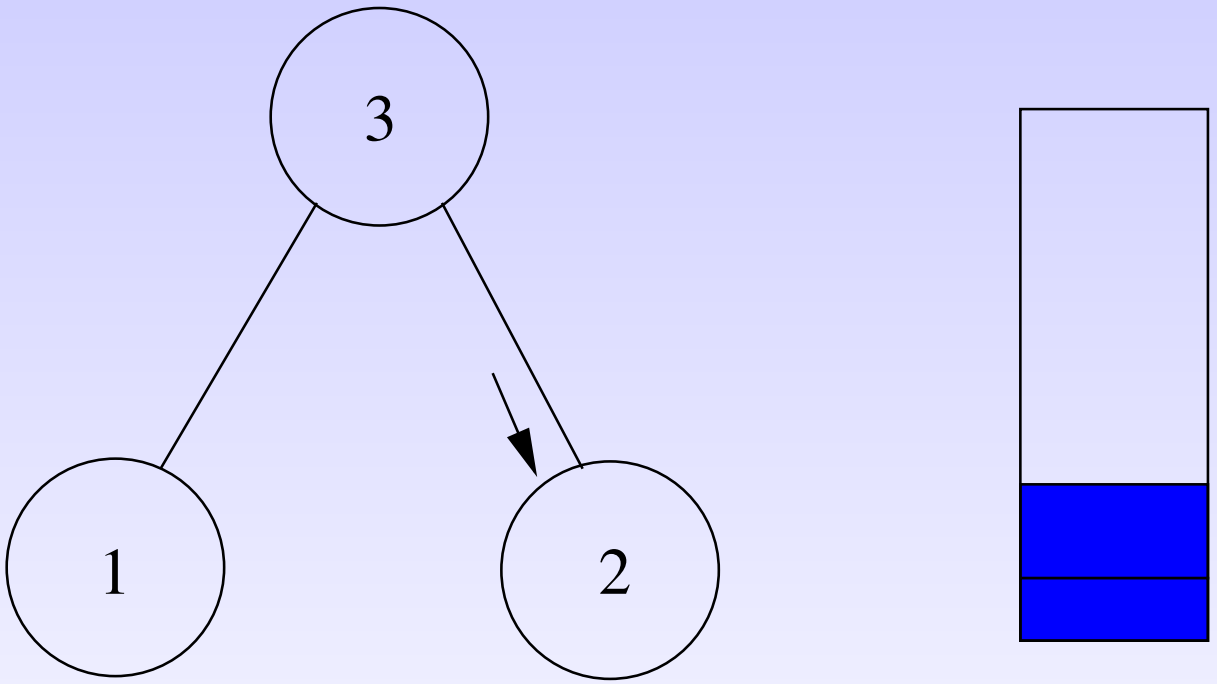


Figure: Example illustrating the memory behaviour.

Sequential case: Memory behaviour (1/2)

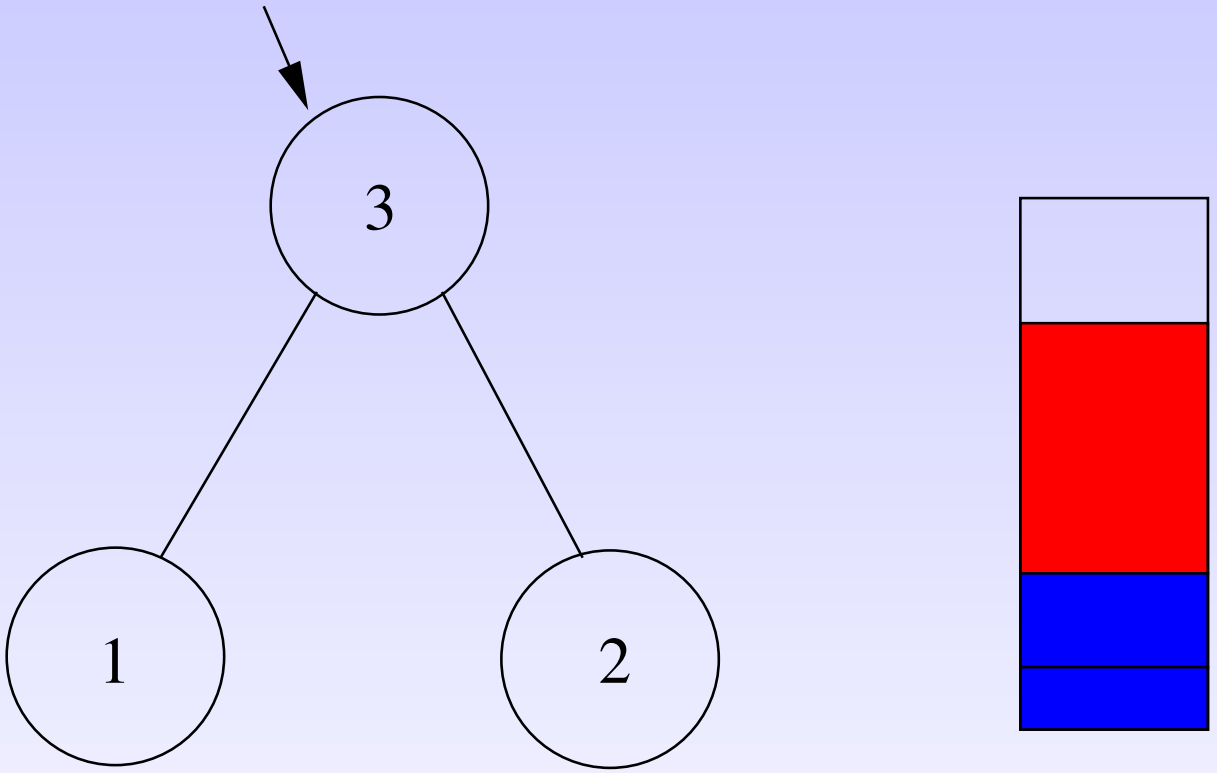


Figure: Example illustrating the memory behaviour.

Sequential case: Memory behaviour (1/2)

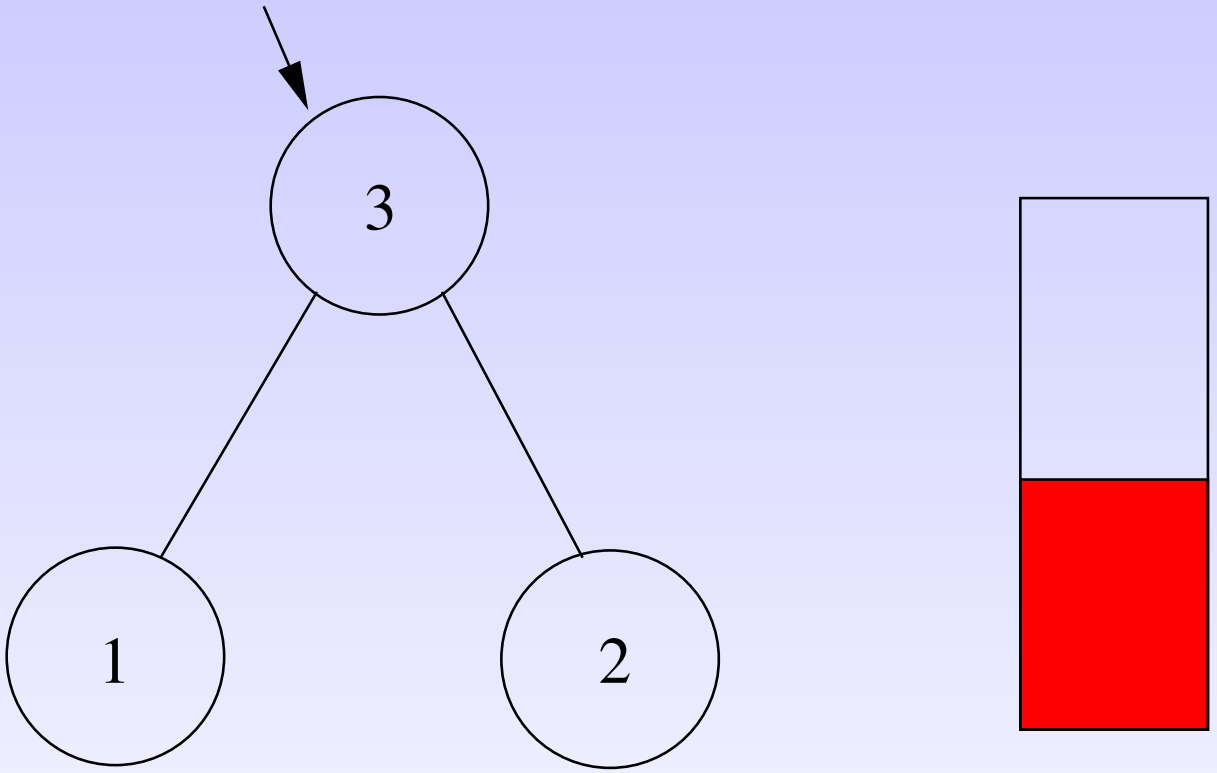


Figure: Example illustrating the memory behaviour.

Sequential case: Memory behaviour (1/2)

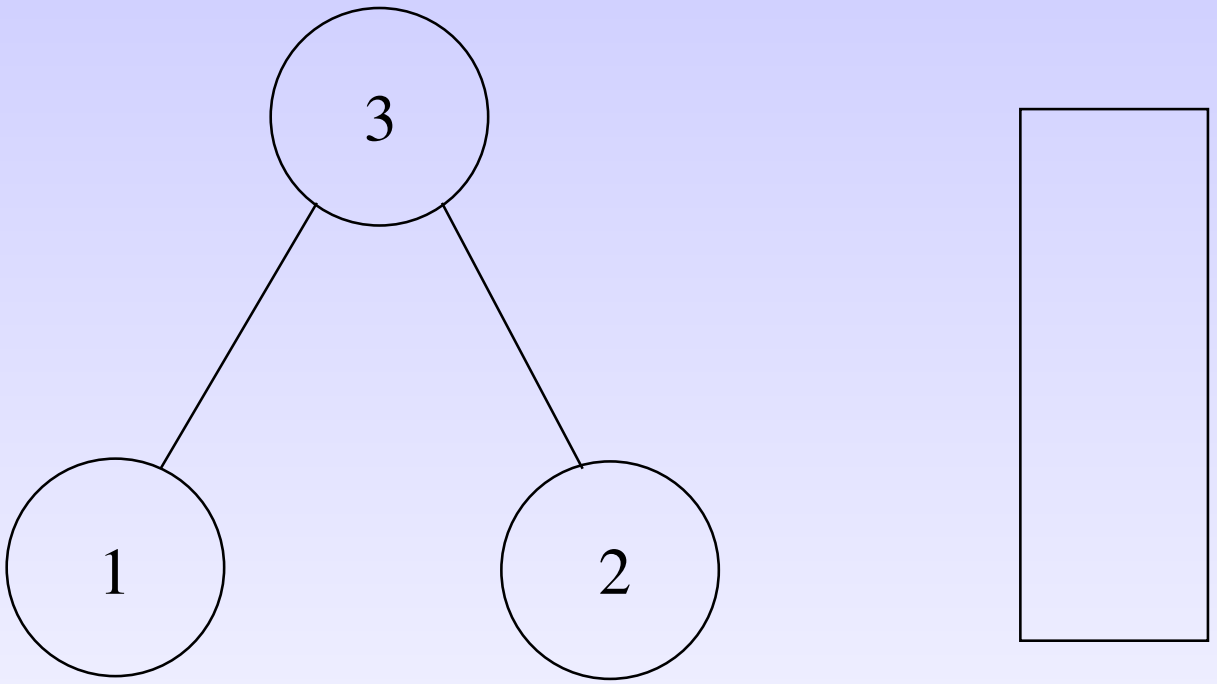
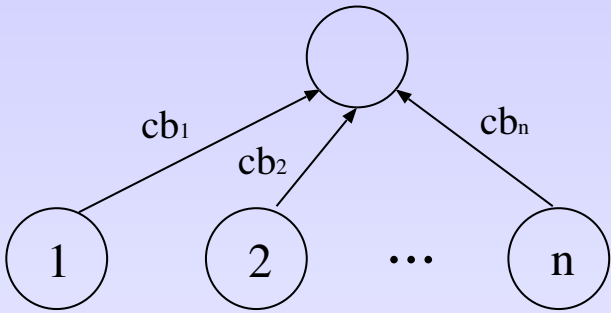


Figure: Example illustrating the memory behaviour.

Sequential case: Memory behaviour (2/2)

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).



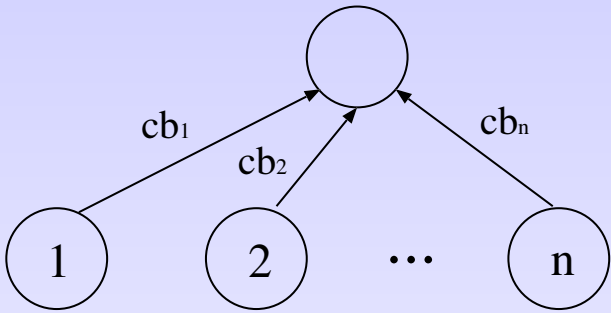
The assembly step requires a storage:

$$m + \sum_{j=1}^{n} cb_j$$

Sequential case: Memory behaviour (2/2)

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).



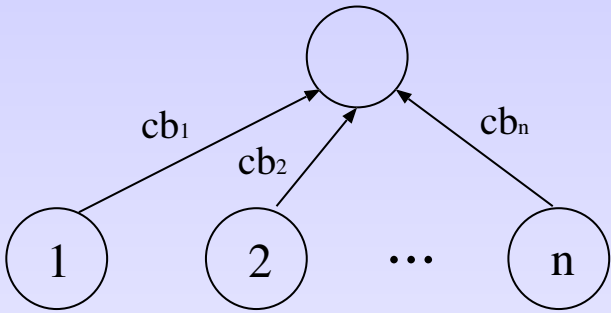
The storage required to process child j is:

$$A_j + \sum_{k=1}^{j-1} cb_k$$

Sequential case: Memory behaviour (2/2)

Consider a parent node in the tree:

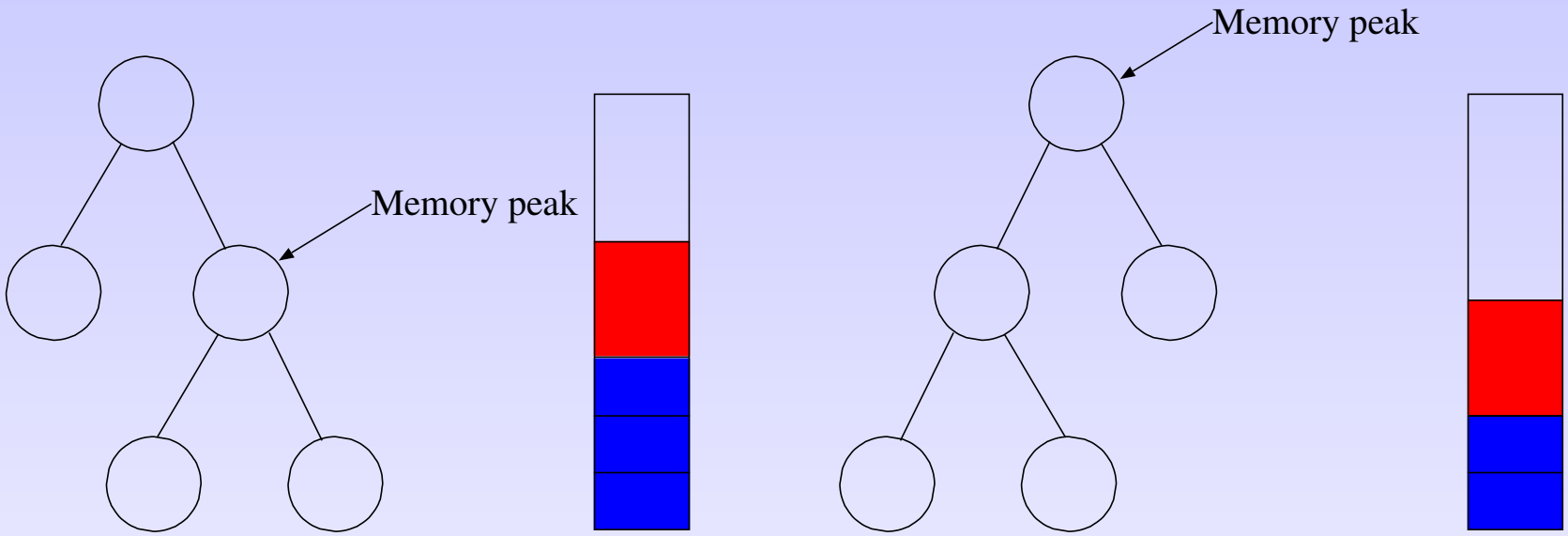
- n is the number of children.
- j denotes the j^{th} child of the node.
- cb_j is the size of the contribution block of child j.
- m is the memory size of the frontal matrix of the parent.
- A (resp. A_j) is the amount of active memory needed to process the parent (resp. child j).



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

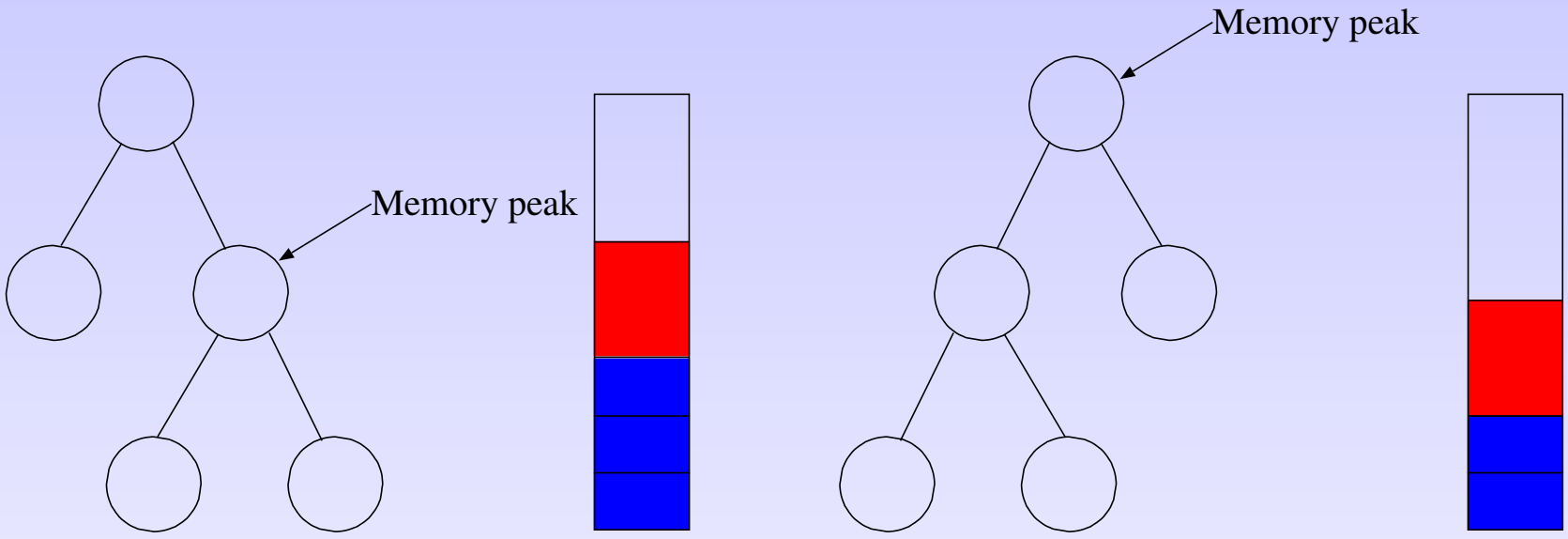


Worst case. Best case.

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



Worst case. Best case.

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

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

Remarks and properties

Liu's Theorem (Tree pebbling theorem)

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

Consequence:

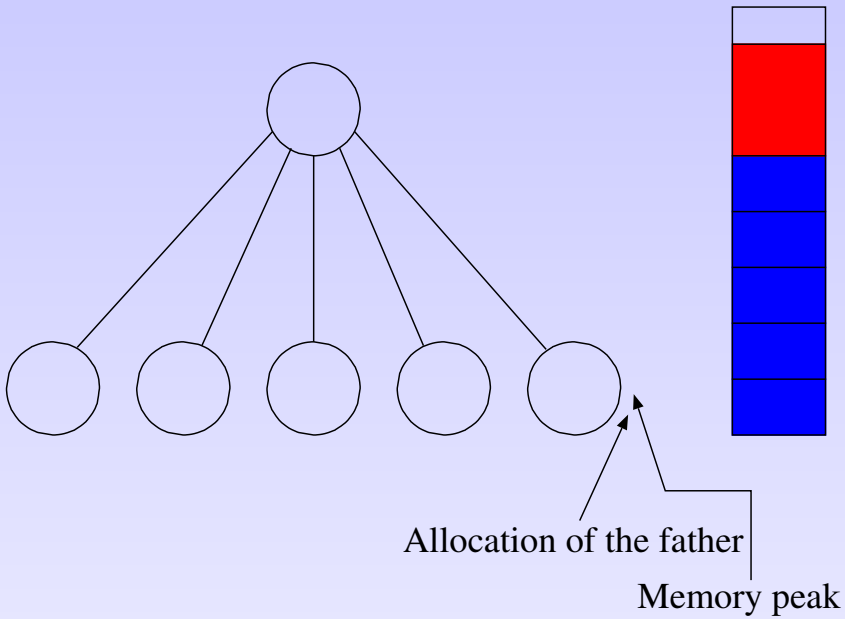
An optimal child sequence is obtained by rearranging the children nodes in decreasing order of $A_i - cb_i$.

Algorithm:

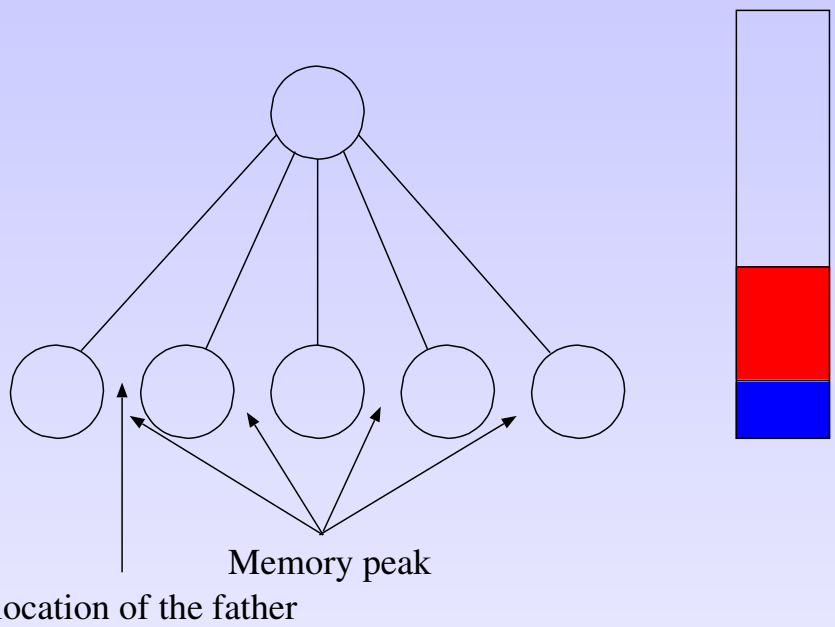
- Bottom-up greedy process.
- Apply Liu's theorem at each level of the tree.

Outline

Limitation of the Classical scheme



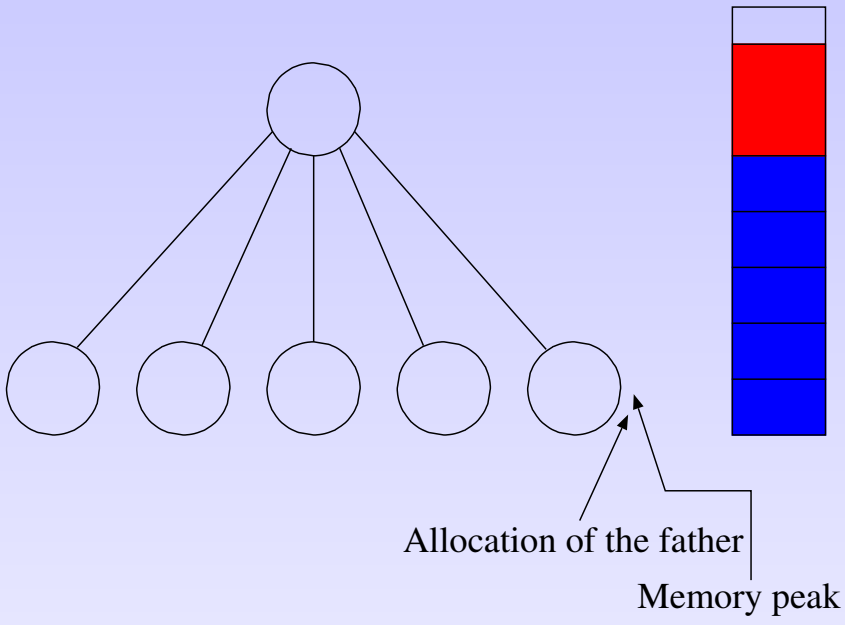
Classical approach.



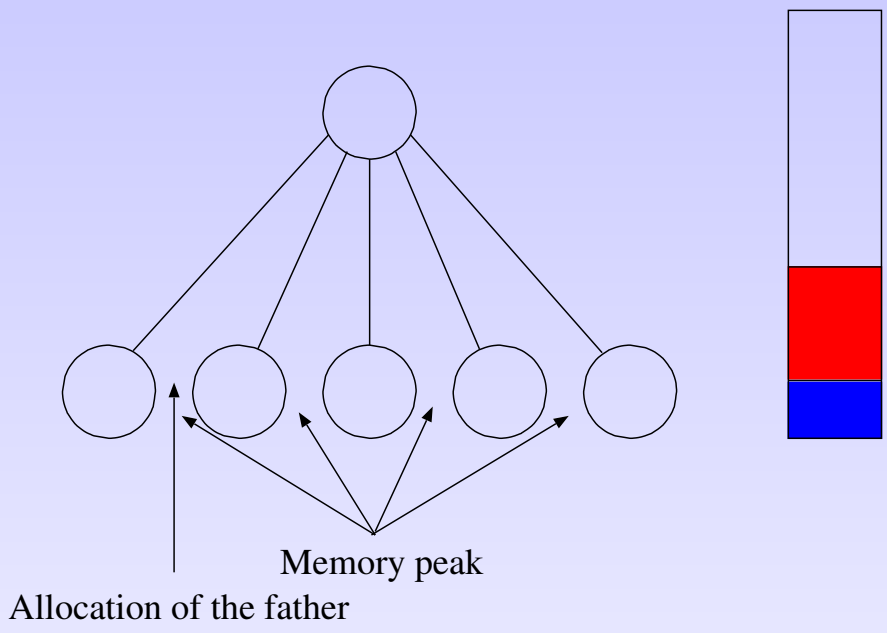
Flexible scheme.

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

Limitation of the Classical scheme



Classical approach.

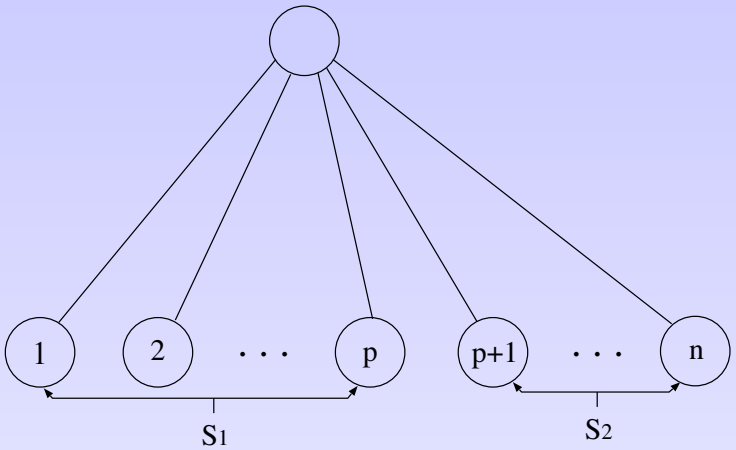


Flexible scheme.

$\longrightarrow$ Decoupling the allocation and the computations can improve the memory behaviour

Outline

Flexible multifrontal scheme

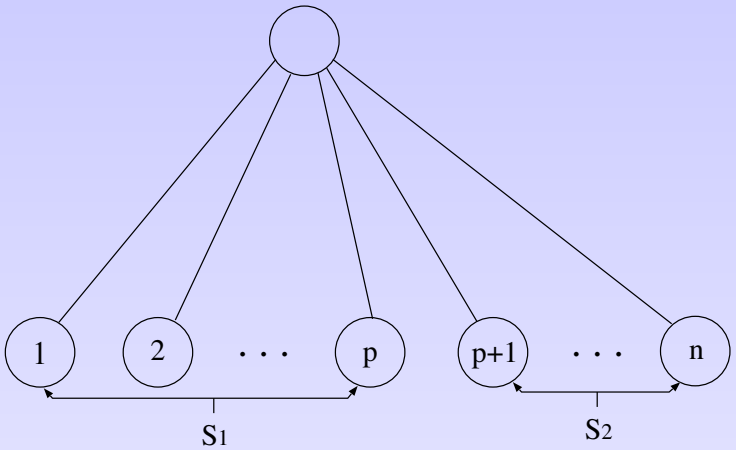


- p is the position of the allocation of the parent.
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent.
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent.

- The memory behaviour inside $\mathcal{S}_1$ is similar to the case of the classical multifrontal scheme.
- Inside $\mathcal{S}_2$, the order of the children has no impact on the memory behaviour.

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

Flexible multifrontal scheme

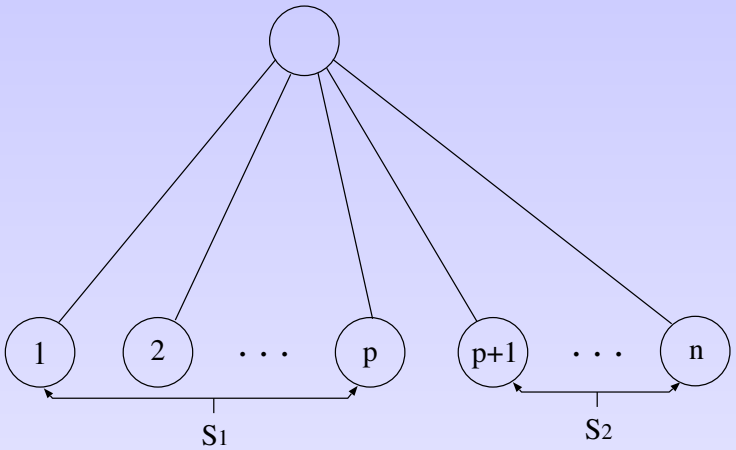


- p is the position of the allocation of the parent.
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent.
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent.

- The memory behaviour inside $\mathcal{S}_1$ is similar to the case of the classical multifrontal scheme.
- Inside $\mathcal{S}_2$, the order of the children has no impact on the memory behaviour.

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

Flexible multifrontal scheme



- p is the position of the allocation of the parent.
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent.
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent.

- The memory behaviour inside $\mathcal{S}_1$ is similar to the case of the classical multifrontal scheme.
- Inside $\mathcal{S}_2$, the order of the children has no impact on the memory behaviour.

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

Outline

A new memory minimization algorithm

Theorem

An optimal sequence can be obtained by :

- *Sorting the children in decreasing order of* A_j^{flex}.
- *Trying all the possible positions for the allocation of the parent and sorting the children belonging to* $\mathcal{S}_1$ *according to Liu's Theorem.*
- *Selecting the configuration that gives the smallest peak.*

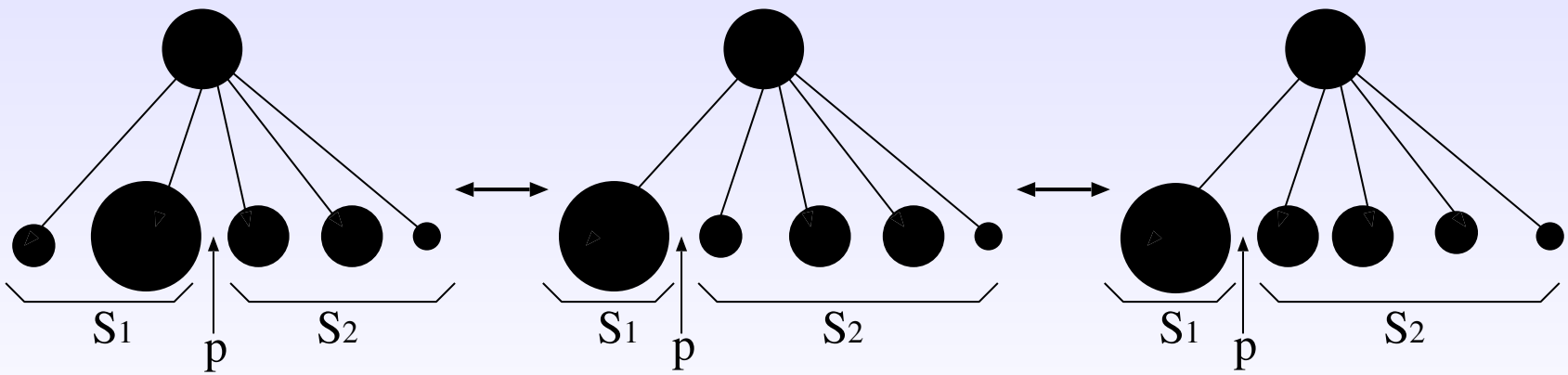
Algorithm:

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

Proof

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

- Inside $\mathcal{S}_2$, the order of the children has no impact on the memory behaviour.
- If $\exists j \in \mathcal{S}_1 \,/\, A_j^{flex} \leq \max_{i \in \mathcal{S}_2}(A_i^{flex}) \rightarrow j$ can be moved from $\mathcal{S}_1$ to $\mathcal{S}_2$ without increasing the peak.



Optimal configuration

Active memory minimization Algorithm

Algorithm:

Set $\mathcal{S}_1 = \{1, \ldots, n\}$, $\mathcal{S}_2 = \emptyset$ and $p = n$;
Find the schedule providing an optimal A^{flex} value for partition ($\mathcal{S}_1$, $\mathcal{S}_2$);
repeat
  Find j such that $A_j^{flex} = \min_{k \in \mathcal{S}_1} A_k^{flex}$;
  Set $\mathcal{S}_1 = \mathcal{S}_1 \setminus \{j\}$, $\mathcal{S}_2 = \mathcal{S}_2 \cup \{j\}$, and $p = p - 1$;
  Find the schedule providing an optimal A'^{flex} value for partition ($\mathcal{S}_1$, $\mathcal{S}_2$);
  if $A'^{flex} \leq A^{flex}$ **then**
    Keep the value of p, and the schedule of children in $\mathcal{S}_1$ and $\mathcal{S}_2$ corresponding to A'^{flex};
    Set $A^{flex} = A'^{flex}$;
  end if
until $p == 1$ or $A'^{flex} > A^{flex}$

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.

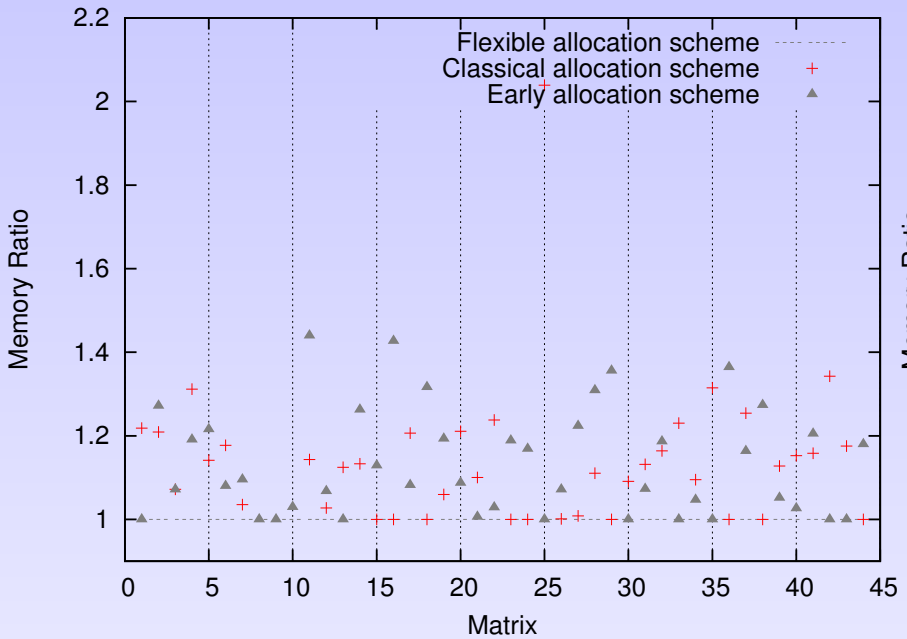
Experimental environment

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 :

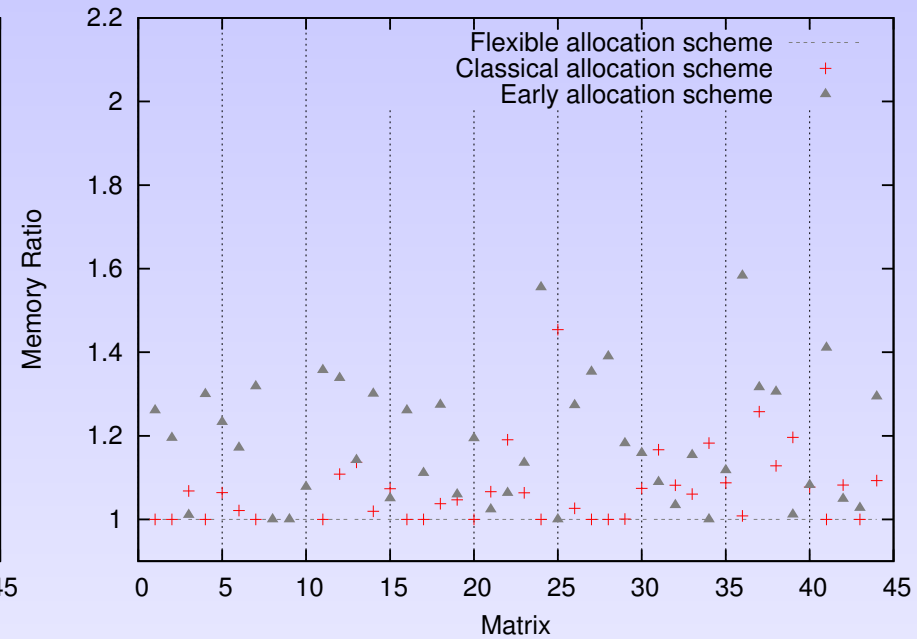
- Classical multifrontal scheme (parent allocated after all its children).
- Anticipated parent allocation scheme (parent allocated after its first child).
- Flexible parent allocation scheme (parent allocated at an arbitrary position).

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

Experimental results



AMD.

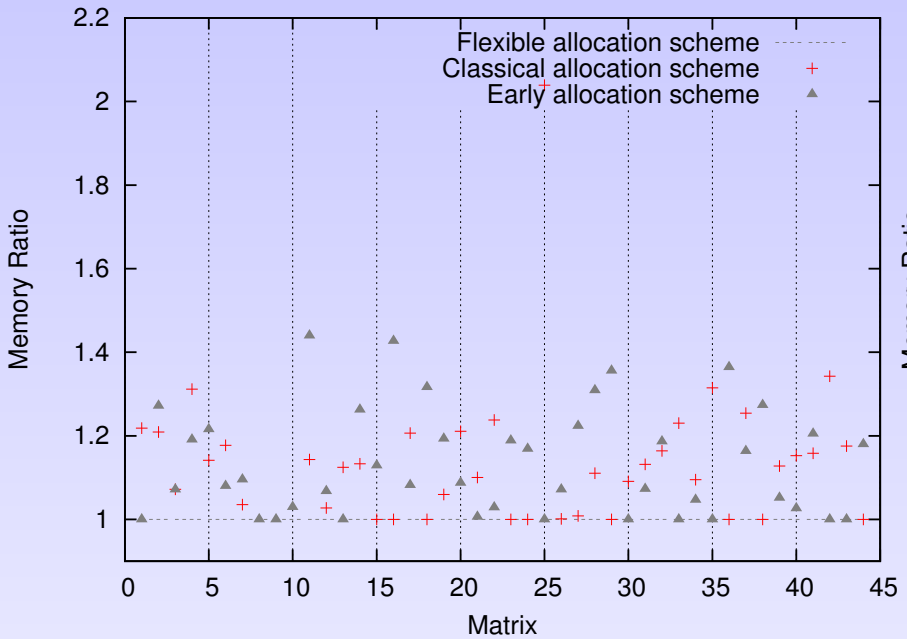


METIS.

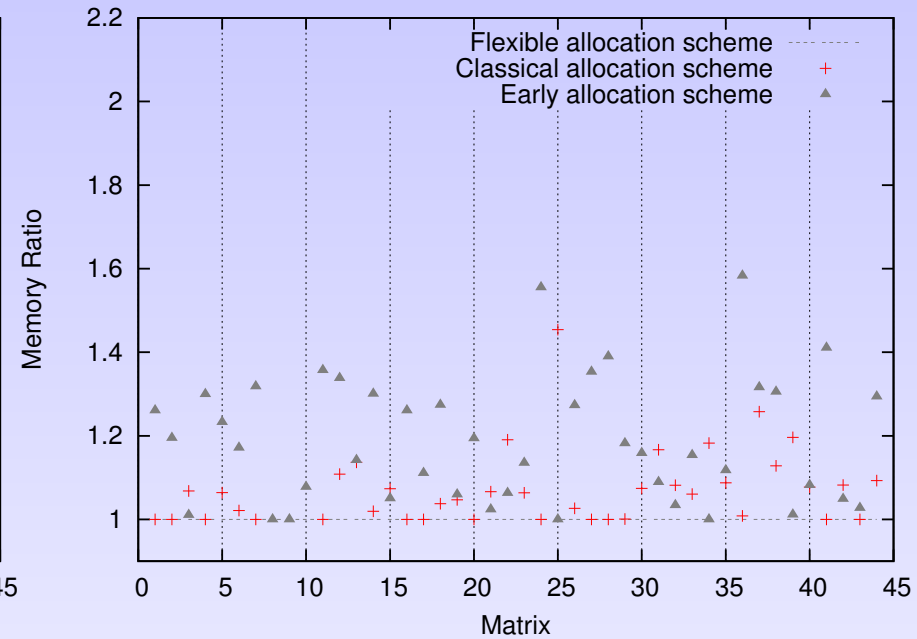
Figure: Active memory ratios.

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

Experimental results



AMD.



METIS.

Figure: Active memory ratios.

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

Total memory minimization (1/3)

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

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

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

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

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

The order in $\mathcal{S}_2$ 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:

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

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

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

The order in $\mathcal{S}_2$ 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:

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

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

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

The order in $\mathcal{S}_2$ 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:

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

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

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

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

The order in $\mathcal{S}_2$ 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:

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

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

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

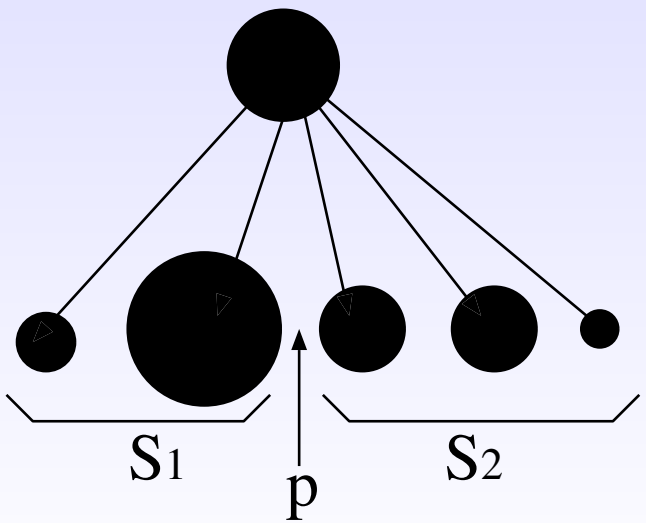
The order in $\mathcal{S}_2$ has an impact on the memory occupation.

Total memory minimization (2/3)

	$\mathcal{S}_1$	$\mathcal{S}_2$
Children sequence	$T_i^{flex} - (cb_i + F_i)$	$T_i^{flex} - F_i$

Total memory minimizing sequences inside $\mathcal{S}_1$ and $\mathcal{S}_2$.

Property:



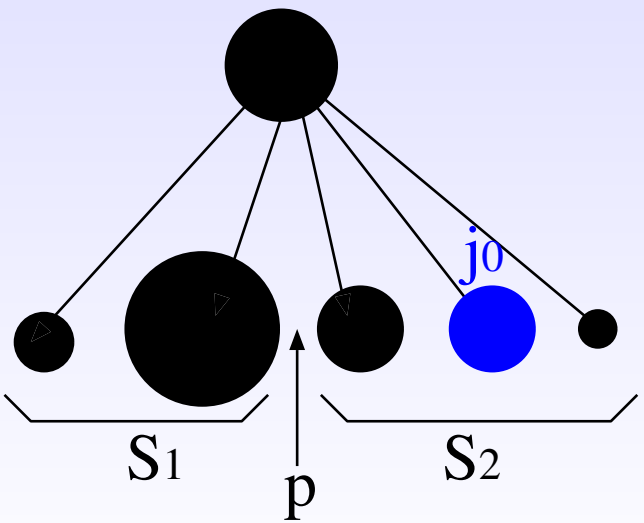
let $j_0 \in \mathcal{S}_2$ be the child for which the peak is reached inside $\mathcal{S}_2$.
$\rightarrow$ The total memory peak cannot decrease if j_0 remains in $\mathcal{S}_2$ for all configurations where $\mathcal{S}_1 \subset \mathcal{S'}_1$.

Total memory minimization (2/3)

	$\mathcal{S}_1$	$\mathcal{S}_2$
Children sequence	$T_i^{flex} - (cb_i + F_i)$	$T_i^{flex} - F_i$

Total memory minimizing sequences inside $\mathcal{S}_1$ and $\mathcal{S}_2$.

Property:



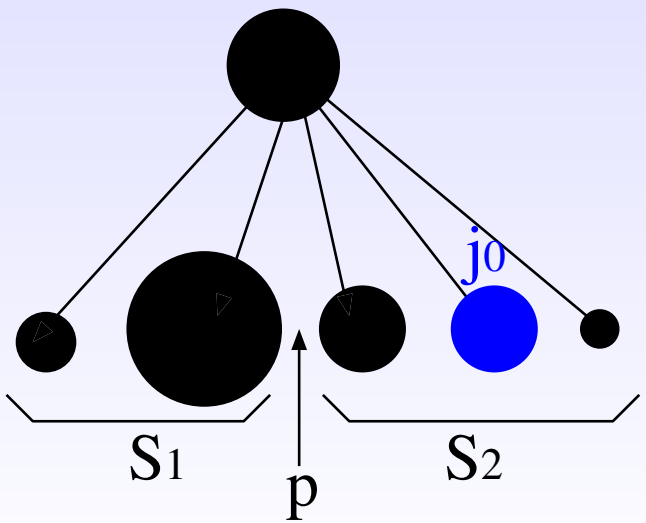
let $j_0 \in \mathcal{S}_2$ be the child for which the peak is reached inside $\mathcal{S}_2$.
$\rightarrow$ The total memory peak cannot decrease if j_0 remains in $\mathcal{S}_2$ for all configurations where $\mathcal{S}_1 \subset \mathcal{S}'_1$.

Total memory minimization (2/3)

	$\mathcal{S}_1$	$\mathcal{S}_2$
Children sequence	$T_i^{flex} - (cb_i + F_i)$	$T_i^{flex} - F_i$

Total memory minimizing sequences inside $\mathcal{S}_1$ and $\mathcal{S}_2$.

Property:



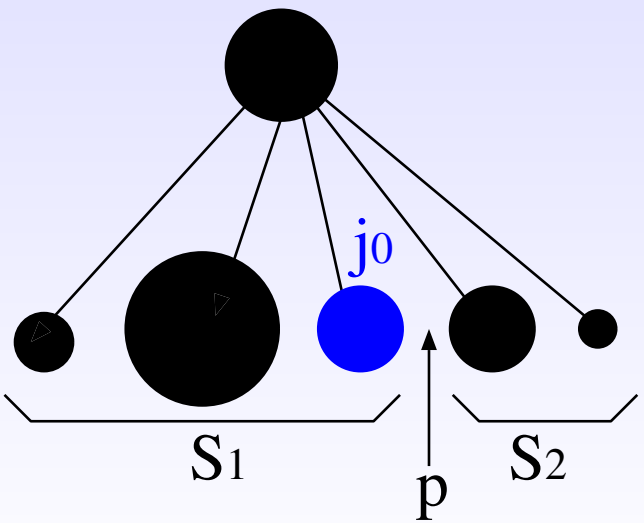
let $j_0 \in \mathcal{S}_2$ be the child for which the peak is reached inside $\mathcal{S}_2$.
$\rightarrow$ The total memory peak cannot decrease if j_0 remains in $\mathcal{S}_2$ for all configurations where $\mathcal{S}_1 \subset \mathcal{S}'_1$.

Total memory minimization (2/3)

	$\mathcal{S}_1$	$\mathcal{S}_2$
Children sequence	$T_i^{flex} - (cb_i + F_i)$	$T_i^{flex} - F_i$

Total memory minimizing sequences inside $\mathcal{S}_1$ and $\mathcal{S}_2$.

Property:



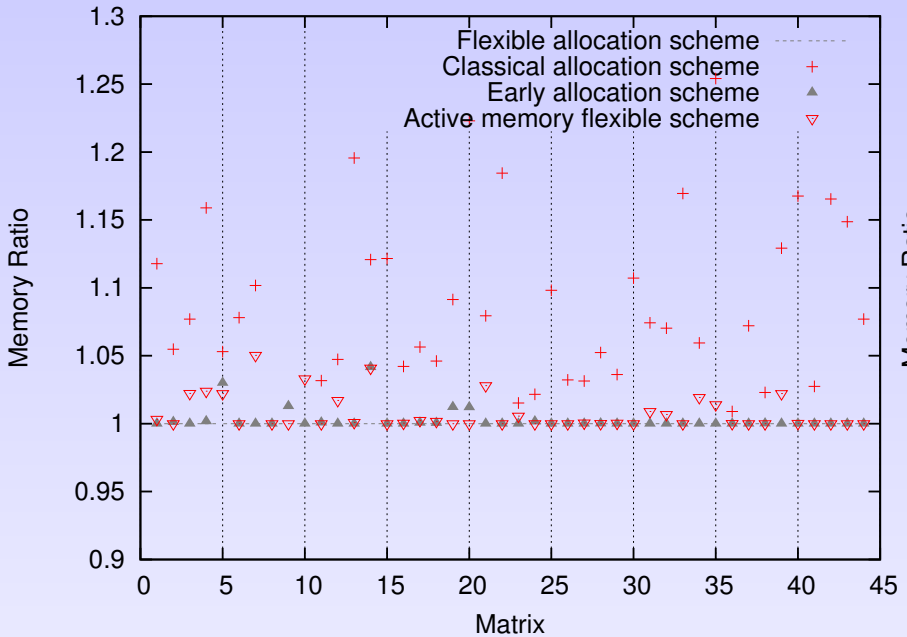
let $j_0 \in \mathcal{S}_2$ be the child for which the peak is reached inside $\mathcal{S}_2$.
$\rightarrow$ The total memory peak cannot decrease if j_0 remains in $\mathcal{S}_2$ for all configurations where $\mathcal{S}_1 \subset \mathcal{S}'_1$.

Total memory minimization (3/3)

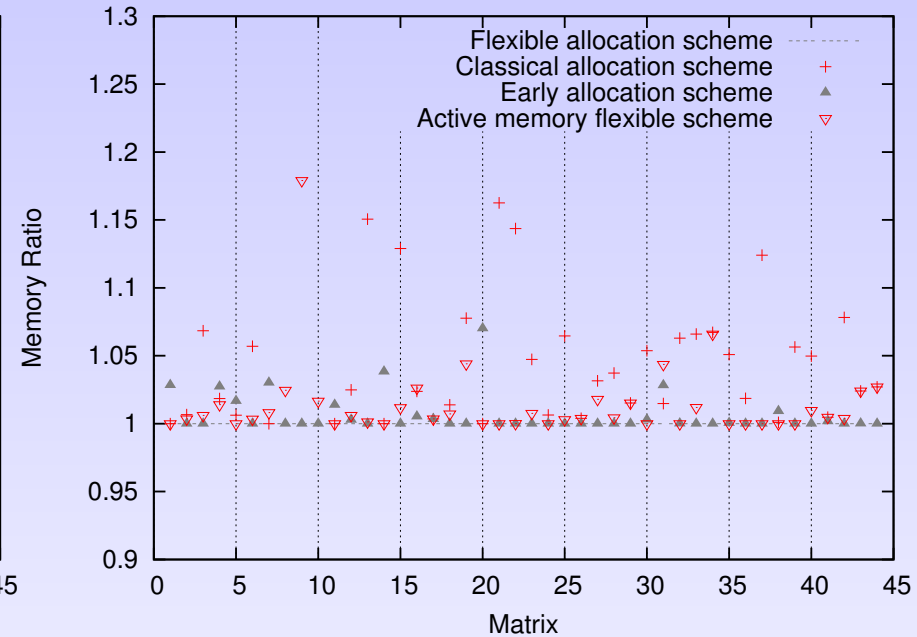
Algorithm:

Set $\mathcal{S}_1 = \emptyset$, $\mathcal{S}_2 = \{1, \ldots, n\}$ and $p = 0$;
Sort $\mathcal{S}_2$ according in decreasing order of $T_j^{flex} - F_j$;
Compute $T^{flex} = \mathcal{P}_2$;
repeat
  Find j_0 such that the peak in $\mathcal{P}_2$ is obtained for j_0;
  Set $\mathcal{S}_1 = \mathcal{S}_1 \cup \{j_0\}$, $\mathcal{S}_2 = \mathcal{S}_2 \setminus \{j_0\}$, and $p = p + 1$;
  (Remark: j_0 is inserted at the position in $\mathcal{S}_1$ so that the order inside this set is decreasing in terms of $T_j^{flex} - (cb_j + F_j)$.)
  Compute $\mathcal{P}_1$, $\mathcal{P}_2$, and $T'^{flex} = \max(\mathcal{P}_1, \mathcal{P}_2)$;
  if $T'^{flex} \leq T^{flex}$ **then**
    Keep the values of p, $\mathcal{S}_1$ and $\mathcal{S}_2$ and set $T^{flex} = T'^{flex}$;
  end if
until $p = n$ or $\mathcal{P}_1 \geq \mathcal{P}_2$

Experimental results



AMD.



METIS.

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).