



# Some Memory Issues in the Multifrontal Method

---

Abdou Guer mouche, Jean-Yves L'Excellent and Gil Utard

REMAP Project

LIP (ENS-Lyon, INRIA and CNRS)

Lyon, France



# Introduction

---

Solve  $Ax=b$

Sparse direct factorization methods : Frontal, Supernodal, Multifrontal,...

**Multifrontal method** (Duff,Reid '74): Sparse direct method for matrix factorization based on a tree.

- Efficiency due to good locality (BLAS3).
- Good potential for parallelism.
- Large memory requirement (compared to iterative methods).

—————▶ Memory problems may arise with large matrices.

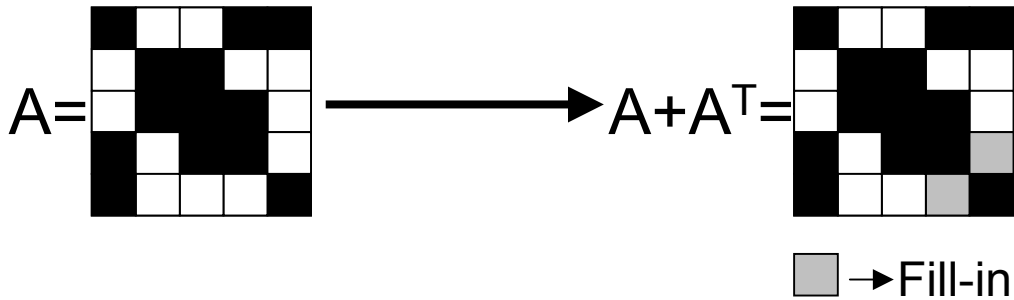


# Outline

---

- Memory behaviour in the multifrontal method.
  - Impact of reordering techniques on the shape of the assembly tree and memory.
  - Study of the memory occupation of a parallel multifrontal solver.
  - Memory-based scheduling strategies (dynamic).
- Improvement of low-level paging mechanisms for the multifrontal method.
- Summary of ongoing work

# Multifrontal Method



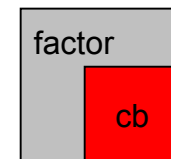
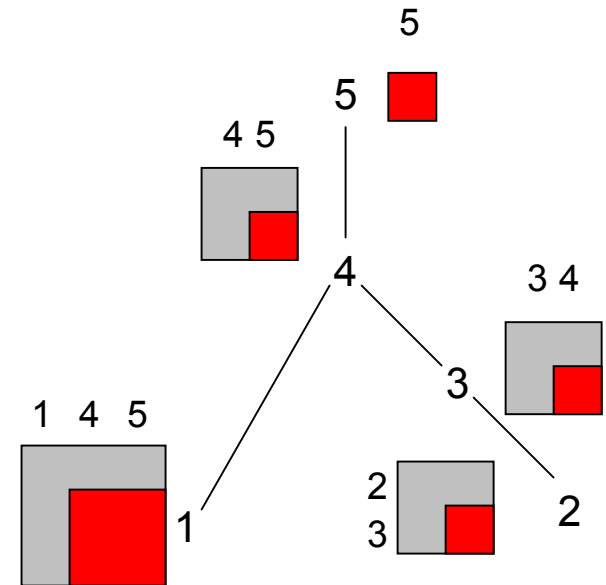
## Advantage:

once they are computed, entries of  $L$  and  $U$  are not reused.

## Drawback:

the method uses a stack that consumes memory.

## dependency tree

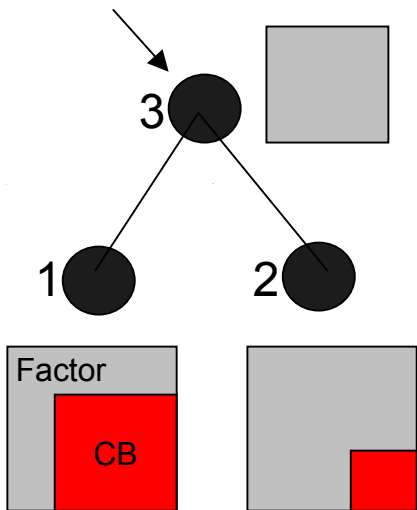
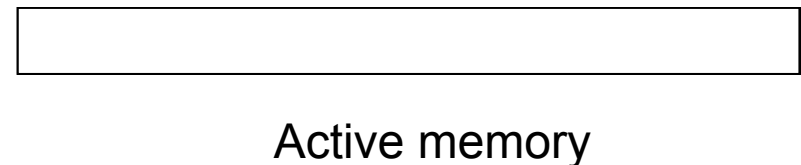
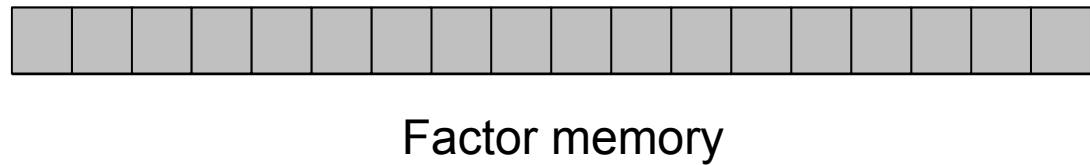
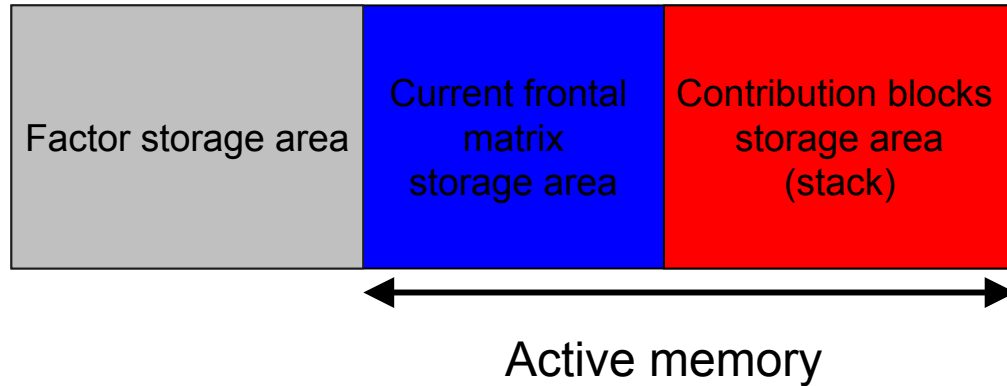


Frontal matrix structure

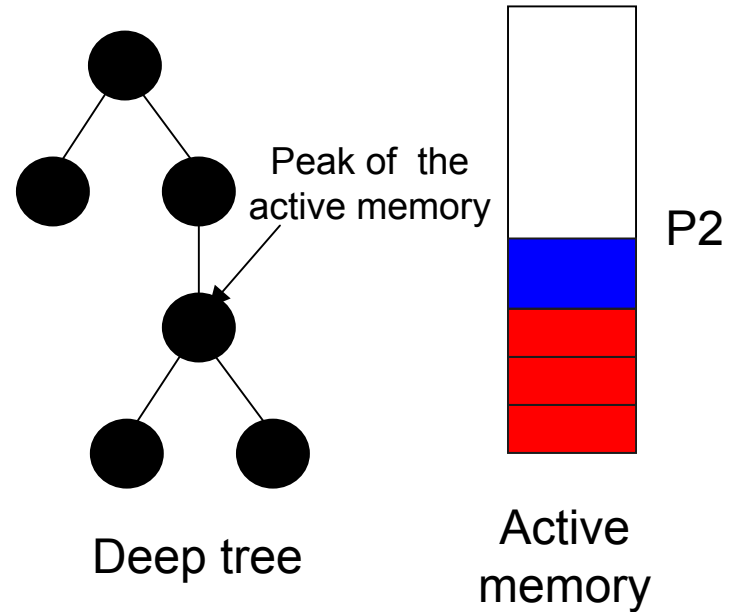
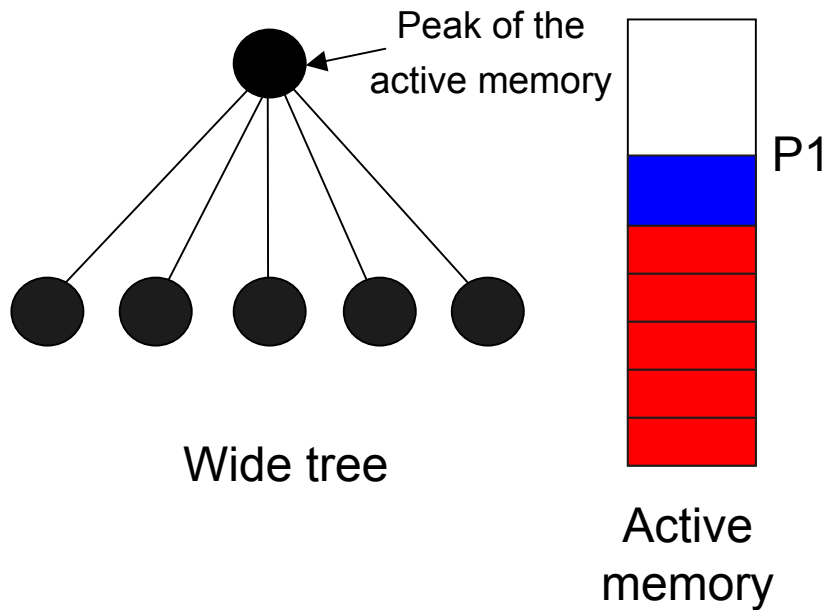
# Multifrontal Method (memory aspects) (1)

Memory divided in two parts:

- Active memory
- Factor memory



## Multifrontal Method (memory aspects) (2)



$P1 > P2$

All the trees are reordered with a variant of Liu's algorithm  
→ Optimal memory usage for the sequential case



# Reordering techniques

---

**Reordering techniques** : Heuristics to decrease flops and size of factors.

→ Large impact on the assembly tree topology.

→ Large impact on the active memory (peak and evolution with time).

**Extensive experimental study** : Impact of reordering techniques on the factors, the assembly tree and the active memory.



# Test problems and environment

---

## Test problems :

Large range of symmetric and unsymmetric assembled matrices from various collections (Rutherford Boeing collection, Tim Davis collection and PARASOL collection).

## Environment :

**MUMPS** (Amestoy, Duff, Koster, L'Excellent) → Multifrontal parallel solver with threshold partial pivoting for both **LU** and **LDL<sup>T</sup>**.

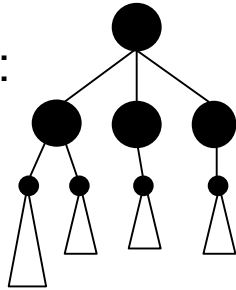
## Reordering techniques :

**Local methods** : Minimum degree (AMD), minimum fill(AMF).

**Hybrid methods** : Using nested dissection on the top of the tree and local methods elsewhere: METIS (METIS permutation only+MUMPS symbolic factorization), SCOTCH, PORD.

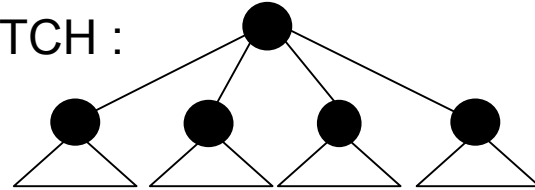
# Impact of the reordering techniques on the assembly tree

AMD :



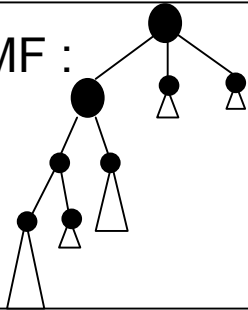
- Deep well balanced tree.
- Large frontal matrices on the top of the tree.

SCOTCH :



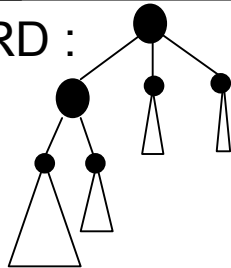
- Very wide well-balanced tree.
- Large frontal matrices.
- Small number of nodes.

AMF :



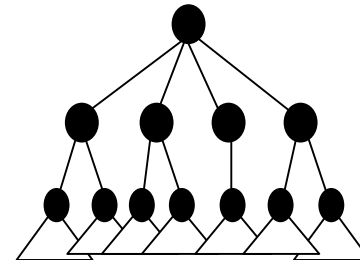
- Very deep unbalanced tree.
- Small frontal matrices.
- Very large number of nodes.

PORD :



- Deep unbalanced tree.
- Small frontal matrices.
- Large number of nodes.

METIS :



- Wide well-balanced tree.
- Large number of nodes.
- Smaller matrices (than SCOTCH).



# Impact of the reordering on the memory usage (Summary)

---

Test problem	Peak of active memory		Size of factors	
	Small	Large	Small	Large
<b>METIS</b>	+	-	-	-
<b>SCOTCH</b>	+	-	-	-
<b>PORD</b>	--	--	--	-
<b>AMF</b>	-	+	-	+
<b>AMD</b>	++	++	++	++

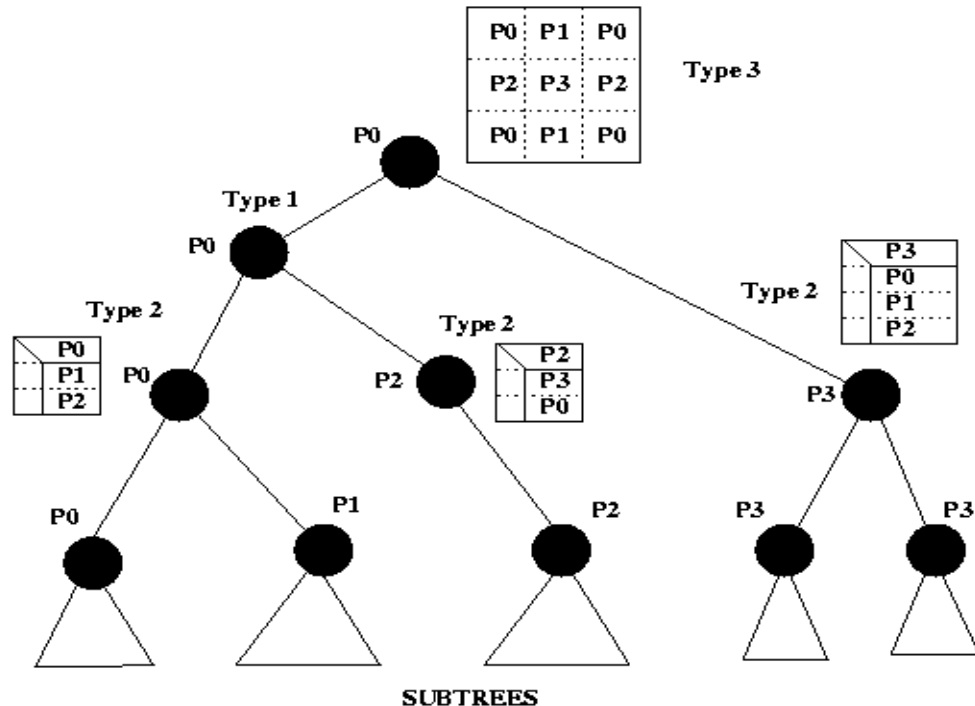
(+): big, (-): small

Memory occupation according to reordering techniques.

# MUMPS (parallel scheme)

Three kinds of nodes :

- **Type 1** : nodes treated in sequential on the concerned processor.
- **Type 2** : nodes processed with a 1D blocking factorization scheme.
- **Type 3** : 2D blocking factorization scheme (only for the root node).



# MUMPS slave selection Strategy (Type 2 nodes)

Flops-based strategy :

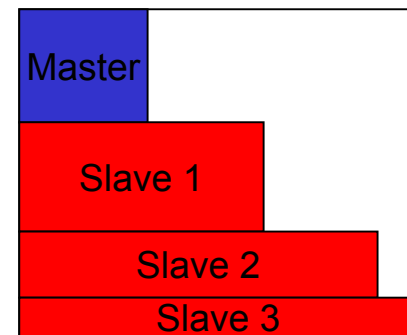
→ Each master tries to choose only the processors less loaded than himself.

Unsymmetric case



Same amount of work to each slave.

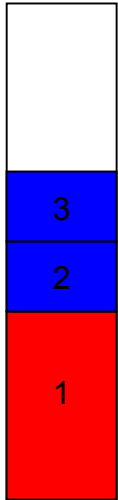
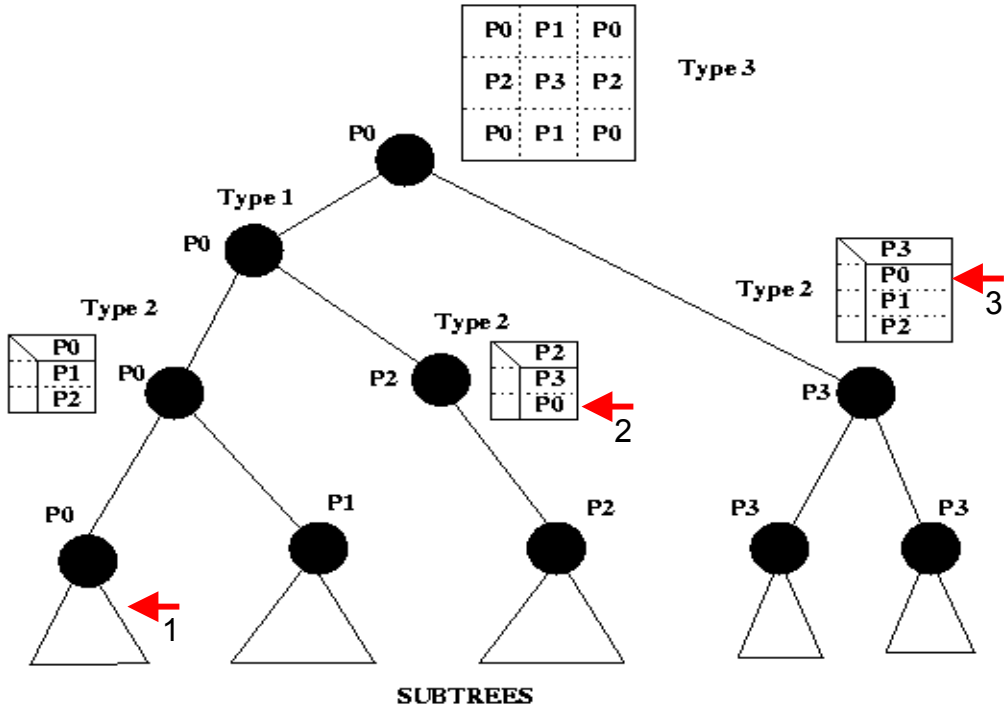
Symmetric case



Try to balance the amount of work (with an upper band which is the amount of work for the master) .

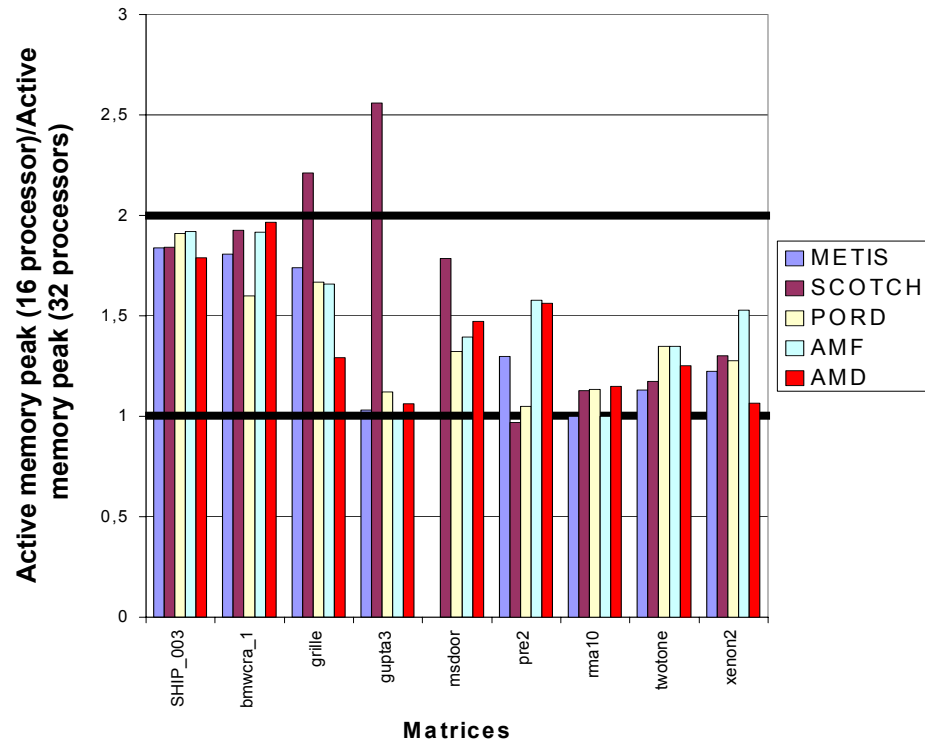
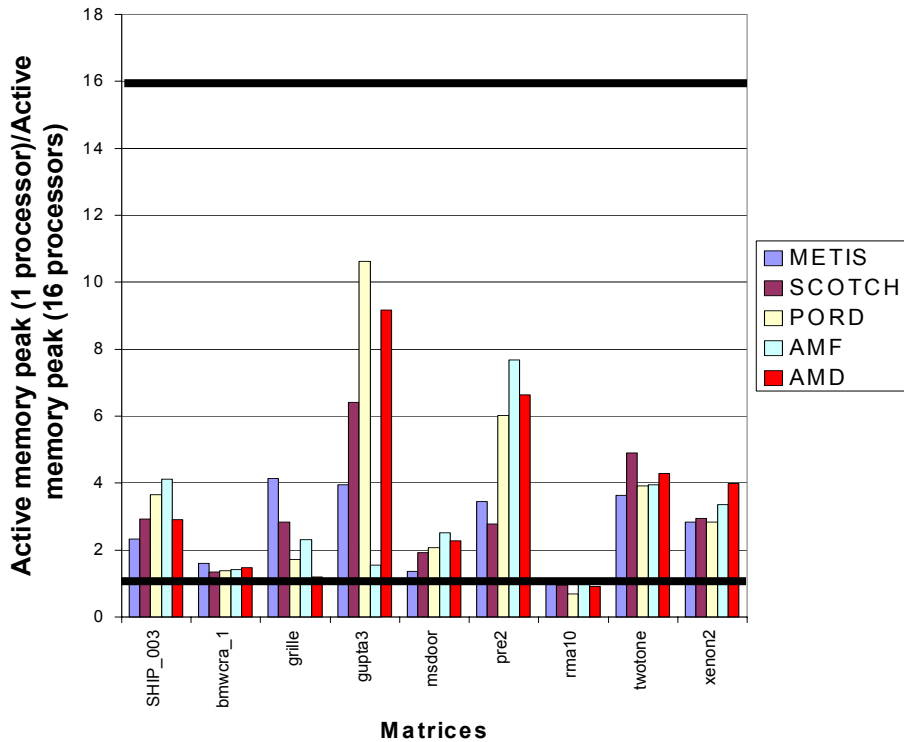
# Memory burden

Scheduling is load based, not memory aware.



Active memory

# Active memory peak for parallel executions



The active memory scalability is not linear.



# Flops-based and memory-based strategies

---

- Strategies implemented in MUMPS are based only on the floating-point operations of partial factorizations.
- Necessity to design memory-based strategies (for large problems and/or a future out-of-core approach).
- Importance of the mechanism needed by a processor to obtain the workload/memory informations about others.

## **Floating-point strategies**

The notion of workload gives informations about what will happen.

## **Memory strategies**

The memory informations used are very instantaneous.

The active memory occupation has huge variations.



## Memory-based slave selection (1)

---

Irregular matrix blocks for both symmetric and unsymmetric cases.

Several strategies based on active memory informations:

- Choose the smallest set of processors that does not increase the current peak of memory.
- Choose slave processors to have a good balance of memory occupation.



## Some results

---

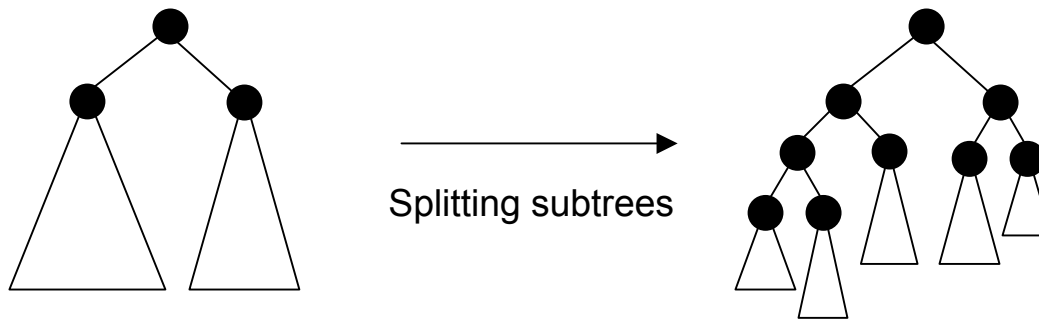
	<b>METIS</b>	<b>SCOTCH</b>	<b>PORD</b>	<b>AMF</b>	<b>AMD</b>
Twotone	24.3 %	0.2 %	36.4 %	12.7 %	31.9 %
Wang3	0 %	0 %	0 %	18.9 %	6.2 %
Xenon2	23.8 %	23.3 %	0 %	14.6 %	18.5 %

Percentage of decrease of active memory (stack memory) peak on 16 processors with the memory-based slave selection strategy.

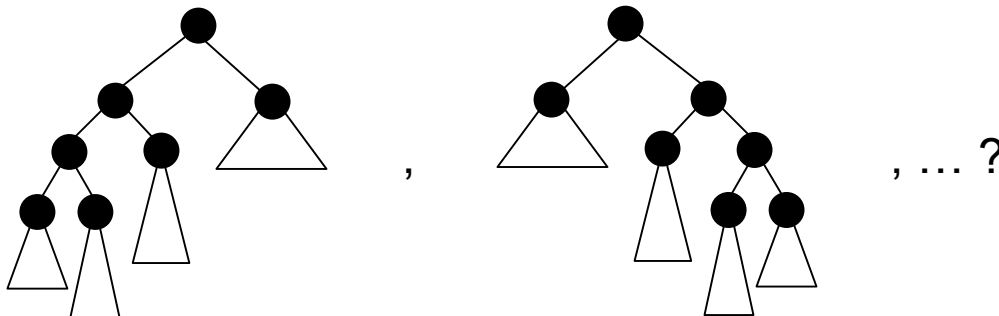


## Ongoing work on static scheduling

- Work on the mapping phase to take memory constraints into account.



- Work on the initial pool of tasks (find the best order on which the initial tasks assigned to a processor are treated).





# Study and improvement of the paging in MUMPS (work with O.Cozette)

---

**Observation:** factorization of matrices too large w.r.t. physical memory lead to very costly disk paging

- Use of a low-level library (MMUSEL).
- Implementation of a monitor that manages the paging mechanisms.
- Initially, focus only on the sequential case.
- Application provides memory access information to the monitor to ensure prefetching



# MMUSEL

---

MMUM

Memory Management in User Mode (Linux mode)

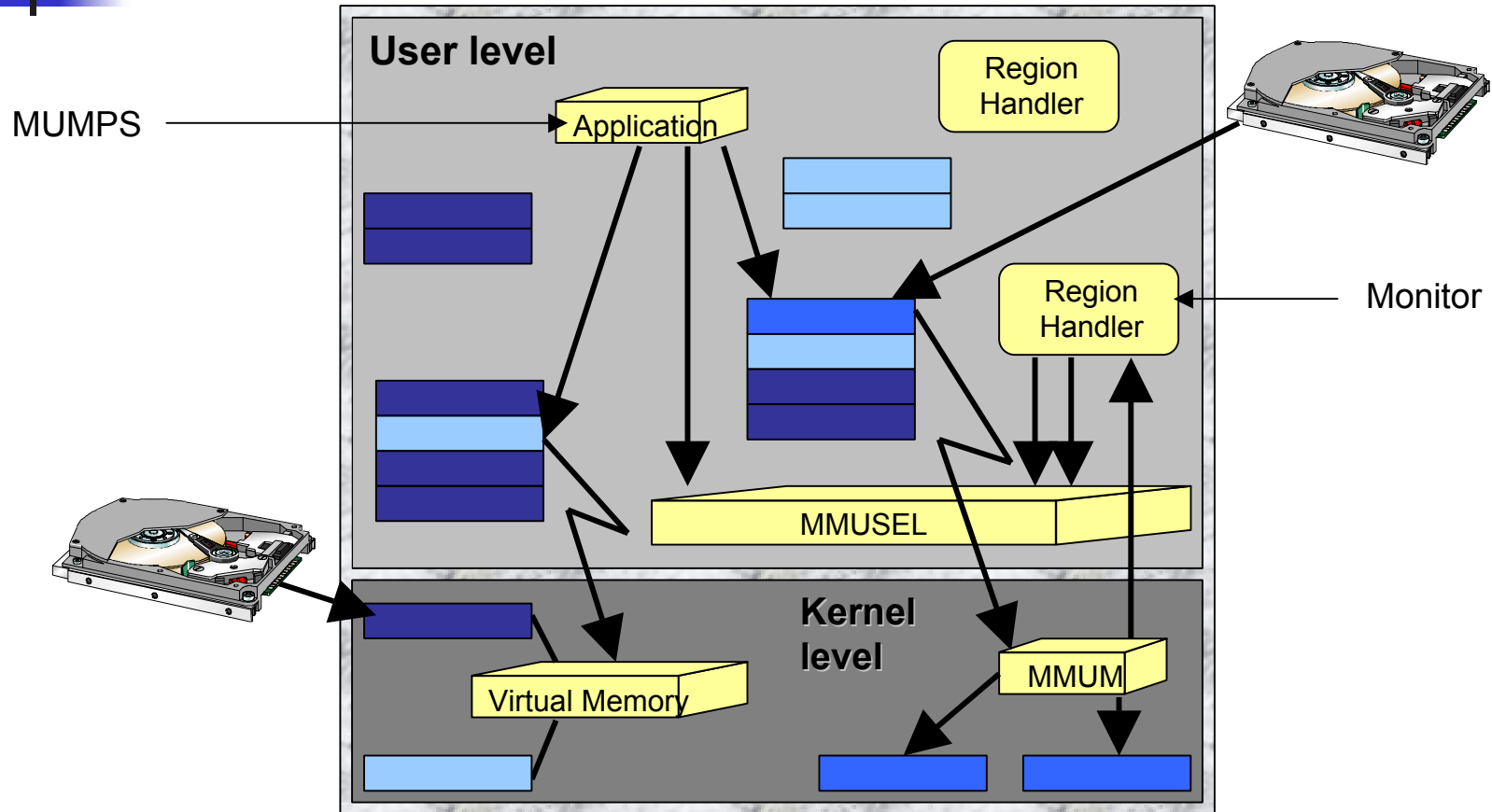
MMUSEL

Memory Management at User Space Level (Library)

Management of paging for intensive computations

[O. Cozette 98]

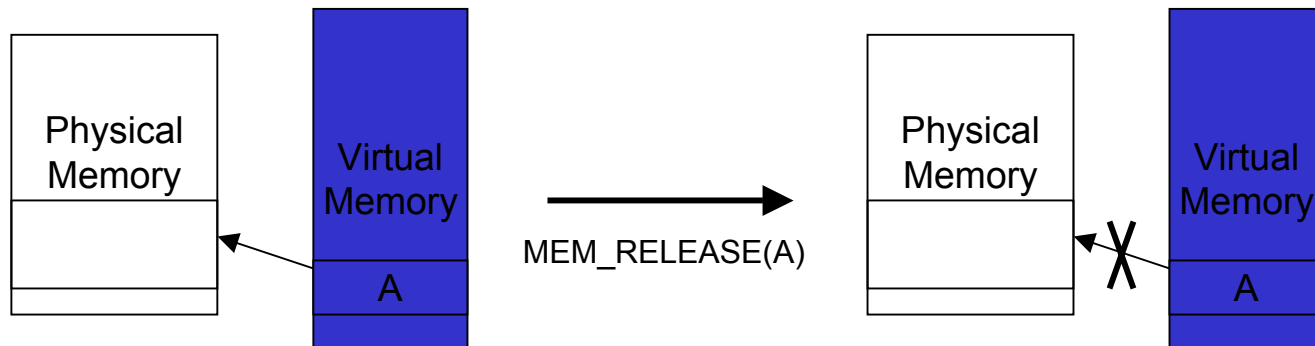
# Architecture of MMUSEL



## Two types of operations

### Releasing memory area :

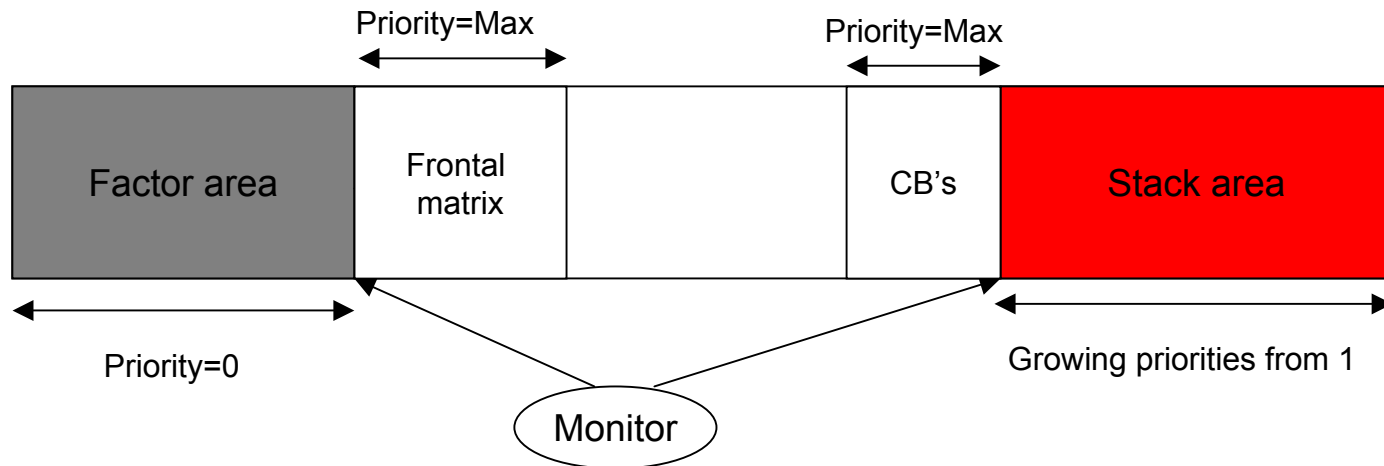
Dereferencing the physical memory page (well-adapted for areas that will only be accessed in write mode).



### Setting a priority to a memory area :

This operation will help the monitor to know which pages must be in memory.

# Interfacing MUMPS and MMUSEL



The current frontal matrix must be in memory → Priority=Max

The contribution blocks needed for the assembly step must be in memory → Priority=Max

Once the contribution is consumed it will not be reused → Call MEM\_RELEASE()

Once the factors are computed they will not be accessed until the solve → Priority=0



## Some results

---

	GUPTA3	SHIP_003	THREAD	XENON2
MUMPS+ Monitor	417.6	2938.6	1160	1392
MUMPS	460.6	3973.6	1594.1	2242.2

- Sequential execution times (seconds) of MUMPS
- Alpha EV56 (533 Mhz) with 64 MB of physical memory
- METIS used



# Conclusion

---

- Study of memory aspects of the multifrontal method.
- Large impact of reordering techniques:
  - Local methods → deep unbalanced tree.
  - Hybrid methods → wide well-balanced tree.
- Scalability of memory usage is not perfect  
→ design of memory-based scheduling strategies.
- Influence low-level memory paging mechanisms of the multifrontal method:  
→ design and optimization of a MUMPS/MMUSEL coupling.

Growing  
active memory  
occupation ↓



## Ongoing work

---

### ➤ **Scheduling:**

- Find hybrid approaches that are well-adapted for both active memory usage and execution time.
- Work on static scheduling to take memory constraints into account.

### ➤ **Paging (MUMPS+MMUSEL):**

- Improvement to sequential version and extension to parallel case.
- Provide at the application level (MUMPS) the memory accesses pattern to ensure « an optimal » paging scheme.

### ➤ **out-of-core extension of MUMPS:**

- Storage of the factors on disk as soon as they are computed (solve step?).
- Add out-of-core stack memory management for critical cases.

### ➤ Compare an out-of-core implementation of MUMPS with MUMPS+MMUSSEL