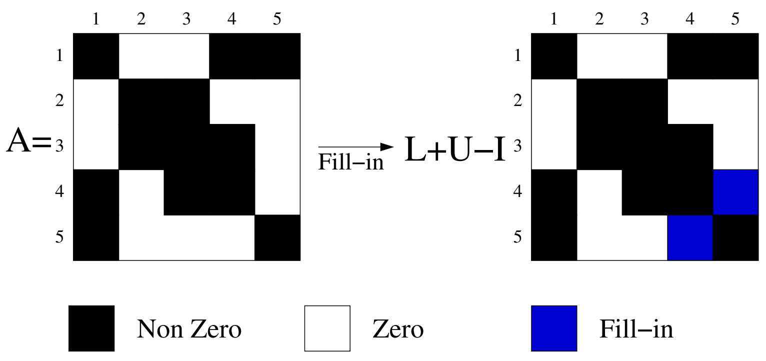
An out-of-core extension of a parallel sparse multifrontal solver

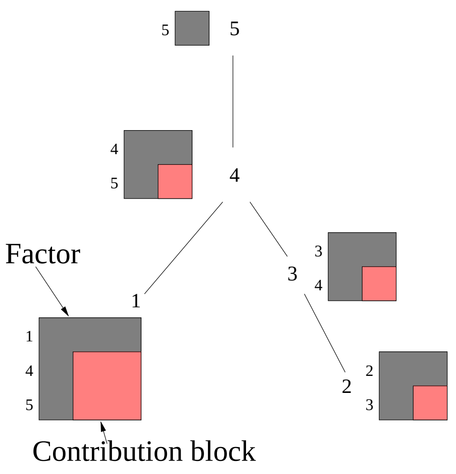
E. Agullo in collaboration with
A. Guermouche and J.-Y. L'Excellent

CSC05, 21-23 June 2005

Introduction

- The multifrontal method (Duff, Reid'83)

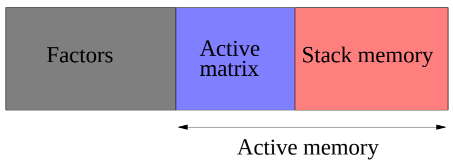




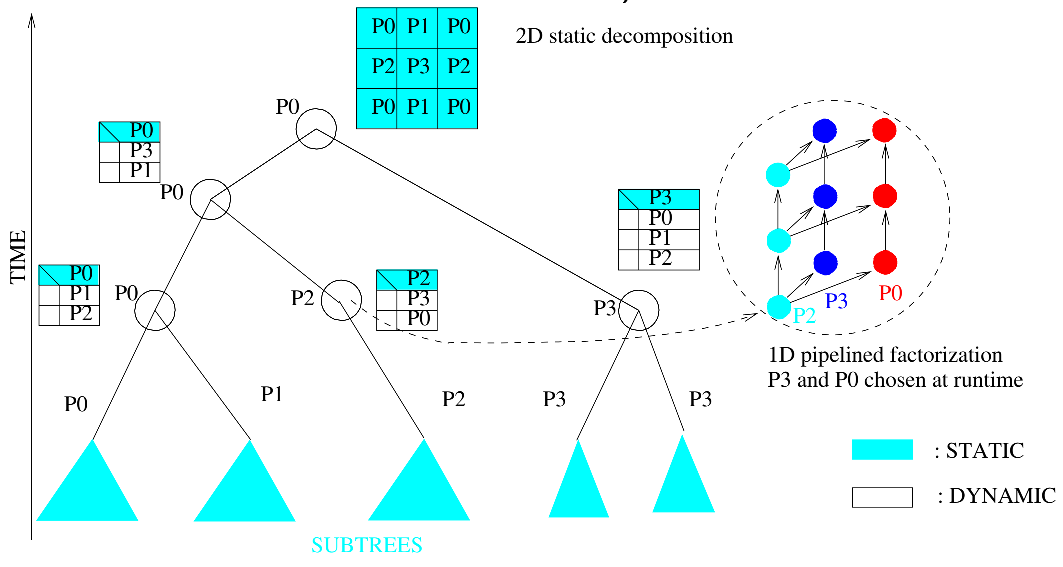
Dependency tree

Memory divided in two parts:

- Active memory
- Factors



- MUMPS: MUltifrontal Massively Parallel Solver (Amestoy, Duff, Guermouche, Koster, L'Excellent, Pralet, ...)



see http://graal.ens-lyon.fr/MUMPS

or http://www.enseeiht.fr/apo/MUMPS

State-of-the-art

- I/O tools:
  C/Fortran libraries (+ threads) ; AIO ; MPIIO ; FG
- Previous *out-of-core* approaches (sequential):

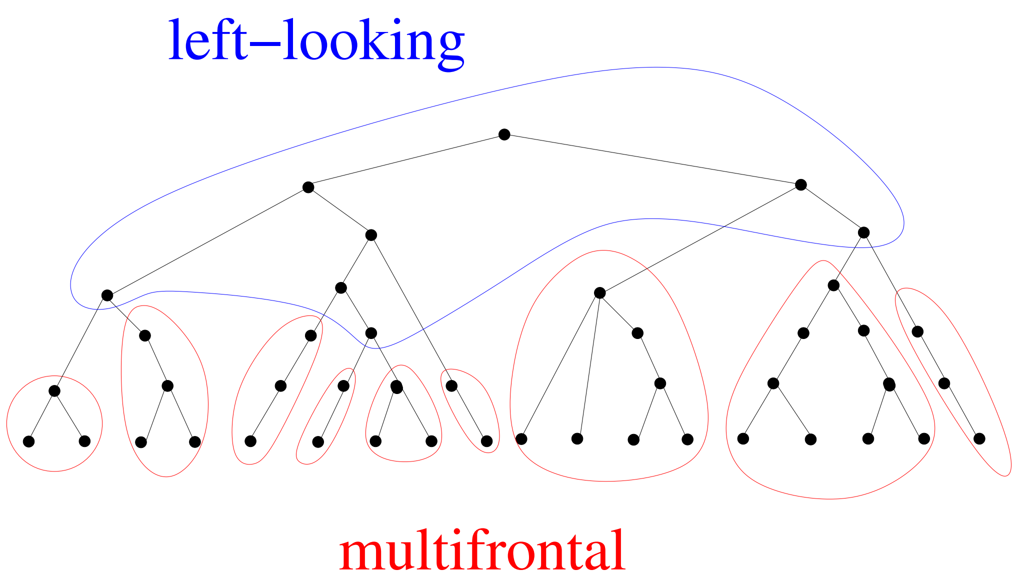


Figure: E. Rothberg and R. Schreiber, 1999

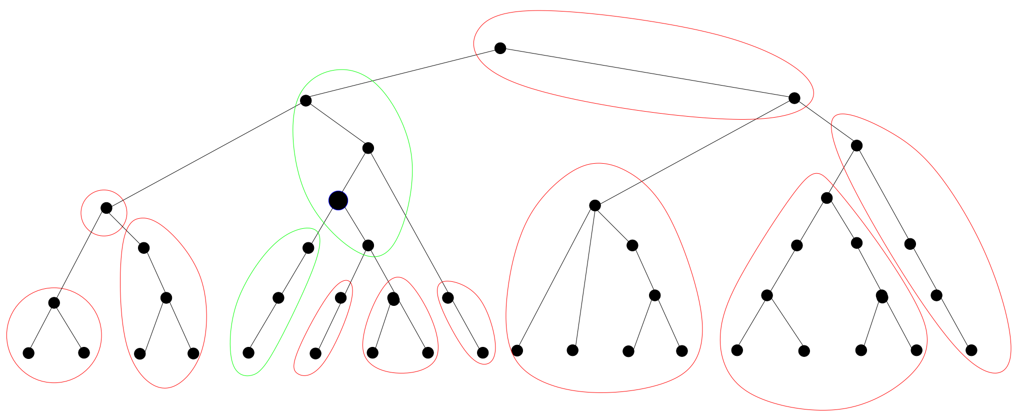


Figure: V. Rotkin and S.Toledo, 2004

Preliminary Study: Experimental Environment

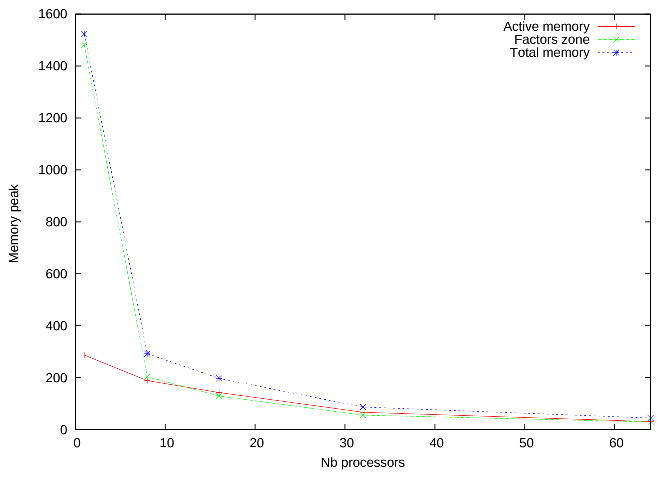
- MUMPS: Multifrontal Parallel Solver for both LU and LDL^T.
- Reordering techniques: `METIS`, `PORD`.
- Test platform: *IBM* platform at *IDRIS* (Orsay, France) composed of 4-way and 32-way Power4+ processors.
  Memory limits per processor:

Number of procs	1	2-16	17-64	65-
Max memory	16 GB	4GB	3.5GB	1.3GB

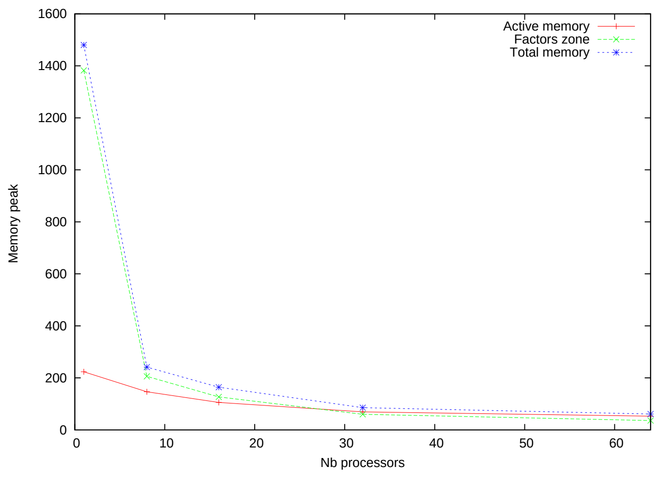
- Test problems: range of large matrices extracted from standard collections or provided by MUMPS users.
- Simulation of an *out-of-core* behaviour:
  - Free factors as soon as they are computed
  - Only factorization step is possible (factors are lost)
- Selected values: the bigger over all processors for :
  - The size of factors
  - The peak of active memory
  - The peak of total memory

Preliminary Study: Experimental Results

- Typical memory behaviour (`AUDIKW_1` matrix)

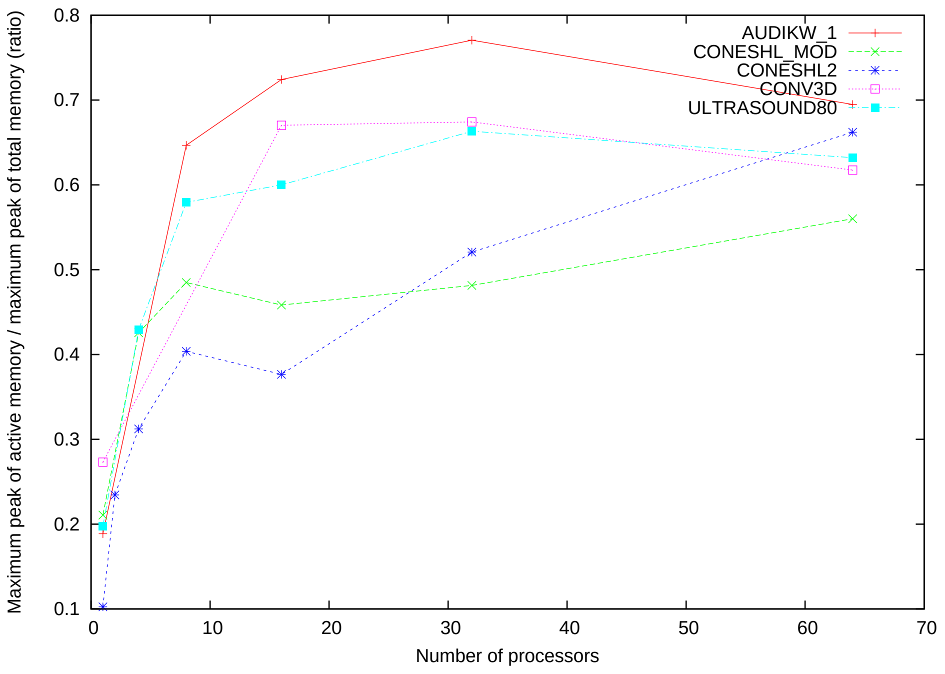


`METIS`



`PORD`

- Active memory / total memory ratio



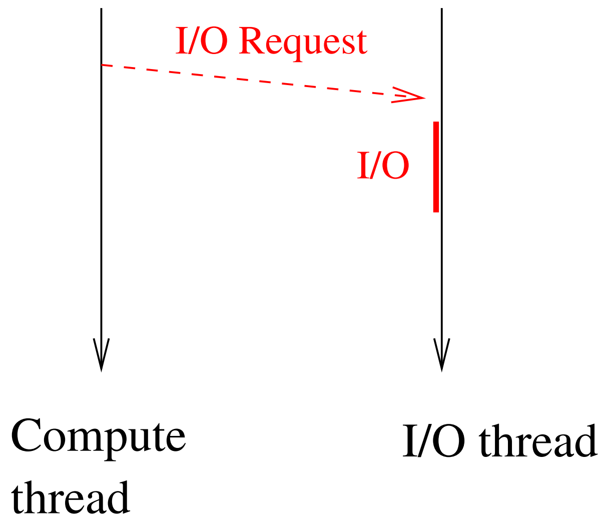
Consequence

- First step: factors *out-of-core* (well adapted for few processors)
- Second step: factors *and* stack *out-of-core* (largest problems or many processors)

Out-of-core Storage of the Factors

- Synchronous Version:
  - Use standard write operations
  - Factors are written to disk (possibly with low-level system buffering) as soon as they are computed
  - Solution step:
    1. Read a factor block
    2. Work with the factor

    ⇒ Factors may be read twice (forward elimination and backward substitution)
- Asynchronous Version:
  - Threaded version with buffers
  - Solution step: still synchronous



- Results: we can solve
  - bigger problems
  - same problems with less memory (cf preliminary study)

    example: ULTRASOUND80

	total mem per proc	active mem per proc
1 proc (16GB memory)	1101 million reals	218 million reals
4 procs	360 million reals	154 million reals

  - same problems with less processors

Matrix	Strategy	min procs
ULTRASOUND80	in-core	8
	out-of-core	2
CONV3D	in-core	32
	out-of-core	16

  - CONV3D on 1 proc with 16 GB memory: *out-of-core* version ok, *in-core* version runs out of memory

Preliminary Performance Analysis

- Same environment: IDRIS platform
- Ordering selected: METIS
- Different strategies:
  - synchronous I/O
  - asynchronous I/O with a buffer
  - in core
- Time for factorization: typical behaviour of the factorization step

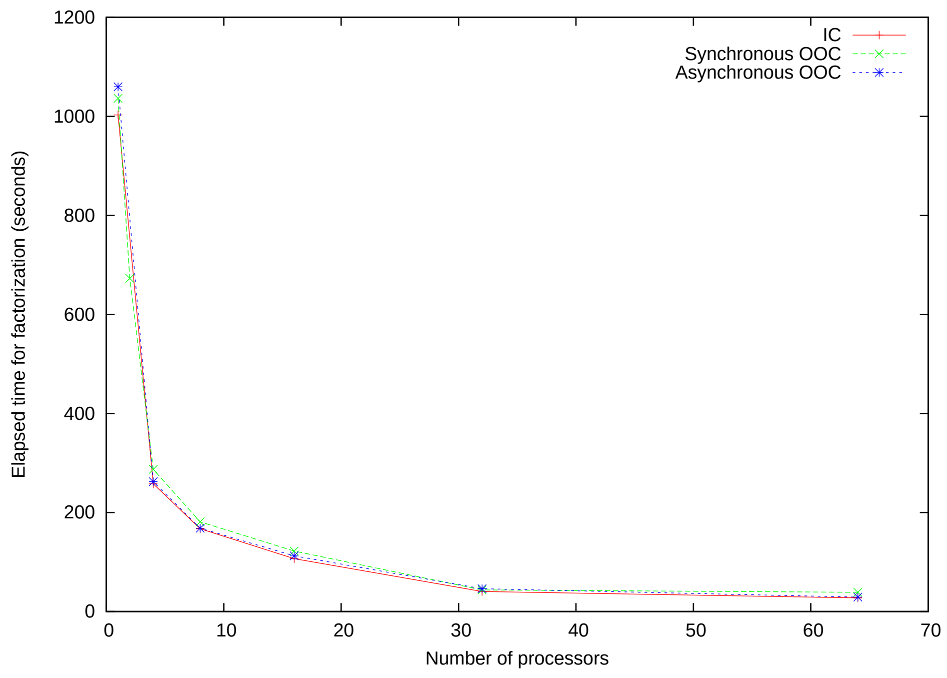
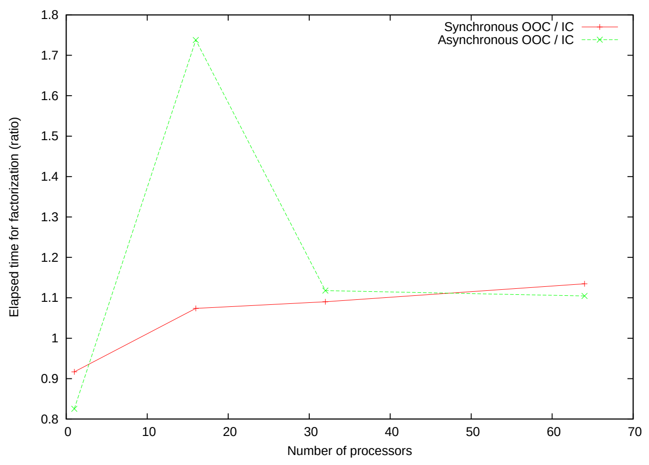
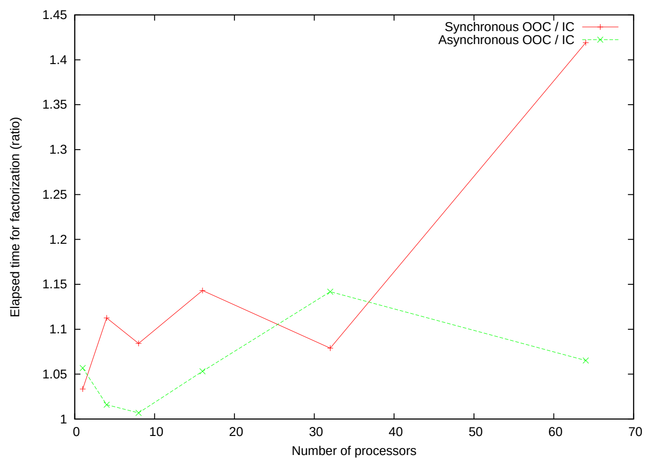


Figure: CONESHL_MOD matrix

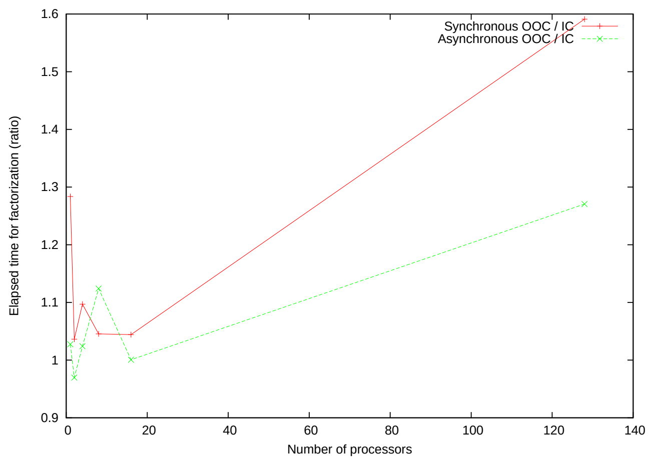
Current Performance Results



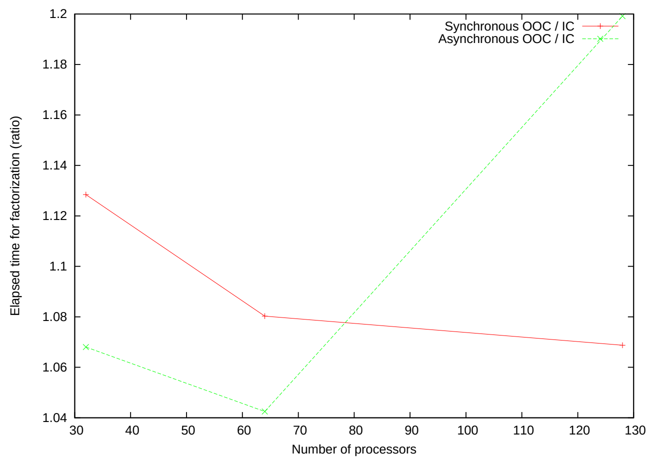
AUDIKW_1



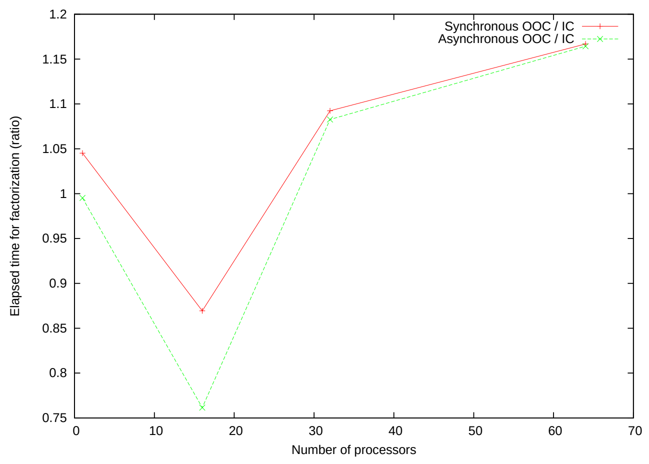
CONESHL_MOD



CONESHL2



CONV3D64



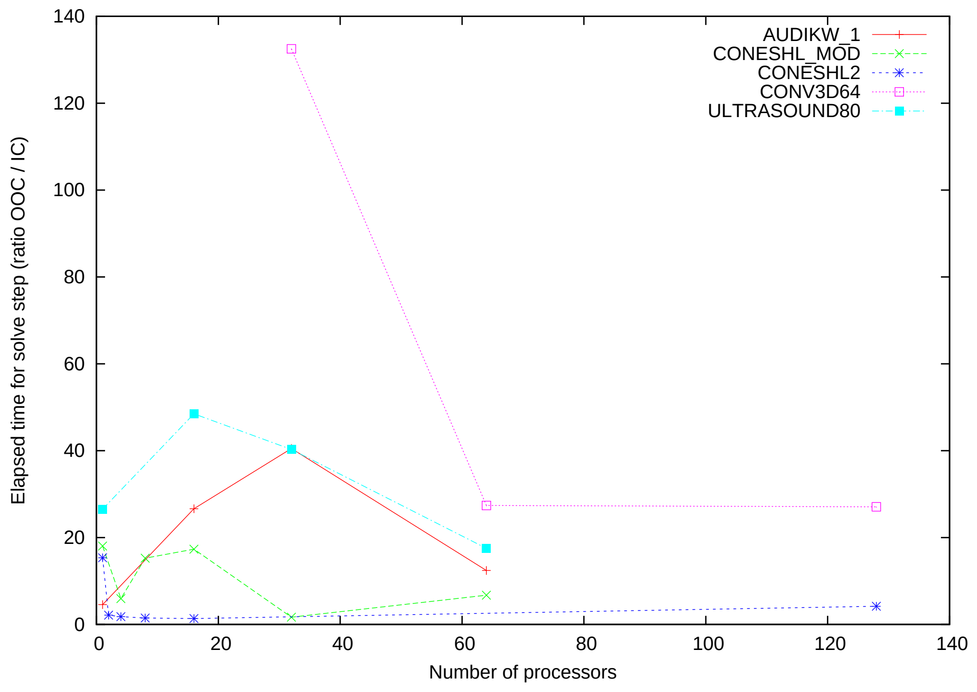
ULTRASOUND80

RED: $\frac{\textit{time synchronous version}}{\textit{time in-core}}$ GREEN: $\frac{\textit{time asynchronous version}}{\textit{time in-core}}$

Performance Analysis

- Some irregular behaviour observed
- Impact of buffer size
  - Larger than largest frontal matrix . . .
  - For large numbers of processors, buffer size too large
    ⇒ The last I/O request concerns a huge amount of factors in the asynchronous approach !!
- Impact of platform
  - No guarantee that each processor accesses its own disk...

  ⇒ Explains some of the surprising behaviour
- Impact of locality
  - In some cases, *out-of-core* version faster than *in-core* version !
  - Explanation: better memory locality (frontal matrix always in the same area of memory)
- Study of the solution step

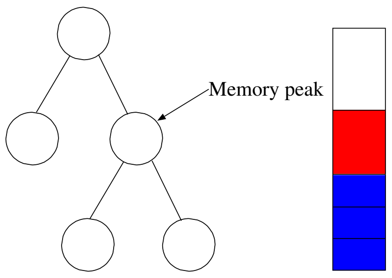


  - Solution time becomes critical: same order as factorization time
  - Improve I/O performance: increase granularity of I/O operations
  - Asynchronous prefetch mechanisms: requires
    - asynchronous I/O (thread)
    - more complex memory management (multiple or cyclic workspaces)
  - Separate L and U factors for unsymmetric matrices

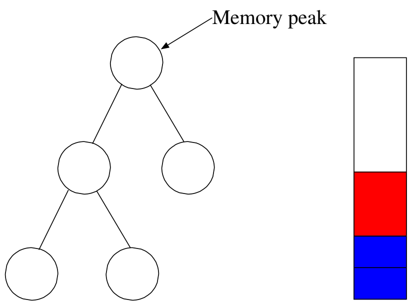
Out-of-core Active Memory (sequential case)

- We assume that the factors are written to disk as soon as computed $\Rightarrow$ need for an *out-of-core* active memory management (even more critical for the parallel case)

- The active memory highly depends on the tree traversal:



Worst case.

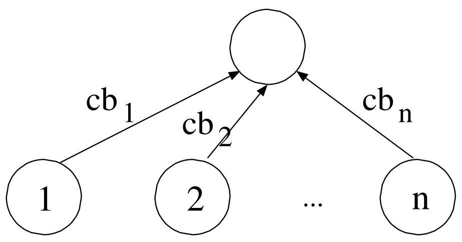


Best case.

- LIU'86 has proposed an optimal algorithm to minimize the peak of active memory

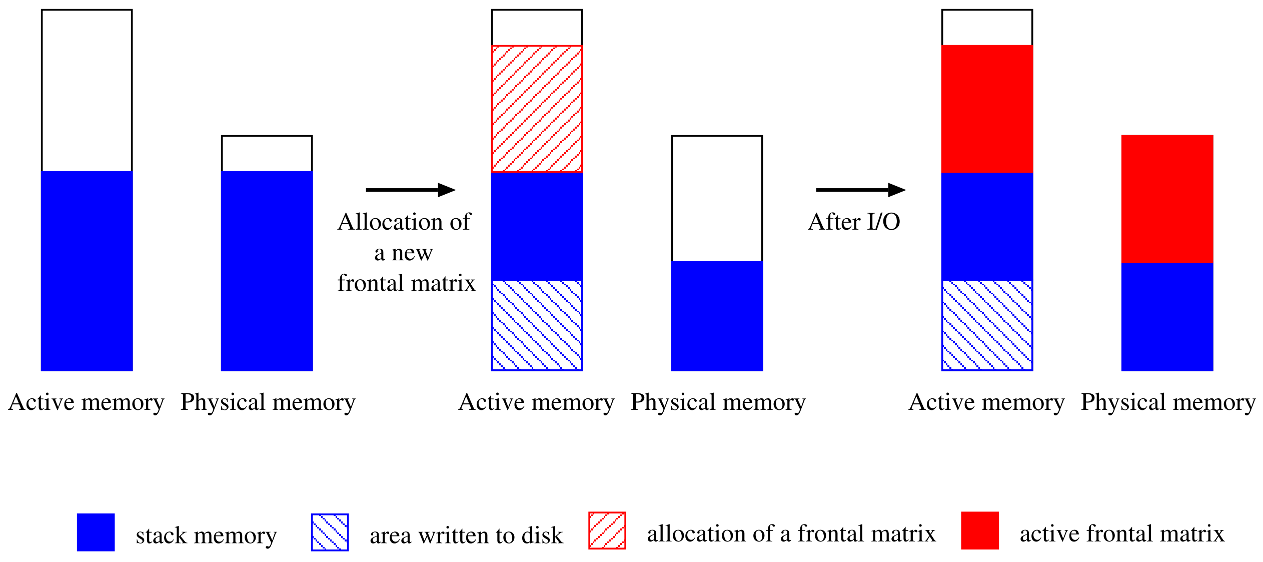
- Our theoretical study: how to minimize the volume of I/O when the active memory does not fit into a physical memory of size M_0 ?

Some Notations and Assumptions



- M_0 : Amount of memory allowed for factorization (e.g. physical memory of the target machine).
- n : Number of children.
- j : j^{th} child of the node.
- cb_j : Size of the contribution block of child j.
- F : Memory size of the parent's frontal matrix ($F < M_0$).
- M_i : Amount of memory needed to process alone an i-rooted subtree.

Assumptions:



- All frontal matrices fit in memory (i.e. $\forall i : F_i < M_0$)
- Factors are written to disk as soon as they are computed.
- Once extra I/O have to been done (I/O concerning contribution blocks), eldest contribution blocks are written first.

Memory Occupation & I/O Volume

Consider a parent node with its children.

Case 1: $\forall i \in$ set of children : $M_i < M_0$

The assembly step requires a storage:

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

The storage required to process child j is:

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

M_{parent} is thus defined by:

$$M_{\text{parent}} = \max\Big(\max_{j=1,n}(M_j + \sum_{k=1}^{j-1} cb_k), F + \sum_{j=1}^{n} cb_j\Big)$$

The assembly step requires an amount of I/O of :

$$\max(0, F + \sum_{j=1}^{n} cb_j - M_0)$$

The amount of I/O required to process child j is:

$$\max(0, M_j + \sum_{k=1}^{j-1} cb_k - M_0)$$

$V^{I/O}$ is thus defined by:

$$V^{I/O} = \max\Big(\max_{j=1,n}(M_j + \sum_{k=1}^{j-1} cb_k), F + \sum_{j=1}^{n} cb_j\Big) - M_0$$

Case 2: General case (M_i can be greater than M_0 for a given child).

- The expression for the active memory size does not change
- I/O volume needs to be recomputed:
  - If $M_i > M_0$ then :
    1. the amount of I/O $V_i^{I/O}$ is independent from the position of child i in the schedule
    2. all contribution blocks that were in memory have to be written to disk when i is processed

$$\Longrightarrow \quad V^{I/O} = \max\Big(\max_{j=1,n}(\min(M_j, M_0) + \sum_{k=1}^{j-1} cb_k), F + \sum_{j=1}^{n} cb_j\Big) - M_0 + \sum_{i=1}^{n} V_i^{I/O}$$

$\Longrightarrow$ I/O volume can be computed on the complete tree using a greedy bottom-up process

Minimizing the Volume of I/O

In which order shall we process children for minimizing $V^{I/O}$?

- Minimizing $V^{I/O}$ equivalent to minimize:

$$\max_{j=1,n}\left(\min(M_j, M_0) + \sum_{k=1}^{j-1} cb_k\right)$$

Theorem (Tree pebbling theorem)

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

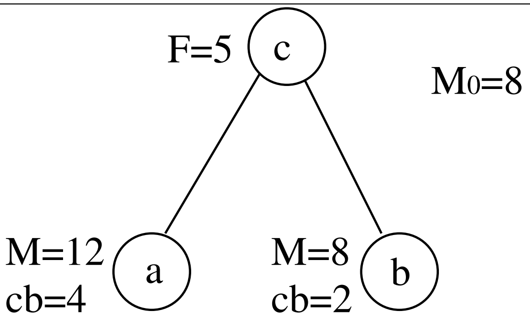
Consequence:

An optimal child sequence is obtained by rearranging the children nodes in decreasing order of $\min(M_j, M_0) - cb_j$

Algorithm:

- Bottom-up greedy process
- Apply Tree pebbling theorem at each level of the tree

Toy example:

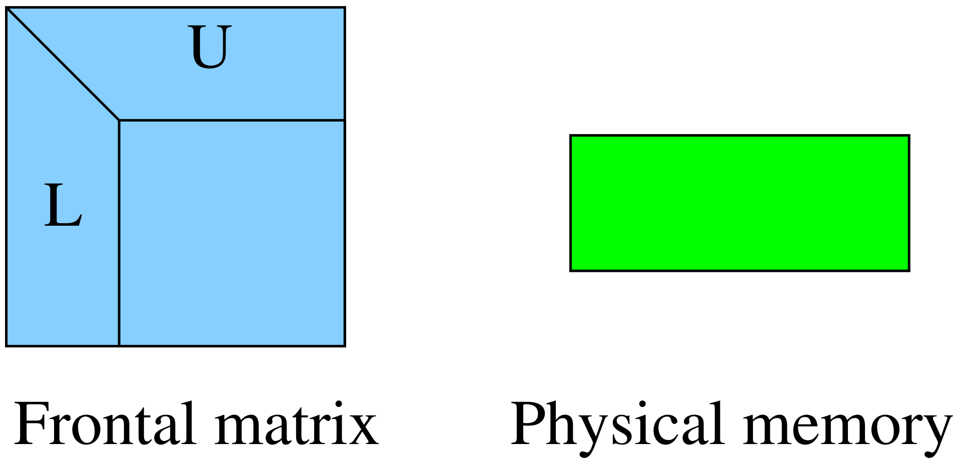
	Liu's Algorithm	I/O minimization Algorithm
	sequence a-b-c $V^{I/0}$ 8	sequence b-a-c $V^{I/0}$ 7

Experimental results:

5% decrease of I/O volume on real-life test cases compared to Liu's algorithm.

Large Active Frontal Matrices

- In the multifrontal method, frontal matrices can be larger than physical memory available (memory bottleneck for sequential codes):



- Previous techniques: use (hybrid) left-looking approaches:
  - [Rothberg, Schreiber 99]
  - [Toledo 2004]

- Multifrontal approach (future work):
  - Out-of-core factorization (and assembly) of frontal matrices (panel by panel)
  - Parallel approach to this sequential bottleneck:
    Use many processors for large frontal matrices that appear at the top of the tree.

Future Work

- Prefetching techniques to read the factors at the (parallel) solve step
- Matrices that *almost* fit in memory:
  Keep most of the factors in memory after factorization step
- *Out-of-core* stack memory in the parallel case:
  The stack memory is not exactly accessed in LIFO order in the dynamic distributed case $\Rightarrow$ Find good heuristics to write / prefetch contribution blocks
- Scheduling strategies for parallel *out-of-core* factorization and solution steps
- Flexible allocation of the parent node:
  How to extend [Guermouche,L'Excellent'05] (memory-minimizing schedules) to minimize the volume of I/O ?
- Impact of reordering, amalgamation, node splitting, ... on I/O volume
- Robust implementation in MUMPS (currently, only the factors are *out-of-core* in an experimental version)

Test Problems

- AUDIKW_1 comes from Automotive crankshaft model with over 900,000 TETRA elements (available in the PARASOL collection)
- CONESHL[_MOD|2] come from 3D finite element problems (cone with shell and solid element connected by linear constraints with Lagrange multiplier technique). These two matrices were provided by SAMTECH and created using SAMCEF
- CONV3D64 has been provided by CEA-CESTA and was generated using AQUILON (http ://www.enscpb.fr/master/aquilon)
- ULTRASOUND80 comes from propagation of 3D ultrasound waves and has been provided by Masha Sosonkina

	Order	nnz	$nnz(L\|U) \times 10^6$	Ops$\times 10^9$
Symmetric matrices				
AUDIKW_1	943695	39297771	1368.6	5682
CONESHL_MOD	1262212	43007782	790.8	1640
CONESHL2	837967	22328697	266.6	218.5
Unsymmetric matrices				
CONV3D64	836550	12548250	2693.9	23880
ULTRASOUND80	531441	33076161	981.4	3915

Statistics with METIS