

Parallel multifrontal method with out-of-core techniques

A. Guermouche

Univ. Bordeaux 1 and INRIA
Bordeaux, France

Outline

Context

The multifrontal method

Concepts

preliminary study

Out-of-core factorization step

Out-of-core solution step

Operating system I/O mechanisms

Direct I/O

Concluding remarks

Outline

Context

The multifrontal method

Concepts

preliminary study

Out-of-core factorization step

Out-of-core solution step

Operating system I/O mechanisms

Direct I/O

Concluding remarks

Context

Solve $Ax = b$ where A is large and sparse

Parallel Multifrontal Algorithm: Sparse direct method for matrix factorization based on a tree structure

- ▶ Efficiency due to good locality (level 3 BLAS)
- ▶ Good potential for parallelism
- ▶ Numerical robustness (pivoting)
- ▶ Large memory requirements (compared to iterative methods).

→ Design/use *out-of-core* techniques in the context of the parallel multifrontal method

Outline

Context

The multifrontal method

Concepts

preliminary study

Out-of-core factorization step

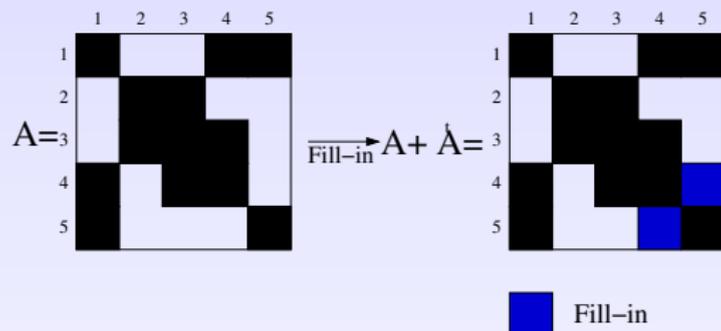
Out-of-core solution step

Operating system I/O mechanisms

Direct I/O

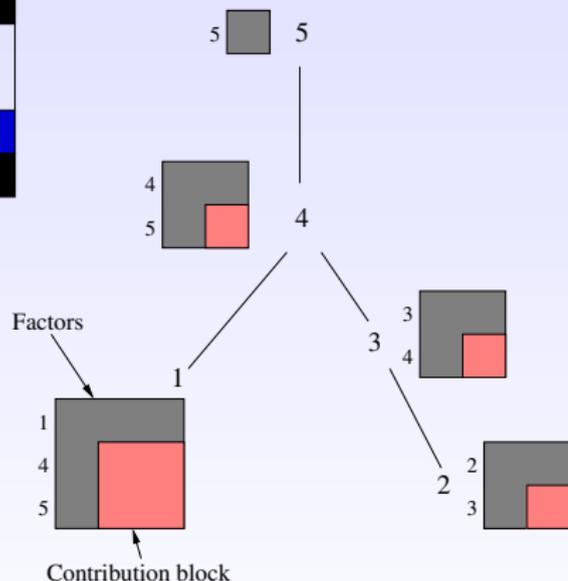
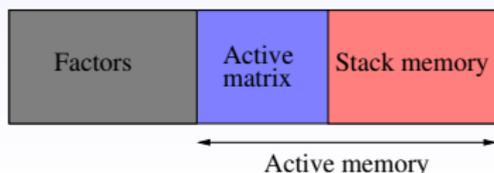
Concluding remarks

The multifrontal method (Duff, Reid'83)



Memory divided into two parts:

- ▶ Active memory
- ▶ Factors



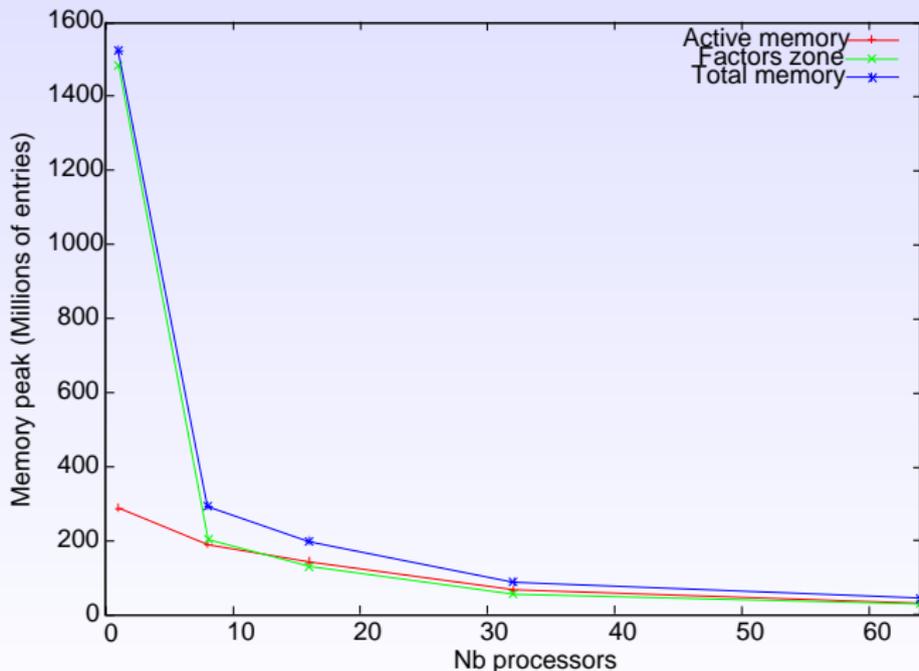
Dependency tree

Preliminary Study: Experimental Environment

- ▶ **MUMPS**: Multifrontal Parallel Solver for both LU and LDL^T .
- ▶ **Reordering technique**: METIS.
- ▶ **Test platform**:
 - ▶ *IBM* platform at *IDRIS* (Orsay, France) composed of 4-way and 32-way Power4+ processors.
 - ▶ *Cray XD1* system at *CERFACS* (Toulouse, France), composed of 48 2-way nodes with 4 GB of memory per node.
- ▶ **Test problems**: range of large matrices extracted from standard collections or provided by MUMPS users.
- ▶ **Simulation of an *out-of-core* behavior**:
 - ▶ 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

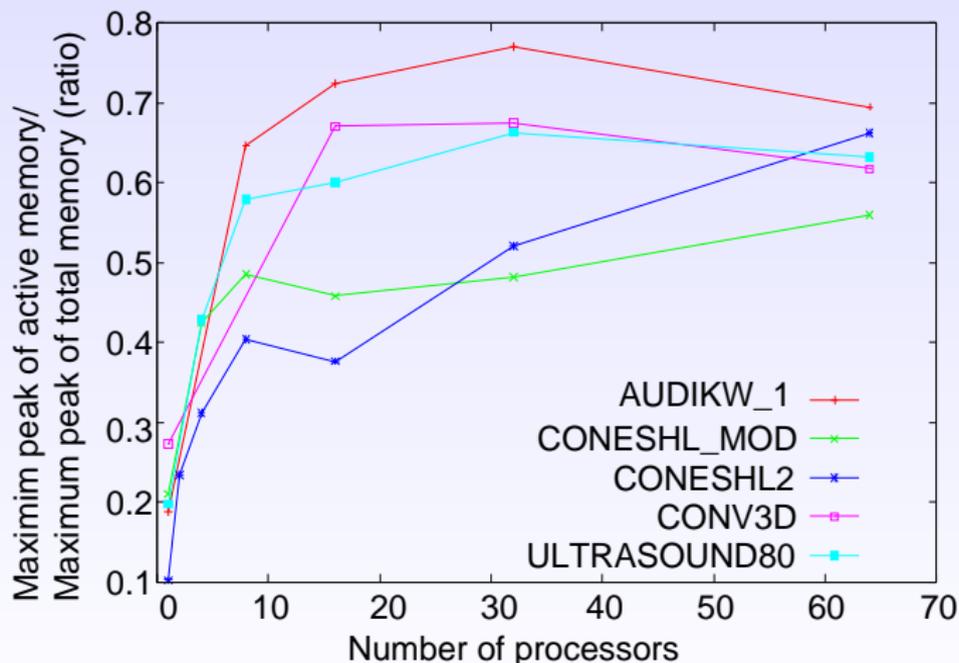
Preliminary Study: Experimental Results

Typical memory behavior : (AUDIKW_1 matrix) with METIS



Preliminary Study: Experimental Results

Typical memory behavior : Active memory / total memory ratio



Outline

Context

The multifrontal method

Concepts

preliminary study

Out-of-core factorization step

Out-of-core solution step

Operating system I/O mechanisms

Direct I/O

Concluding remarks

Out-of-core factorization (Phd of E. Agullo)

Out-of-core storage of factors :

→ write factor to disk as soon as they are computed.

Out-of-core factorization (Phd of E. Agullo)

Out-of-core storage of factors :

→ write factor to disk as soon as they are computed.

Synchronous Version:

- ▶ Use standard write operations
- ▶ Factors are written to disk as soon as they are computed

Out-of-core factorization (Phd of E. Agullo)

Out-of-core storage of factors :

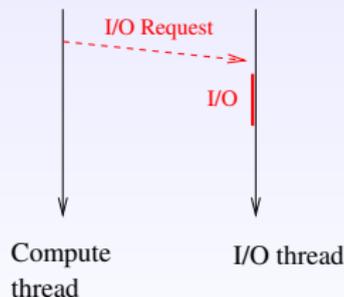
→ write factor to disk as soon as they are computed.

Synchronous Version:

- ▶ Use standard write operations
- ▶ Factors are written to disk as soon as they are computed

Asynchronous Version:

- ▶ Copy factors to a user buffer as soon as they are computed
- ▶ A dedicated I/O thread writes factors from the user buffer to disk



Out-of-core factorization (Phd of E. Agullo)

Out-of-core storage of factors :

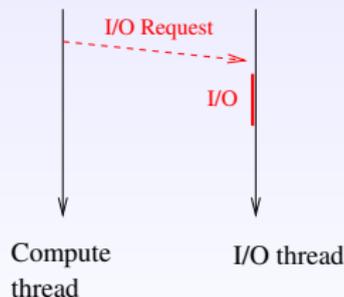
→ write factor to disk as soon as they are computed.

Synchronous Version:

- ▶ Use standard write operations
- ▶ Factors are written to disk as soon as they are computed

Asynchronous Version:

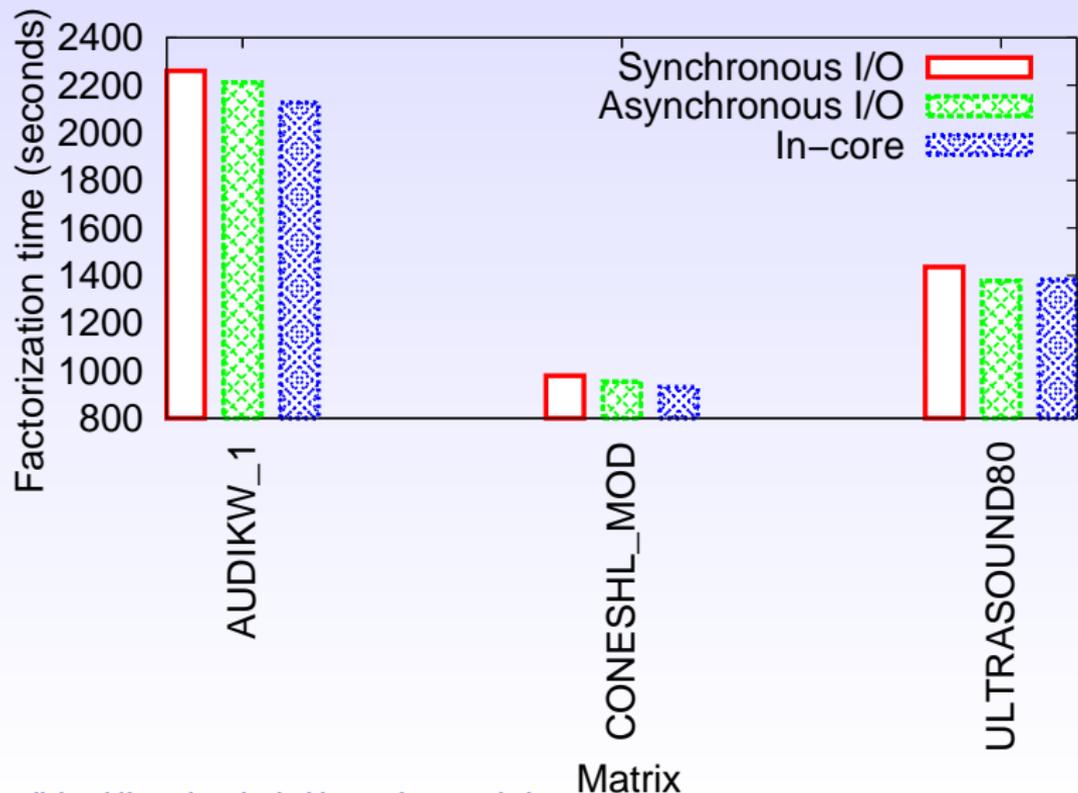
- ▶ Copy factors to a user buffer as soon as they are computed
- ▶ A dedicated I/O thread writes factors from the user buffer to disk



Next step → factors *and* stack *out-of-core* (largest problems or many processors)

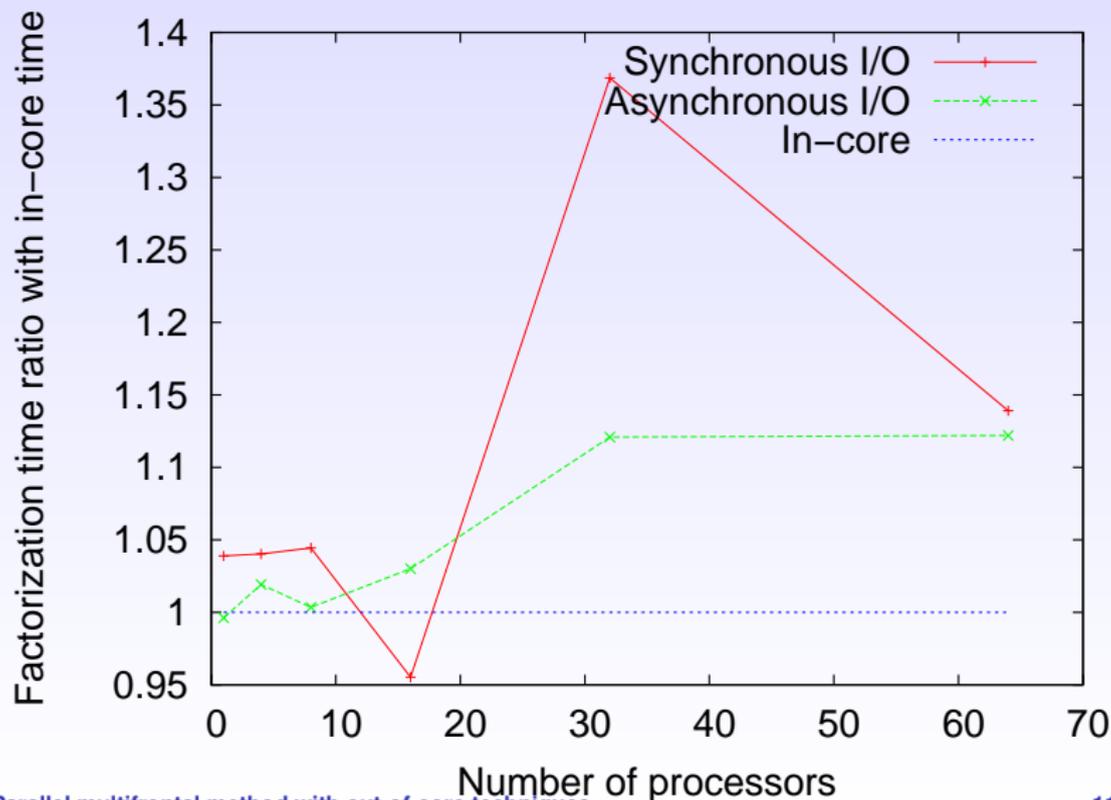
Preliminary results

Sequential factorization:



Preliminary results

Parallel factorization: (ULTRASOUND80 matrix)



Outline

Context

The multifrontal method

Concepts

preliminary study

Out-of-core factorization step

Out-of-core solution step

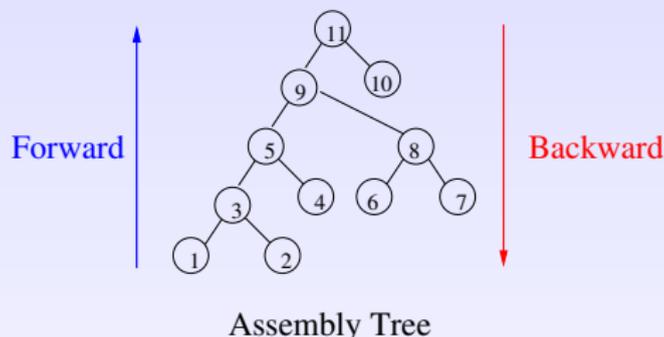
Operating system I/O mechanisms

Direct I/O

Concluding remarks

Solution step

Solution step \rightarrow solve the given system using the factored matrix.



Sequential case:

- ▶ forward step(Fwd): postordering as in the factorization phase
- ▶ backward step(Bwd): in the reverse order

Parallel case:

- ▶ no guarantee of the order in which the nodes are processed

Out-of-core Solution step (Phd of T. Slavova)

Assumptions:

- ▶ During factorization **ALL** factors are written to local disks
- ▶ No factors are kept in memory at the beginning of the solution step

How to load efficiently data from disk?

- ▶ Each factor-block is loaded only once
- ▶ User control of number and size of buffers
- ▶ One Emergency buffer (EMG), to hold largest front (demand driven)
- ▶ Other buffers used to automatically prefetch data with a look-ahead mechanism

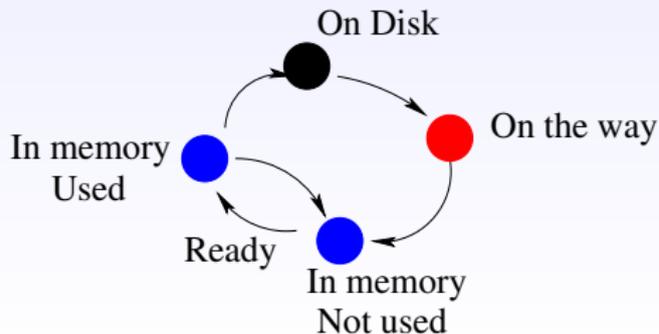
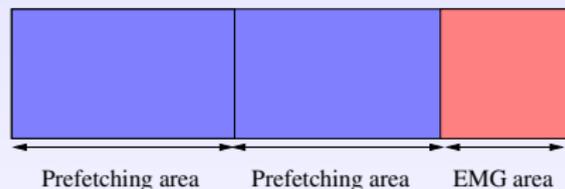
Out-of-core Solution step (Phd of T. Slavova)

Assumptions:

- ▶ During factorization **ALL** factors are written to local disks
- ▶ No factors are kept in memory at the beginning of the solution step

How to load efficiently data from disk?

- ▶ Each factor-block is loaded only once
- ▶ User control of number and size of buffers
- ▶ One Emergency buffer (EMG), to hold largest front (demand driven)
- ▶ Other buffers used to automatically prefetch data with a look-ahead mechanism



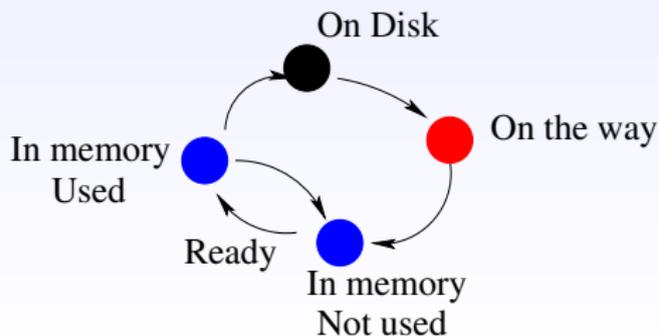
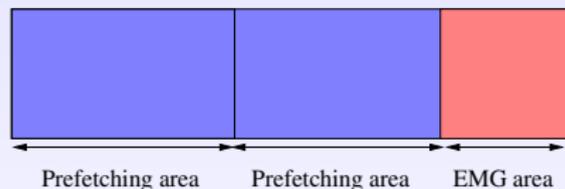
Out-of-core Solution step (Phd of T. Slavova)

Assumptions:

- ▶ During factorization **ALL** factors are written to local disks
- ▶ No factors are kept in memory at the beginning of the solution step

How to load efficiently data from disk?

- ▶ Each factor-block is loaded only once
- ▶ User control of number and size of buffers
- ▶ One Emergency buffer (EMG), to hold largest front (demand driven)
- ▶ Other buffers used to automatically prefetch data with a look-ahead mechanism



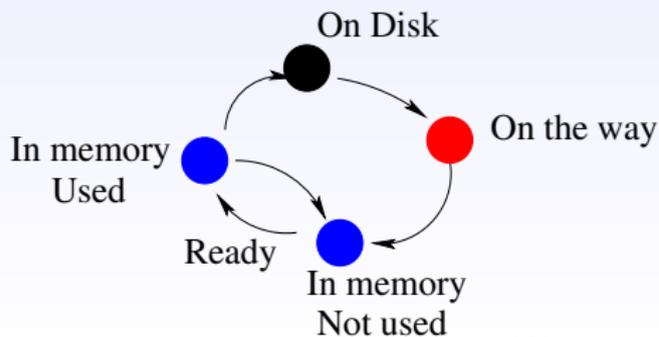
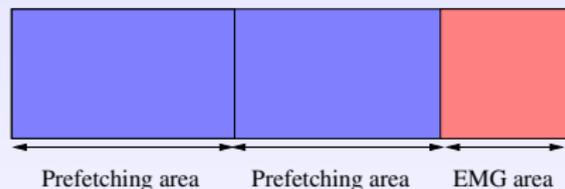
Out-of-core Solution step (Phd of T. Slavova)

Assumptions:

- ▶ During factorization **ALL** factors are written to local disks
- ▶ No factors are kept in memory at the beginning of the solution step

How to load efficiently data from disk?

- ▶ Each factor-block is loaded only once
- ▶ User control of number and size of buffers
- ▶ One Emergency buffer (EMG), to hold largest front (demand driven)
- ▶ Other buffers used to automatically prefetch data with a look-ahead mechanism



Preliminary results

Time needed for solution step:

Preliminary results

Time needed for solution step:

- ▶ “Small problem” : Grid 300-100-10

	Factor	Solve		
	Time (sec)	Fwd (sec)	Bwd (sec)	Disk access (MB/s)
In-core	34.9	0.4	0.4	-
OOO	34.9	1.3	1.2	616

Preliminary results

Time needed for solution step:

- ▶ “Small problem” : Grid 300-100-10

	Factor	Solve		
	Time (sec)	Fwd (sec)	Bwd (sec)	Disk access (MB/s)
In-core	34.9	0.4	0.4	-
OOO	34.9	1.3	1.2	616

- ▶ “large problem” : Qimonda07

	Factor	Solve		
	Time (sec)	Fwd (sec)	Bwd (sec)	Disk access (MB/s)
In-core	40.4	0.9	0.9	-
OOO	191.0	186.4	207.7	13

Preliminary results

Time needed for solution step:

- ▶ “Small problem” : Grid 300-100-10

	Factor	Solve		
	Time (sec)	Fwd (sec)	Bwd (sec)	Disk access (MB/s)
In-core	34.9	0.4	0.4	-
OOO	34.9	1.3	1.2	616

- ▶ “large problem” : Qimonda07

	Factor	Solve		
	Time (sec)	Fwd (sec)	Bwd (sec)	Disk access (MB/s)
In-core	40.4	0.9	0.9	-
OOO	191.0	186.4	207.7	13

Outline

Context

The multifrontal method

Concepts

preliminary study

Out-of-core factorization step

Out-of-core solution step

Operating system I/O mechanisms

Direct I/O

Concluding remarks

I/O Mechanisms

read and write operations use a *cache* mechanism (*page cache*)

- ▶ For each call to `read` or `write`, data is kept in the page cache at the kernel level
- ▶ User doesn't know when data is "really" written to disk (unless by explicit synchronization)
- ▶ User has no control on the size of the page cache
- ▶ The page cache is usually managed with a LRU scheme

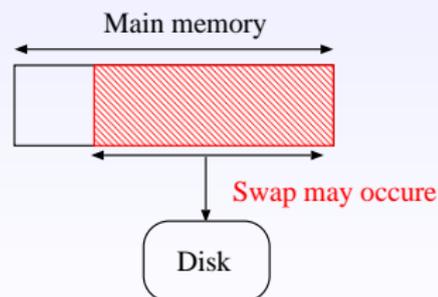
I/O Mechanisms

read and write operations use a *cache* mechanism (*page cache*)

- ▶ For each call to `read` or `write`, data is kept in the page cache at the kernel level
- ▶ User doesn't know when data is “really” written to disk (unless by explicit synchronization)
- ▶ User has no control on the size of the page cache
- ▶ The page cache is usually managed with a LRU scheme

In our context, page cache can be **dangerous**.

- ▶ I/O may not have the same speed (depending on whether disk is accessed or not)
- ▶ The kernel may dramatically slowdown the performance of I/O's.



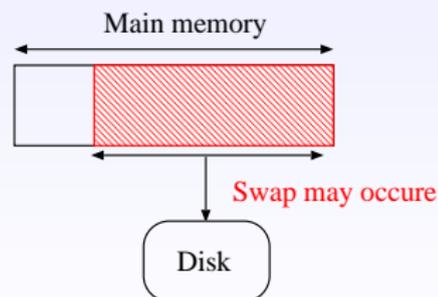
I/O Mechanisms

read and write operations use a *cache* mechanism (*page cache*)

- ▶ For each call to `read` or `write`, data is kept in the page cache at the kernel level
- ▶ User doesn't know when data is “really” written to disk (unless by explicit synchronization)
- ▶ User has no control on the size of the page cache
- ▶ The page cache is usually managed with a LRU scheme

In our context, page cache can be **dangerous**.

- ▶ I/O may not have the same speed (depending on whether disk is accessed or not)
- ▶ The kernel may dramatically slowdown the performance of I/O's.



⇒ Use of **direct** I/O mechanisms

Direct I/O scheme

Advantages:

- ▶ Data is directly written to disk (data is not copied in the page cache)
- ▶ Very efficient I/O operations

Drawbacks:

- ▶ A disk access is made at each call to `read` or `write`
- ▶ Data needs to be aligned in memory

Direct I/O scheme \Rightarrow Use of more sophisticated algorithms but ensures robustness.

Direct I/O scheme

Advantages:

- ▶ Data is directly written to disk (data is not copied in the page cache)
- ▶ Very efficient I/O operations

Drawbacks:

- ▶ A disk access is made at each call to `read` or `write`
- ▶ Data needs to be aligned in memory

Direct I/O scheme \Rightarrow Use of more sophisticated algorithms but ensures robustness.

Preliminary results: Factorization time (seconds)

	Direct I/O Sync.	Direct I/O Async.	P.C. Sync.	P.C. Async.	<i>in-core</i>
AUDIkw_1	2417.1	2217.3	2260.8	2211.3	2126.4
CONESHL_MOD	995.6	967.2	979.2	953.6	930.4
CONV3D64	10826.9	7599.4	8078.4	7981.6	-
ULTRASOUND80	1446.9	1389.8	1436.4	1377.3	1382.5

Direct I/O scheme

Advantages:

- ▶ Data is directly written to disk (data is not copied in the page cache)
- ▶ Very efficient I/O operations

Drawbacks:

- ▶ A disk access is made at each call to `read` or `write`
- ▶ Data needs to be aligned in memory

Direct I/O scheme \Rightarrow Use of more sophisticated algorithms but ensures robustness.

Preliminary results: Factorization time (seconds)

	Direct I/O Sync.	Direct I/O Async.	P.C. Sync.	P.C. Async.	<i>in-core</i>
AUDIkw_1	2417.1	2217.3	2260.8	2211.3	2126.4
CONESHL_MOD	995.6	967.2	979.2	953.6	930.4
CONV3D64	10826.9	7599.4	8078.4	7981.6	-
ULTRASOUND80	1446.9	1389.8	1436.4	1377.3	1382.5

Direct I/O scheme

Advantages:

- ▶ Data is directly written to disk (data is not copied in the page cache)
- ▶ Very efficient I/O operations

Drawbacks:

- ▶ A disk access is made at each call to `read` or `write`
- ▶ Data needs to be aligned in memory

Direct I/O scheme \Rightarrow Use of more sophisticated algorithms but ensures robustness.

Preliminary results: Time for solution step (Qimonda07 matrix)

	Forward	Backward
Direct I/O (Demand-driven)	1149.2	1279.2
Direct I/O (Look-ahead)	174.0	183.7
P.C. (Demand-driven)	186.4	207.7

Outline

Context

The multifrontal method

Concepts

preliminary study

Out-of-core factorization step

Out-of-core solution step

Operating system I/O mechanisms

Direct I/O

Concluding remarks

Concluding remarks & future work

- ▶ First implementation of an *out-of-core* extension of the MUMPS solver
- ▶ Efficient *out-of-core* factorization
- ▶ Good performance of solution step

Ongoing work:

Factorization step: (Phd. of E. Agullo)

- ▶ *out-of-core* management of the contribution blocks
- ▶ design new scheduling strategies adapted to the *out-of-core* context
- ▶ work on algorithms to minimize the I/O volume

Solution step: (Phd. of T. Slavova)

- ▶ design specific scheduling/prefetching algorithms for the parallel solution step
- ▶ study the case of linear systems with multiple right-hand sides