

Patrick AMESTOY

Iain DUFF

Abdou GERMOUCHE

Tzvetomila SLAVOVA

(in collaboration with Jean-Yves L'EXCELLENT)

A Preliminary Analysis of the Out-Of-Core Solution Phase of a Parallel Multifrontal Approach

PMAA – IRISA

Rennes 2006

CERFACS, Toulouse

Overview

- Introduction
- System Based Method
- Direct IO Method
- Concluding Remarks

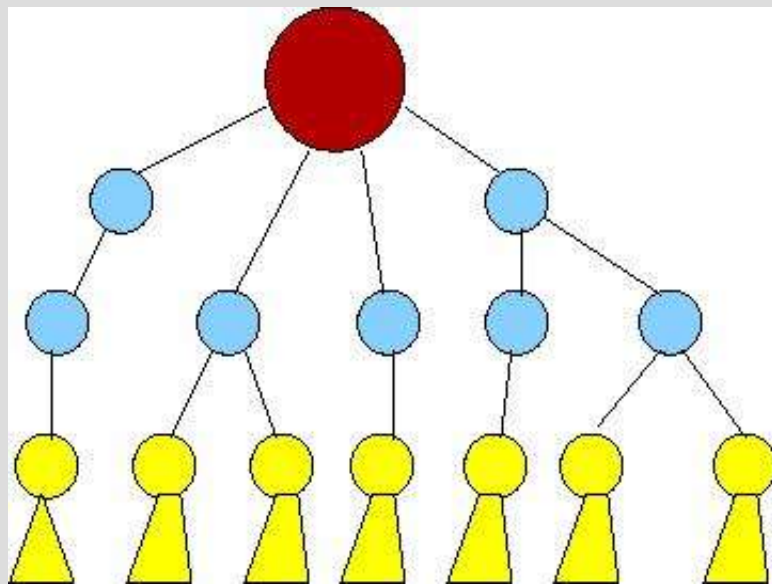
Objective: Reduce time for solution of $Ax=b$
in a parallel limited memory environment

Context:

- Direct Method $A=LU$ or $A=LDL^T$
- Distributed Memory Parallel Solution
- Factors stored on disk (Out-Of-Core, OOC)
- MUMPS (distributed memory multifrontal solver)

Parallelism in MUMPS Solver

3 levels of parallelism for both factorization and solve phases



- 2D block cyclic parallelism
- irregular 1D decomposition
- sequential processing of subtree

Introduction

System Based Method

Direct IO Method

Concluding Remarks

For a symmetric case: $Ax=b \Rightarrow LD y = b$ (forward)
 $L^T x = y$ (backward)

Introduction

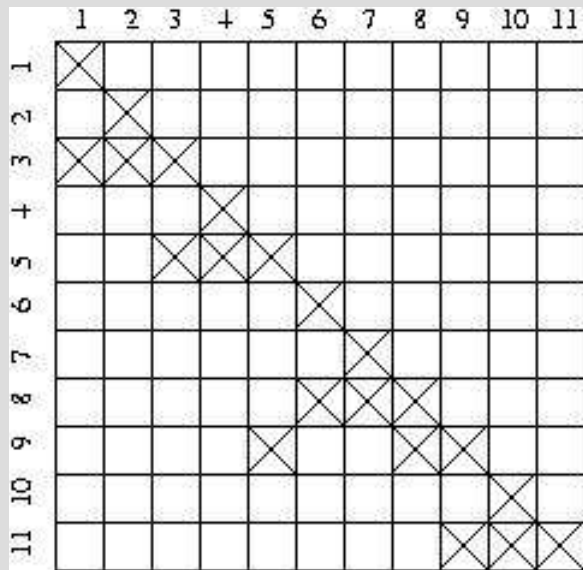
System Based Method

Direct IO Method

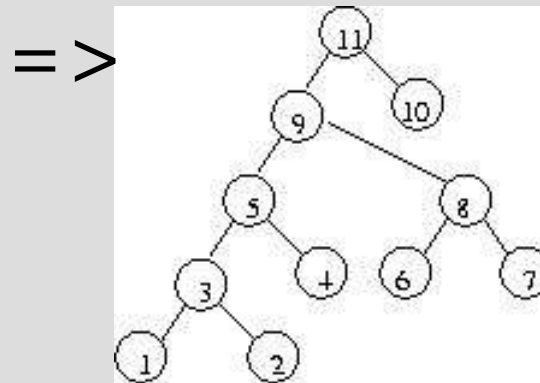
Concluding Remarks

For a symmetric case: $Ax=b \Rightarrow LD y = b$ (forward)
 $L^T x = y$ (backward)

matrix pattern



assembly tree

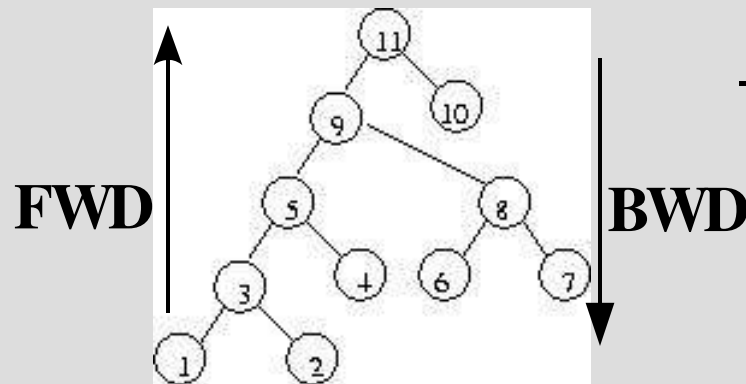


| Hard Disk | | | | | | | | | | |
|-----------|---|---|---|---|---|---|---|---|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Tree traversal during solve: for the **sequential case**

assembly tree

--> forward step(Fwd):
postordering as in the
factorization phase



--> backward step(Bwd):
in the reverse order

Tree traversal during solve:

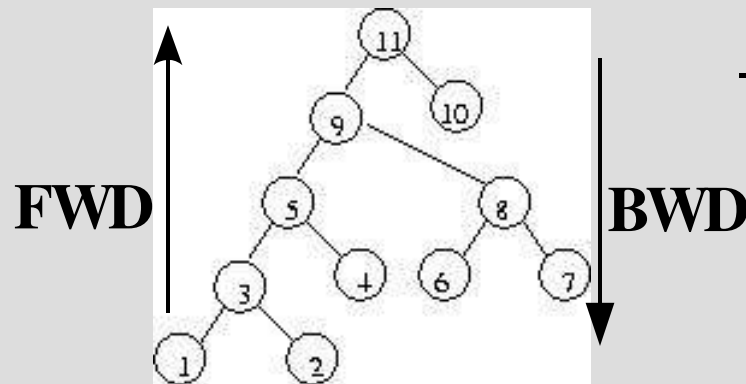
for the **parallel case: no guarantee of the order**

in which the nodes are processed

--> Fwd: topological ordering

(not necessarily postordering)

assembly tree



--> Bwd:

top-down traversal

Implementation issue: the **pool of tasks**

definition: list of all tasks ready to be executed

(In core: Last-In-First-Out strategy to extract tasks)

Out-Of-Core issues:

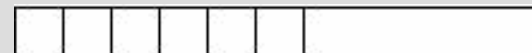
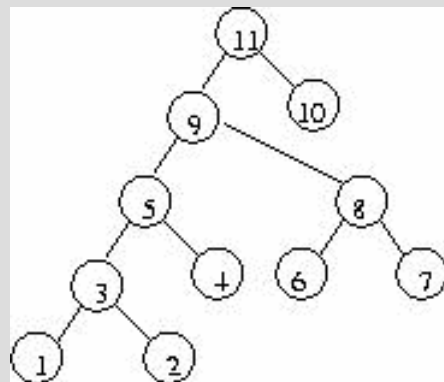
How to process the pool

to have a reasonably "regular" access to disk in both

- sequential and
- parallel context

Illustration: sequential processing of the tree

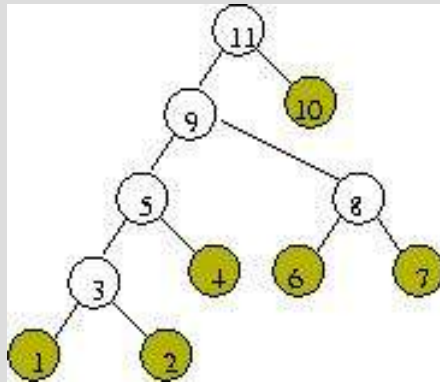
Pool at the beginning of FWD



Hard Disk

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|----|----|

Illustration: sequential processing of the tree



Pool at the beginning of FWD

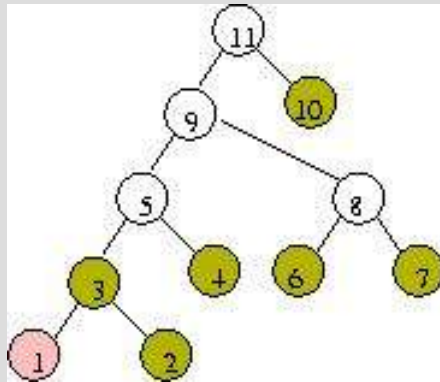
I step

| | | | | | | |
|----|---|---|---|---|---|--|
| 10 | 7 | 6 | 4 | 2 | 1 | |
|----|---|---|---|---|---|--|

Hard Disk

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|----|----|

Illustration: sequential processing of the tree



Pool at the beginning of FWD

II step



Hard Disk

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|----|----|

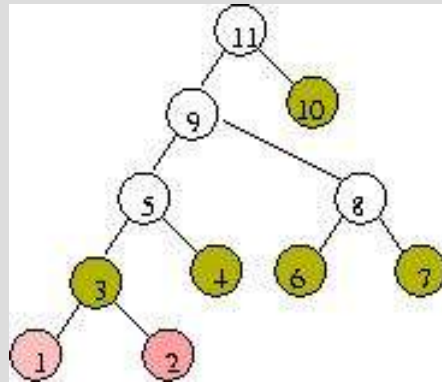
Introduction

System Based Method

Direct IO Method

Concluding Remarks

Illustration: sequential processing of the tree



Pool at the beginning of FWD

I - II step

| | | | | | | | | | |
|----|---|---|---|---|---|--|--|--|--|
| 10 | 7 | 6 | 4 | 2 | 1 | | | | |
|----|---|---|---|---|---|--|--|--|--|

III step

| | | | | | | | | | |
|----|---|---|---|---|--|--|--|--|--|
| 10 | 7 | 6 | 4 | 3 | | | | | |
|----|---|---|---|---|--|--|--|--|--|

FWD



Hard Disk

| | | | | | | | | | | | |
|--|---|---|---|---|---|---|---|---|---|----|----|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|--|---|---|---|---|---|---|---|---|---|----|----|

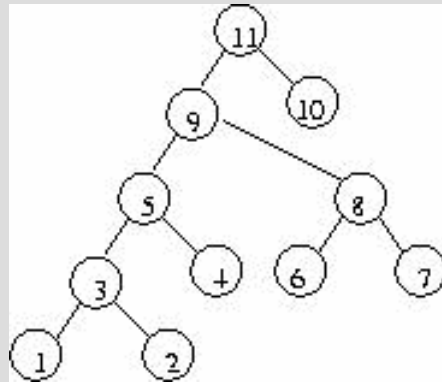
Introduction

System Based Method

Direct IO Method

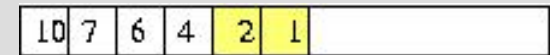
Concluding Remarks

Illustration: sequential processing of the tree

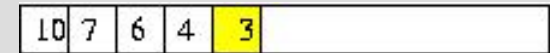


Pool at the beginning of FWD

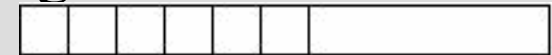
I - II step



III step



Pool at the beginning of BWD



FWD



Hard Disk

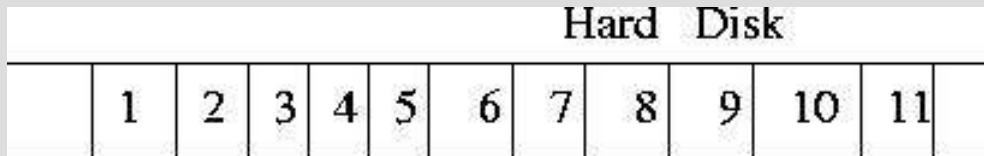
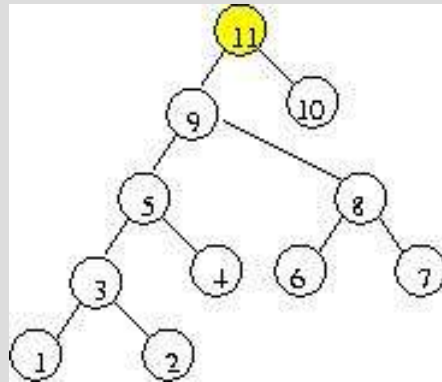
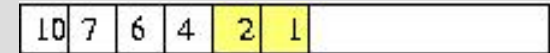


Illustration: sequential processing of the tree

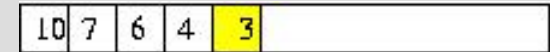


Pool at the beginning of FWD

I - II step

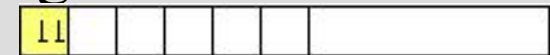


III step



Pool at the beginning of BWD

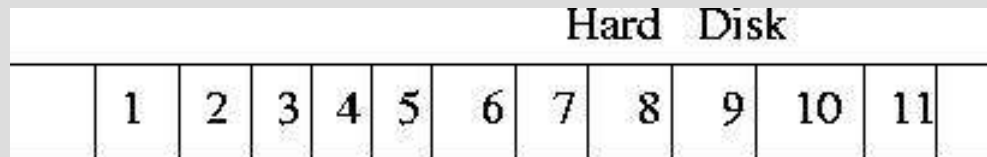
I step



FWD



Hard Disk



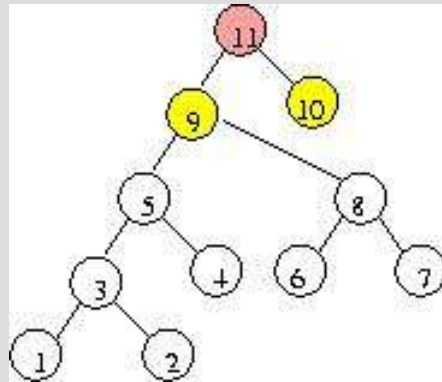
Introduction

System Based Method

Direct IO Method

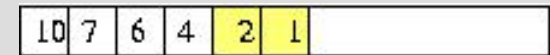
Concluding Remarks

Illustration: sequential processing of the tree

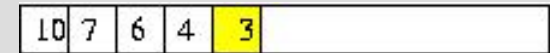


Pool at the beginning of FWD

I - II step

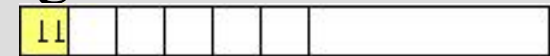


III step



Pool at the beginning of BWD

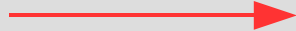
I step



II - III step



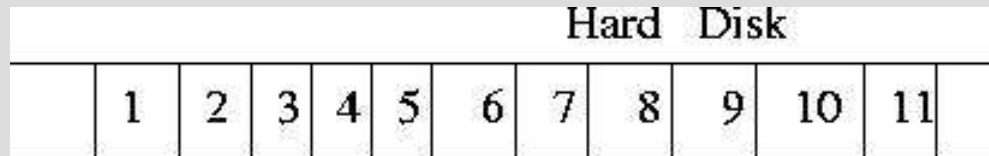
FWD



BWD



Hard Disk



OOC context

- During factorization **ALL** factors are written to local disks
- **No factors kept in memory** at the beginning of the solve part (**and between** forward and backward steps)
- We load data from the local disks to the main memory
- Two possibilities to access data on disk:
 - The simple/natural/default way: do nothing and exploit system properties so called **System Based Method**
 - User defined way, more critical to implement so called **Direct IO Method**

Testing environment

- Multiprocessor Cray XD1 (CERFACS)
- 58 nodes with 2 processors per node
- 4 GB memory per node
- IDE disk managing for each node
- Reiserfs file system (max bandwidth for read operation: 16 MB/sec)
- 2 matrices will be used:
 - Grid 300-100-10 (factor size **748 MB**, **order: 300 000**)
 - Qimonda07, from Qimonda company (factor size **2 534 MB**
order: 8 613 291)

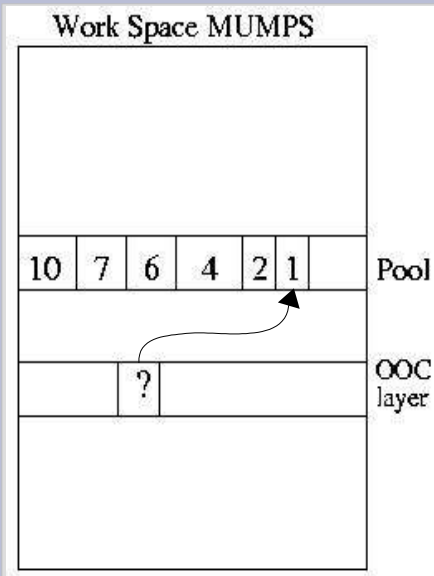
Introduction

System Based Method

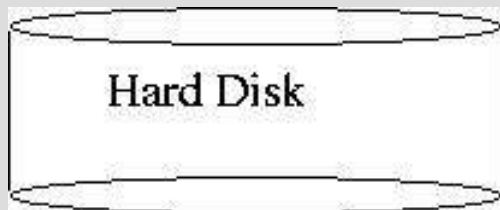
Direct IO Method

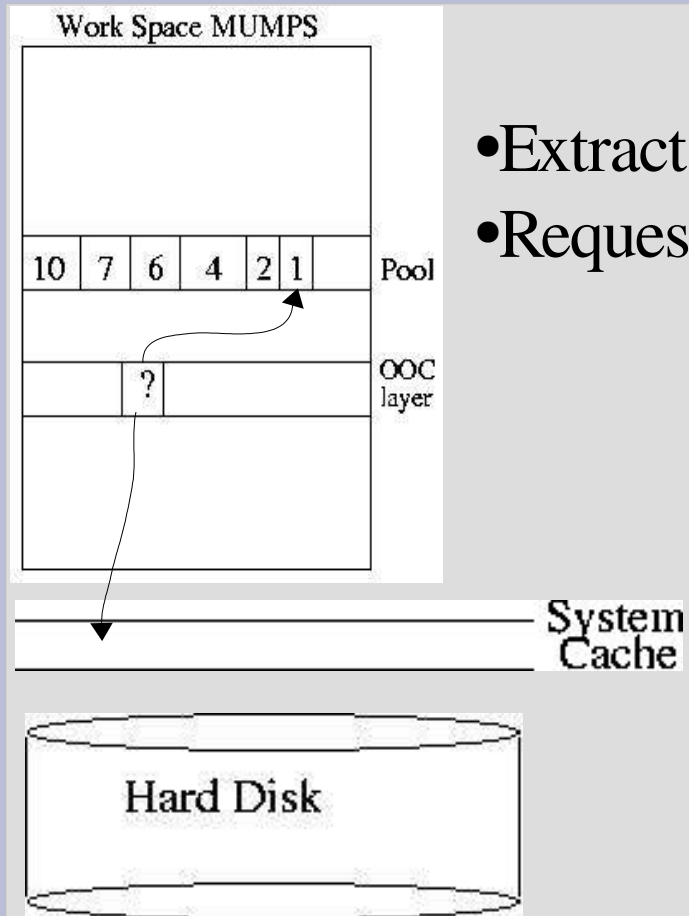
Concluding Remarks

System Based Method

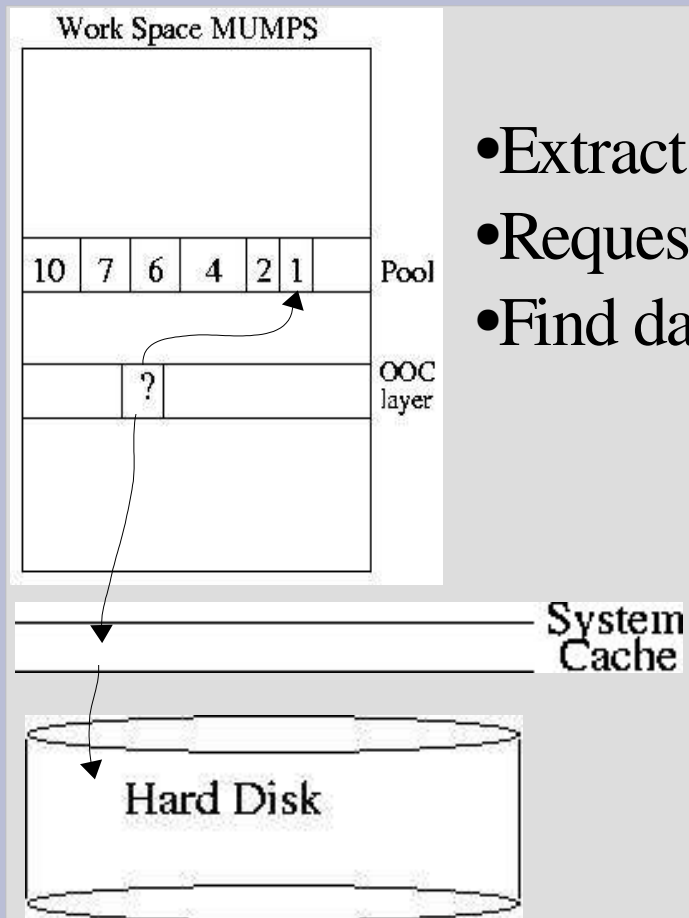


- Extract task from pool

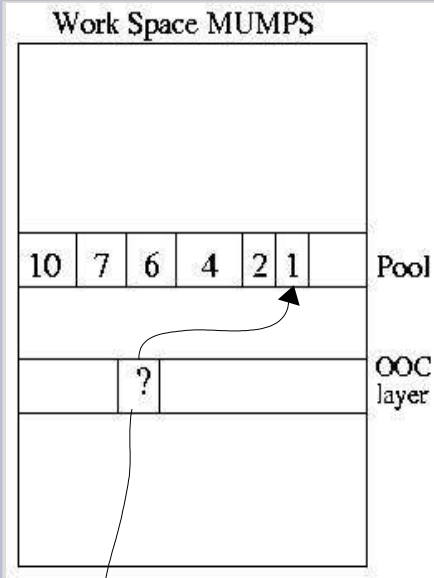




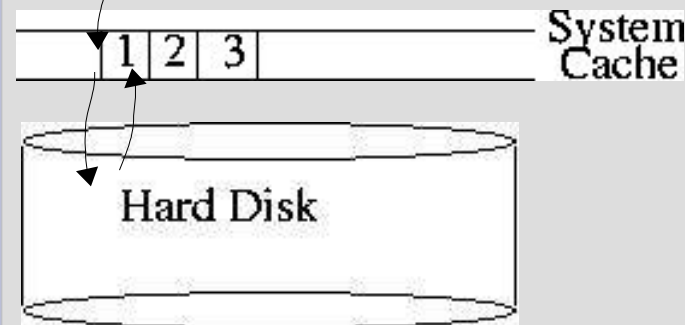
- Extract task from pool
- Request to load data

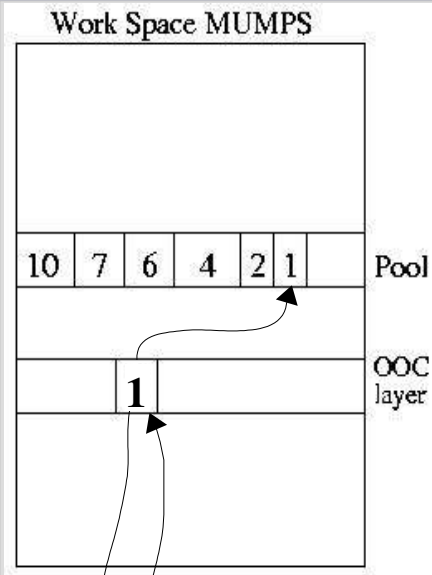


- Extract task from pool
- Request to load data
- Find data on the disk

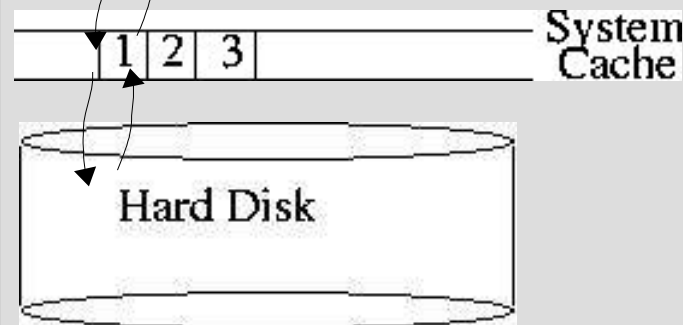


- Extract task from pool
- Request to load data
- Find data on the disk
- Load data in the system cache (+ look-ahead)





- Extract task from pool
- Request to load data
- Find data on the disk
- Load data in the system cache (+ look-ahead)
- Copy requested data to the work space



Advantage

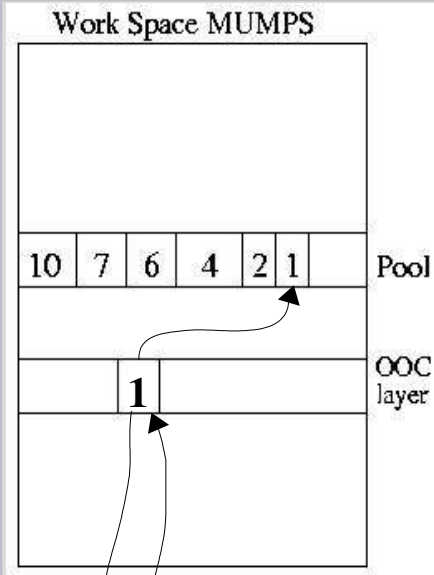
- Easy to implement, demand driven strategy

| Strategy | Factor | Solve | | |
|-----------------|---------------|--------------|--------------|-----------------------|
| | Time (sec) | Fwd (sec) | Bwd (sec) | Disk access (MB/s) |
| Grid 300-100-10 | | | | |
| In-core | 34.91 | 0.39 | 0.37 | |
| OOB | 34.90 | 1.26 | 1.17 | 616 |

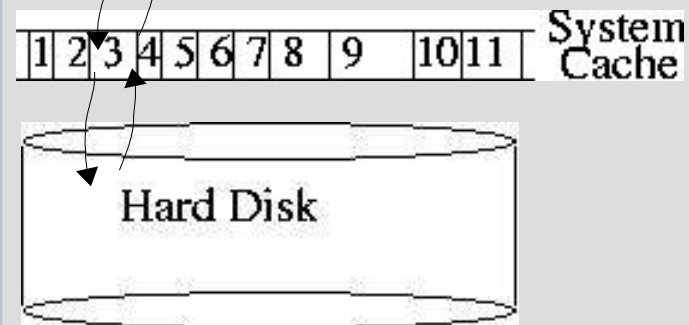
Table 1-a) System Based Approach;

Remark: Average speed of disk access is 616 MB/sec.

616 MB/sec >> 16 MB/sec WHY ?



Factors written on disk are
in fact kept in the system cache.



How will this work **when memory is critical** ?

| Strategy | Factor | Solve | | |
|----------|---------------|--------------|--------------|-----------------------|
| | Time (sec) | Fwd (sec) | Bwd (sec) | Disk access (MB/s) |
| | Qimonda07 | | | |
| In-core | 40.40 | 0.9 | 0.9 | |
| OOC | 191.00 | 186.4 | 207.7 | 13 |

Table 1-b) System Based Approach

Remark: Average speed of disk access is 13 MB/sec

Drawback

- The system cache grows at each disk (read/write) access
- Impossible to control memory use (size and speed)
- Problem for large matrices, when the volume of disk access is larger than memory size
- Swapping effect
 - System decision often based on a variant of Least Recently Used strategy
 - System has no knowledge of data access pattern
 - System cache management may lead to user space swap

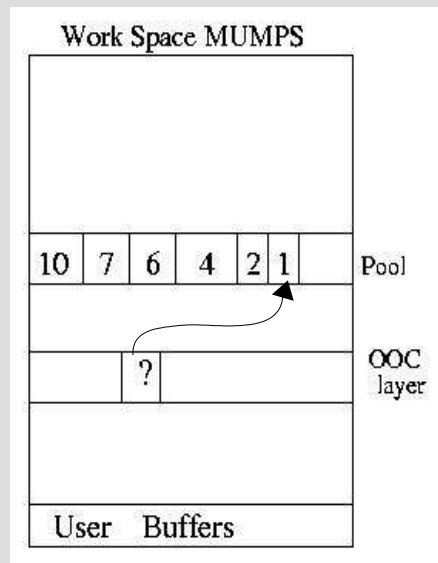
Introduction

System Based Method

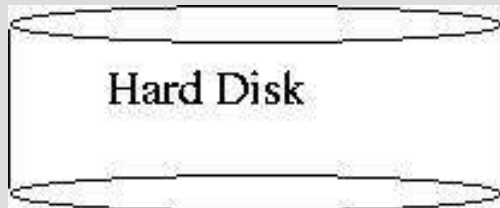
Direct IO Method

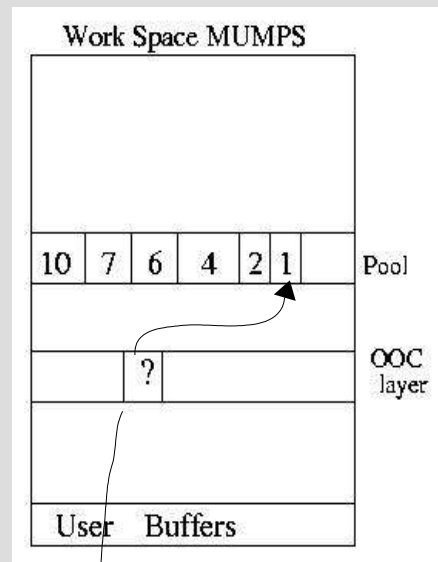
Concluding Remarks

Direct IO Method

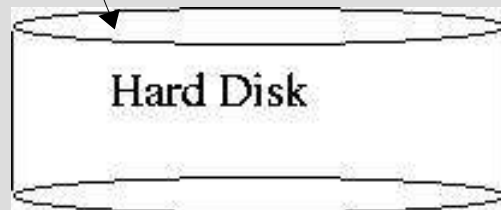


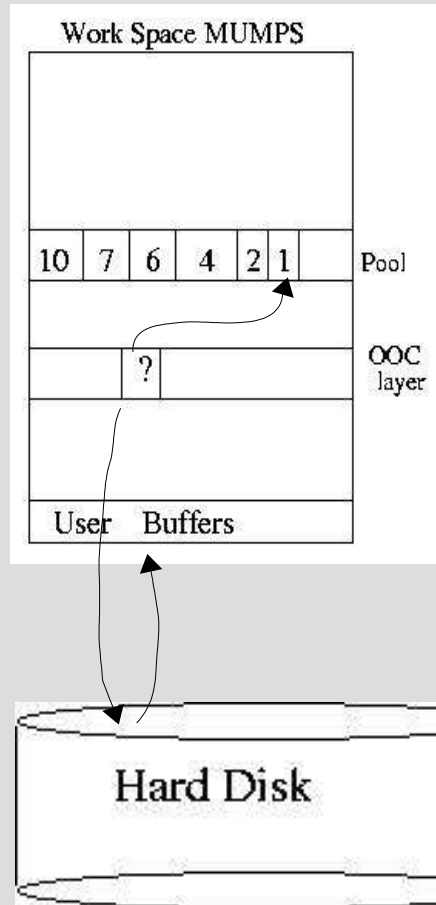
- Extract task from pool



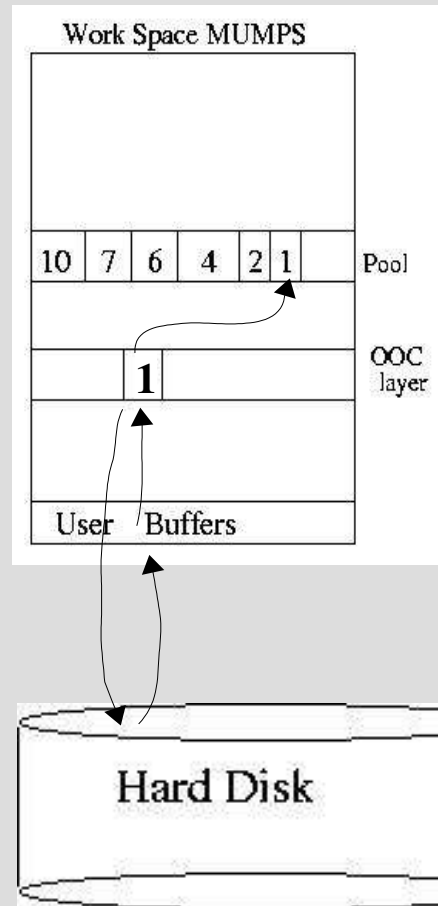


- Extract task from pool
- Request to load data





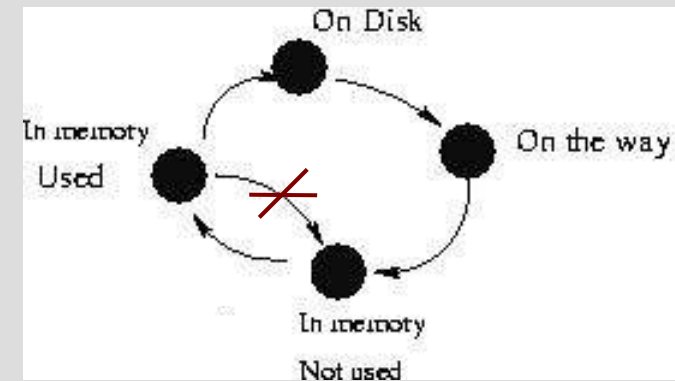
- Extract task from pool
- Request to load data
- Load data in the user's buffer (+some look-ahead mechanisms)



- Extract task from pool
- Request to load data
- Load data in the user's buffer (+some look-ahead mechanisms)
- Wait for availability of data

Implementation details

- 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 look-ahead mechanism



Introduce user buffers to improve performance

| Method | User Buffers | T_factor (sec) | T_fwd (sec) | T_bwd (sec) | Nb_Req 1 buffer FWD | Nb_Req EMG zone FWD | Nb_Req 1 buffer BWD | Nb_Req EMG zone BWD |
|--------------|--------------|-------------------|----------------|----------------|---------------------------|---------------------------|---------------------------|---------------------------|
| System Based | - | 191.0 | 186.4 | 207.7 | - | - | - | - |
| Direct IO | EMG | 107.8 | 1 149.2 | 1 279.2 | 0 | 3 083 998 | 0 | 3 083 998 |
| | EMG+1buffer | 107.5 | 174.0 | 183.7 | 543 | 0 | 494 | 1 |

EMG buffer = **1 MB**, 1 buffer = **10 MB**, factor size = **2 534 MB**

- EMG- significant number of requests to the disk => overhead
- EMG+1 buffer- anticipate mechanism
 - Stable performance; almost suppress the EMG use

Influence of Scheduling

- Influence of pool management strategy on sequential performance

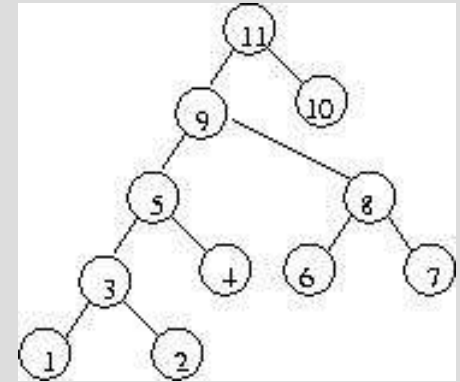
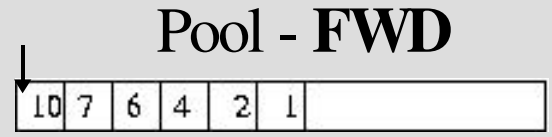
| Strategy | Nb of Procs | T_factor (sec) | T_min (sec) | T_fwd (sec) | T_bwd (sec) | Nb_Req 1 buffer FWD | Nb_Req EMG zone FWD | Nb_Req 1 buffer BWD | Nb_Req EMG zone BWD |
|----------|-------------|----------------|-------------|-------------|-------------|---------------------|---------------------|---------------------|---------------------|
| LIFO | 1 | 107.5 | 158.4 | 174.0 | 183.7 | 543 | 0 | 494 | 1 |
| FIFO | 1 | 107.8 | 158.4 | 2307.9 | 1403.7 | 87 | 3 054 879 | 252 | 3 073 355 |

Table 3: Influence of the scheduling of the tasks on Qimonda07. T_min is the min time to load factors from disk. EMG=1 MB, 1 buffer = 10 MB per processor

- Note that FWD is slower than the BWD

- FWD step might be slower than BWD for FIFO !
 - Illustration of **FIFO** access during solve

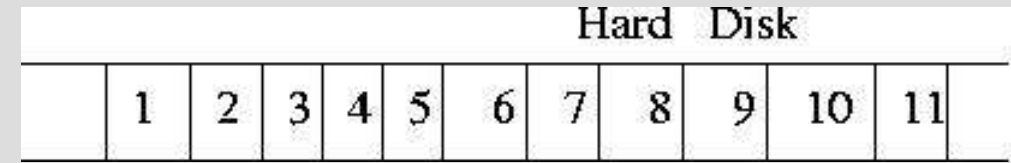
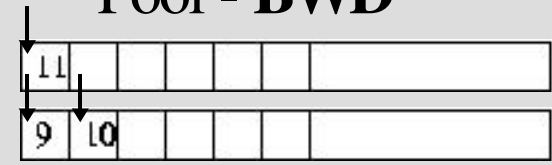
| | |
|-------------|-----|
| User buffer | Emg |
| 1 2 3 4 | 10 |
| 1 2 3 4 | 7 |
| 1 2 3 4 | 6 |



...

| | |
|----------|--|
| 9 10 11 | |
| 9 10 7 8 | |
| 5 6 7 8 | |

Pool - **BWD**



Advantage

- User has full control of:
 - Memory Used
 - User buffers
 - Strategies to prefetch (to exploit algorithm properties)
- Stable behaviour for small as well as for large problems

Drawback

- System based cache mechanism not available
- More complex to implement
- Algorithmic effort required (asynchronous and look-ahead mechanisms)

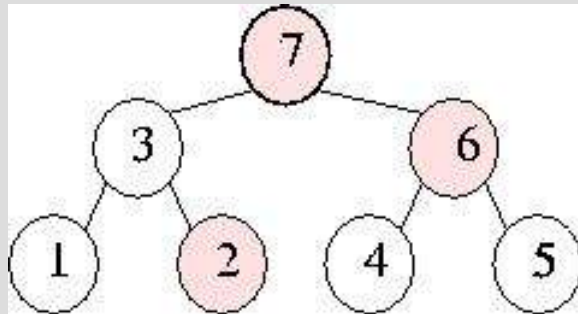
Influence of parallelism (LIFO strategy)

- In parallel, the normal/natural order to access is not respected
- Large number of emergency calls (BWD step)

| Strategy | Nb of Procs | T_factor (sec) | T_min (sec) | T_fwd (sec) | T_bwd (sec) | Nb_Req 1 buffer FWD | Nb_Req emg zone FWD | Nb_Req 1 buffer BWD | Nb_Req emg zone BWD |
|----------|-------------|-------------------|----------------|----------------|----------------|---------------------------|---------------------------|---------------------------|---------------------------|
| LIFO | 2 | 76.0 | 78.8 | 90.0 | 97.4 | 267 | 0 | 240 | 4 520 |
| LIFO | 6 | 51.2 | 24.3 | 33.8 | 365.7 | 75 | 0 | 71 | 426 014 |
| LIFO | 8 | 42.2 | 20.8 | 24.7 | 212.0 | 61 | 0 | 32 | 195 990 |

Table4: Influence of the scheduling of the tasks on Qimonda07. T_min is the min time to load factors from disk. EMG=1 MB, 1 buffer = 10 MB per processor

In parallel scheduling may also be a critical issue



Hard Disk

P0:

| | | | | |
|---|---|---|---|---|
| 1 | 3 | 4 | 5 | 7 |
|---|---|---|---|---|

P1:

| | | |
|---|---|---|
| 2 | 6 | 7 |
|---|---|---|

User Buffer

| | | |
|---|---|---|
| 4 | 5 | 7 |
|---|---|---|

 : P0

| | |
|---|---|
| 6 | 7 |
|---|---|

 : P1

POOL : LIFO strategy

P0:

| | |
|---|--|
| 7 | |
|---|--|

 P1:

| | |
|---|--|
| 7 | |
|---|--|

| | |
|---|--|
| 3 | |
|---|--|

| | |
|---|--|
| 6 | |
|---|--|

| | | | |
|---|---|---|---|
| 4 | 5 | 1 | 2 |
|---|---|---|---|

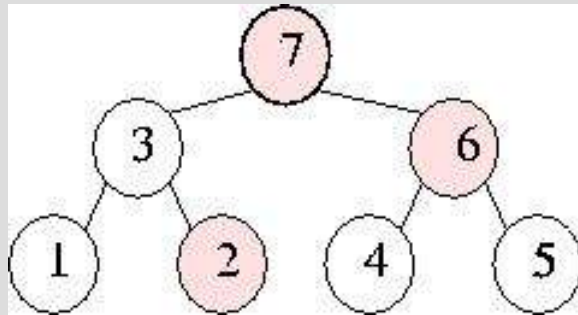
 Too late !

Introduction

System Based Method

Direct IO Method

Concluding Remarks



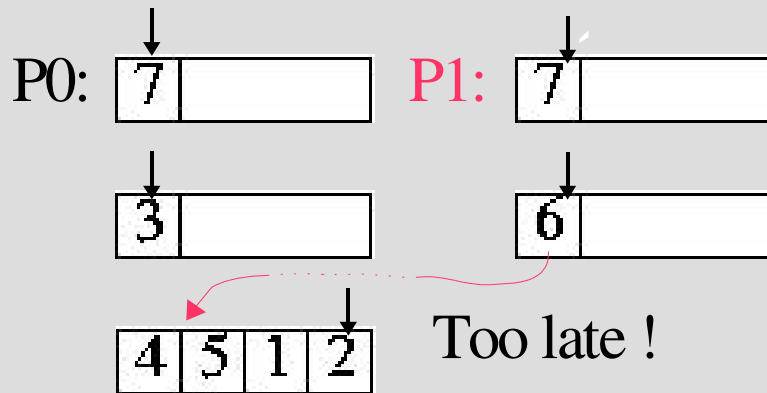
Hard Disk



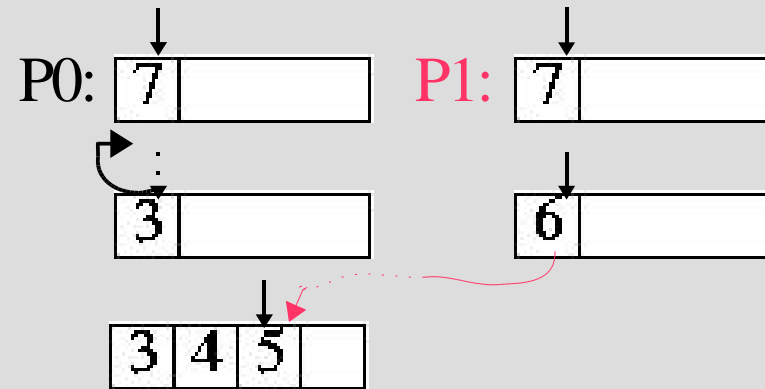
User Buffer



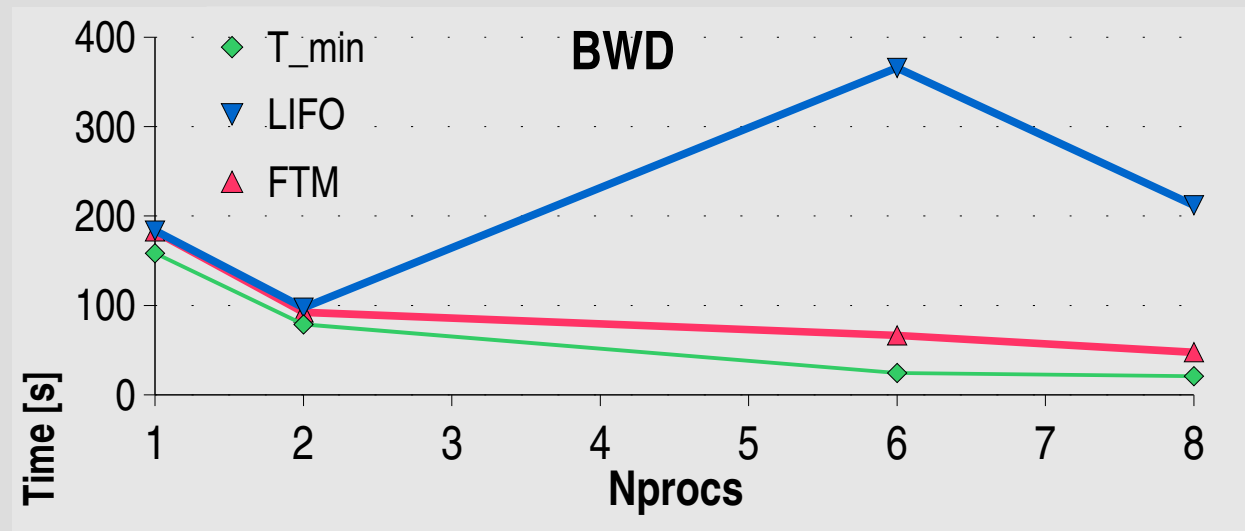
POOL : LIFO strategy



New FTM strategy



Influence of parallelism - a new FTM strategy driven by the write sequence of the factorisation



Remark:

Only master node write sequence is followed, slave tasks processed on the fly.

Concluding remarks (on going work ...)

- Time for solve is critical
- Scheduling is also critical
- Future work:
 - Scheduling for parallel case (exploit slave task write sequence)
 - Overlapping computation and disk access in a context of multiple RHS
- Influence the scheduler of the factorisation to improve the solve (collaboration with Agullo and L'Excellent, ENS Lyon)

Influence of parallelism - a new FTM strategy

- Force to respect the write sequence of the factorisation

| Strategy | Nb of Procs | T_factor (sec) | T_min (sec) | T_fwd (sec) | T_bwd (sec) | Nb_Req 1 buffer FWD | Nb_Req emg zone FWD | Nb_Req 1 buffer BWD | Nb_Req emg zone BWD |
|----------|-------------|----------------|-------------|-------------|-------------|---------------------|---------------------|---------------------|---------------------|
| FTM | 2 | 76.0 | 78.8 | 92.3 | 92.9 | 275 | 0 | 249 | 0 |
| FTM | 6 | 51.2 | 31.5 | 34.2 | 66.4 | 101 | 5 | 93 | 0 |
| FTM | 8 | 42.2 | 21.8 | 24.9 | 47.6 | 71 | 1 | 64 | 1 |

Table5: Influence of the scheduling First of the Top of the Memory (FTM) of the tasks on Qimonda07.

EMG=1 MB, 1 buffer = 10 MB per processor