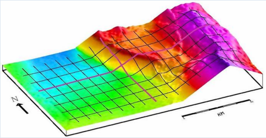
Memory-Aware Scheduling for Sparse Direct Methods

Emmanuel AGULLO, ICL - University of Tennessee
Abdou GUERMOUCHE, LaBRI, Université de Bordeaux
Jean-Yves L'EXCELLENT, LIP - INRIA

May 15, 2009

Context

Solving sparse linear systems



$$Ax = b$$

$\Rightarrow$ Direct methods: $A = LU$

Typical matrix: BRGM matrix

- 3.7×10^6 variables
- 156×10^6 non zeros in A
- 4.5×10^9 non zeros in LU
- 26.5×10^{12} flops

Hardware paradigm

- Many-core architecture.
- Large global amount of memory.
- Limited memory per core.

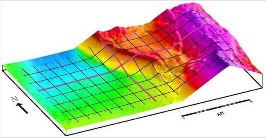
Software challenge

$\rightarrow$ Need for algorithms whose memory usage scales with the number of processors.

- Case study: MUMPS

Context

Solving sparse linear systems



$$Ax = b$$

$\Rightarrow$ Direct methods: $A = LU$

Typical matrix: BRGM matrix

- ⋆ 3.7×10^6 variables
- ⋆ 156×10^6 non zeros in A
- ⋆ 4.5×10^9 non zeros in LU
- ⋆ 26.5×10^{12} flops

Hardware paradigm

- ⋆ Many-core architecture.
- ⋆ Large global amount of memory.
- ⋆ Limited memory per core.

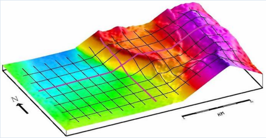
Software challenge

→ Need for algorithms whose memory usage scales with the number of processors.

- ⋆ Case study: MUMPS

Context

Solving sparse linear systems



$$Ax = b$$

$\Rightarrow$ Direct methods: $A = LU$

Typical matrix: BRGM matrix

- 3.7×10^6 variables
- 156×10^6 non zeros in A
- 4.5×10^9 non zeros in LU
- 26.5×10^{12} flops

Hardware paradigm

- Many-core architecture.
- Large global amount of memory.
- Limited memory per core.

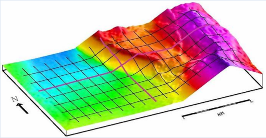
Software challenge

$\rightarrow$ Need for algorithms whose memory usage scales with the number of processors.

- Case study: MUMPS

Context

Solving sparse linear systems



$$Ax = b$$

$\Rightarrow$ Direct methods: $A = LU$

Typical matrix: BRGM matrix

- ⋆ 3.7×10^6 variables
- ⋆ 156×10^6 non zeros in A
- ⋆ 4.5×10^9 non zeros in LU
- ⋆ 26.5×10^{12} flops

Hardware paradigm

- ⋆ Many-core architecture.
- ⋆ Large global amount of memory.
- ⋆ Limited memory per core.

Software challenge

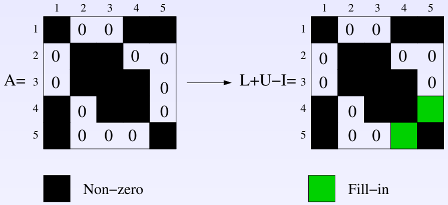
$\rightarrow$ Need for algorithms whose memory usage scales with the number of processors.

- ⋆ Case study: MUMPS

Outline

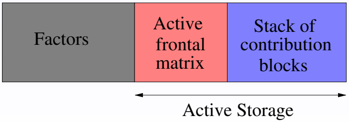
Outline

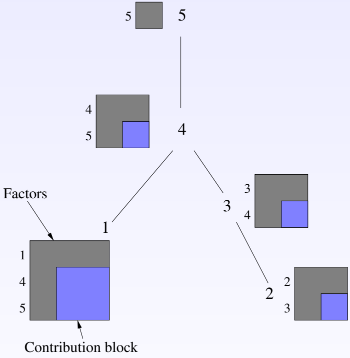
The multifrontal method (Duff, Reid'83)



Storage divided into two parts:

- ★ Factors *systematically* written to disk;
- ★ Active Storage kept in memory.



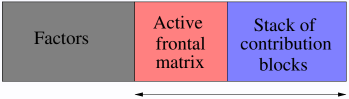


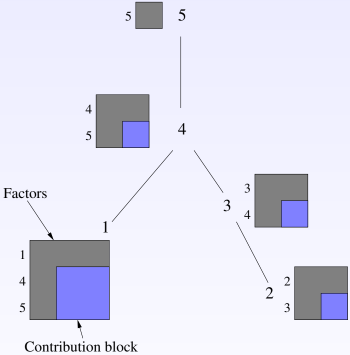
Elimination tree

The multifrontal method (Duff, Reid'83)

Storage divided into two parts:

- Factors *systematically* written to disk;
- Active Storage kept in memory.



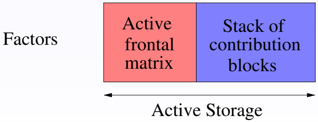


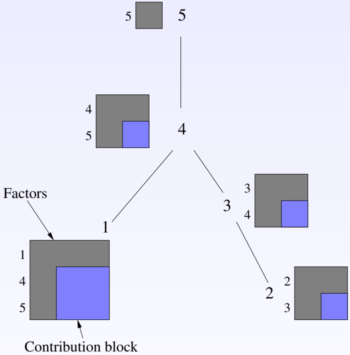
Elimination tree

The multifrontal method (Duff, Reid'83)

Storage divided into two parts:

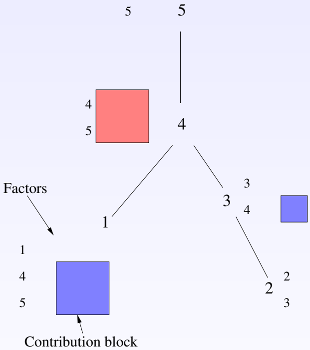
- Factors *systematically* written to disk;
- Active Storage kept in memory.





Elimination tree

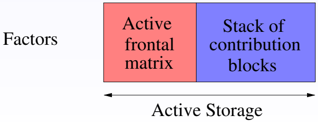
The multifrontal method (Duff, Reid'83)



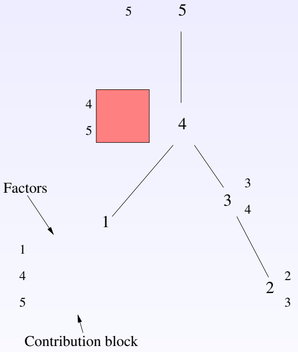
Elimination tree

Storage divided into two parts:

- Factors *systematically* written to disk;
- Active Storage kept in memory.



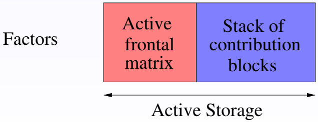
The multifrontal method (Duff, Reid'83)



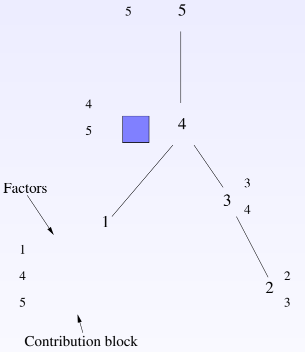
Elimination tree

Storage divided into two parts:

- Factors *systematically* written to disk;
- Active Storage kept in memory.



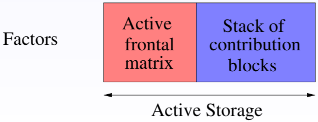
The multifrontal method (Duff, Reid'83)



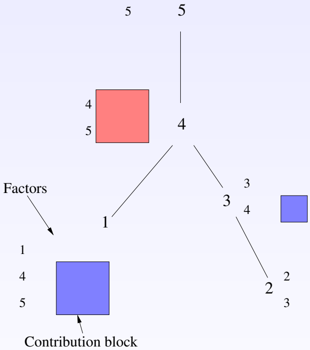
Elimination tree

Storage divided into two parts:

- ⋆ Factors *systematically* written to disk;
- ⋆ Active Storage kept in memory.

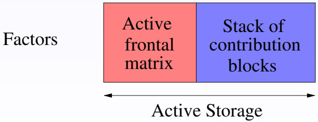


The multifrontal method (Duff, Reid'83)



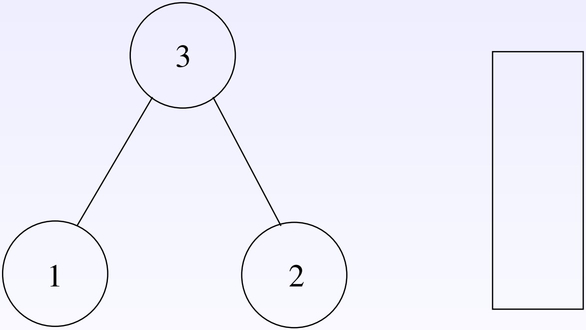
Storage divided into two parts:

- ⋆ Factors *systematically* written to disk;
- ⋆ Active Storage kept in memory.

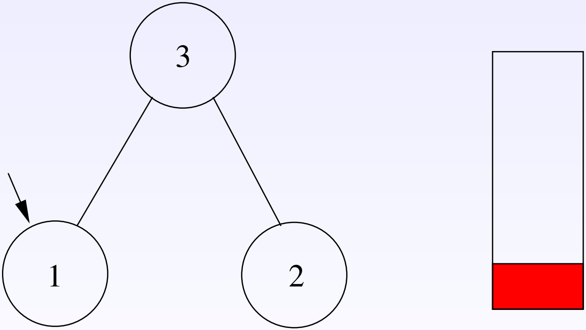


Elimination tree

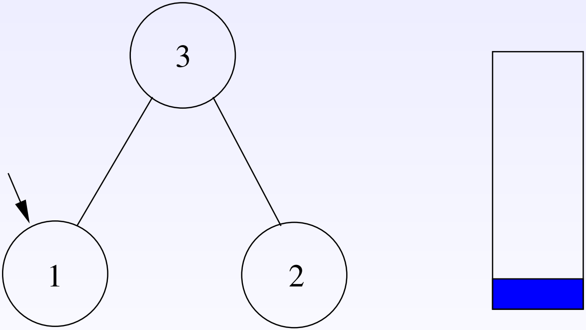
Memory behaviour (serial postorder traversal)



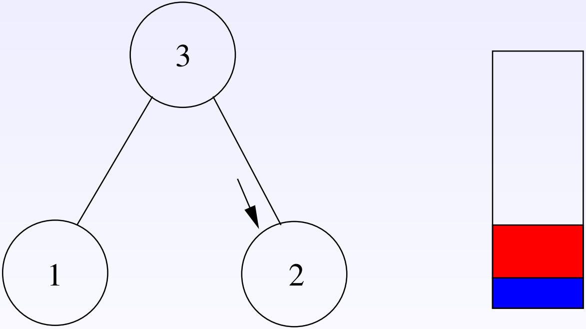
Memory behaviour (serial postorder traversal)



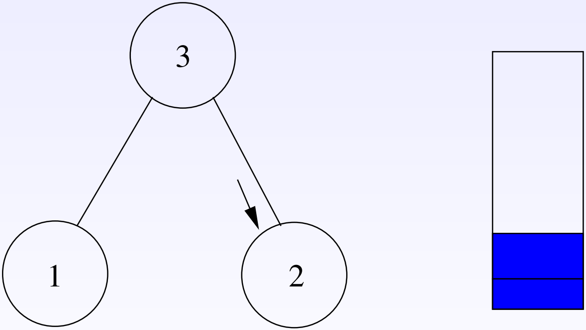
Memory behaviour (serial postorder traversal)



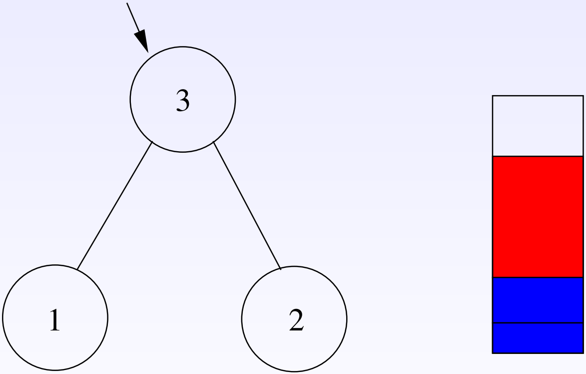
Memory behaviour (serial postorder traversal)



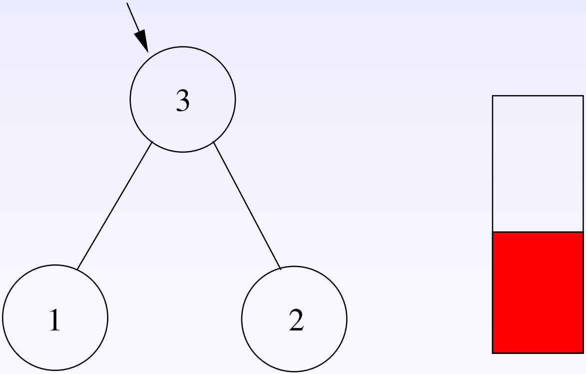
Memory behaviour (serial postorder traversal)



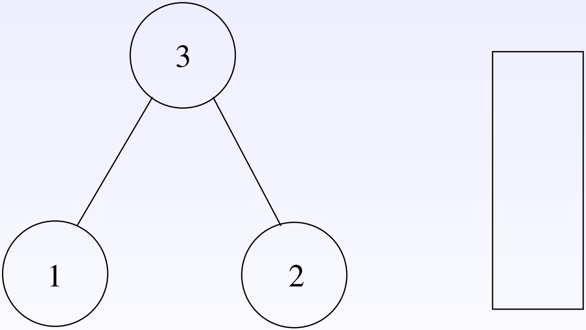
Memory behaviour (serial postorder traversal)



Memory behaviour (serial postorder traversal)



Memory behaviour (serial postorder traversal)



Sequential case results

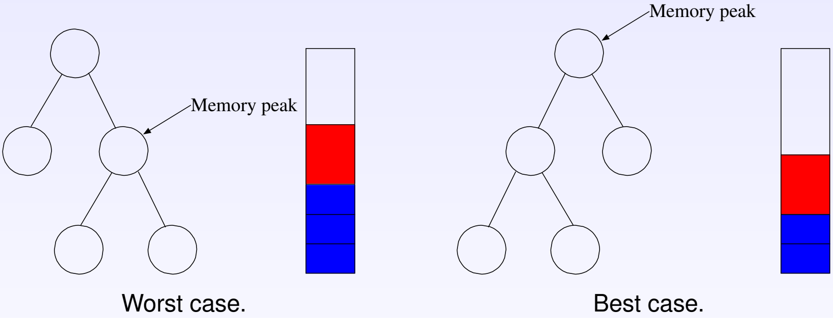
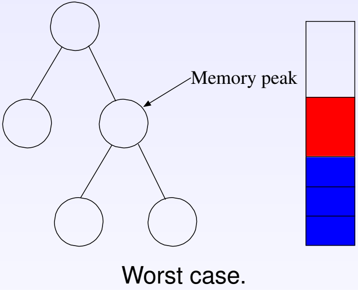


Figure: Impact of the tree traversal on the memory behavior.

→ Algorithms to find the optimal tree traversal have been proposed

Sequential case results



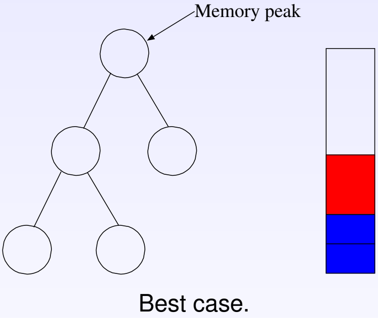


Figure: Impact of the tree traversal on the memory behavior.

→ Algorithms to find the optimal tree traversal have been proposed

Memory efficiency

Definition: *Memory Efficiency* on p processors (or cores)

$e(p) = \frac{S_{seq}}{p \times S_{max}(p)}$, S_{seq}: serial storage, S_{max}: parallel storage

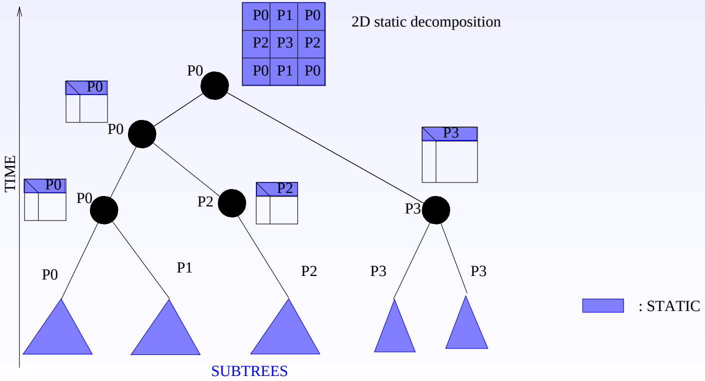
Results: Memory Efficiency of MUMPS (with factors on disk)

Number p of processors	16	32	64	128
AUDI_KW_1	0.16	0.12	0.13	0.10
CONESHL_MOD	0.28	0.28	0.22	0.19
CONV3D64	0.42	0.40	0.41	0.37
QIMONDA07	0.30	0.18	0.11	-
ULTRASOUND80	0.32	0.31	0.30	0.26

Outline

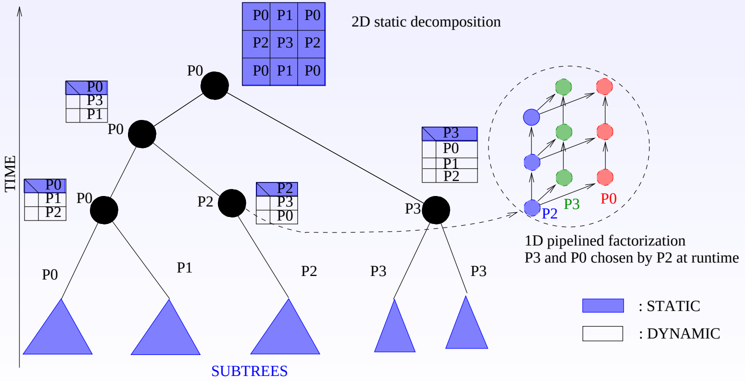
Parallel multifrontal scheme

- **Type 1** : Nodes processed on a single processor
- **Type 2** : Nodes processed with a parallel 1D blocked factorization
- **Type 3** : Parallel 2D cyclic factorization (root node)

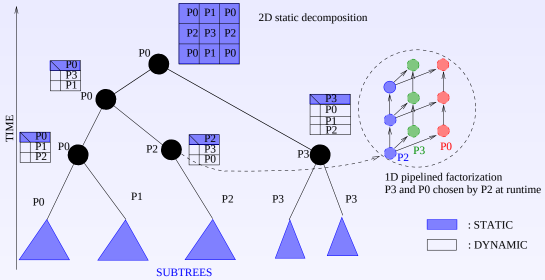


Parallel multifrontal scheme

- **Type 1** : Nodes processed on a single processor
- **Type 2** : Nodes processed with a parallel 1D blocked factorization
- **Type 3** : Parallel 2D cyclic factorization (root node)

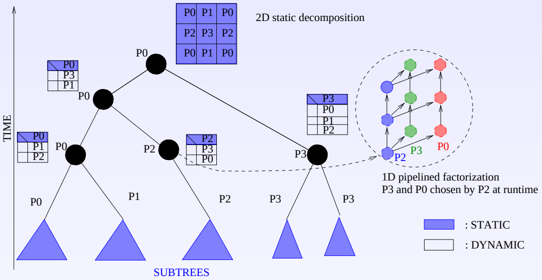


Limits to memory scalability



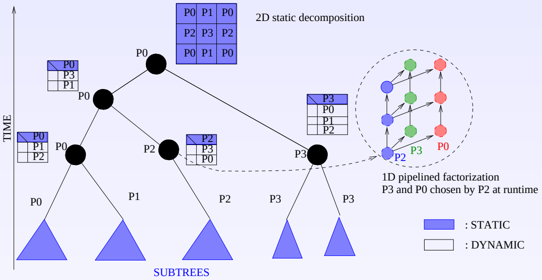
- Many simultaneous active tasks;
- Large master tasks;
- Large subtrees;
- Proportional mapping.

Limits to memory scalability



- ⋆ Many simultaneous active tasks;
- ⋆ Large master tasks;
- ⋆ Large subtrees;
- ⋆ Proportional mapping.

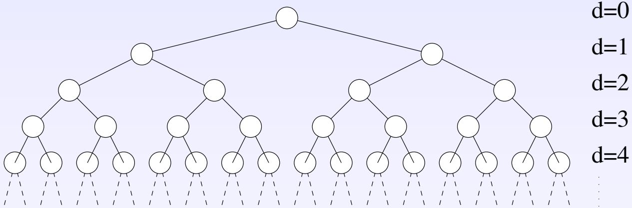
Limits to memory scalability



- ⋆ Many simultaneous active tasks;
- ⋆ Large master tasks;
- ⋆ Large subtrees;
- ⋆ Proportional mapping.

Proportional mapping VS postorder traversal (1/2)

Elimination tree :



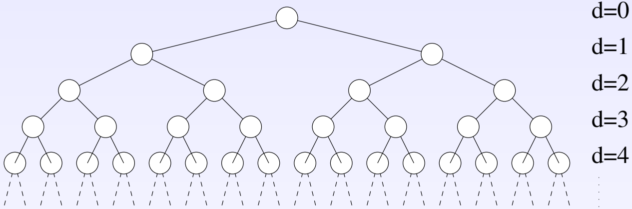
Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees.

Advantages and drawbacks

Proportional mapping VS postorder traversal (1/2)

Proportional mapping:



Mapping

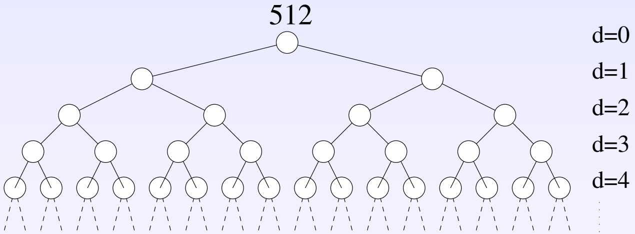
- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees.

Advantages and drawbacks

- Tree-level + task-level parallelism;
- Bad memory efficiency.

Proportional mapping VS postorder traversal (1/2)

Proportional mapping:



Mapping

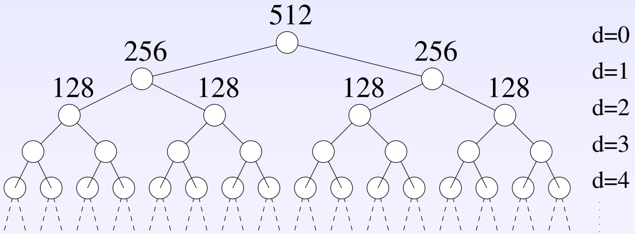
- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees.

Advantages and drawbacks

- Tree-level + task-level parallelism;
- Bad memory efficiency

Proportional mapping VS postorder traversal (1/2)

Proportional mapping:



Mapping

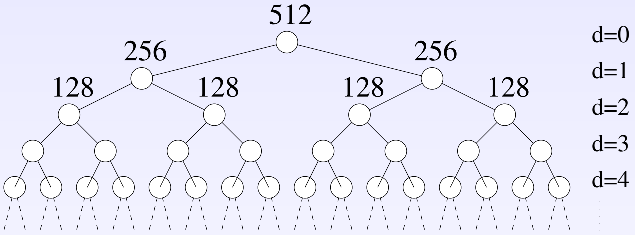
- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees.

Advantages and drawbacks

- Tree-level + task-level parallelism;
- Bad memory efficiency.

Proportional mapping VS postorder traversal (1/2)

Proportional mapping:



Mapping

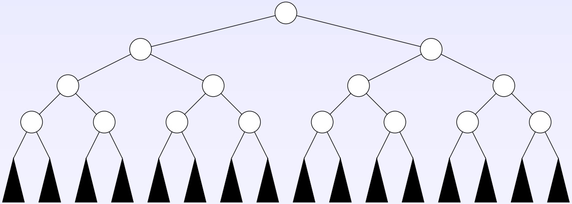
- ⋆ Initially: all processors on root node;
- ⋆ Recursively split the set of processors on child subtrees.

Advantages and drawbacks

- ☺ Tree-level + task-level parallelism;
- ☹ Bad memory efficiency.

Proportional mapping VS postorder traversal (1/2)

Proportional mapping:



Mapping

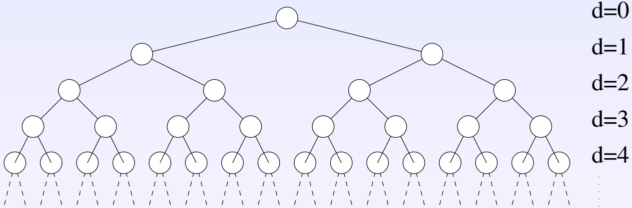
- ⋆ Initially: all processors on root node;
- ⋆ Recursively split the set of processors on child subtrees.

Advantages and drawbacks

- ☺ Tree-level + task-level parallelism;
- ☹ Bad memory efficiency.

Proportional mapping VS postorder traversal (2/2)

Elimination tree :



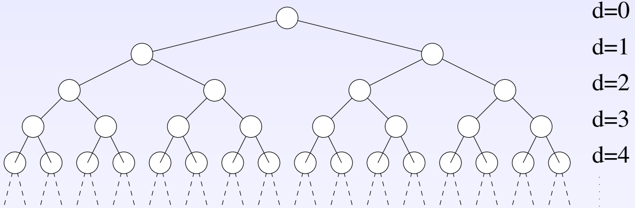
Traversal

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks

Proportional mapping VS postorder traversal (2/2)

Postorder traversal :



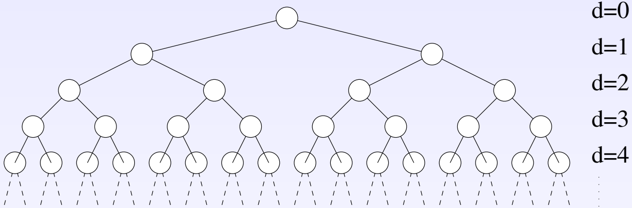
Traversal

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks

Proportional mapping VS postorder traversal (2/2)

Postorder traversal :



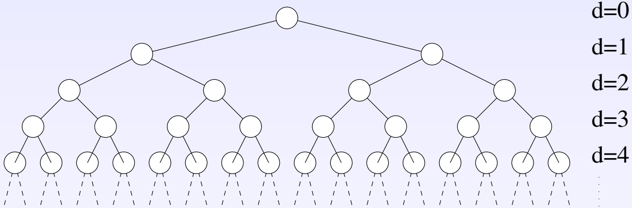
Traversal

- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks

Proportional mapping VS postorder traversal (2/2)

Postorder traversal :



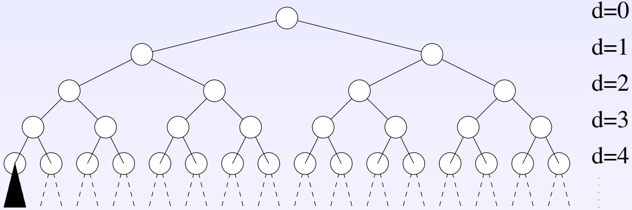
Traversal

- ⋆ Postorder traversal, node by node;
- ⋆ All processors on each node.

Advantages and drawbacks

Proportional mapping VS postorder traversal (2/2)

Postorder traversal :



Traversal

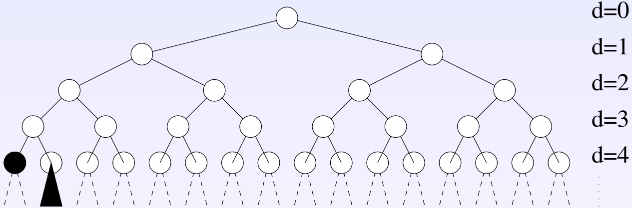
- Postorder traversal, node by node;
- All processors on each node.

Advantages and drawbacks

- Only task level parallelism;
- High memory efficiency.

Proportional mapping VS postorder traversal (2/2)

Postorder traversal :



Traversal

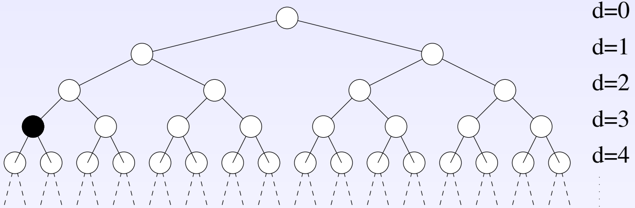
- ⋆ Postorder traversal, node by node;
- ⋆ All processors on each node.

Advantages and drawbacks

- Only task-level parallelism;
- High memory efficiency.

Proportional mapping VS postorder traversal (2/2)

Postorder traversal :



Traversal

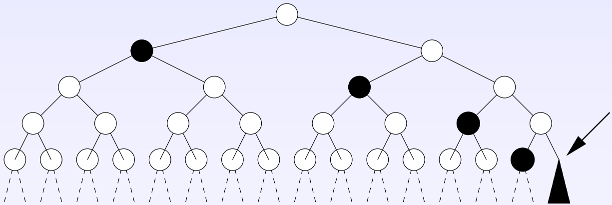
- ⋆ Postorder traversal, node by node;
- ⋆ All processors on each node.

Advantages and drawbacks

- Only task-level parallelism;
- High memory efficiency.

Proportional mapping VS postorder traversal (2/2)

Postorder traversal :



Traversal

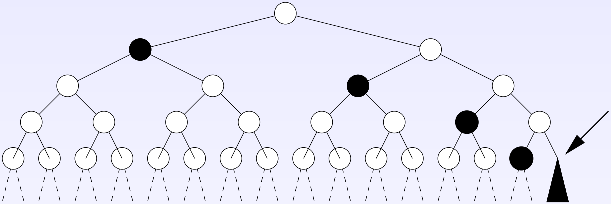
- ⋆ Postorder traversal, node by node;
- ⋆ All processors on each node.

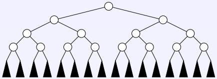
Advantages and drawbacks

- ☹ Only task-level parallelism;
- ☺ High memory efficiency.

Proportional mapping VS postorder traversal (2/2)

Postorder traversal :





Traversal

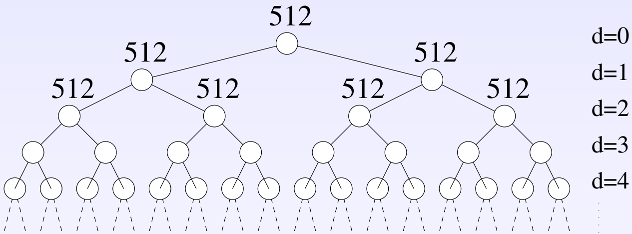
- ⋆ Postorder traversal, node by node;
- ⋆ All processors on each node.

Advantages and drawbacks

- ☹ Only task-level parallelism;
- ☺ High memory efficiency.

Proportional mapping VS postorder traversal (2/2)

Postorder traversal :



Traversal

- ⋆ Postorder traversal, node by node;
- ⋆ All processors on each node.

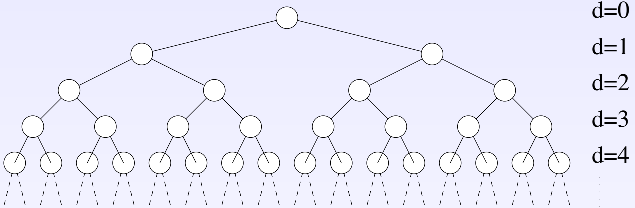
Advantages and drawbacks

- ☹ Only task-level parallelism;
- ☺ High memory efficiency.

Outline

Memory-aware mapping algorithm

Elimination tree :

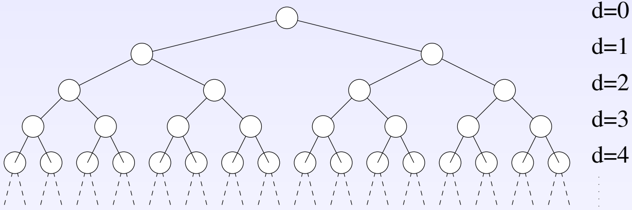


Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees if memory allows for it.

Memory-aware mapping algorithm

Memory-aware mapping:

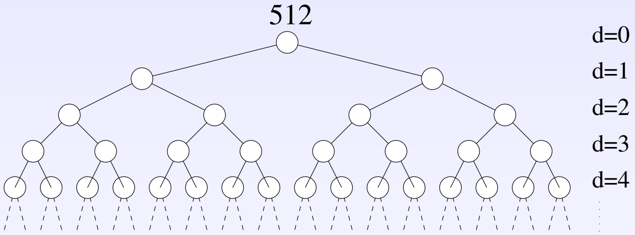


Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees if memory allows for it.

Memory-aware mapping algorithm

Memory-aware mapping:

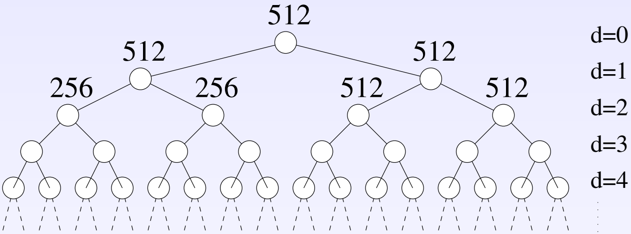


Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees if memory allows for it.

Memory-aware mapping algorithm

Memory-aware mapping:

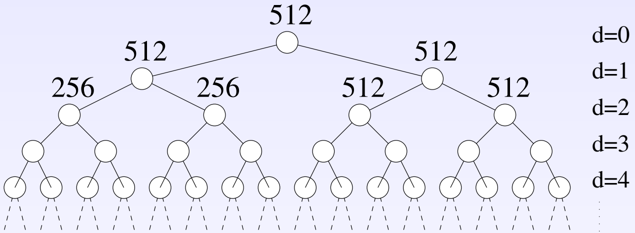


Mapping

- Initially: all processors on root node;
- Recursively split the set of processors on child subtrees if memory allows for it.

Memory-aware mapping algorithm

Memory-aware mapping:



Advantages

- ☺ Robust: guaranteed (if memory $M_0 < \frac{S_{seq}}{p}$).
- ☺ Efficient: available memory provides tree-level parallelism.

Outline

MUMPS: a MUltifrontal Massively Parallel sparse direct Solver

Solution of large sparse linear systems with:

- ⋆ Symmetric positive definite matrices;
- ⋆ General symmetric matrices;
- ⋆ General unsymmetric matrices.

Implementation

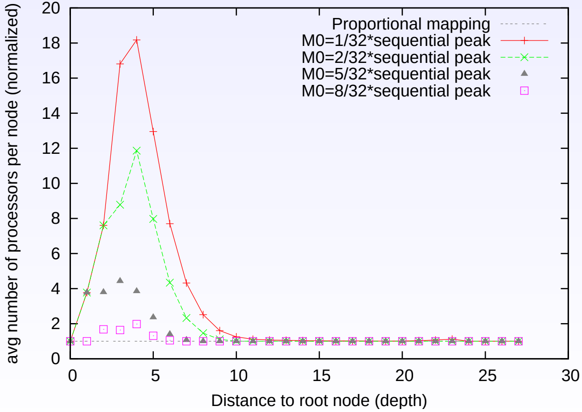
- ⋆ Distributed Multifrontal Solver (F90, MPI based);
- ⋆ Dynamic Distributed Scheduling;
- ⋆ Use of BLAS, BLACS, ScaLAPACK.

Interfaces

- ⋆ Fortran, C, Matlab, Scilab, Visual Studio.

Preliminary results

- Excellent memory scalability:
  - memory efficiency closed to 1.
- Competitive (time) efficiency
  - closed to proportional mapping (if enough memory);
  - memory provides tree-level parallelism:



Outline

Conclusion

Prototype of a *memory-aware* algorithm

- ⋆ Maximizes the amount of tree-level parallelism with respect to the amount of memory available per processor/core.
- ⋆ New static mapping implemented, with constraints on dynamic schedulers; experimented within the OOC version of MUMPS.
- ⋆ Very good memory scalability obtained.

On-going work

- ⋆ Further tuning and validation.
- ⋆ Generalization to the in-core case.
- ⋆ Reinject dynamic information to schedulers.