Sparse direct linear solvers
Woudschoten conference on
Parallel numerical linear algebra
6-7 Octobre 2010

Patrick Amestoy
INPT-IRIT (University of Toulouse)
http://amestoy.perso.enseeiht.fr/

in collaboration with members of MUMPS team
A. Buttari, A. Guermouche, J.Y. L'Excellent, B. Ucar, and F.H. Rouet
http://mumps.enseeiht.fr or
http://graal.ens-lyon.fr/MUMPS/

Outline

Introduction-Context

Memory related issues

- Out-of-core to “extend” memory
- Memory scalability to equilibrate active memory

Efficiency of the solution phase

- Sparsity in the right hand side and/or solution
- Multiple entries of A^{-1}

Concluding remarks

Sparse direct linear solvers (II) - advanced features

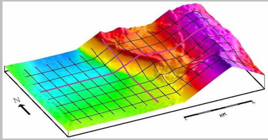
Woudschoten conference 2010

Outline

Introduction-Context

Context

Solving sparse linear systems



$$Ax = b$$

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

Typical matrix (BRGM)

- 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

- Focus recent work (> 2006) within MUMPS project by Emanuel Agullo, Patrick Amestoy, Alfredo Buttari, Abdou Guermouche, Jean-Yves L'Excellent, François-Henry Rouet, Mila Slavova, Bora Uçar and Clément Weisbecker.
- Memory issues
- Performance of the solution phase ($Ly = b$ and $Ux = y$)

What is MUMPS (MUltifrontal Massively Parallel Solver) ?

http://graal.ens-lyon.fr/MUMPS and http://mumps.enseeiht.fr

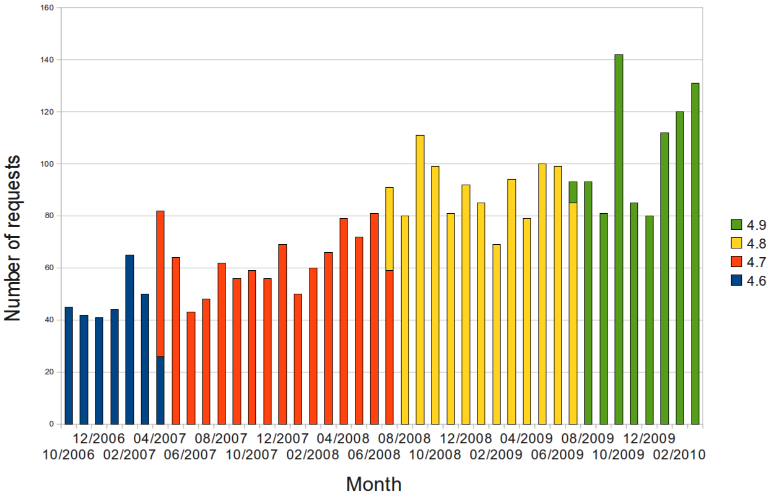
Initially funded by LTR (Long Term Research) European project PARASOL (1996-1999) PARASOL

Platform for research and collaboration with industries

Competitive software package used worldwide

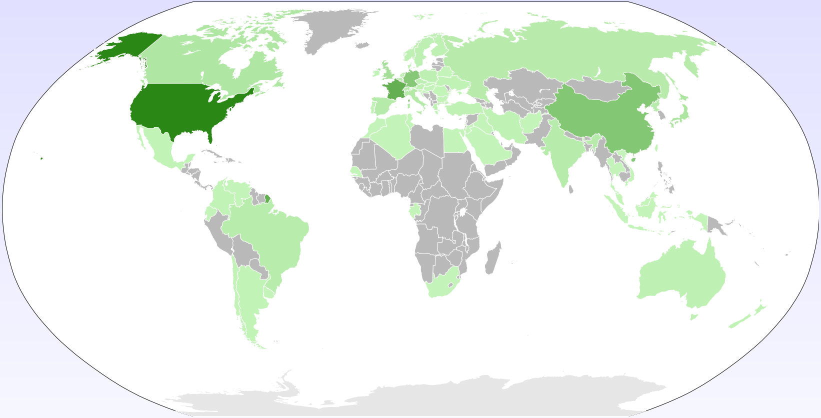
- Co-developed by Lyon-Toulouse-Bordeaux
- Latest release : MUMPS 4.9.2, Nov. 2009, ≈ 250 000 lines of C and Fortran code
- 1000+ downloads per year from our website, half from industries : Boeing, EADS, EDF, Petroleum industries, Samtech, etc.
- Integrated within commercial and academic packages (Samcef from Samtech, FEMTown from Free Field Technologies, *Code_Aster* or Telemac from EDF, IPOPT, Petsc, Trilinos, ...).

Download Requests from the MUMPS website

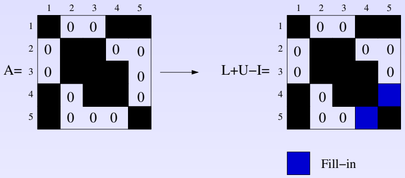


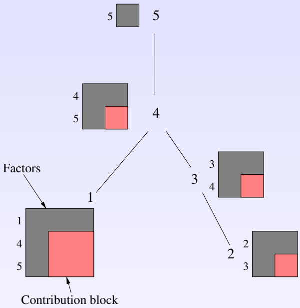
User's distribution map

1000+ download requests per year



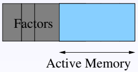
Multifrontal method : *Duff and Reid, '83*





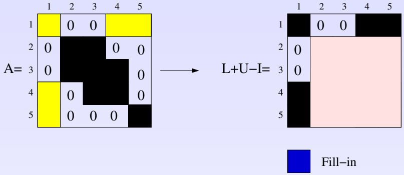
Memory is divided into two parts :

- the factors
- the active memory



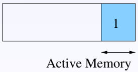
Elimination tree represents the dependencies of the tasks

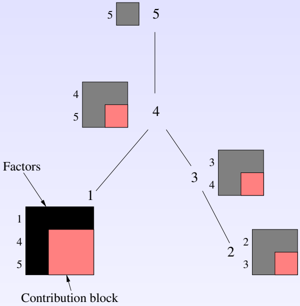
Multifrontal method : *Duff and Reid, '83*



Memory is divided into two parts :

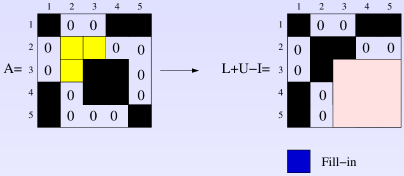
- the factors
- the active memory





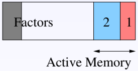
Elimination tree represents the dependencies of the tasks

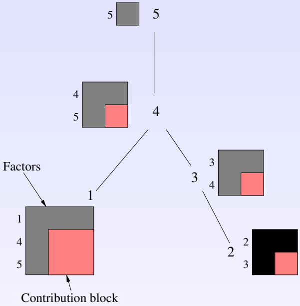
Multifrontal method : *Duff and Reid, '83*



Memory is divided into two parts :

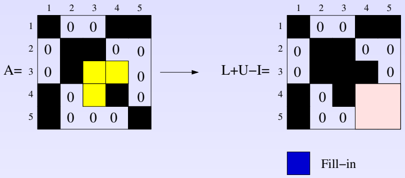
- the factors
- the active memory

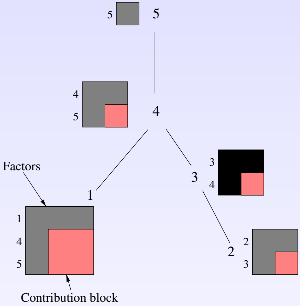




Elimination tree represents the dependencies of the tasks

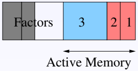
Multifrontal method : *Duff and Reid, '83*





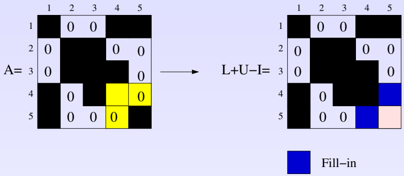
Memory is divided into two parts :

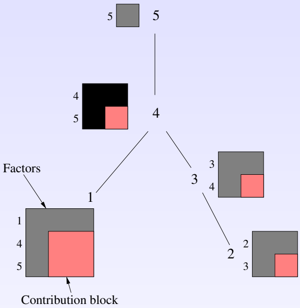
- the factors
- the active memory



Elimination tree represents the dependencies of the tasks

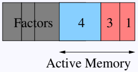
Multifrontal method : *Duff and Reid, '83*





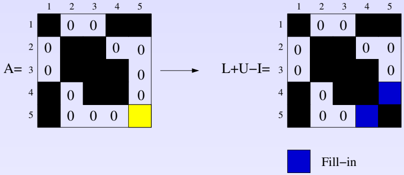
Memory is divided into two parts :

- the factors
- the active memory



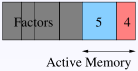
Elimination tree represents the dependencies of the tasks

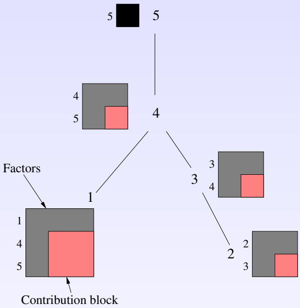
Multifrontal method : *Duff and Reid, '83*



Memory is divided into two parts :

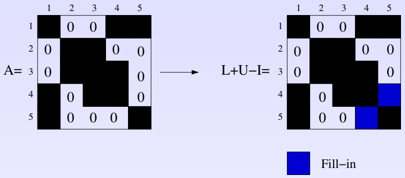
- the factors
- the active memory





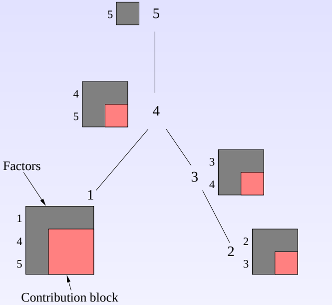
Elimination tree represents the dependencies of the tasks

Multifrontal method : *Duff and Reid, '83*



Memory is divided into two parts :

- the factors
- the active memory



Elimination tree represents the dependencies of the tasks

Outline

Memory related issues

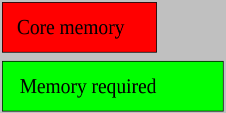
- Out-of-core to "extend" memory
- Memory scalability to equilibrate active memory

Memory related issues

- Out-of-core to “extend” memory
- Memory scalability to equilibrate active memory

Out-of-core to extend memory

Physical constraint



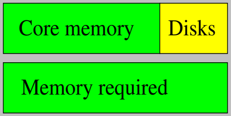
Memory crash

Software challenge

- Implementation of an out-of-core execution scheme within MUMPS

Out-of-core to extend memory

Physical constraint



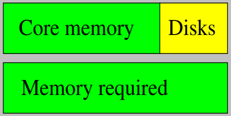
Use of disks

Software challenge

- Implementation of an out-of-core execution scheme within `MUMPS`

Out-of-core to extend memory

Out-of-core



Use of disks

Software challenge

- Implementation of an out-of-core execution scheme within MUMPS

- Compatibility with : numerical pivoting (partial pivoting, 2x2), distributed memory environment
- logical unit of transfer to disk independent of both computational unit (frontal matrices) and independent of low level caches used to perform effective I/O.

OOC factorization and solution

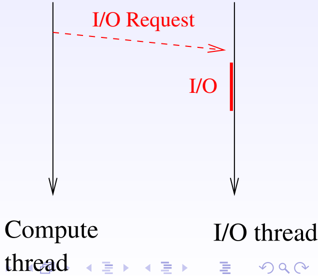
Work performed in the context of the PhD thesis of E. Agullo, ENS-Lyon (2006-2008) and M. Slavova CERFACS-Toulouse (2006-2009)

- Models and algorithms to reduce I/O traffic, in case the active storage goes to disk
- Out-of-core storage of factors :

→ write factors to disk as soon as they are computed

Asynchronous approach

- Factors copied to a user buffer (panel-oriented approach)
- Dedicated I/O thread writes buffers to disk
- Low-level I/O can avoid system buffering



Out-of-core factorization : performance

Factorization time (seconds) on AMD Opteron cluster :

	Direct I/O		Pagecache		In-core
Matrix	Synch.	Asynch.	Synch.	Asynch	
SHIP003	43.6	36.4	37.7	35.0	33.2
XENON2	45.4	33.8	42.1	33.0	31.9
CONESHL2	158.7	123.7	144.1	125.1	Out-of-mem
QIMONDA07 ∗	159.2	98.6	190.1	171.1	Out-of-mem

∗ Special matrix with huge factors and few computations.

Out-of-core and parallelism : critical issues

Epicure matrix (EDF, $N = 853632$)

- 1 proc :
  - Total memory (InCore) = 20.8 GBytes
  - Active memory (OOC) = 3.7 GBytes
- 16 procs :
  - Total memory (InCore) = 2.4 GBytes
  - Active memory (OOC) = 1.4 GBytes
- 24 procs :
  - Total memory (InCore) = 1.5 GBytes
  - Active memory (OOC) = 1.0 GBytes

Active memory per processor thus need be controlled

Tree traversals and memory-aware mapping algorithms need be designed.

Memory related issues

- Out-of-core to "extend" memory
- Memory scalability to equilibrate active memory

Memory scalability of a multifrontal solver

Problem

- Memory consumption is often a bottleneck for direct solvers.
- We want to redesign the mapping, that is the choice of a set of processors for each node of the tree. It should be able to handle different contexts (in-core, out-of-core...) and objectives (factorization, solve phase...).

Memory efficiency

Definition : *Memory Efficiency* on p processors

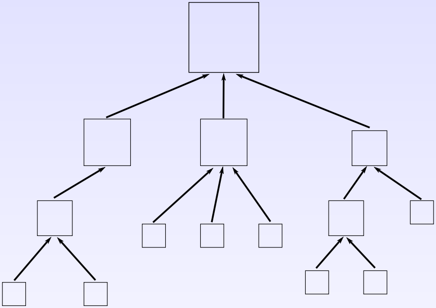
$e(p) = \frac{M_{seq}}{p \times M_{max}(p)}$, M_{seq} : serial storage, M_{max} : parallel storage

Results : Memory Efficiency (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

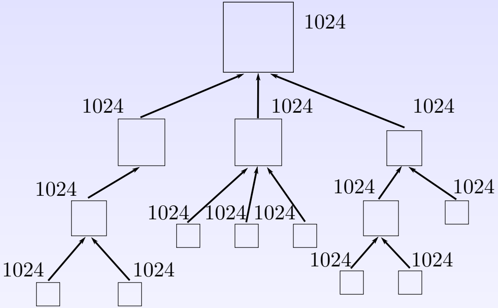
Mapping techniques

Processor-to-node mapping :



Mapping techniques

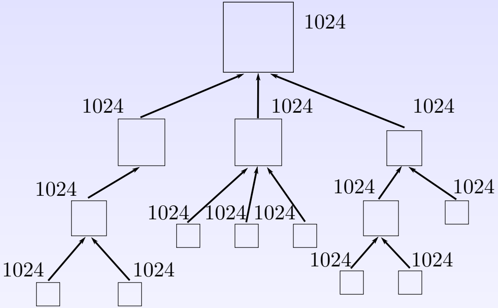
Processor-to-node mapping : all-to-one mapping (postorder traversal)



- Optimal memory scalability : $M_{max} = M_{seq}/p$.
- Poor parallelism : only intra-node parallelism is exploited.

Mapping techniques

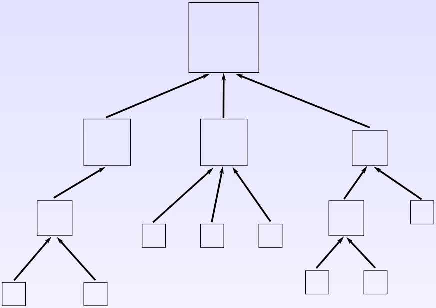
Processor-to-node mapping : all-to-one mapping (postorder traversal)



- Optimal memory scalability : $M_{max} = M_{seq}/p$.
- Poor parallelism : only intra-node parallelism is exploited.

Mapping techniques

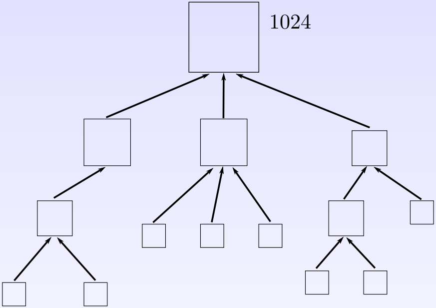
Processor-to-node mapping : proportional mapping



- Good properties for parallelism :
  Flops aware, inter-node and intra-node parallelism, communication locality.

Mapping techniques

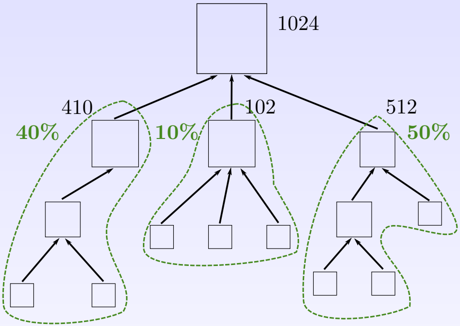
Processor-to-node mapping : proportional mapping



- Good properties for parallelism :
  Flops aware, inter-node and intra-node parallelism, communication locality.

Mapping techniques

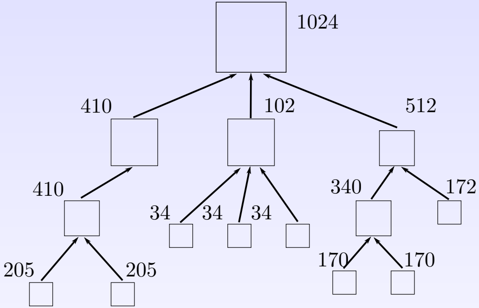
Processor-to-node mapping : proportional mapping



- Good properties for parallelism :
  Flops aware, inter-node and intra-node parallelism, communication locality.

Mapping techniques

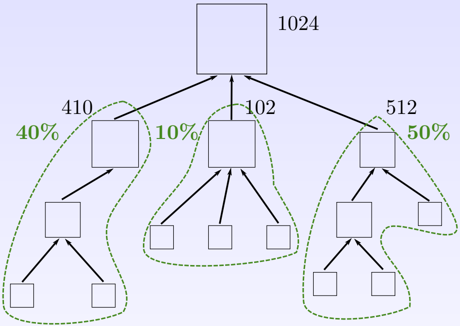
Processor-to-node mapping : proportional mapping



- Good properties for parallelism :
  Flops aware, inter-node and intra-node parallelism, communication locality.

Mapping techniques

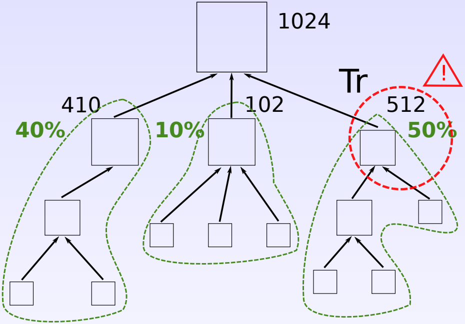
Processor-to-node mapping : "memory-aware" mapping



1. Try to apply proportional mapping.
2. Check constraint for each subtree : is there enough memory ?
   If not, node factorizations are serialized.

Mapping techniques

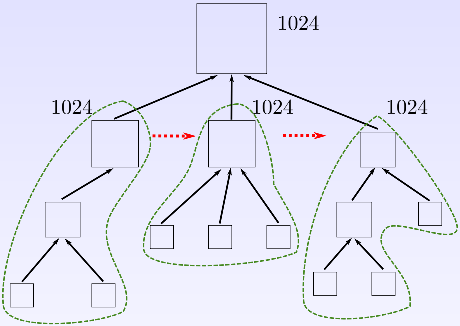
Processor-to-node mapping : "memory-aware" mapping



1. Try to apply proportional mapping.
2. Check constraint for each subtree : is there enough memory ?
   If not, node factorizations are serialized.

Mapping techniques

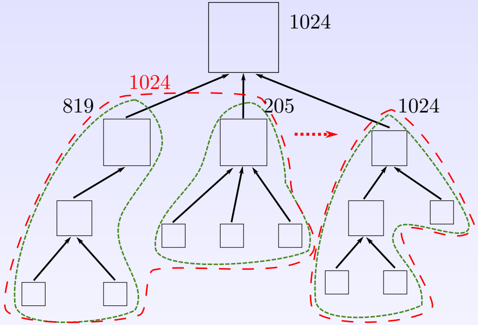
Processor-to-node mapping : "memory-aware" mapping



1. Try to apply proportional mapping.
2. Check constraint for each subtree : is there enough memory ?
   If not, node factorizations are serialized.

Mapping techniques

Processor-to-node mapping : a finer "memory-aware" mapping ?



1. Try to find groups of subtrees on which proportional mapping works.
2. Serialize these groups.

Preliminary work : scheduling influences memory

- Modify tree mapping to reduce the memory requirement during parallel executions

- Estimated core memory (MB) - `AUDIKW_1`, 16 procs :

Factors		Current (MUMPS 4.9.2)	Memory-oriented mapping
In-Core	Max	4038	2587
	Avg	3345	2446
Out-Of-Core	Max	3028	968
	Avg	2251	827

- under development (PhD thesis of Rouet, in continuation of preliminary work by Agullo et al.)
- Another critical issue to address is the reliability of the memory estimates in a dynamic scheduling context.

Outline

Efficiency of the solution phase

- Sparsity in the right hand side and/or solution
- Multiple entries of A^{-1}

Efficiency of the solution phase

Sparsity in the right hand side and/or solution

Multiple entries of A^{-1}

Exploit sparsity of the right-hand-side/solution

Applications

- Highly reducible matrices and/or sparse right-hand-sides (linear programming, seismic processing)
- Null-space basis computation
- Partial computation of A^{-1}
  - Computing variances of the unknowns of a data fitting problem = computing the diagonal of a so-called variance-covariance matrix.
  - Computing short-circuit currents = computing blocks of a so-called impedance matrix.
  - Approximation of the condition number of a SPD matrix.

Core idea

An efficient algorithm has to take advantage of the sparsity of A and of both the right-hand sides and the solution.

Exploit sparsity of the right-hand-side/solution

Applications

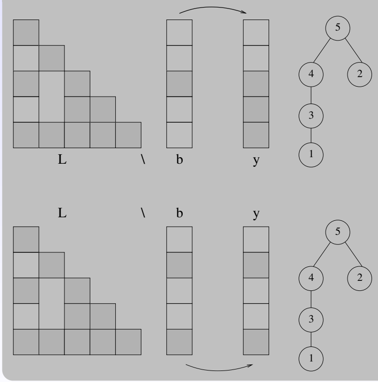
- Highly reducible matrices and/or sparse right-hand-sides (linear programming, seismic processing)
- Null-space basis computation
- Partial computation of A^{-1}
  - Computing variances of the unknowns of a data fitting problem = computing the diagonal of a so-called variance-covariance matrix.
  - Computing short-circuit currents = computing blocks of a so-called impedance matrix.
  - Approximation of the condition number of a SPD matrix.

Core idea

An efficient algorithm has to take advantage of the sparsity of A and of both the right-hand sides and the solution.

Exploit sparsity in RHS : an quick insight of main properties

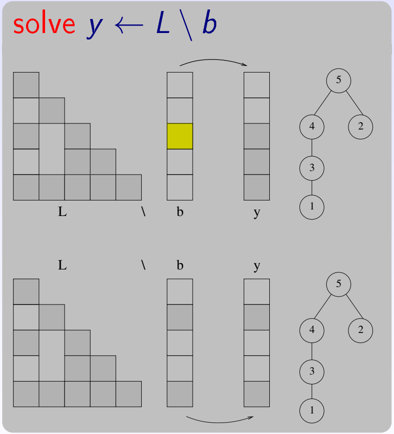
solve $y \leftarrow L \setminus b$



- In all application cases, only part of factors needs to be loaded
- Objectives with sparse RHS
  - Efficient use of the RHS sparsity
  - Characterize LU factors to be loaded from disk
  - Efficiently load only needed factors from disk

(1) Predicting structure of the solution vector, *Gilbert-Liu*, '93

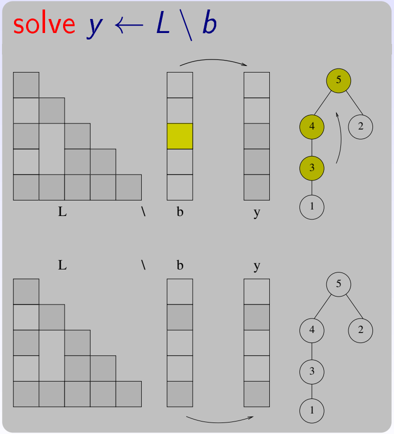
Exploit sparsity in RHS : an quick insight of main properties



- In all application cases, only part of factors needs to be loaded
- Objectives with sparse RHS
  - Efficient use of the RHS sparsity
  - Characterize LU factors to be loaded from disk
  - Efficiently load only needed factors from disk

(1) Predicting structure of the solution vector, *Gilbert-Liu*, '93

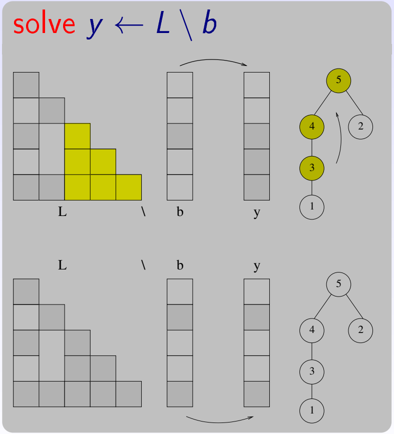
Exploit sparsity in RHS : an quick insight of main properties



- In all application cases,
  only part of factors needs to be loaded
- Objectives with sparse RHS
  - Efficient use of the RHS sparsity
  - Characterize LU factors to be loaded from disk
  - Efficiently load only needed factors from disk

(1) Predicting structure of the solution vector, *Gilbert-Liu*, '93

Exploit sparsity in RHS : an quick insight of main properties

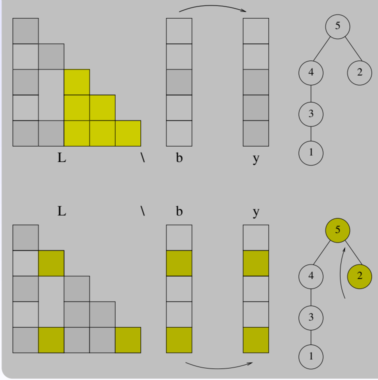


- In all application cases, only part of factors needs to be loaded
- Objectives with sparse RHS
  - Efficient use of the RHS sparsity
  - Characterize LU factors to be loaded from disk
  - Efficiently load only needed factors from disk

(1) Predicting structure of the solution vector, *Gilbert-Liu*, '93

Exploit sparsity in RHS : an quick insight of main properties

solve $y \leftarrow L \setminus b$



- In all application cases, only part of factors needs to be loaded
- Objectives with sparse RHS
  - Efficient use of the RHS sparsity
  - Characterize LU factors to be loaded from disk
  - Efficiently load only needed factors from disk

(1) Predicting structure of the solution vector, *Gilbert-Liu*, '93

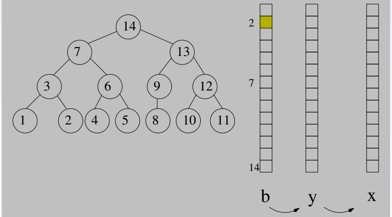
Application : elements in A^{-1}

$AA^{-1} = I$, specific entry : $a_{ij}^{-1} = (A^{-1}e_j)_i$,
$A^{-1}e_j$ – column j of A^{-1}

Theorem : structure of x (based on Gilbert and Liu '93)

For any matrix A such that $A = LU$, the structure of the solution (x) is given by the set of nodes reachable from nodes associated with right-hand side entries by paths in the e-tree.

compute some elements in A^{-1}



Which factors needed to compute a_{82}^{-1}?
$a_{82}^{-1} = (U^{-1}(L^{-1}e_2))_8$

Application : elements in A^{-1}

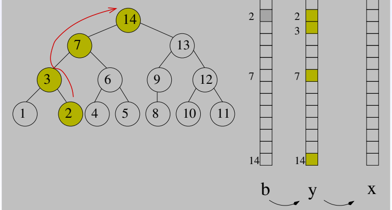
$AA^{-1} = I$,   specific entry : $a_{ij}^{-1} = (A^{-1}e_j)_i$,

$A^{-1}e_j$ – column j of A^{-1}

Theorem : structure of x (based on Gilbert and Liu '93)

For any matrix A such that $A = LU$, the structure of the solution (x) is given by the set of nodes reachable from nodes associated with right-hand side entries by paths in the e-tree.

compute some elements in A^{-1}



Which factors needed to compute a_{82}^{-1}?

$a_{82}^{-1} = (U^{-1}(L^{-1}e_2))_8$

We have to load :

L factors associated with nodes $2, 3, 7, 14$

Application : elements in A^{-1}

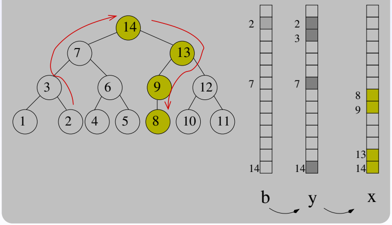
$AA^{-1} = I$, specific entry : $a_{ij}^{-1} = (A^{-1}e_j)_i$,

$A^{-1}e_j$ – column j of A^{-1}

Theorem : structure of x (based on Gilbert and Liu '93)

For any matrix A such that $A = LU$, the structure of the solution (x) is given by the set of nodes reachable from nodes associated with right-hand side entries by paths in the e-tree.

compute some elements in A^{-1}



Which factors needed to compute a_{82}^{-1}?

$a_{82}^{-1} = (U^{-1}(L^{-1}e_2))_8$

We have to load :

L factors associated with nodes $2, 3, 7, 14$

and U factors associated with nodes $14, 13, 9, 8$

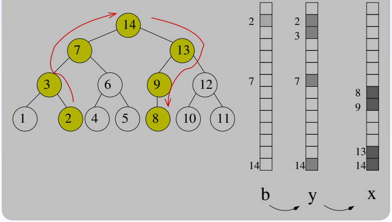
Application : elements in A^{-1}

$AA^{-1} = I$, specific entry : $a_{ij}^{-1} = (A^{-1}e_j)_i$,
$A^{-1}e_j$ – column j of A^{-1}

Theorem : structure of x (based on Gilbert and Liu '93)

For any matrix A such that $A = LU$, the structure of the solution (x) is given by the set of nodes reachable from nodes associated with right-hand side entries by paths in the e-tree.

compute some elements in A^{-1}



Which factors needed to compute a_{82}^{-1}?
$a_{82}^{-1} = (U^{-1}(L^{-1}e_2))_8$

We have to load :
L factors associated with nodes $2, 3, 7, 14$
and U factors associated with nodes $14, 13, 9, 8$

Note :

A part of the tree is concerned

Entries of the inverse : a single one

Notation for later use

$P(i)$: denotes the nodes in the unique path from the node i to the root node r (including i and r).

$P(S)$: denotes $\bigcup_{s \in S} P(s)$ for a set of nodes S.

Use the elimination tree

For each requested (diagonal) entry a_{ii}^{-1},

(1) visit the nodes of the elimination tree from the node i to the root : at each node access necessary parts of $\mathbf{L}$,

(2) visit the nodes from the root to the node i again ; this time access necessary parts of $\mathbf{U}$.

Experiments : interest of exploiting sparsity

Implementation

These ideas have been implemented in MUMPS during Tz. Slavova's PhD.

Experiments : computation of the diagonal of the inverse of matrices from data fitting in Astrophysics (CESR, Toulouse)

Matrix size	Time (s)	
	No ES	ES
46,799	6,944	472
72,358	27,728	408
148,286	>24h	1,391

Interest

Exploiting sparsity of the right-hand sides reduces the number of accesses to the factors (in-core : number of flops, out-of-core : accesses to hard disks).

Efficiency of the solution phase

- Sparsity in the right hand side and/or solution
- Multiple entries of A^{-1}

Entries of the inverse : multiple entries

Same as before...

For each requested (diagonal) entry a_{ii}^{-1},

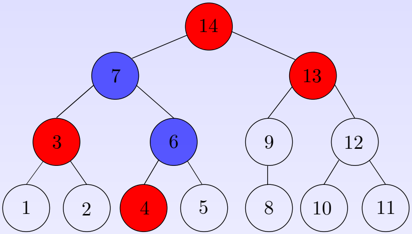
(1) visit the nodes in the path from node i to the root (access to parts of $\mathbf{L}$,

(2) visit the same nodes again (in reverse order) ; this time access necessary parts of $\mathbf{U}$.

...only this time

- a block-wise solve is necessary,
- we access parts of $\mathbf{L}$ for all the solves in the upward traversal of the tree only once,
- we access parts of $\mathbf{U}$ for all the solves in the downward traversal of the tree only once.

Entries of the inverse : multiple entries

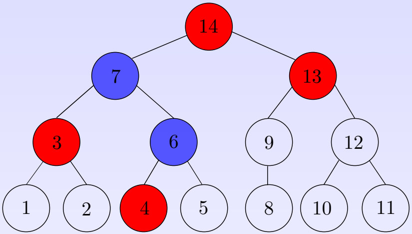


[The requested entries in the diagonal of the inverse are shown in red]

Requested	accesses
$a_{3,3}^{-1}$	$\{3, 7, 14\}$
$a_{4,4}^{-1}$	$\{4, 6, 7, 14\}$
$a_{13,13}^{-1}$	$\{13, 14\}$
$a_{14,14}^{-1}$	$\{14\}$

If we were to compute all these four entries, we just need to access the data associated with the nodes in red and blue.

Entries of the inverse : multiple entries



[The requested entries S in the diagonal of the inverse are in red.]

Requested	accesses
$a^{-1}_{3,3}$	$\{3, 7, 14\}$
$a^{-1}_{4,4}$	$\{4, 6, 7, 14\}$
$a^{-1}_{13,13}$	$\{13, 14\}$
$a^{-1}_{14,14}$	$\{14\}$

If we compute all at the same time, we access the data associated with nodes in $P(S) = \{3, 4, 6, 7, 13, 14\}$ shown in red and blue.

$$\mathrm{Cost}(S) = \sum_{i \in P(S)} w(i) = w(3) + w(4) + w(6) + w(7) + w(13) + w(14)$$

Entries of the inverse : multiple entries

In reality (or in a particular setting)...

We are to compute a set R of requested entries. Usually $|R|$ is large.

The memory requirement for the solution vectors is $|R| \times n$, where n is the number of rows/cols of the matrix.

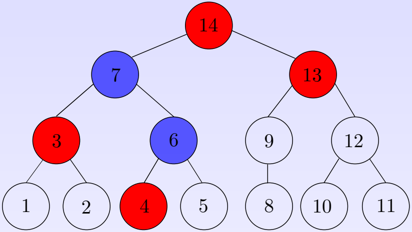
We can hold at most B many solution vectors, requiring $B \times n$ memory.

Tree-Partitioning problem

Given a set R of nodes of a node-weighted tree and a number B (*blocksize*), find a partition $\Pi(R) = \{R_1, R_2, \ldots\}$ such that $\forall R_k \in \Pi, |R_k| \leq B$, and has minimum cost

$$\mathrm{Cost}(\Pi) = \sum_{R_k \in \Pi} \mathrm{Cost}(R_k) \quad \text{where} \quad \mathrm{Cost}(R_k) = \sum_{i \in P(R_k)} w(i)$$

Entries of the inverse : multiple entries



$[R = \{3, 4, 13, 14\}$ and $B = 3]$

Bare minimum cost (mc) :

$$\mathrm{Cost}(R) = w(3)+w(4)+w(6) + w(7) + w(13) + w(14)$$

	Partition	Accesses	$\mathrm{Cost}(\Pi)$
Π'	$R_1 = \{3, 13, 14\}$ $R_2 = \{4\}$	$P(R_1) = \{3, 7, 13, 14\}$ $P(R_2) = \{4, 6, 7, 14\}$	$mc + w(7) + w(14)$
Π''	$R_1 = \{3, 4, 14\}$ $R_2 = \{13\}$	$P(R_1) = \{3, 4, 6, 7, 14\}$ $P(R_2) = \{13, 14\}$	$mc + w(14)$

Permuting multiple entries : performance

With a postorder (Po in the table) ordering of the requested entries we can obtain good tree locality properties and decrease memory requirements by a factor of 2 or 3 !

Experiments the set of matrices from Astrophysics :

Matrix size	Lower bound	Factors loaded [MB]		
		No ES	Nat	Po
46,799	11,105	137,407	12,165	11,628
72,358	1,621	433,533	5,800	1,912
148,286	9,227	1,677,479	18,143	9,450

On-going work and open issues

General case of selected set of entries in A^1

For multiple off-diagonal entries hypergraph modelling and partitionning can further improve the performance

Parallel processing

- By construction, columns in the same block must be associated to nodes close to each other in the tree.
- In a distributed memory context, to limit memory communication volumes, nodes close to each other are often mapped on a small subset of the set of processors.
- Efficient partitioning for sparsity seems to be bad for parallelism ?
- On going work Phd of F.H. Rouet (Toulouse) : some promising algorithms/results.

Outline

Concluding remarks

Towards a state of the art parallel direct solver (I)

Preprocessing

Fully parallel on distributed matrices/graphs ;
Mixed symbolic and numerical issues ;
Design specific algorithms for important classes of problems (for ex. augmented systems matrices)

Memory use

- Memory aware algorithms ;
- Memory peak (per processor) is difficult to control in a dynamic context : efficient preprocessing critical to have good memory estimates.

Memory locality

Design algorithms providing good locality of memory accesses :
"Old" algorithms designed for Out-Of-Core or for distributed memory context might be relevant for multicore.

Towards a state of the art parallel direct solver (II)

Efficient solution phases (forward and backward)

Take into account sparse multiple right-hand-sides problems ;
Analysis and factorization stategies might be guided by the performance of the solve (factor size and distribution)

Exploiting large number of cores ?

Can we keep memory demanding strategies such as numerical pivoting ?
Hybrid approaches (Domain Decomposition, Schur, Block Cimmino) provide an additional level of parallelism.

More questions than answers and certainly much work in perspective !

Outline

Appendix

Unsymmetric test problems

	Order	nnz	$nnz(L\|U)$ $\times 10^6$	Ops $\times 10^9$	Origin
conv3d64	836550	12548250	2693.9	23880	CEA/CESTA
fidapm11	22294	623554	11.3	4.2	Matrix market
lhr01	1477	18427	0.1	0.007	UF collection
qimonda07	8613291	66900289	556.4	45.7	QIMONDA AG
twotone	120750	1206265	25.0	29.1	UF collection
ultrasound80	531441	33076161	981.4	3915	Sosonkina
wang3	26064	177168	7.9	4.3	Harwell-Boeing
xenon2	157464	3866688	97.5	103.1	UF collection

Ops and $nnz(L|U)$ when provided obtained with `METIS` and default `MUMPS` input parameters.
UF Collection : University of Florida sparse matrix collection.
Harwell-Boeing : Harwell-Boeing collection.

PARASOL : Parasol collection

Symmetric test problems

	Order	nnz	$nnz(L)$ $\times 10^6$	Ops $\times 10^9$	Origin
audikw_1	943695	39297771	1368.6	5682	PARASOL
brgm	3699643	155640019	4483.4	26520	BRGM
coneshl2	837967	22328697	239.1	211.2	Samtech S.A.
coneshl	1262212	43007782	790.8	1640	Samtech S.A.
cont-300	180895	562496	12.6	2.6	Maros & Meszanos
cvxqp3	17500	69981	6.3	4.3	CUTEr
gupta2	62064	4248386	8.6	2.8	A. Gupta, IBM
ship_003	121728	4103881	61.8	80.8	PARASOL
stokes128	49666	295938	3.9	0.4	Arioli
thread	29736	2249892	24.5	35.1	PARASOL