Computation of a matrix inverse in MUMPS

François-Henry Rouet and Bora Uçar,
In collaboration with: P. R. Amestoy, I. S. Duff and Y. Robert.

Toulouse, April 16th, 2010

Problem definition and motivations

Problem definition

Given a large sparse matrix $\mathbf{A}$, compute the entries in the diagonal of $\mathbf{A}^{-1}$.

Motivation/applications

- Linear least-squares solutions: Variance of the variables is at the diagonal of the inverse of a large sparse matrix.
- Quantum-scale device simulation: The use of Green's function reduces the problem to computing the diagonal entries of the inverse of a large sparse matrix.
- Various other simulations: computation of short-circuit currents, approximation of condition numbers.

How to compute the entries of the inverse?

Computing a set of entries in $\mathbf{A}^{-1}$ involves the solution of a set of linear systems. For each requested diagonal entry, we solve

$$a_{ii}^{-1} = e_i^T \mathbf{A}^{-1} e_i \; .$$

An efficient algorithm has to take advantage of the sparsity of $\mathbf{A}$ and the canonical vectors e_i.

- In numerical linear algebra, one never computes the inverse of a matrix.
- The above equation can be solved with Gaussian elimination, a.k.a., LU decomposition: Assume we have $\mathbf{LU} = \mathbf{A}$, then

$$\begin{cases} x = \mathbf{L}^{-1} e_i & \triangleright \text{ solve for } x \\ y = \mathbf{U}^{-1} x & \triangleright \text{ solve for } y \\ a_{ii}^{-1} = e_i^T y & \triangleright \text{ get the } i\text{th component} \end{cases}$$

Sparse LU decomposition

A common variant of LU decomposition

Has $n-1$ steps; at step $k = 1, 2, \ldots, n-1$, the formulae

$$a_{ij}^{(k+1)} \leftarrow a_{ij}^{(k)} - \left(a_{ik}^{(k)}/a_{kk}^{(k)}\right) a_{kj}^{(k)} , \quad \text{for } i,j > k$$

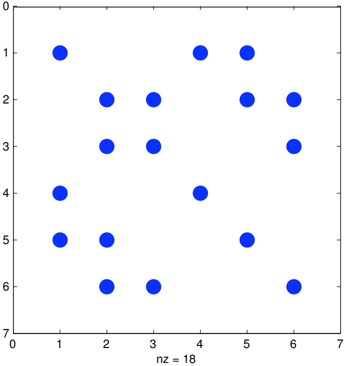
are used to create zeros below the diagonal entry in column k.

Each updated entry $a_{ij}^{(k+1)}$ overwrites $a_{ij}^{(k)}$, and the multipliers $l_{ik} = a_{ik}^{(k)}/a_{kk}^{(k)}$ may overwrite $a_{ik}^{(k)}$.

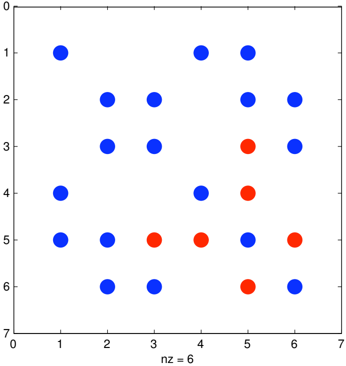
The process results in a unit lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$ such that $\mathbf{A} = \mathbf{LU}$.

Fill-in occurs: some zeros in $\mathbf{A}^{(k)}$ become nonzero in $\mathbf{A}^{(k+1)}$.

Sparse LU decomposition: filled-in matrix

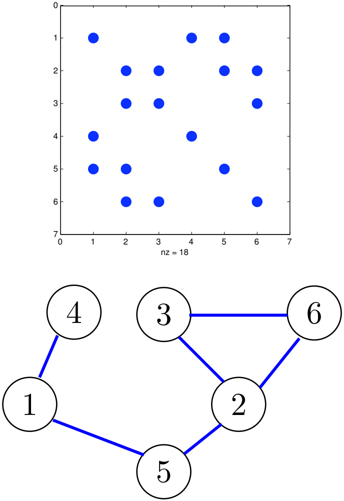


The pattern of **A**

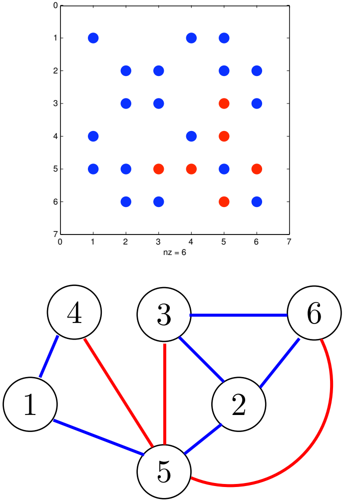


The pattern of **L** + **U**

Sparse LU decomposition: the graphs

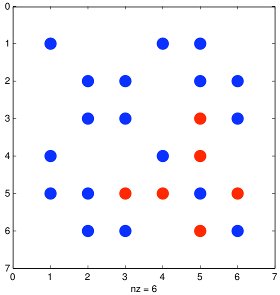


[The graph of **A**]



[The graph of **L** + **U**]

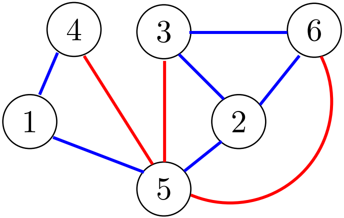
Sparse LU decomposition: the elimination tree



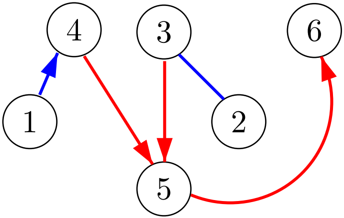
The elimination tree

The elimination tree is a spanning tree of the graph of **L** + **U**.

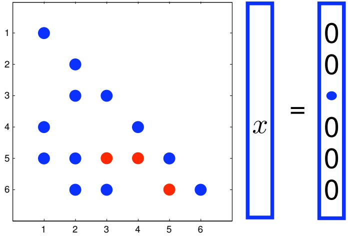
Node i is the father of node j if $l_{ij} \neq 0$ and i is the smallest such index.

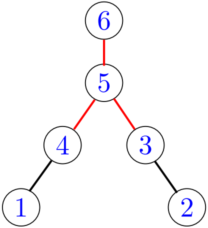


[The graph of **L** + **U**]



Solve $\mathbf{L}x = e_3$ for x





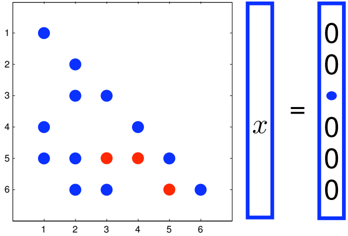
[Elimination tree redrawn]

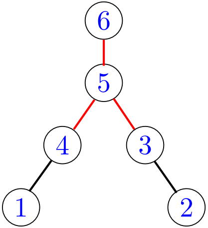
$$l_{11}x_1 = 0 \Rightarrow x_1 = 0$$
$$l_{22}x_2 = 0 \Rightarrow x_2 = 0$$

$$l_{41}x_1 + l_{44}x_4 = 0 \Rightarrow x_4 = 0$$

The elimination tree: what to do with it?

Solve $\mathbf{L}x = e_3$ for x





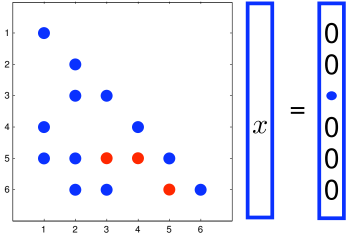
[Elimination tree redrawn]

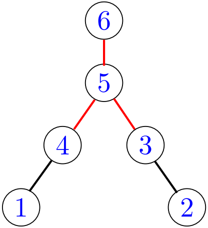
$$l_{32}x_2 + l_{33}x_3 \neq 0 \Rightarrow x_3 \neq 0$$

$$\cdots + l_{53}x_3 + l_{55}x_5 = 0 \Rightarrow x_5 \neq 0$$

$$l_{63}x_3 + l_{65}x_5 + l_{66}x_6 = 0 \Rightarrow x_6 \neq 0$$

Solve $\mathbf{L}x = e_3$ for x





[Elimination tree redrawn]

$$l_{32}x_2 + l_{33}x_3 \neq 0 \Rightarrow x_3 \neq 0$$

$$\cdots + l_{53}x_3 + l_{55}x_5 = 0 \Rightarrow x_5 \neq 0$$

$$l_{63}x_3 + l_{65}x_5 + l_{66}x_6 = 0 \Rightarrow x_6 \neq 0$$

Visit the nodes of the tree starting from node 3 to the root; they will be the nonzero entries of x; solve the associated equations.

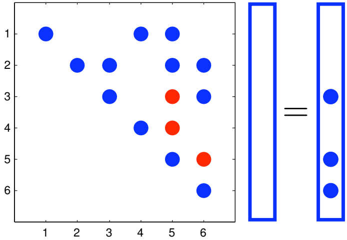
Entries of the inverse: Back to the equations

To find a_{ii}^{-1}, solve the equations

$$\begin{cases} x = \mathbf{L}^{-1} e_i & \triangleright \text{ solve for } x \\ y = \mathbf{U}^{-1} x & \triangleright \text{ solve for } y \\ a_{ii}^{-1} = e_i^T y & \triangleright \text{ get the } i\text{th component} \end{cases}$$

Assume we are looking for a_{33}^{-1}. We have seen how we solve for x.

Solve $\mathbf{U}y = x$ until we get the 3rd entry.



We need to solve:

$$u_{33}y_3 + u_{35}y_5 + u_{36}y_6 = x_3$$

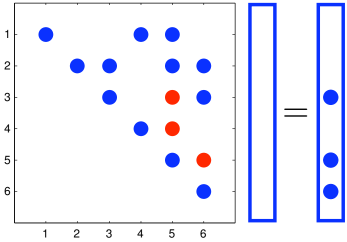
So we need y_5, y_6

To find a_{ii}^{-1}, solve the equations

$$\begin{cases} x = \mathbf{L}^{-1} e_i & \triangleright \text{ solve for } x \\ y = \mathbf{U}^{-1} x & \triangleright \text{ solve for } y \\ a_{ii}^{-1} = e_i^T y & \triangleright \text{ get the } i\text{th component} \end{cases}$$

Assume we are looking for a_{33}^{-1}. We have seen how we solve for x.

Solve $\mathbf{U}y = x$ until we get the 3rd entry.



We need to solve:

$$u_{33}y_3 + u_{35}y_5 + u_{36}y_6 = x_3$$

So we need y_5, y_6

$$u_{55}y_5 + u_{56}y_6 = x_5$$

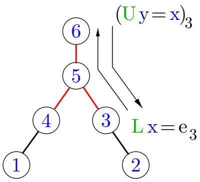
$$u_{66}y_6 = x_6$$

Forget the other vars/eqns

Entries of the inverse: a single one

To find a_{ii}^{-1}, solve the equations

$$\begin{cases} x = \mathbf{L}^{-1} e_i & \triangleright \text{ solve for } x \\ y = \mathbf{U}^{-1} x & \triangleright \text{ solve for } y \\ a_{ii}^{-1} = e_i^T y & \triangleright \text{ get the } i\text{th entry} \end{cases}$$



$$\begin{aligned} l_{33} x_3 &= 1 \\ l_{53} x_3 + l_{55} x_5 &= 0 \\ l_{63} x_3 + l_{65} x_5 + l_{66} x_6 &= 0 \end{aligned}$$

$$\begin{aligned} u_{33} y_3 + u_{35} y_5 + u_{36} y_6 &= x_3 \\ u_{55} y_5 + u_{56} y_6 &= x_5 \\ u_{66} y_6 &= x_6 \end{aligned}$$

Forget the other vars/eqns

$x = \mathbf{L}^{-1} e_3$, visit the nodes of the tree starting from node 3 to the root; solve the equations associated with $\mathbf{L}$.

$a_{33}^{-1} = \left(\mathbf{U}^{-1} x\right)_3$, visit the nodes of the tree starting from the root to node 3; solve the equations associated with $\mathbf{U}$.

Entries of the inverse: a single one

To find a_{ii}^{-1}, solve the equations

$$\begin{cases} x = \mathbf{L}^{-1} e_i & \triangleright \text{solve for } x \\ y = \mathbf{U}^{-1} x & \triangleright \text{solve for } y \\ a_{ii}^{-1} = e_i^T y & \triangleright \text{get the } i\text{th entry} \end{cases}$$

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 $\mathbf{U}$.

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 **L**,

(2) visit the nodes from the root to the node i again; this time access necessary parts of **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).

Entries of the inverse: multiple entries

Same as before...

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

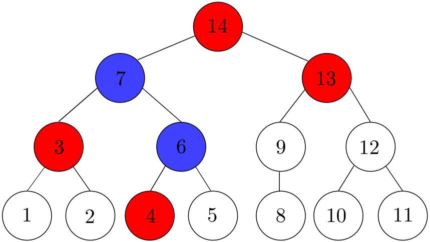
(1) visit the nodes in $P(i)$: at each node access necessary parts of **L**,

(2) visit the nodes in $P(i)$ again (in reverse order); this time access necessary parts of **U**.

...only this time

- a block-wise solve is necessary,
- we access parts of **L** for all the solves in the upward traversal of the tree only once,
- we access parts of **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
$a^{-1}_{3,3}$	$\{3, 7, 14\}$
$a^{-1}_{4,4}$	$\{4, 6, 7, 14\}$
$a^{-1}_{13,13}$	$\{13, 14\}$
$a^{-1}_{14,14}$	$\{14\}$

If we 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

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

Matrices are factored, e.g., the LU-decomposition is computed, in a coarser scheme, and the factors are represented as a (sparse) collection of dense (much) smaller submatrices.

Those submatrices are stored on disks (*out-of-core* setting).

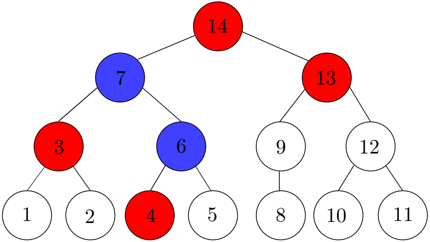
When we access a part of **L** (or **U**), we load the associated dense submatrix from the disk; at node i of the tree the cost of the load is proportional to $w(i)$: the weight of the node.

Cost

Given a set of requested entries S, we visit all the nodes in $P(S)$, and the total cost is $\mathrm{Cost}(S) = \sum_{i \in P(S)} w(i)$.

Assuming we can hold S many solution vectors in memory, this is the absolute minimum we can do for a given set S of requested entries.

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 need to access the data associated with the 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.

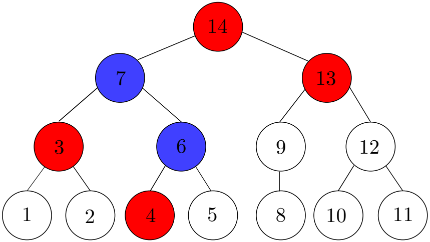
We can hold at most B many solution vectors, requiring $B \times n$ memory.

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''	$R_1 = \{3, 4, 14\}$ $R_2 = \{13\}$	$P(R_1) = \{3, 4, 6, 7, 14\}$ $P(R_2) = \{13, 14\}$	$mc + w(14)$

Entries of the inverse: multiple entries

Can we get significant differences in pratice ?
Experiments on the same 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

Motivations

A simple strategy (postorder, presented later), can decrease memory requirements by a factor of 2 or 3 !
Can we go further ?

Tree-Partitioning problem

Tree-Partitioning problem

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

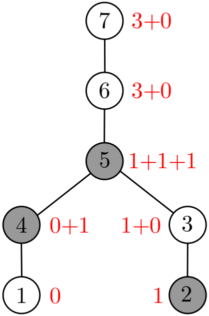
$$\mathrm{Cost}(\Pi) = \sum_{R_k \in \Pi} \mathrm{Cost}(R_k) \quad \text{where} \quad \mathrm{Cost}(R_k) = \sum_{i \in P(R_k)} w(i)$$

- We showed that it is NP-complete.
- There is a non-trivial lower bound.
- The case $B = 2$ is special and can be solved in polynomial time.
- A simple algorithm gives an approximation guarantee.
- We have a heuristic which gives extremely good results.
- We have hypergraph models that address the most general cases.

Number of requests

nr(i): number of requested nodes in the subtree rooted at node i

$$\mathrm{nr}(i) = \sum_{j \in children(i)} \mathrm{nr}(j) + \mathrm{req}(i)$$



Then, the lower bound is given by:

$$\sum_i w(i) \left\lceil \frac{\mathrm{nr}(i)}{B} \right\rceil$$

A simplistic heuristic

The simplistic heuristic $\mathcal{H}$

Put the requested nodes in post-order (in the increasing order) and cut in blocks of size B.

Simple; runs in $\mathcal{O}(n)$ for a tree with n nodes. And it comes with an approximation guarantee.

Approximation guarantee

Let $\mathrm{Cost}^{\mathcal{H}}$ be the cost of the heuristic $\mathcal{H}$ and $\mathrm{Cost}^{\star}$ be the optimal cost. Then

$$\mathrm{Cost}^{\mathcal{H}} \leq 2 \times \mathrm{Cost}^{\star}$$

A simplistic heuristic

The simplistic heuristic $\mathcal{H}$

Put the requested nodes in post-order (in the increasing order) and cut in blocks of size B.

Approximation guarantee

$$\mathrm{Cost}^{\mathcal{H}} \leq 2 \times \mathrm{Cost}^{\star}$$

Why? In a post-order all nodes in a subtree are numbered consecutively. At node i the lower bound $\mathcal{L}$ was counting $w(i)\left\lceil \frac{\mathrm{nr}(i)}{B} \right\rceil$. Due to consecutive number of the nodes, post-order can incur, at node i, at most

$$w(i)\left(\left\lceil \frac{\mathrm{nr}(i)}{B} \right\rceil + 1\right)$$

The sum of the excess is $\leq \mathcal{L}$, hence

$$\mathrm{Cost}^{\mathcal{H}} \leq 2 \times \mathcal{L} \leq 2 \times \mathrm{Cost}^{\star}$$

A simplistc heuristic: experiments. . .

Experiments on a set a various matrices: the ratio of number of accesses over the lower bound is measured:

Matrix	10% diagonal	10% off-diag
CESR(46799)	1.01	1.28
af2356	1.02	2.09
boyd1	1.03	1.92
ecl32	1.01	2.31
gre1107	1.17	1.89
saylr4	1.06	1.92
sherman3	1.04	2.51
grund/bayer07	1.05	1.96
mathworks/pd	1.09	2.10
stokes64	1.05	2.35

⇒ topological orders provide good results for the diagonal case, but are not efficient enough for the general case.

A special case and the general case

A special case: $B = 2$

We have an exact algorithm running in $\mathcal{O}(n)$ time, for a tree with n nodes.

Essential idea: find the best matching $\mathcal{M}$ among the requested nodes.

The general case: A bisection based heuristic $\mathcal{B}$

Or almost a general case: $B = 2^{\ell}$ ☺

1: **for** level $= 1$ **to** ℓ **do**
2: Find the best matching $\mathcal{M}$ among the requested nodes
3: for each pair in $\mathcal{M}$ remove one, mark the remaining one as the representative for the other node(s)
4: **end for**
5: Put each vertex in the part of the representative (of its representative of its...)

Running time is $\mathcal{O}(n \log B)$. Preliminary results are very good (work in progress).

Experiments: hypergraph model

We use PaToH [Çatalyürek and Aykanat, '99] for the tests. Here we measure the ratio hypergraph / post-order:

Matrix	10% diagonal	10% off-diag
CESR(46799)	1.01	0.75
af2356	1.03	0.69
boyd1	1.03	0.54
ecl32	1,05	0.56
gre1107	0.86	0.80
saylr4	0.98	0.80
sherman3	0.97	0.65
grund/bayer07	0.97	0.72
mathworks/pd	0.94	0.60
stokes64	0.99	0.80

- Diagonal case: no gain, except for "tough" problems.
- General case: on average, a gain of 30%.

Conclusions

- A new feature in MUMPS (available in the next release ! ☺)
- It raises an interesting combinatorial problems, with many possible approaches.

Perspectives and work in progress

Several extensions and improvements can be studied:

- In-core case.
- Parallel environment.

Thank you for your attention !

Any questions ?