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

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

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

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

An efficient algorithm has to take advantage of the sparsity of $A$ 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 $LU = A$, then

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

A common variant of LU decomposition

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

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

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 \( L \) and an upper triangular matrix \( U \) such that \( A = LU \).

Fill-in occurs: some zeros in \( A^{(k)} \) become nonzero in \( A^{(k+1)} \).
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 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 $Lx = e_3$ for $x$

\[
\begin{align*}
\ell_{11} x_1 &= 0 \Rightarrow x_1 = 0 \\
\ell_{22} x_2 &= 0 \Rightarrow x_2 = 0 \\
\ell_{41} x_1 + \ell_{44} x_4 &= 0 \Rightarrow x_4 = 0
\end{align*}
\]

[Elimination tree redrawn]
The elimination tree: what to do with it?

Solve $Lx = e_3$ for $x$

\[
\begin{bmatrix}
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
0 \\
0 \\
v \\
v \\
v \\
\end{bmatrix}
\]

[Elimination tree redrawn]

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.

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

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

\[l_{63}x_3 + l_{65}x_5 + l_{66}x_6 = 0 \Rightarrow x_6 \neq 0\]
Solve $Lx = e_3$ for $x$

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.

\[
\begin{align*}
132x_2 + 133x_3 & \neq 0 \Rightarrow x_3 \neq 0 \\
\cdots + 153x_3 + 155x_5 & = 0 \Rightarrow x_5 \neq 0 \\
163x_3 + 165x_5 + 166x_6 & = 0 \Rightarrow x_6 \neq 0 
\end{align*}
\]
To find $a_{ii}^{-1}$, solve the equations

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

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

Solve $Uy = x$ until we get the 3rd entry.

We need to solve:

\[
\begin{align*}
 u_{33} y_3 + u_{35} y_5 + u_{36} y_6 &= x_3 \\
\text{So we need } y_5, y_6
\end{align*}
\]
Entries of the inverse: back to the equations

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

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

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

Solve $Uy = x$ 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
To find $a_{ii}^{-1}$, solve the equations

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

\[
l_{33}x_3 = 1 \\
l_{53}x_3 + l_{55}x_5 = 0 \\
l_{63}x_3 + l_{65}x_5 + l_{66}x_6 = 0
\]

\[
 u_{33}y_3 + u_{35}y_5 + u_{36}y_6 = x_3 \\
 u_{55}y_5 + u_{56}y_6 = x_5 \\
 u_{66}y_6 = x_6
\]

Forget the other vars/eqns.
Entries of the inverse: a single one

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

\[
\begin{align*}
  x &= L^{-1}e_i \quad \triangleright \text{solve for } x \\
  y &= U^{-1}x \quad \triangleright \text{solve for } y \\
  a_{ii}^{-1} &= e_i^T y \quad \triangleright \text{get the } i\text{th entry}
\end{align*}
\]

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 $U$. 
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)

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Time (s)</th>
<th>No ES</th>
<th>ES</th>
</tr>
</thead>
<tbody>
<tr>
<td>46,799</td>
<td></td>
<td>6,944</td>
<td>472</td>
</tr>
<tr>
<td>72,358</td>
<td></td>
<td>27,728</td>
<td>408</td>
</tr>
<tr>
<td>148,286</td>
<td>&gt;24h</td>
<td>1,391</td>
<td></td>
</tr>
</tbody>
</table>

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]

<table>
<thead>
<tr>
<th>Requested Entry</th>
<th>Accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_{3,3}^{-1})</td>
<td>({3, 7, 14})</td>
</tr>
<tr>
<td>(a_{4,4}^{-1})</td>
<td>({4, 6, 7, 14})</td>
</tr>
<tr>
<td>(a_{13,13}^{-1})</td>
<td>({13, 14})</td>
</tr>
<tr>
<td>(a_{14,14}^{-1})</td>
<td>({14})</td>
</tr>
</tbody>
</table>

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 $\text{Cost}(S) = \sum_{i \in P(S)} w(i)$.

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.]

<table>
<thead>
<tr>
<th>Requested $a_{ij}$</th>
<th>accesses</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{3,3}^{-1}$</td>
<td>${3, 7, 14}$</td>
<td></td>
</tr>
<tr>
<td>$a_{4,4}^{-1}$</td>
<td>${4, 6, 7, 14}$</td>
<td></td>
</tr>
<tr>
<td>$a_{13,13}^{-1}$</td>
<td>${13, 14}$</td>
<td></td>
</tr>
<tr>
<td>$a_{14,14}^{-1}$</td>
<td>${14}$</td>
<td></td>
</tr>
</tbody>
</table>

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

$$\text{Cost}(\Pi) = \sum_{R_k \in \Pi} \text{Cost}(R_k) \quad \text{where} \quad \text{Cost}(R_k) = \sum_{i \in P(R_k)} w(i)$$
Entries of the inverse: multiple entries

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

Bare minimum cost (mc):

\[
\text{Cost}(R) = w(3) + w(4) + w(6) + w(7) + w(13) + w(14)
\]

<table>
<thead>
<tr>
<th>Partition</th>
<th>Accesses ( P(R) )</th>
<th>Cost ( \text{Cost}(\Pi) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Pi' )</td>
<td>( R_1 = {3, 13, 14} )</td>
<td>( P(R_1) = {3, 7, 13, 14} )</td>
</tr>
<tr>
<td>( R_2 = {4} )</td>
<td>( P(R_2) = {4, 6, 7, 14} )</td>
<td></td>
</tr>
<tr>
<td>( \Pi'' )</td>
<td>( R_1 = {3, 4, 14} )</td>
<td>( P(R_1) = {3, 4, 6, 7, 14} )</td>
</tr>
<tr>
<td>( R_2 = {13} )</td>
<td>( P(R_2) = {13, 14} )</td>
<td></td>
</tr>
</tbody>
</table>
Entries of the inverse: multiple entries

Can we get significant differences in practice?
Experiments on the same set of matrices from Astrophysics:

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Lower bound</th>
<th>Factors loaded [MB]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No ES</td>
</tr>
<tr>
<td>46,799</td>
<td>11,105</td>
<td>137,407</td>
</tr>
<tr>
<td>72,358</td>
<td>1,621</td>
<td>433,533</td>
</tr>
<tr>
<td>148,286</td>
<td>9,227</td>
<td>1,677,479</td>
</tr>
</tbody>
</table>

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

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

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

- 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.
Lower Bound $L$

Number of requests

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

\[
nr(i) = \sum_{j \in \text{children}(i)} nr(j) + \text{req}(i)
\]

Then, the lower bound is given by:

\[
\sum_{i} w(i) \left\lceil \frac{nr(i)}{B} \right\rceil
\]
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 $\text{Cost}^\mathcal{H}$ be the cost of the heuristic $\mathcal{H}$ and $\text{Cost}^\star$ be the optimal cost. Then

$$\text{Cost}^\mathcal{H} \leq 2 \times \text{Cost}^\star$$
A simplistic heuristic

The simplistic heuristic $H$

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

Approximation guarantee

$$\text{Cost}^H \leq 2 \times \text{Cost}^*$$

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

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

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

$$\text{Cost}^H \leq 2 \times L \leq 2 \times \text{Cost}^*$$
Experiments on a set of various matrices: the ratio of number of accesses over the lower bound is measured:

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

⇒ 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 \( B \)

1: \textbf{for} level = 1 \textbf{to} \( \ell \) \textbf{do} \\
2: \quad \text{Find the best matching } \mathcal{M} \text{ among the requested nodes} \\
3: \quad \text{for each pair in } \mathcal{M} \text{ remove one, mark the remaining one as the } \\
\quad \text{representative for the other node(s)} \\
4: \quad \textbf{end for} \\
5: \quad \text{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).
We use PaToH [Çatalyürek and Aykanat, ’99] for the tests. Here we measure the ratio hypergraph / post-order:

<table>
<thead>
<tr>
<th>Matrix</th>
<th>10% diagonal</th>
<th>10% off-diag</th>
</tr>
</thead>
<tbody>
<tr>
<td>CESR(46799)</td>
<td>1.01</td>
<td>0.75</td>
</tr>
<tr>
<td>af2356</td>
<td>1.03</td>
<td>0.69</td>
</tr>
<tr>
<td>boyd1</td>
<td>1.03</td>
<td>0.54</td>
</tr>
<tr>
<td>ecl32</td>
<td>1.05</td>
<td>0.56</td>
</tr>
<tr>
<td>gre1107</td>
<td>0.86</td>
<td>0.80</td>
</tr>
<tr>
<td>saylr4</td>
<td>0.98</td>
<td>0.80</td>
</tr>
<tr>
<td>sherman3</td>
<td>0.97</td>
<td>0.65</td>
</tr>
<tr>
<td>grund/bayer07</td>
<td>0.97</td>
<td>0.72</td>
</tr>
<tr>
<td>mathworks/pd</td>
<td>0.94</td>
<td>0.60</td>
</tr>
<tr>
<td>stokes64</td>
<td>0.99</td>
<td>0.80</td>
</tr>
</tbody>
</table>

- 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?