Parallel direct solvers for finite element problems

J. M. Vargas-Felix, S. Botello-Rionda

miguelvargas@cimat.mx, botello@cimat.mx

Abstract: In this paper we will describe the software implementation of parallel direct solvers for sparse matrices and their use in the solution of linear finite element problems, particularly an example of structural mechanics.

We will describe four strategies to reduce the time and the memory usage when solving linear systems using Cholesky and LU factorization.

1. The usage of compressed schema to store sparse matrices.
2. Reordering of rows and columns of the system of equations matrix to reduce factorization fill-in.
3. Symbolic Cholesky factorization to produce an exact factorization with only non-zero entries.
4. Parallelize the factorization algorithms.

1 Linear finite element problems

A problem defined in a domain Ω modeled with a linear differential operator could be seen as a systems of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

with certain conditions (Dirichlet o Neumann) on the domain boundary $\partial\Omega$ (figure 1),

$$\mathbf{x} = \mathbf{x}_c \text{ in } \Gamma_x,\ \mathbf{b} = \mathbf{b}_c \text{ in } \Gamma_b.$$

In structural mechanics, the matrix $\mathbf{A}$ is called stiffness matrix.

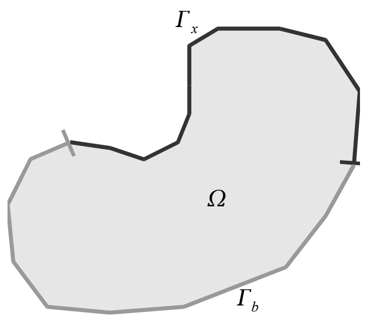
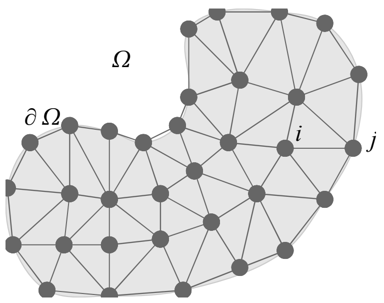


Figure 1: Problem domain.

We discretized the domain dividing it with geometric elements that approximately cover the domain (figure 2), generating then a mesh of nodes and edges. The relation among two nodes i y j corresponds to a value in the entry a_{ij} of the matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, where n is the number of nodes. Due to there is a relation from node i to node j, exists also a relation (not necessarily with the same value) from node j to node i, this will produce a matrix with symmetric structure, and not necessarily symmetric because of its values. Entries in the diagonal represents the nodes. These will be all entries of the matrix different to zero. It is easily seen that this kind of problems produce sparse matrices.



$$\mathbf{A} = \begin{pmatrix} \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\ \circ & a_{ii} & \circ & a_{ij} & \circ & \circ & \cdots \\ \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\ \circ & a_{ji} & \circ & a_{jj} & \circ & \circ & \cdots \\ \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\ \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Figure 2: Matrix representation of a discretized domain.

For problems with m degrees of freedom per node, we will have a matrix $\mathbf{A} \in \mathbb{R}^{mn \times mn}$. The entries of the matrix different to zero will be $(im-k, jm-l)$, $(jm-k, im-l)$, $(im-k, im-k)$ and $(jm-k, jm-k)$, with $k, l = 0 \ldots m$.

We will introduce the notation $\eta(\mathbf{A})$, this indicates the number of non-zero entries of a matrix $\mathbf{A}$.

The figure 3 was taken from a finite element problem, it shows with black dots the non-zero entries a matrix $\mathbf{A} \in \mathbb{R}^{556 \times 556}$ that contains 309,136 entries, with $\eta(\mathbf{A}) = 1810$, thus only 0.58% of the entries are non-zero.

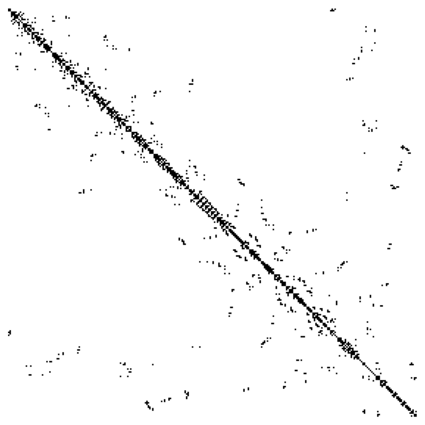


Figure 3: An example of the sparsity of a matrix from a finite element problem.

To save memory and processing time we will only store the entries of $\mathbf{A}$ that are non-zero.

An efficient method to store and operate this kind of matrices is the Compressed Row Storage (CRS) [Saad03 p362]. This method is suitable when we want to access the entries of each row of $\mathbf{A}$ sequentially.

$$\begin{pmatrix} 8 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 & 0 & 0 \\ 2 & 0 & 1 & 0 & 7 & 0 \\ 0 & 9 & 3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{pmatrix}$$

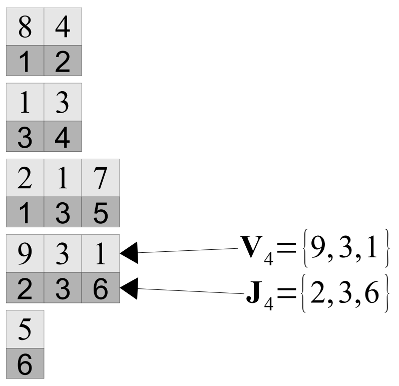


Figure 4: Storage of a sparse matrix using CRS.

For each row i of $\mathbf{A}$ we will have two vectors, a vector $\mathbf{V}_i(\mathbf{A})$ that will contain the non-zero values of the row, and a vector $\mathbf{J}_i(\mathbf{A})$ with their respective column indexes (figure 4 shows an example). The size of the row will be denoted by $|\mathbf{V}_i(\mathbf{A})|$ or by $|\mathbf{J}_i(\mathbf{A})|$. Therefore the qth non zero value of the row i of $\mathbf{A}$ will be denoted by $\mathbf{V}_i^q(\mathbf{A})$ and the index of this value as $\mathbf{J}_i^q(\mathbf{A})$, with $q = 1 \ldots |\mathbf{J}_i(\mathbf{A})|$.

If we dont order the entries of each row, then to search an entry with certain column index will have a cost of $O(n)$ in the worst case. To improve it we will keep $\mathbf{V}_i(\mathbf{A})$ and $\mathbf{J}_i(\mathbf{A})$ ordered by the column indexes ($\mathbf{J}_i(\mathbf{A})$). Then we could perform a binary algorithm to have an search cost of $O(\log_2 n)$.

The main advantage of using Compressed Row Storage is when data in each row is stored continuously and accessed in a sequential way, this is important because we will have and efficient processor cache usage [Drep07]. The next table shows how important this is, it shows the number of clock cycles needed to access each kind of memory by a Pentium M processor.

Access to	**Cycles**
CPU registers	<= 1
L1 cache	~ 3
L2 cache	~ 14
RAM	~ 240

2 Cholesky factorization

A system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{1}$$

with a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ symmetric positive definite can be solved applying to this matrix the Cholesky factorization

$$\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathrm{T}}, \tag{2}$$

where $\mathbf{L}$ is a triangular inferior matrix. This factorization exists and is unique [Quar00 p80].

The formulae to calculate the values of $\mathbf{L}$ are

$$L_{ij} = \frac{1}{L_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} \right), \text{ for } i > j, \tag{3}$$

$$L_{jj} = \sqrt{A_{jj} - \sum_{k=1}^{j-1} L_{jk}^2}. \tag{4}$$

Substituting (2) in (1), we have

$$\mathbf{L}\mathbf{L}^{\mathrm{T}}\mathbf{x} = \mathbf{y},$$

by doing $\mathbf{z} = \mathbf{L}^{\mathrm{T}}\mathbf{x}$ we will have two triangular systems of equations

$$\mathbf{L}\mathbf{z} = \mathbf{y}, \tag{5}$$

$$\mathbf{L}^{\mathrm{T}}\mathbf{x} = \mathbf{z}, \tag{6}$$

easily solvable.

3 Reordering rows and columns

By reordering the rows and columns of $\mathbf{A}$ we could reduce the fill-in (the number of non-zero entries) of $\mathbf{L}$.

Figure 5 shows an example of the factorization of a sparse matrix in terms of non-zero entries. To the left, we have a stiffness matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, with $\eta(\mathbf{A}) = 1810$, to the right a lower triangular matrix $\mathbf{L}$, with $\eta(\mathbf{L}) = 8729$, resulting from the Cholesky factorization of $\mathbf{A}$.

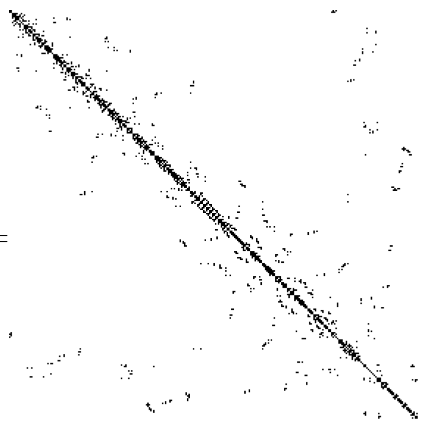
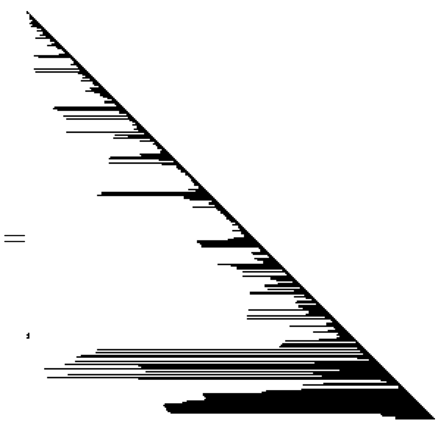
$\mathbf{A} =$  $\mathbf{L} =$ 

Figure 5: Representation of non-zero elements of a symmetric sparse matrix and its corresponding Cholesky factorization.

Now in figure 6, applying reordering to $\mathbf{A}$, we will have the stiffness matrix $\mathbf{A}'$ with $\eta(\mathbf{A}') = 1810$ (obviously the same number of non-zero entries than $\mathbf{A}$) and its corresponding factorization $\mathbf{L}'$ will have $\eta(\mathbf{L}') = 3215$. Both factorizations allow us to solve the same system of equations.

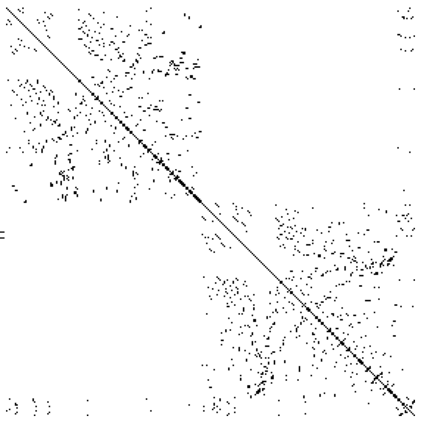
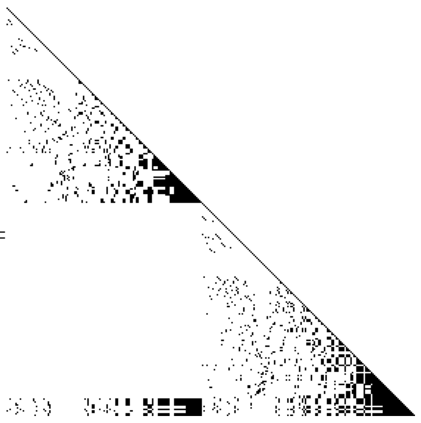
$\mathbf{A}' =$  $\mathbf{L}' =$ 

Figure 6: Representation of non-zero elements of a reordered symmetric matrix and its corresponding Cholesky factorization.

We reduced the fill-in of the factorization by

$$\frac{\eta(\mathbf{L}')}{\eta(\mathbf{L})} = \frac{3215}{8729} = 0.368.$$

A well known algorithm to produce good reordering is the minimum degree method, we will introduce it later. To determinate the reordering in fiigure 6 we used the METIS library [Kary99].

4 Permutation matrices

Give $\mathbf{P}$ a permutation matrix, the permutation (reordering) of columns

$$\mathbf{A}' \leftarrow \mathbf{PA},$$

or rows

$$\mathbf{A}' \leftarrow \mathbf{AP},$$

alone will destroy the symmetry of $\mathbf{A}$ [Golu96 p148]. To keep the symmetry of $\mathbf{A}$ we could only consider reordering of the entries of the form

$$\mathbf{A}' \leftarrow \mathbf{PAP}^{\mathrm{T}}.$$

We have to notice that this kind of symmetric permutations do not move elements outside the diagonal to the diagonal. The diagonal of $\mathbf{PAP}^{\mathrm{T}}$ is a reordering of the diagonal of $\mathbf{A}$.

$\mathbf{PAP}^{\mathrm{T}}$ is also symmetric positive definite for any permutation matrix $\mathbf{P}$, we could solve the reordered system

$$\left(\mathbf{PAP}^{\mathrm{T}}\right)(\mathbf{Px}) = (\mathbf{Py})\,.$$

The selected $\mathbf{P}$ will have an determinant effect in the number of non-zero entries of $\mathbf{L}$. To obtain the best reordering of the matrix $\mathbf{A}$ that minimizes the non-zero entries of $\mathbf{L}$ is an NP-complete problem [Yann81], but there are heuristics that could generate an acceptable ordering in a reduced amount of time.

5 Representation of sparse matrices as undirected graphs

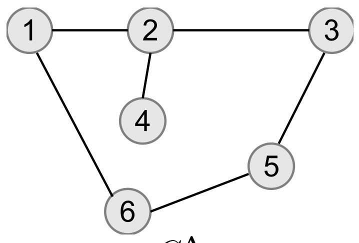
We saw previously that a finite element mesh could be represented as a sparse matrix with symmetric structure. This mesh could be seen as an undirected graph. In other words, we could represent a sparse matrix with symmetric structure as undirected graph.

An undirected graph $G = (X, E)$ consists of a finite set of nodes X and a set E of edges. Edges are non ordered pairs of nodes.

An ordering (or tag) α of G is simply a map of the set $\{1, 2, \ldots, N\}$ to X, where N denotes the number of nodes of G. The graph ordered by α will be denoted by $G^\alpha = (X^\alpha, E^\alpha)$.

Let $\mathbf{A}$ an $n \times n$ matrix with symmetric structure, the ordered graph of $\mathbf{A}$, denoted by $G^{\mathbf{A}} = \left(X^{\mathbf{A}}, E^{\mathbf{A}}\right)$ in which the N nodes of $G^{\mathbf{A}}$ are numbered from 1 to N, and $(x_i, x_j) \in E^{\mathbf{A}}$ if and only if $a_{ij} \neq 0$ and $a_{ji} \neq 0$, for $i \neq j$. Here x_i is the node of $X^{\mathbf{A}}$ with tag i. Figure 7 shows an example.

$$\underset{\mathbf{A}}{\begin{pmatrix} a_{11} & a_{12} & & & & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & & \\ & a_{32} & a_{33} & & a_{35} & \\ & a_{42} & & a_{44} & & \\ & & a_{53} & & a_{55} & a_{56} \\ a_{61} & & & & a_{65} & a_{66} \end{pmatrix}}$$



$G^{\mathbf{A}}$

Figure 7: Matrix (only non-zero entries are shown), and its ordered graph.

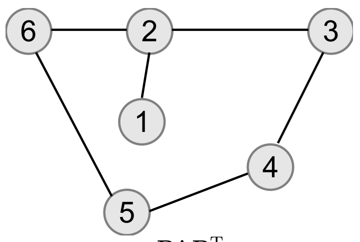
For any permutation matrix $\mathbf{P} \neq \mathbf{I}$, the non ordered (non tagged) graphs of $\mathbf{A}$ and $\mathbf{PAP}^{\mathrm{T}}$ are the same, but their associated tag is different.

Then, a non tagged graph of $\mathbf{A}$ represents the structure of $\mathbf{A}$ without suggesting a particular ordering. This represents the equivalence of the matrices of class $\mathbf{PAP}^{\mathrm{T}}$.

To find a good permutation of $\mathbf{A}$ is the same as to find a good ordering of its graph [Geor81]. Figure 8 shows the same graph but with other ordering (tag).

$$\begin{pmatrix} b_{11} & b_{12} & & & & \\ b_{21} & b_{22} & b_{23} & & & b_{26} \\ & b_{32} & b_{33} & b_{34} & & \\ & & b_{43} & b_{44} & b_{45} & \\ & & & b_{54} & b_{55} & b_{56} \\ & b_{62} & & & b_{65} & b_{66} \end{pmatrix}$$

$$\mathbf{B} = \mathbf{PAP}^{\mathrm{T}}$$



$G^{\mathbf{PAP}^{\mathrm{T}}}$

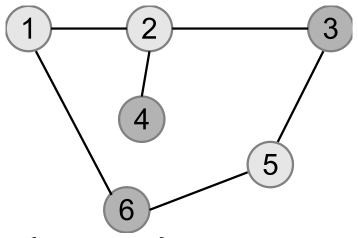
Figure 8: A different ordering (with a permutation matrix) of previous figure.

Two nodes $x, y \in X$ of a graph $G = (X, E)$ are adjacent if $(x, y) \in E$. For $Y \subset X$, the adjacent set of Y, denoted as $\mathrm{adj}(Y)$, is

$$\mathrm{adj}(Y) = \{x \in X - Y \mid (x, y) \in E \text{ for some } y \in Y\}.$$

In other words, $\mathrm{adj}(Y)$ is simply the set of nodes of G that does no belong to Y but are adjacent to at leas a node of Y. An example is shown in figure 9.

$$\left(\begin{array}{cc|cccc} a_{11} & a_{12} & & & & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & & \\ \hline & a_{32} & a_{33} & & a_{35} & \\ & a_{42} & & a_{44} & & \\ & & a_{53} & & a_{55} & a_{56} \\ a_{61} & & & & a_{65} & a_{66} \end{array}\right)$$



$Y = \{x_1, x_2\}$; $\mathrm{adj}(Y) = \{x_3, x_4, x_6\}$

Figure 9: Example of adjacency of a set $Y \subset X$.

For a set $Y \subset X$, the degree of Y, denoted by $\mathrm{dg}(Y)$, is the number $|\mathrm{adj}(Y)|$. For a single element set, we consider $\mathrm{dg}(\{x_2\}) \equiv \mathrm{dg}(x_2)$.

6 Reordering algorithms

The most common heuristic for graph reordering is the minimum degree algorithm. Algorithm 1 shows a basic version of this heuristic [Geor81 p116].

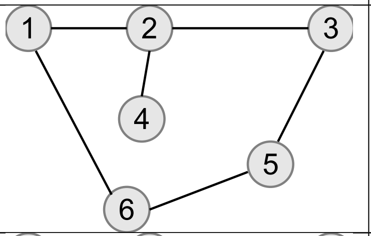
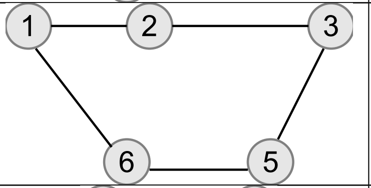
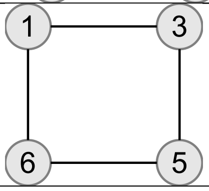
Algorithm 1 Minimum degree.

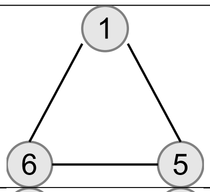
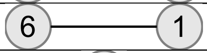
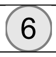
```
Require: A a symetric structure matrix and $G^{\mathbf{A}}$ its corresponding graph
  $G_0(X_0, E_0) \leftarrow G^{\mathbf{A}}$
  $i \leftarrow 1$
  repeat
    In $G_{i-1}(X_{i-1}, E_{i-1})$, choose the node $x_i$ with minimum degree
    Create the elimination graph $G_i(X_i, E_i)$ as follow:
    Remove node $x_i$ from $G_{i-1}(X_{i-1}, E_{i-1})$ and the edges with this node.
    Add edges to the graph to have adj($x_i$) as adjacent pairs in $G_i(X_i, E_i)$.

    $i \leftarrow i + 1$
  until $i < |X|$
```

When we have the same minimum degree in several nodes, usually we choose arbitrarily.

An example of this algoritm is shown next.

i	Elimination graph G_{i-1}	x_i	dg(x_i)
1		4	1
2		2	2
3		3	2

4		5	3
5		1	1
6		6	0

This elimination sequence is: 4, 2, 3, 5, 1, 6. This new order for the matrix **A**, corresponds to a permutation matrix

$$\mathbf{P} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

More advanced versions of this algorithm could be found in [Geor89].

7 Symbolic Cholesky factorization

By reordering **A** we have now a more sparse factor matrix **L**, now, we have to identify the non-zero entries of **L** in order to perform efficiently (3) and (4). The algorithm to do this is known as symbolic Cholesky factorization [Gall90 p86-88]. This method is shown in algorithm 2.

For a sparse matrix **A**, we define

$$\mathbf{a}_j := \{k > j \mid A_{kj} \neq 0\},\ j = 1, \ldots, n,$$

as the sets of indexes of the non-zero entries of the column of the strict lower triangular part of **A**.

In the same way, we define for **L**, the sets

$$\mathbf{l}_j := \{k > j \mid L_{kj} \neq 0\},\ j = 1, \ldots, n.$$

We will need also the sets $\mathbf{r}_j$ that will be used to store the columns of **L** which structure will affect the column j of **L**.

Algorithm 2 Symbolic Cholesky factorization.

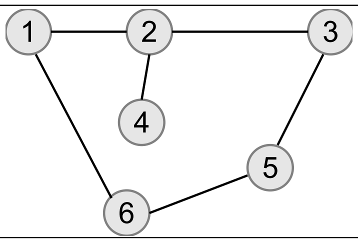
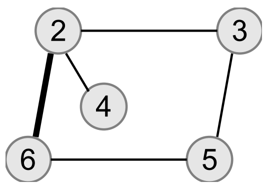
```
for $j \leftarrow 1, \dots, n$ do
    $\mathbf{r}_j \leftarrow \emptyset$
    $\mathbf{l}_j \leftarrow \mathbf{a}_j$
    for all $i \in \mathbf{r}_j$ do
        $\mathbf{l}_j \leftarrow \mathbf{l}_j \cup \mathbf{l}_i \setminus \{j\}$
    $p \leftarrow \begin{cases} \min\{i \in \mathbf{l}_j\} & \text{if } \mathbf{l}_j \neq \emptyset \\ j & \text{other case} \end{cases}$
    $\mathbf{r}_p \leftarrow \mathbf{r}_p \cup \{j\}$
```

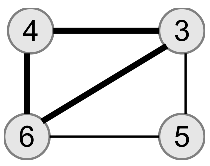
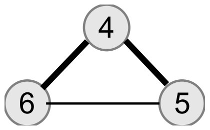
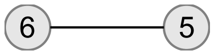
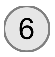
This algorithm is very efficient, complexity of time and storage space has an order of $O(\eta(\mathbf{L}))$.

Now we will show visually how this symbolic factorization works. It could be seen as a sequence of elimination graphs [Geor81 pp92-100]. Given $\mathbf{H}_0 = \mathbf{A}$, we can define a transformation from $\mathbf{H}_0$ to $\mathbf{H}_1$ as the changes to the corresponding graphs. We denote $\mathbf{H}_0$ by $G_0^{\mathbf{H}}$ and $\mathbf{H}_1$ by $G_1^{\mathbf{H}}$. For an ordering α implied by $G^{\mathbf{A}}$, let denote a node $\alpha(i)$ by x_i. In the next figure, the graph $\mathbf{H}_{i+1}$ is obtained from $\mathbf{H}_i$ by:

1. Removing the node x_j and its edges.

2. Add edges adj(x_j) in $G^{\mathbf{H}_i}$ as adjacent pairs in $G^{\mathbf{H}_{i+1}}$. New edges are indicated with a bold line, and new entries in the matrix with an **X**.

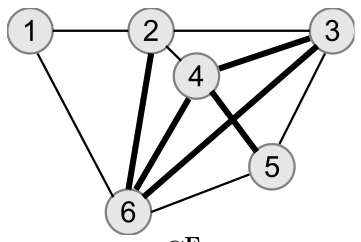
$G^{\mathbf{H}_0}$ 	$\mathbf{H}_0 = \begin{pmatrix} x & x & & & & x \\ x & x & x & x & & \\ & x & x & & x & \\ & x & & x & & \\ & & x & & x & x \\ x & & & & x & x \end{pmatrix}$
$G^{\mathbf{H}_1}$ 	$\mathbf{H}_1 = \begin{pmatrix} x & x & & & & x \\ x & x & x & x & & \mathbf{X} \\ & x & x & & x & \\ & x & & x & & \\ & & x & & x & x \\ x & \mathbf{X} & & & x & x \end{pmatrix}$

$G^{\mathbf{H}_2}$ 	$\mathbf{H}_2 = \begin{pmatrix} x & x & & & & x \\ x & x & x & x & & X \\ & x & x & \mathbf{X} & x & \mathbf{X} \\ & x & \mathbf{X} & x & & \mathbf{X} \\ & & x & & x & x \\ x & X & \mathbf{X} & \mathbf{X} & x & x \end{pmatrix}$
$G^{\mathbf{H}_3}$ 	$\mathbf{H}_3 = \begin{pmatrix} x & x & & & & x \\ x & x & x & x & & X \\ & x & x & X & x & X \\ & x & X & x & \mathbf{X} & X \\ & & x & \mathbf{X} & x & x \\ x & X & X & X & x & x \end{pmatrix}$
$G^{\mathbf{H}_4}$ 	$\mathbf{H}_4 = \begin{pmatrix} x & x & & & & x \\ x & x & x & x & & X \\ & x & x & X & x & X \\ & x & X & x & X & X \\ & & x & X & x & x \\ x & X & X & X & x & x \end{pmatrix}$
$G^{\mathbf{H}_5}$ 	$\mathbf{H}_5 = \begin{pmatrix} x & x & & & & x \\ x & x & x & x & & X \\ & x & x & X & x & X \\ & x & X & x & X & X \\ & & x & X & x & x \\ x & X & X & X & x & x \end{pmatrix}$

Let $\mathbf{L}$ be the lower triangular factor of the matrix $\mathbf{A}$. Now we defined the filled graph $G^{\mathbf{A}}$ as the undirected graph $G^{\mathbf{F}} = \left(X^{\mathbf{F}}, E^{\mathbf{F}}\right)$, where $\mathbf{F} = \mathbf{L} + \mathbf{L}^{\mathrm{T}}$. Where the set of edges $E^{\mathbf{F}}$ is conformed with all the edges in $E^{\mathbf{A}}$ plus all edges added during the factorization. Obviously $X^{\mathbf{F}} = X^{\mathbf{A}}$. Figure 10 shows the result of the elimination sequece.

8 Cholesky factorization in parallel

The most time consuming part of Cholesky factorization for sparse matrices is the calculus of the entries of L_{ij}. By using the symbolic factorization we can now identify the non-zero entries of $\mathbf{L}$ before factorization, replacing (3)



$$G^{\mathbf{F}} \qquad \begin{pmatrix} x & x & & & & x \\ x & x & x & x & & X \\ & x & x & X & x & X \\ & x & X & x & X & X \\ & & x & X & x & x \\ x & X & X & X & x & x \end{pmatrix}$$

$$\mathbf{F} = \mathbf{L} + \mathbf{L}^{\mathrm{T}}$$

Figure 10: Result of the elimination sequence.

and (4) by

$$L_{ij} = \frac{1}{L_{jj}} \left(A_{ij} - \sum_{k \in J_i(\mathbf{L}) \cap J_j(\mathbf{L}),\ k<j} L_{ik} L_{jk} \right), \text{ for } i > j,$$

$$L_{jj} = \sqrt{A_{jj} - \sum_{k \in J_j(\mathbf{L}),\ k<j} L_{jk}^2}.$$

We can see that the calculus of an entry L_{ij} could be performed at the same time than the calculus of L_{kj} with $i \neq k$, if all the columns $1, \ldots, (j-1)$ are already calculated [Heat91]. Thus we can parallelize the Cholesky factorization if we calculate L_{ij} entries advancing by columns. We fill the first column of $\mathbf{L}$ in parallel, then we proceed to the second column, etc. (figure 11).

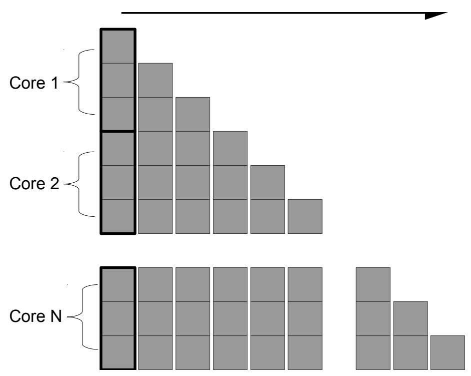


Figure 11: Cholesky factorization can be done filling column by column.

When performing forward and back substitution in (5) and (6), we have to access the entries of $\mathbf{L}$ and $\mathbf{L}^{\mathrm{T}}$ by row, therefore is convenient to have both **L** and $\mathbf{L}^{\mathrm{T}}$ stored using CRS. This will double the memory usage, but will perform fast, especially considering efficient processor cache usage is improved when accessed memory is allocated continuously. This is shown in algoritm 3.

Algorithm 3 Parallel Cholesky factorization using sparse matrices.

```
for $j \leftarrow 1, \dots, n$ do
    $L_{jj} \leftarrow A_{jj}$
    for $q \leftarrow 1, \dots, |\mathbf{J}_j(\mathbf{L})|$ do
        $L_{jj} \leftarrow L_{jj} - \mathbf{V}_j^q(\mathbf{L})\mathbf{V}_j^q(\mathbf{L})$
    $L_{jj} \leftarrow \sqrt{L_{jj}}$
    $L_{jj}^{\mathrm{T}} \leftarrow L_{jj}$
    for in parallel $q \leftarrow 1, \dots, |\mathbf{J}_j(\mathbf{L}^{\mathrm{T}})|$ do
        $i \leftarrow \mathbf{J}_j^q(\mathbf{L}^{\mathrm{T}})$
        $L_{ij} \leftarrow A_{ij}$
        $r \leftarrow 1$; $\rho \leftarrow \mathbf{J}_i^r(\mathbf{L})$
        $s \leftarrow 1$; $\sigma \leftarrow \mathbf{J}_j^s(\mathbf{L})$
        loop
            while $\rho < \sigma$ do
                $r \leftarrow r + 1$; $\rho \leftarrow \mathbf{J}_i^r(\mathbf{L})$
            while $\rho > \sigma$ do
                $s \leftarrow s + 1$; $\sigma \leftarrow \mathbf{J}_j^s(\mathbf{L})$
            while $\rho = \sigma$ do
                if $\rho = j$ then
                    exit loop
                $L_{ij} \leftarrow L_{ij} - \mathbf{V}_i^r(\mathbf{L})\mathbf{V}_j^s(\mathbf{L})$
                $r \leftarrow r + 1$; $\rho \leftarrow \mathbf{J}_i^r(\mathbf{L})$
                $s \leftarrow s + 1$; $\sigma \leftarrow \mathbf{J}_j^s(\mathbf{L})$
        $L_{ij} \leftarrow \frac{L_{ij}}{L_{jj}}$
        $L_{ji}^{\mathrm{T}} \leftarrow L_{ij}$
```

9 LU factorization

Symbolic Cholesky factorization could be use to determine the structure of the **LU** factorization if the matrix has symmetric structure, like the ones resulting of the finite element method. The minimum degree algorithm gives also a good ordering for factorization. In this case **L** and $\mathbf{U}^{\mathrm{T}}$ will have the same structure.

Formulae to calculate the **L** and **U** (using Doolittles algorithm) are

$$U_{ij} = A_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj}, \text{ for } i > j, \tag{7}$$

$$L_{ji} = \frac{1}{U_{ii}} \left(A_{ji} - \sum_{k=1}^{i-1} L_{jk} U_{ki} \right), \text{ for } i > j, \tag{8}$$

$$U_{ii} = A_{ii} - \sum_{k=1}^{i-1} L_{jk} U_{ki}. \tag{9}$$

By storing these matrices using sparse compressed row, we can rewrite them as

$$U_{ij} = A_{ij} - \sum_{k \in J_i(\mathbf{L}) \cap J_j(\mathbf{L}),\, k<j} L_{ik} U_{kj}, \text{ for } i > j,$$

$$L_{ji} = \frac{1}{U_{ii}} \left(A_{ji} - \sum_{k \in J_i(\mathbf{L}) \cap J_j(\mathbf{L}),\, k<j} L_{jk} U_{ki} \right), \text{ for } i > j,$$

$$U_{ii} = A_{ii} - \sum_{k \in J_j(\mathbf{L}),\, k<j} L_{jk} U_{ki}.$$

Similarity to the Cholesky algorithm, to improve performance we will store **L**, **U** and $\mathbf{U}^{\mathrm{T}}$ matrices using CRS. It is shown in the algorithm 4.

10 Numerical examples

The program was implemented using C++ with OpenMP as a parallelization schema. It was compiled using GCC-4.5.1 and tested in a computer with 8 Intel Xeon E5620 cores running at 2.40GHz and with 32GB of memory.

10.1 Heat equation 2D

The problem is to determine the temperature inside a circular plate, only a quarter of the circle is analyzed (figure 12). Imposed boundary conditions on

Algorithm 4 Parallel LU factorization using sparse matrices.

```
for $j \leftarrow 1, \dots, n$ do
    $U_{jj} \leftarrow A_{jj}$
    for $q \leftarrow 1, \dots, |\mathbf{J}_j(\mathbf{L})|$ do
        $U_{jj} \leftarrow U_{jj} - \mathbf{V}_j^q(\mathbf{L})\mathbf{V}_j^q(\mathbf{U}^{\mathrm{T}})$
    $L_{jj} \leftarrow 1$
    $U_{jj}^{\mathrm{T}} \leftarrow U_{jj}$
    for in parallel $q \leftarrow 1, \dots, |\mathbf{U}_j(\mathbf{L}^{\mathrm{T}})|$ do
        $i \leftarrow \mathbf{U}_j^q(\mathbf{U})$
        $L_{ij} \leftarrow A_{ij}$
        $U_{ji} \leftarrow A_{ji}$
        $r \leftarrow 1$; $\rho \leftarrow \mathbf{J}_i^r(\mathbf{L})$
        $s \leftarrow 1$; $\sigma \leftarrow \mathbf{J}_j^s(\mathbf{L})$
        loop
            while $\rho < \sigma$ do
                $r \leftarrow r + 1$; $\rho \leftarrow \mathbf{J}_i^r(\mathbf{L})$
            while $\rho > \sigma$ do
                $s \leftarrow s + 1$; $\sigma \leftarrow \mathbf{J}_j^s(\mathbf{L})$
            while $\rho = \sigma$ do
                if $\rho = j$ then
                    exit loop
                $L_{ij} \leftarrow L_{ij} - \mathbf{V}_i^r(\mathbf{L})\mathbf{V}_j^s(\mathbf{U}^{\mathrm{T}})$
                $U_{ji} \leftarrow U_{ji} - \mathbf{V}_j^s(\mathbf{L})\mathbf{V}_i^r(\mathbf{U}^{\mathrm{T}})$
                $r \leftarrow r + 1$; $\rho \leftarrow \mathbf{J}_i^r(\mathbf{L})$
                $s \leftarrow s + 1$; $\sigma \leftarrow \mathbf{J}_j^s(\mathbf{L})$
        $L_{ij} \leftarrow \frac{L_{ij}}{U_{jj}}$
        $U_{ji} \leftarrow U_{ji}$
        $U_{ij}^{\mathrm{T}} \leftarrow U_{ji}$
```

the exterior of the circle is a temperature of zero degrees. A source of heat is imposed in all the volume. Temperature is unknown inside of the circle. The number of equations is equal to the number of nodes in discretization. Element types are linear triangles.

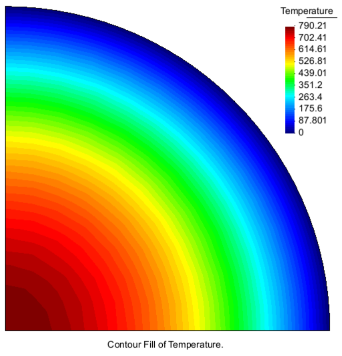
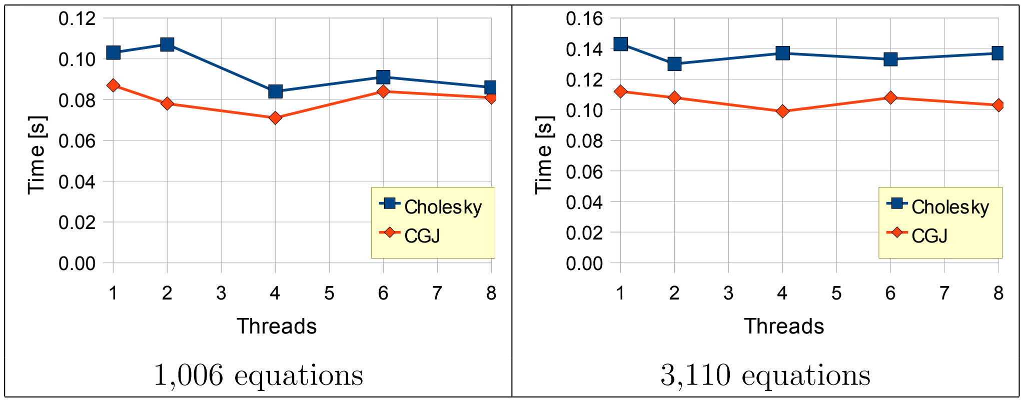
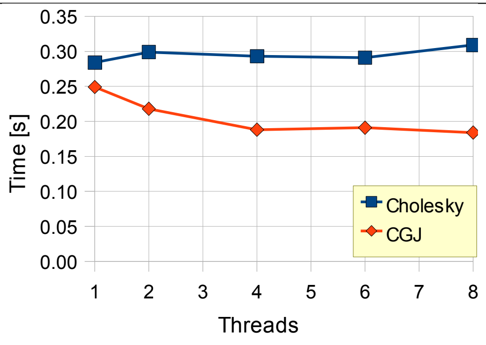


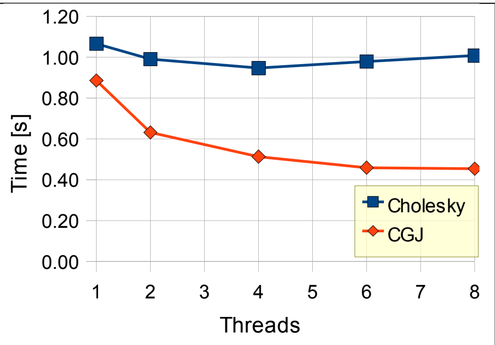
Figure 12: Numerical example of heat equation.

We tested Cholesky algorithm using a different number of matrix sizes and number of threads (1, 2, 4, 6 and 8 threads where used). Next images shows a comparison of the solution times. As a comparison the conjugate gradient method with Jacobi preconditioner (CGJ) was used in the same problems with a tolerance of 1×10^{-5}. It is notizable that paralelization with OpenMP performs poorly when matix size is small.

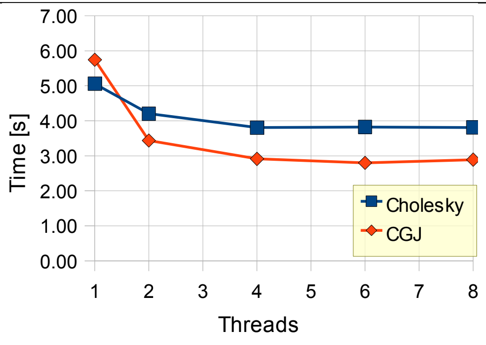




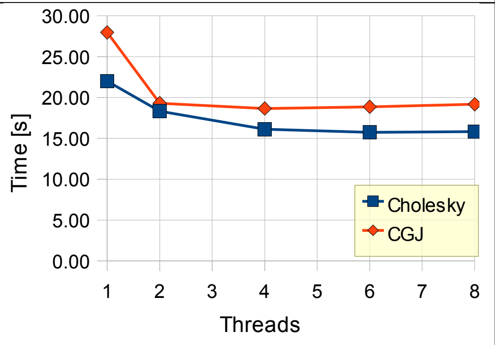
10,014 equations



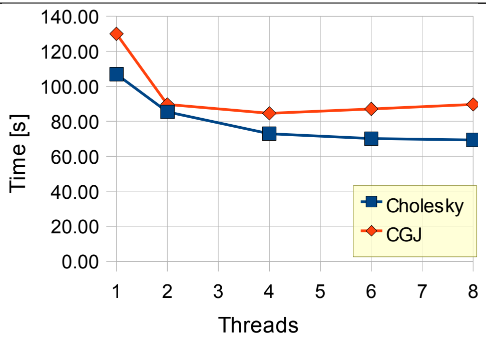
31,615 equations



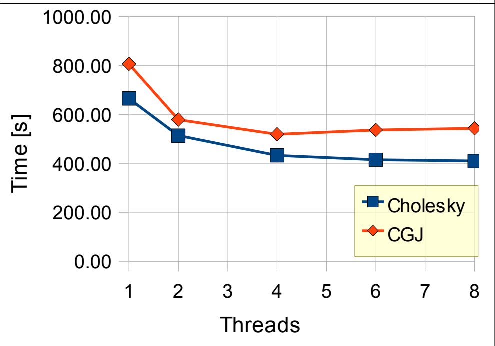
102,233 equations



312,248 equations



909,540 equations



3'105,275 equations

Next table shows solution times using both Cholesky and CGJ with 8 threads, the correspondig figure is 13.

Number of equations	$\eta(\mathbf{A})$	$\eta(\mathbf{L})$	Cholesky Time [s]	CGJ Time [s]
1,006	6,140	14,722	0.086	0.081
3,110	20,112	62,363	0.137	0.103
10,014	67,052	265,566	0.309	0.184
31,615	215,807	1’059,714	1.008	0.454
102,233	705,689	4’162,084	3.810	2.891
312,248	2’168,286	14’697,188	15.819	19.165
909,540	6’336,942	48’748,327	69.353	89.660
3’105,275	21’681,667	188’982,798	409.365	543.110
10’757,887	75’202,303	743’643,820	2,780.734	3,386.609

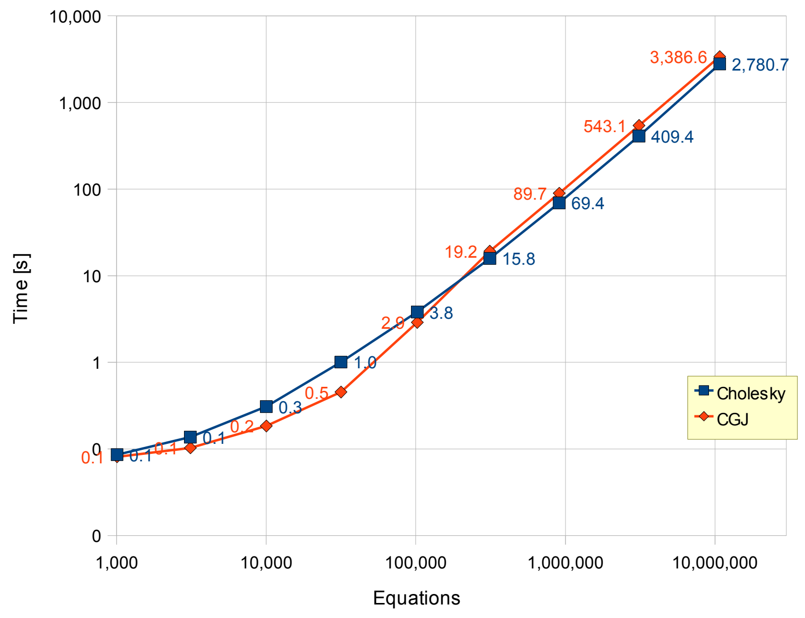


Figure 13: Time to complete solution, comparing Cholesky and CGJ.

Memory usage is shown in figure 14.

10.2 Solid deformation 3D

The problem shown here is the finite element implementation of the linear deformation of a building that has body forces due to self weight of building material.

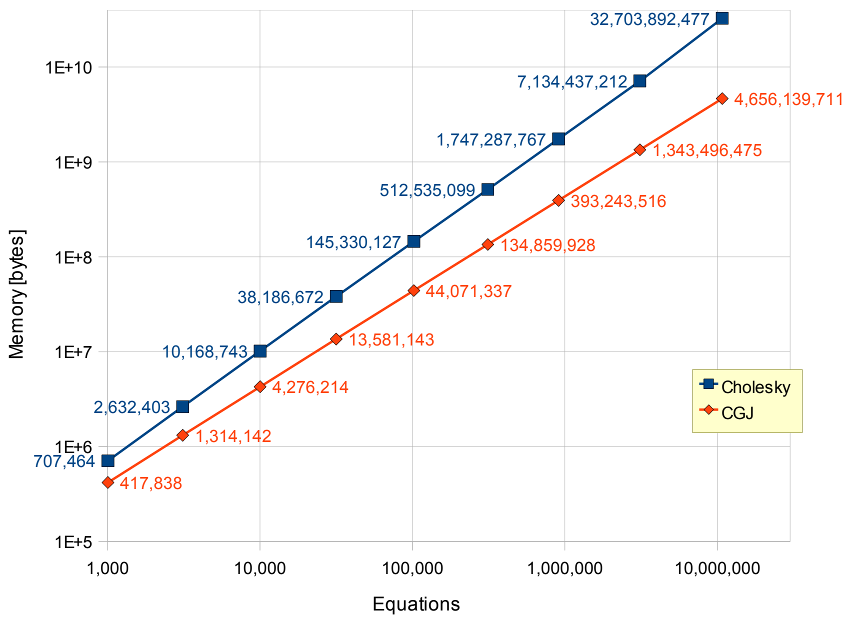


Figure 14: Memory usage, comparing Cholesky and CGJ.

Problem	Building
Dimension	3
Elements	264,250
Element type	Linear hexaedra
Nodes	326,228
Equations	978,684
$\eta(\mathbf{A})$	69,255,522
$\eta(\mathbf{L})$	787,567,656

The domain (figure 15) was divided in 264,250 elements and 326,228 nodes. The stiffness matrix size is 978,684. We solved the problem varying the number of processor used. As a comparison, we show results obtained using the parallel conjugate gradient method to solve the problem. The conjugate gradient method was used with and without preconditioning, the preconditioner used is Jacobi. Tolerance on the norm of the gradient used for iterative solvers is 1×10^{-5}. In the next charts, the values between parenthesis represents the number of processor used in parallel.

Comparison of the solution time using different solvers and number of processors in parallel is shown in figure 16.

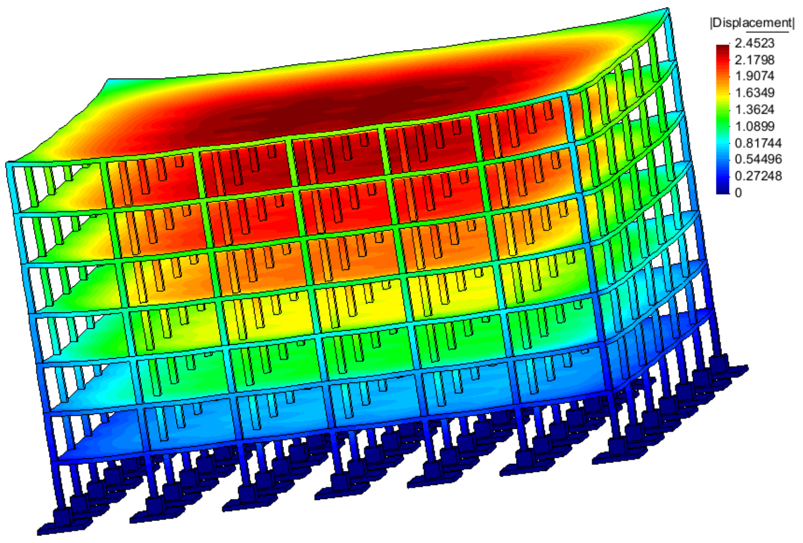


Figure 15: Numerical example of solid deformation.

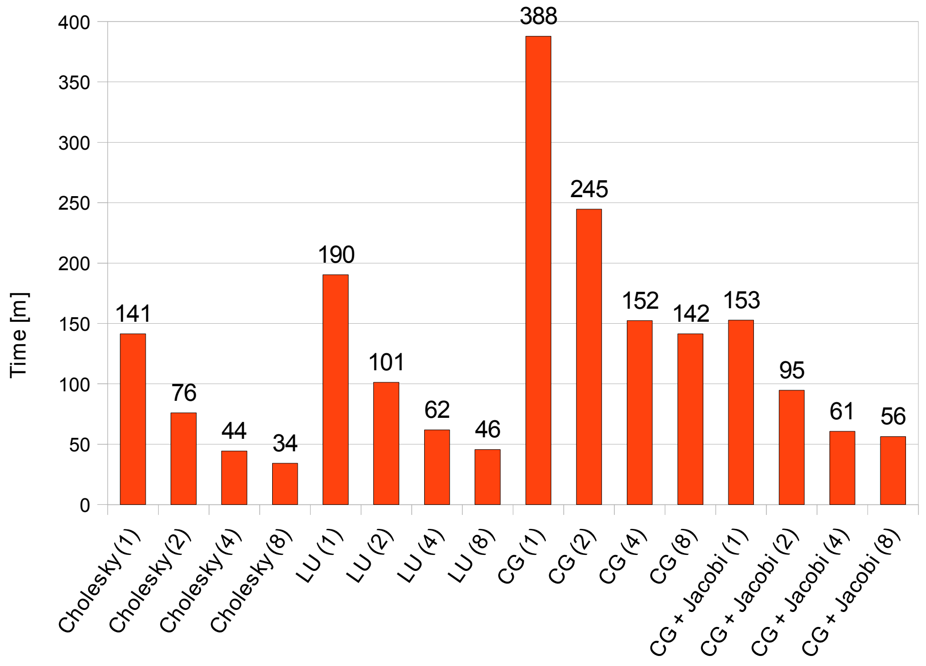


Figure 16: Solution time comparison of different parallel solvers.

Memory usage is shown in figure 17.

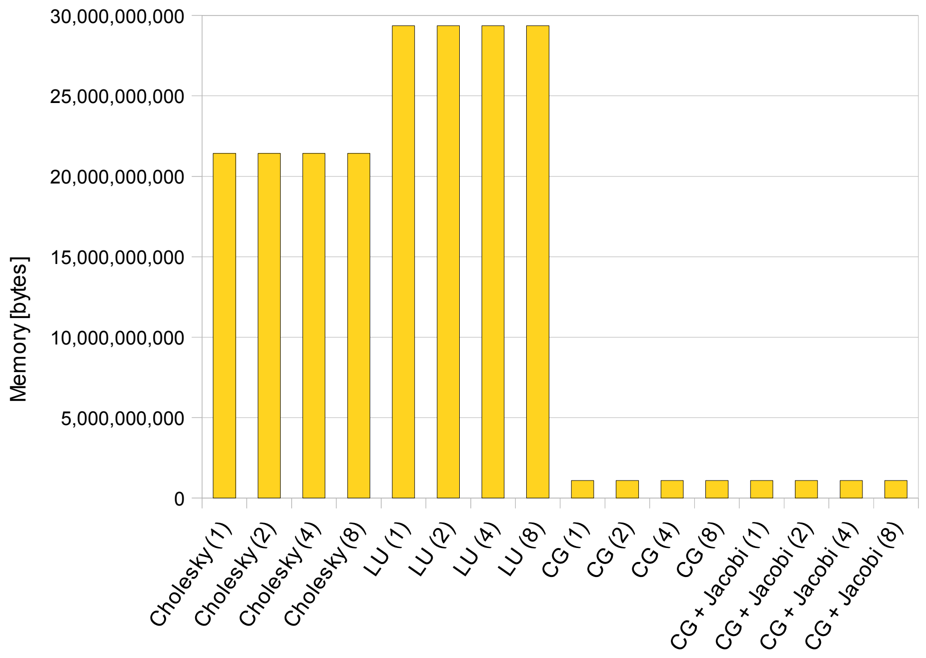


Figure 17: Maximum memory usage comparison of different parallel solvers.

11 Conclusion

We presented a parallel implementation of Cholesky and LU factorizations that are comparable in speed to iterative solvers. The drawback is of course the memory usage. We have shown numerical examples big enough to fit in a modern computer.

The real advantage of this kind of solvers is shown when direct solvers are applied in conjunction to domain decomposition techniques, like the Schwarz alternating method. In this case we only need to factorize the stiffness matrix once, and all Schwarz iterations consists only in forward and back substitutions that are performed very fast. Moreover by partitioning the domain we will have many small stiffness matrices that are factorized faster. We will present this results in a future paper.

12 References

Drep07 U. Drepper. What Every Programmer Should Know About Memory. Red Hat, Inc. 2007.

Gall90 K. A. Gallivan, M. T. Heath, E. Ng, J. M. Ortega, B. W. Peyton, R. J. Plemmons, C. H. Romine, A. H. Sameh, R. G. Voigt, Parallel Algorithms for Matrix Computations, SIAM, 1990.

Geor81 A. George, J. W. H. Liu. Computer solution of large sparse positive definite systems. Prentice-Hall, 1981.

Geor89 A. George, J. W. H. Liu. The evolution of the minimum degree ordering algorithm. SIAM Review Vol 31-1, pp 1-19, 1989.

Golu96 G. H. Golub, C. F. Van Loan. Matrix Computations. Third edidion. The Johns Hopkins University Press, 1996.

Heat91 M T. Heath, E. Ng, B. W. Peyton. Parallel Algorithms for Sparse Linear Systems. SIAM Review, Vol. 33, No. 3, pp. 420-460, 1991.

Kary99 G. Karypis, V. Kumar. A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM Journal on Scientific Computing, Vol. 20-1, pp. 359-392, 1999.

Lipt77 R. J. Lipton, D. J. Rose, R. E. Tarjan. Generalized Nested Dissection, Computer Science Department, Stanford University, 1997.

Quar00 A. Quarteroni, R. Sacco, F. Saleri. Numerical Mathematics. Springer, 2000.

Yann81 M. Yannakakis. Computing the minimum fill-in is NP-complete. SIAM Journal on Algebraic Discrete Methods, Volume 2, Issue 1, pp 77-79, March, 1981.