Universität des Saarlandes

Fachrichtung 6.1 – Mathematik

Preprint Nr. 154

Hierarchical Cholesky decomposition of sparse matrices arising from curl-curl-equation

Ilgis Ibragimow, Sergej Rjasanow and
Katharina Straube

Saarbrücken 2005

Fachrichtung 6.1 – Mathematik
Universität des Saarlandes

Preprint No. 154
submitted: July 12, 2005

Hierarchical Cholesky decomposition of sparse matrices arising from curl-curl-equation

Ilgis Ibragimow
Saarland University
Department of Mathematics
Postfach 15 11 50
D–66041 Saarbrücken
Germany
ilgis@num.uni-sb.de

Sergej Rjasanow
Saarland University
Department of Mathematics
Postfach 15 11 50
D–66041 Saarbrücken
Germany
rjasanow@num.uni-sb.de

Katharina Straube
Robert Bosch GmbH

Postfach 10 60 50
D–70049 Stuttgart
Germany
katharina.straube@de.bosch.com

Edited by
FR 6.1 – Mathematik
Universität des Saarlandes
Postfach 15 11 50
66041 Saarbrücken
Germany

Fax: + 49 681 302 4443
e-Mail: preprint@math.uni-sb.de
WWW: http://www.math.uni-sb.de/

Hierarchical Cholesky decomposition of sparse matrices arising from curl-curl-equation

I. Ibragimov, S. Rjasanow and K. Straube

July 12, 2005

Abstract

A new hierarchical renumbering technique for sparse matrices arising from the application of the Finite Element Method (FEM) to three-dimensional Maxwell's equations is presented. It allows the complete Cholesky decomposition of the matrix, which leads to a direct solver of $O(N^{4/3})$ memory requirement. In addition, an approximate factorisation yielding a preconditioner for the matrix can be constructed. For this, two algorithms using low-rank approximation are presented which have almost linear arithmetic complexity and memory requirement. The efficiency of the methods is demonstrated on several numerical examples.

AMS Subject Classification:
Keywords: sparse, reordering, hierarchical matrix, clustering, approximate Cholesky decomposition

1 Introduction and motivation

Three-dimensional problems in electromagnetic field calculation can be solved with a coupling of the boundary element method (BEM) and the FEM. Fine discretisation of complex problems generates large systems of equations. The BEM part can be solved with asymptotically optimal complexity by using block-wise adaptive cross approximation (ACA) [4]. In larger problems the main cost is caused by the FEM part. Especially in the case of non-linear time dependent problems efficient solvers for the FE-system are essential. In this paper, we address the solution of such large sparse linear systems with a symmetric and positive definite system matrix.
One way of solving such systems is the Cholesky decomposition of the matrix in order to find the exact solution. In general, the number of non-zeros

strongly increases during the factorisation (fill-in). Several reordering algorithms based on graph theoretical heuristics have been developed in order to reduce occurring fill-in, e.g. the bandwidth reducing algorithms by Cuthill and McKee, and minimum degree strategies [1, 8]. Here, we concentrate on the idea of nested dissection first discussed in [6]. It is based on recursively finding vertex separators dividing the graph which describes the matrix structure. This gives a reordering which provides a block structure such that zero blocks remain empty in the factorised matrix.
For general sparse matrices not arising from FEM the so called automatic nested dissection [7, 13] can be applied to find separators. For this, no geometry information of the degrees of freedom is needed.
Nested dissection reordering was first applied to linear systems arising from 2D finite element discretisation forming a regular grid. There, the graph of the non-zero matrix pattern is a 2D tensor uniform grid. A cut along a straight line of this grid gives a separator of the mesh and of the matrix graph [6]. Later, mesh partitioning methods were used in order to find separators of finite element matrices arising from non-uniform meshes [9].
Our discretisation of Maxwell's equations uses an edge-based formulation, the regularisation of which increases the number of non-zeros in the stiffness matrix. This property does not allow using mesh partitioning as usual.
We will present a new reordering method that combines a geometric approach and the usage of the matrix structure in order to recursively find nested dissection interfaces. A cluster tree of the degrees of freedom will be constructed. Interfaces of decreasing size occur on every recursive level. If k_E is the number of unknowns, approximately $k_E^{2/3}$ of them are associated to the interface cluster that belongs to the first clustering step. The nested dissection based reordering yields a special block structure. A complete block Cholesky decomposition algorithm (HSLLT) benefiting from the time optimised BLAS-3 library will be used to solve the reordered system.
An evaluation of our method is carried out by a comparison to super-nodal factorisation methods [12] provided by freely available libraries for sparse matrix computation (see Section 8).
For higher dimensions, usually iterative methods (for example, Krylov subspace methods) are used where the stiffness matrix is only treated by matrix-vector multiplications. Because of a high condition number of the stiffness matrix a preconditioner is used to obtain reasonable convergence rates. Very fast methods of constructing preconditioners are based on incomplete factorisation as presented in [14]. It reduces the number of non-zeros in the Cholesky factor by setting negligible entries to zero.
Another possibility to construct a preconditioner is given by an approximate Cholesky decomposition as presented in [3], where $\mathcal{H}$-matrices as introduced

by Hackbusch [10] are used in order to reach almost linear complexity. $\mathcal{H}$-matrices are based on block-wise low-rank approximations yielding a data sparse representation of fully populated matrices.
We will first use the approximate $\mathcal{H}$-Cholesky decomposition in order to construct a preconditioner. For this, the above mentioned cluster tree is applied, which we originally constructed for nested dissection reordering. The second approach in constructing a preconditioner is also based on block-wise low-rank approximations. However, the factorisation is computed non-recursively with a block Cholesky algorithm and it has almost linear complexity.
This paper is organised as follows. First we describe the generation of the system of equations arising from the finite element discretisation of magnetostatic problems using edge elements (Section 2) and its regularisation (Section 3). In Section 4, we address the clustering method with the exact block decomposition method. In Section 5 and 6, the construction of approximate decompositions by block-wise low-rank approximation is described. After deriving complexity estimates (Section 7), we evaluate these methods in comparison to leading sparse matrix solvers by applying them to numerical examples in Section 8.

2 Discretisation of magnetostatic problems

Electromagnetic simulations are based on Maxwell's equations. We will only discuss the treatment of magnetostatic problems with linear materials. For this, the decoupled magnetic part of the equations holds:

$$\begin{aligned} \mathrm{curl}\vec{H} &= \vec{j}, & (1)\\ \mathrm{div}\vec{B} &= 0. & (2) \end{aligned}$$

Here, $\vec{H}$ is the magnetic field strength, $\vec{B}$ the magnetic induction and $\vec{j}$ the electric current density. The material properties are represented by

$$\vec{B} = \mu\vec{H} \qquad (3)$$

with a constant magnetic permeability μ for the linear case. The system of differential equations (1), (2) together with the material properties (3) will be solved on a contractible domain Ω by FEM. The potential ansatz $\vec{B} = \mathrm{curl}\vec{A}$ with the magnetic vector potential $\vec{A}$ is applied and yields the so called curl-curl-equation

$$\frac{1}{\mu}\mathrm{curl}\,\mathrm{curl}\vec{A} = \vec{j}. \qquad (4)$$

On the boundary we assume homogeneous Neumann boundary conditions. A variational formulation is derived in the Hilbert space

$$\mathbb{H}(\mathrm{curl},\Omega) = \{\vec{u} \in [\mathbb{L}_2(\Omega)]^3, \mathrm{curl}\vec{u} \in [\mathbb{L}_2(\Omega)]^3\}.$$

For this, $\mathrm{curl}\,\vec{u}$ exists in a weak sense and the respective scalar product is

$$\langle \vec{u}, \vec{v} \rangle_{\mathbb{H}(\mathrm{curl},\Omega)} = \langle \vec{u}, \vec{v} \rangle_{\mathbb{L}_2(\Omega)} + \langle \mathrm{curl}\vec{u}, \mathrm{curl}\vec{v} \rangle_{\mathbb{L}_2(\Omega)}.$$

Green's formula is applied to the variational representation. After considering the boundary conditions, this yields $a(\vec{u},\vec{v}) = f(\vec{v})$ with $\vec{u},\ \vec{v} \in \mathbb{H}(\mathrm{curl},\Omega)$, the symmetric bilinear form a and the linear functional f:

$$\begin{aligned} a(\vec{u},\vec{v}) &= \int_\Omega \frac{1}{\mu} \mathrm{curl}\vec{u} \cdot \mathrm{curl}\vec{v}\, d\Omega, \\ f(\vec{v}) &= \int_\Omega \vec{j} \cdot \vec{v}\, d\Omega. \end{aligned}$$

On a discretisation Ω_h of the domain Ω, the discrete representation of the magnetic vector potential reads

$$\vec{A}_h = \sum_{i=1}^{k_E} a_i \vec{\beta}_i,$$

where k_E is the number of edges of Ω_h. The degree of freedom a_i is interpreted as integral of $\vec{A}$ along the ith edge . In case of tetrahedrons $\vec{\beta}_i$ describes the Whitney-1-form [5]

$$\vec{\beta}_i = \lambda_k \nabla\lambda_l - \lambda_l \nabla\lambda_k$$

associated to the oriented edge $e_i = \{n_k, n_l\}$ from node n_k to n_l. Here, λ_k and λ_l are the Whitney-0-forms associated to n_k and n_l. In the case of hexahedrons we characterise $\vec{\beta}_i$ in the reference element. Each of its edges is parallel to one of the axis of the coordinate system (ξ,η,ζ). Thus, we find

$$\vec{\beta}_i = \begin{cases} \frac{1}{8}(1+\eta_i\eta)(1+\zeta_i\zeta)\,\vec{e}_\xi & ,\xi\text{−direction} \\ \frac{1}{8}(1+\zeta_i\zeta)(1+\xi_i\xi)\,\vec{e}_\eta & ,\eta\text{−direction} \\ \frac{1}{8}(1+\xi_i\xi)(1+\eta_i\eta)\,\vec{e}_\zeta & ,\zeta\text{−direction} \end{cases}$$

with $\vec{e}$ being the unit vector according to the subscripted coordinate. These tangentially continuous Whitney-1-forms form a finite dimensional discrete subspace of $\mathbb{H}(\mathrm{curl},\Omega)$. The discretisation of (4) by the Galerkin method

yields a linear system of equations $Qx = b$. The system matrix $Q \in \mathbb{R}^{k_E \times k_E}$ is symmetric and sparse with

$$Q_{ij} = \frac{1}{\mu} \int_{\Omega_h} \mathrm{curl}\vec{\beta}_i \cdot \mathrm{curl}\vec{\beta}_j \, d\Omega_h, \quad i, j = 1, \ldots, k_E.$$

The right hand side $b = (b_1, \ldots, b_{k_E})^T$ is given by

$$b_j = \int_{\Omega_h} \vec{j} \cdot \vec{\beta}_j \, d\Omega_h, \quad j = 1, \ldots, k_E,$$

and $x = (a_1, \ldots, a_{k_E})^T$ is the vector of degrees of freedom.
Due to the missing gauging in the potential ansatz, Q has a kernel of dimension $\dim(\ker Q) = k_N - 1$, where k_N is the number of corner nodes of the mesh. This is a result of topological considerations on the discretisation [5]. In order to deal with the singularity, we will describe a general way for regularisation in the next section.

3 Regularisation of the system

Let $A \in \mathbb{C}^{N \times N}$ be a complex valued singular matrix with

$$\begin{aligned} \ker A &= \mathrm{span}(v_1, \ldots, v_M) = \mathrm{Im}\, V\,, \; V = (v_1 \vdots v_2 \vdots \ldots \vdots v_M) \in \mathbb{C}^{N \times M}\,, \\ \ker A^* &= \mathrm{span}(u_1, \ldots, u_M) = \mathrm{Im}\, U\,, \; U = (u_1 \vdots u_2 \vdots \ldots \vdots u_M) \in \mathbb{C}^{N \times M}\,. \end{aligned}$$

The vectors $v_1, \ldots, v_M$ and $u_1, \ldots, u_M$ are assumed to be linearly independent and therefore the columns of the matrices V and U form a basis in $\ker A$ and in $\ker A^*$. Thus, the system of linear equations with the matrix A

$$A\,x = b\,, \; x, b \in \mathbb{C}^N \tag{5}$$

is only solvable if the right hand side b is orthogonal to the kernel of the adjoint matrix A^*,

$$b \in (\ker A^*)^{\perp}\,. \tag{6}$$

In this case, there are infinitely many solutions of the system (5). There are no solutions of this system if condition (6) is violated. The whole vector space $\mathbb{C}^N$ can be decomposed in the following direct orthogonal sums:

$$\mathbb{C}^N = \mathrm{Im}\, A \oplus \ker A^* = \mathrm{Im}\, A^* \oplus \ker A\,. \tag{7}$$

If system (5) is not solvable, the normal system can be considered:

$$A^* A\, x = A^* b\,. \tag{8}$$

This system is always solvable. The sets of solutions of systems (5) and (8) are identical if (5) is solvable. Otherwise the solutions of system (8) are called pseudo-solutions. However, the matrix A^*A of system (8) is still singular, much more populated than the sparse matrix A and possibly its condition number is much higher than that of the matrix A.
If condition (6) is violated, a further possibility to solve system (5) is to modify the right hand side,

$$A\, x = \tilde{b} = \mathcal{P} b\,,$$

where $\mathcal{P}$ is the projector on the image of the matrix A

$$\mathcal{P} \,:\, \mathbb{C}^N \to \operatorname{Im} A\,, \mathcal{P}^2 = \mathcal{P}\,.$$

If the kernel of the adjoint matrix A^* is known then it is theoretically easy to construct the projector $\mathcal{P}$. However, this procedure is not stable from the numerical point of view if the dimension of the kernel is very high ($M \sim N$). On the other hand, system (9) is still singular.
In this section, we suggest a different approach to deal with the singular system (5) which is based on the explicit knowledge of the kernel of the matrix A and its extreme sparsity (cf. (11)).

Lemma 1 *Let the matrix A of the system*

$$A\, x = b\,,\ \ x, b \in \mathbb{C}^N$$

be singular and let the columns of the matrices $V, U \in \mathbb{C}^{N\times M}$ form a basis in $\ker A$ *and in* $\ker A^*$*. Then the matrix $A + U\, V^*$ is regular, the linear system*

$$(A + U\, V^*)x = b \tag{9}$$

is uniquely solvable and its solution x is one of the solutions of the solvable original system. If the original system is not solvable then the vector x is a pseudo-solution.

Proof. Let us consider the square of the norm of the vector $(A + U\, V^*)v$ for an arbitrary vector $v \in \mathbb{C}^N$:

$$\begin{aligned} \|(A + U\, V^*)v\|^2 &= \|Av\|^2 + \|U\, V^* v\|^2 + 2\operatorname{Re}(Av, U\, V^* v) \\ &= \|Av\|^2 + \|U\, V^* v\|^2\,. \end{aligned}$$

Here, the property

$$A^*U = 0$$

implying

$$\big(Av, U\,V^*v\big) = \big(v, A^*U\,V^*v\big) = 0$$

has been used. The norm of the vector $(A + U\,V^*)v$ can thus vanish only if the next two conditions

$$Av = 0\,,\ U\,V^*v = 0$$

are satisfied simultaneously. These conditions are equivalent to

$$v \in \ker A\,,\ v \in \mathrm{Im}\,A^*\,. \tag{10}$$

The first property in (10) is obvious and the second can be checked as follows:

$$U\,V^*v = 0 \ \Leftrightarrow\ V^*v = 0 \ \Leftrightarrow\ v \in (\,\ker A)^{\perp} \ \Leftrightarrow\ v \in \mathrm{Im}\,A^*\,.$$

Here the linear independence of the columns of the matrix U has been used. Corresponding to (7), the vector v belongs to two spaces which are orthogonal to each other. It should be equal to zero. Thus, the matrix $A+U\,V^*$ is regular and the system of linear equations (9) is uniquely solvable for any right hand side b. The solution vector x solves system (8) since

$$A^*(A + U\,V^*)x = A^*Ax = A^*b\,.$$

If the original system is solvable, then x is one of the solutions. It is a pseudo-solution if the original system is not solvable.

■

The matrices Q arising from the FEM as described above are real valued and symmetric. Thus, the matrix $Q + U\,U^*$ can be used instead of the matrix Q without any modification of the right hand side.
In order to perform this regularisation, we need to construct the matrix U spanning the kernel. For this purpose, one has to consider the reason of ambiguity inherent in the potential ansatz. It allows to add gradients of arbitrary scalar functions

$$\vec{B} = \mathrm{curl}\vec{A} = \mathrm{curl}(\vec{A} + \nabla\psi)$$

without changing the magnetic induction. This is due to the vector analytic relation that gradient fields have no circulation.

To describe the discrete kernel, we need to find discrete representatives of gradient fields. The finite element mesh Ω_h forms a polygonal topology by a union of simplices in 3D. On such a topology, d-dimensional simplices can be transferred into $(d-1)$-dimensional simplices with the help of incidence matrices. It is shown in [5] that the discrete counterpart to the gradient operator is formed by the incidence matrix $D \in \mathbb{R}^{k_E \times k_N}$ associated to edges and nodes. In case of a connected domain, $\dim(\ker D) = 1$ and the image dimension amounts to $\dim(\operatorname{Im} D) = k_N - 1$. On a trivial topology for arbitrary vector fields,

$$\operatorname{curl}\vec{w} = 0 \Leftrightarrow \vec{w} = \nabla\vec{v}$$

holds in the continuous and the discrete case [5]. With this, one can conclude that the whole kernel U of the matrix Q is formed by the discrete gradient D (cf. [2]). The product UU^* contains information about the connection of the edges of the topology:

$$\left(UU^*\right)_{ij} = \begin{cases} 2, & i = j \\ \pm 1, & e_i \text{ and } e_j \text{ are adjacent} \\ 0, & e_i \text{ and } e_j \text{ are not adjacent} \end{cases}.$$

Thus, the matrix UU^* is symmetric and extremely sparse. The additional fill-in will be illustrated in detail in the last section of the paper, where some numerical results to the solution of

$$Kx = (Q + \alpha UU^*)x = b$$

will be shown. Here, the factor $\alpha > 0$ is used to improve the condition number of the system.

4 Hierarchical clustering algorithm

Our first approach for solving the system $Kx = b$ with positive definite system matrix K is the construction of the exact Cholesky decomposition. For this, a new hierarchical clustering algorithm is presented which provides a fill-in reducing reordering. Thus, $PKP^T = LL^T$ can be computed exploiting the arising block structure. In Section 5 and 6, we will reuse this hierarchical clustering for the computation of an approximate decomposition by $\mathcal{H}$-matrices.

As already mentioned, the decomposition factor L is in general much more populated than K because fill-in occurs in the Cholesky decomposition. For symmetric matrices, this is analysed within an undirected so called matrix graph $G(E, W)$. The vertices $e_i \in E$ with $i \in \mathcal{I} = \{1, \ldots, k_E\}$ represent

the degrees of freedom of the matrix. Between two vertices e_i and e_j an undirected edge $w \in W$ exists if there is a non-zero entry at the respective position in the matrix. Further graph theoretical details on symmetric matrices can be found in [13]. The natural ordering of the degrees of freedom is given by the sequence in which they occur in the matrix, that is, the sequence of rows and columns. The degrees of freedom are assigned to the index set $\mathcal{I}$.

The Cholesky decomposition step for a column i in the matrix is interpreted as the elimination of the associated vertex e_i in the graph G. The resulting elimination graph $G'(E', W')$ is formed,

$$\begin{aligned} E' &= E \setminus e_i, \\ W' &= W \cup C(\mathrm{adj}(e_i)). \end{aligned}$$

Here, $C(\mathrm{adj}(e_i))$ denotes the complete graph of all vertices adjacent to vertex e_i in the matrix graph G. By repeating this elimination process with G' until one vertex is remaining, one obtains the sparsity structure of the triangular factor L. During this process, the fill-in is counted by the number of additional edges occurring in the elimination graphs.

The ordering of the matrix, that is, the sequence of elimination nodes, affects the amount of fill-in. As mentioned in the introduction, there are several heuristics providing a fill-in reduced ordering of the matrix. We will concentrate on nested dissection strategies. For this, a partitioning of the matrix graph given by a preferably small set S_I, called interface, needs to be found. Removing all vertices associated to S_I and their incident edges results in two clustered graphs with no connection between them.

The reordering is performed by permuting the index set, so that first the vertices of the two non-connected graphs are numbered consecutively, followed by the vertices of the interface. This will be done recursively for the resulting subgraphs as is illustrated in Figure 1 by two steps of partitioning.

On the left, the matrix graph is shown, where grey coloured parts represent the recursively constructed interface layers. On the right, one can see the matrix structure after renumbering the degrees of freedom. The dark grey blocks belong to the nodes of the first level interface layer I. In the remaining 2×2 block, only the two blocks on the diagonal contain non-zeros, whereas the two white shaded blocks are empty. This is because the vertices separated by the interface I do not interact. The corresponding interfaces I_1 and I_2 divide up the previous partitioning. The respective rows and columns of the matrix can be reordered in such a way that the same structure is obtained on the non-zero subblocks.

The construction of such interfaces can be done graph theoretically [7, 13]. Another method is geometrical partitioning of the associated finite element

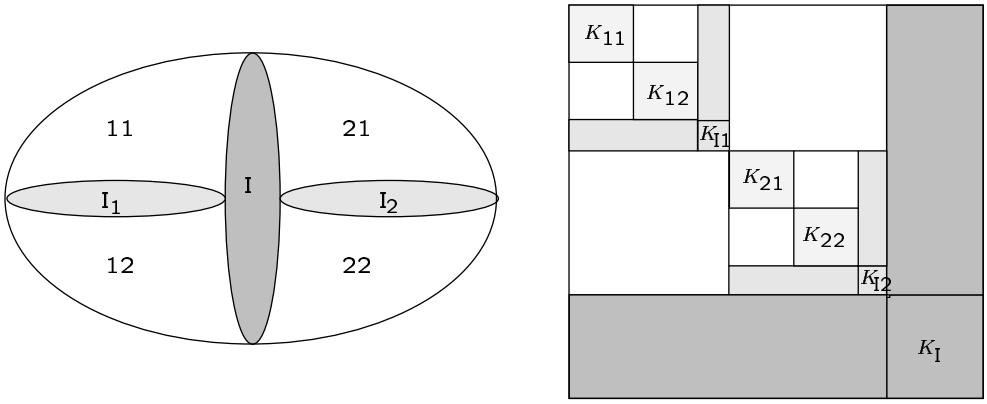

Figure 1: On the left, two steps of hierarchical clustering are illustrated. The associated matrix structure after renumbering is shown on the right.

mesh [9]. For our application, a combination of geometrical partitioning and the usage of the matrix structure is appropriate. In the following sections, we present an algorithm to create a suitable cluster tree which is used to renumber the matrix efficiently.

4.1 The structure of the cluster tree

Starting point for the clustering is the index set $\mathcal{I} = \{1, \ldots, k_E\}$ corresponding with edges of the finite element mesh. A so called cluster tree $T_\mathcal{I}$ is a hierarchical partitioning of $\mathcal{I}$ [10]. Elements of a cluster tree are called clusters and form subsets of $\mathcal{I}$.
For every cluster $S \subset \mathcal{I}$ we define the set of sons $\mathcal{S}(S) = \{S',\ S' = \mathrm{son}(S)\}$. If a cluster has no sons, it is called a leaf. The set of leafs of the tree is denoted by $\mathcal{L}(T_\mathcal{I}) = \{S,\ \mathcal{S}(S) = \emptyset\}$.

Definition 2 (cluster tree) *With the finite index set $\mathcal{I}$ a cluster tree $T_\mathcal{I}$ is defined as follows*

1. $\mathcal{I} \in T_\mathcal{I}$,
2. $\forall S \subset T_\mathcal{I}$: $S = \bigcup_{S' \in \mathcal{S}(S)} S'$,
3. $\forall S \subset T_\mathcal{I}$: $\#\mathcal{S}(S) \neq 1$.

In order to reach a suitable nested dissection structure, every non-leaf cluster is desired to have three sons. If so, one of them contains all indices dividing the underlying matrix graph of its father, that is, the interface. This cluster is a leaf. The two other clusters are leaves if they have less than a certain

number $b > 2$ of degrees of freedom. Alternatively, the set of sons can contain only two clusters. This happens if the father cluster is not connected, so that it is possible to find two sons divided by nature.

4.2 Creation of the cluster tree

In the creation process we make use of the correspondence between the matrix graph of stiffness matrix K and the underlying geometry of the finite element mesh Ω_h. The edges of the mesh correspond with the vertices of the matrix graph $G(E, W)$. One can assign geometrical information to $e_i \in E$, $i \in \mathcal{I}$, by the coordinates of the midpoint of the underlying finite element edge and a weight given by its length. W is given by the matrix entries. Our approach of clustering $S \subset \mathcal{I}$ consists of two steps:

Geometrical bisection

For this, only the geometry of the degrees of freedom, i.e. the finite element edges, of the cluster S is taken into account. If S has the cardinality k, the centre of the cluster is given by the weighted sum

$$\vec{x}_S = \frac{1}{g_S} \sum_{i=1}^{k} g_i \vec{x}_i,$$

where g_S is the sum of all weights g_i and $\vec{x}_i$ are the coordinates of the midpoint of edge e_i. The eigenvectors of the covariance matrix

$$\mathrm{cov}(S) = \sum_{i=1}^{k} g_i (\vec{x}_S - \vec{x}_i)(\vec{x}_S - \vec{x}_i)^T \in \mathbb{R}^{3\times 3}$$

are the axes of an enveloping ellipsoid. The plane orthogonal to the eigenvector $\vec{v}$ corresponding to the largest eigenvalue divides the indices in two sets as follows:

$$\begin{aligned} {S_1}' &= \{i \in \mathcal{I}, \ \ \vec{v} \cdot (\vec{x}_i - \vec{x}_S) < 0\}, \\ {S_2}' &= \{i \in \mathcal{I}, \ \ \vec{v} \cdot (\vec{x}_i - \vec{x}_S) \geq 0\}. \end{aligned}$$

Construction of the interface cluster

The underlying matrix graph of cluster S is denoted by $G(E_S, W_S)$ with $E_S \subset E$ and $W_S \subset W$ containing all interactions within the cluster S. Our next aim is to find the set of indices S_I such that the removal of S_I and its adjacent edges forms a disconnected graph $G(E_{S_1} \cup E_{S_2}, {W_S}')$, with

$$\begin{aligned} S_1 = {S_1}' \setminus S_\mathrm{I}, \ S_2 = {S_2}' \setminus S_\mathrm{I}, \ {W_S}' = W_S \setminus \mathrm{adj}(E_{S_\mathrm{I}}), \\ \forall i \in S_1, \, j \in S_2 : \ w = \{e_i, e_j\} \notin {W_S}'. \end{aligned}$$

S_I should be as small as possible.
We now check $S_1{}'$ and $S_2{}'$ for connecting edges $w = \{e_i, e_j\} \in W_S, i \in S_1', j \in S_2'$ to construct the interface cluster S_I. For this, an array `sort` subscripted over all indices of S is used. Its initial entries are given by the geometrical clustering:

$$\texttt{sort[i]} = \begin{cases} 1, & i \in S_1{}' \\ -1, & i \in S_2{}' \end{cases} .$$

This array is filled with zeros at all indices associated to the interface S_I by the following algorithm.

Algorithm 3 (set interface layer)

1. *for all i - alternately chosen from $S_1{}'$ and $S_2{}'$ as index farthest from the cutting plane*
2. *for all $j \in S$*
3. *if $K_{ij} \neq 0$ and* `sort[j]` $\neq 0$ *and* `sort[i]`$\neq$`sort[j]`
4. `sort[i]`$= 0$
   goto 1
5. *move elements marked by zero into S_I*

Using the sparse matrix format, we do not have to pass over all elements in loop 2. The cost is bounded by the maximum node degree in the matrix graph, so the complexity of this algorithm is $\mathcal{O}(\#S)$. By construction, there are no more interactions between S_1 and S_2. The clustering process continues recursively on cluster S_1 and S_2 while they are non-leaves. As mentioned above, a cluster is a leaf if it is an interface, or its cardinality is less than b, or the algorithm fails to find a further sub-clustering.
From the leaves of the cluster tree, we can create a vector of indices which represents the permutation of the degrees of freedom to obtain a structure as illustrated in Figure 1. This permutation of the system matrix allows the application of the following exact Cholesky decomposition which exploits the block structure.
In the context of the later discussed $\mathcal{H}$-arithmetic for computing an approximate Cholesky decomposition, the cluster tree is used for the construction of the $\mathcal{H}$-matrix.

4.3 Cholesky decomposition

From the cluster tree, a prediction of the sparse matrix pattern of the Cholesky factor L is possible. Thus, we can already reserve space for the future non-zeros in the sparse matrix.

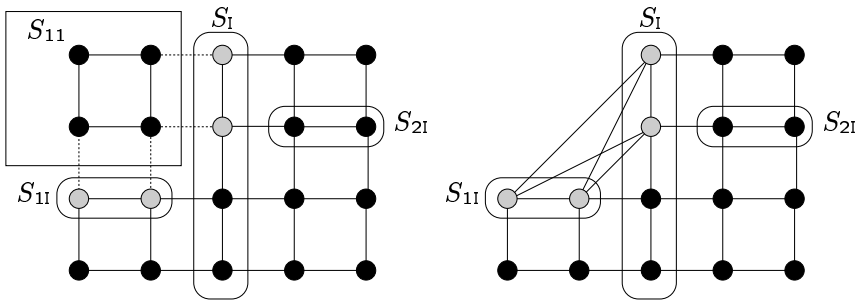

Figure 2: A matrix graph on the left and on the right the elimination graph after eliminating all vertices of S_{11} is shown.

Figure 2 shows a matrix graph $G(E, W)$ with two levels of clustering and the corresponding interfaces S_{I}, $S_{1\mathrm{I}}$, $S_{2\mathrm{I}}$. The degrees of freedom associated to the non-interface cluster S_{11} are permuted to the beginning of the matrix. This means that the Cholesky factorisation starts with these columns. It can be interpreted graph theoretically as forming the elimination graph $G'(E \setminus e_i, W \cup C(\mathrm{adj}(e_i)))$ of a respective vertex e_i. The vertices of S_{11} can produce interactions between vertices contained in S_{11} and in interface clusters. However, there will never occur edges to other non-interface clusters.
We will consider fill-in by assuming the worst case of interactions caused by elimination of S_{11}. All vertices of S_{11} and all e_j, $j \in S_{\mathrm{I}}$ and $S_{1\mathrm{I}}$ connected by $w = \{e_i, e_j\}$ (dotted edges) to S_{11} will cause fill-in. We assume that the connections between them form a complete graph. This is, the elimination of all columns according to S_{11} then results in a non-zeros matrix pattern shown in Figure 3. In the elimination graph $G'(E \setminus S_{11}, W \cup C(\mathrm{adj}(S_{11})))$ the complete graph of all respective interface vertices is contained.
The advantage of this worst case guess of fill-in lies in the same sparsity structure of columns belonging to the same block. Thus, the sub-diagonal non-zeros can be accumulated in the vectors $c_1^T, \ldots, c_n^T$ forming the non-zero rows of the block. The factorisation of the first block column only provides Schur complement updates in the $n \times n$ grey coloured positions in the lower right of Figure 3. We can merge the non-zero rows of the block in a dense array $K_{1*}^T := (c_1^T \vdots \ldots \vdots c_n^T)$. With this, the Cholesky factorisation can be

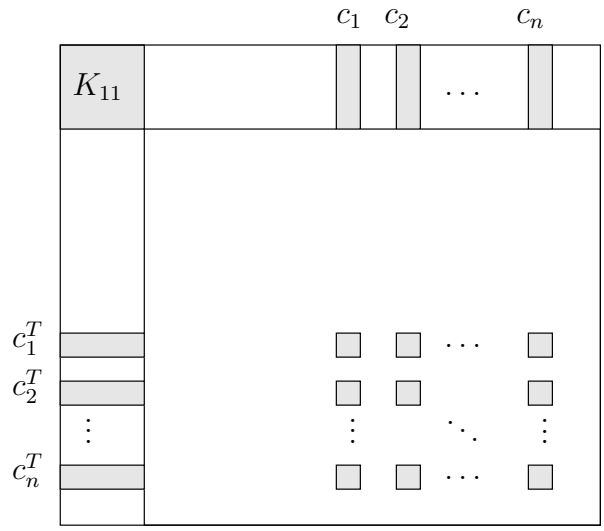

Figure 3: Block Cholesky elimination for columns belonging to K_{11} and resulting fill-in.

done block-wise by dense matrix computations using time optimised BLAS-3 library.

Algorithm 4 (block decomposition) *Do for every block column i:*

1. *decompose* $K_{ii} = L_{ii}L_{ii}^T$
2. $D = L_{ii}^{-1}K_{i*}$
3. *compute* $D^T D = K_{i*}^T L_{ii}^{-T} L_{ii}^{-1} K_{i*}$
4. *update the Schur complement at the correct position of the sparse matrix*

The clustering algorithm and the described decomposition form our exact solver called HSLLT.

5 $\mathcal{H}$-Matrix technique and an approximate decomposition

The $\mathcal{H}$-matrix technique is a way of approximating matrices. With the associated arithmetic [11, 10] one can perform matrix operations. For this, a cluster tree as constructed in the previous section is needed.

The proof given in [3] states that Schur complements can be approximated with $\mathcal{H}$-matrices. With this, the hierarchical decomposition of sparse matrices was described. We want to apply the described clustering method for $\mathcal{H}$-matrix construction and perform the hierarchical Cholesky decomposition (HLLT). Our approach differs from [3] so a short description of the arithmetic is given.
For a recursive approach in storing the $\mathcal{H}$-matrix we provide four types of matrices:

- `Null` – zero matrix of any dimension;
- `Dense` – fully populated matrix $\mathbb{R}^{m\times n}$;
- `LowRank(`r`)` – low-rank matrix with rank r stored as its QR decomposition, i.e. $Q \in \mathbb{R}^{m\times r}$, $R \in \mathbb{R}^{r\times n}$, $Q^TQ = I \in \mathbb{R}^{r\times r}$, and R is the upper triangular matrix
- $\mathcal{H}$`-matrix` – $\mathbb{R}^{m\times n}$ matrix separated by b_m and b_n lines into 4 blocks: $\mathbb{R}^{b_m\times b_n}$, $\mathbb{R}^{(m-b_m)\times b_n}$, $\mathbb{R}^{b_m\times(n-b_n)}$, $\mathbb{R}^{(m-b_m)\times(n-b_n)}$. Each of them is again represented as one of those types.

Inside of this class we establish binary operations: addition and multiplication. Furthermore, the following unary operations are implemented: the LL^T decomposition of $\mathcal{H}$`-matrix` type, a `Dense` to `LowRank(r)` transformation, the post-compression of `LowRank(`r`)` and the transformation of any type to $\mathcal{H}$`-matrix` type with predefined b_m and b_n.
Each matrix operation is designed to save space and computation costs, i.e. if the input data contains only fully populated matrices, then a fully populated result matrix is produced. The post-compression of the result matrix into low-rank type is done only if both matrix dimensions are large. Only in this case we can expect that the compression rate can be large enough.

5.1 The addition

If one of the input matrices is of `Dense` type, it will take at least nm arithmetic operations to refer to it. The result is again stored in `Dense` type. After that, a low-rank approximation can be applied in order to convert the matrix into `LowRank(`r`)` with r being small enough. In case of two `LowRank` matrices with rank r_1, r_2 we perform

$$Q_1R_1 + Q_2R_2 = (Q_1, Q_2)(R_1^T, R_2^T)^T,$$

The matrix (Q_1, Q_2) is no longer orthogonal. Hence, the post-compression algorithm is applied.

The `LowRank(`r`)`+$\mathcal{H}$`-matrix`-operation has less than nm data in the input and could be represented in compact storage format at the output.
Since an $\mathcal{H}$`-matrix` might have full rank, there is no reason to convert the result to `LowRank` type. Hence we convert `LowRank(`r`)` to $\mathcal{H}$`-matrix` type and call the $\mathcal{H}$`-matrix`+$\mathcal{H}$`-matrix` operation.
In case of two $\mathcal{H}$-matrices we have to ensure the same partitioning (b_m, b_n). If they differ, we transform one of these matrices into the partitioning of the other. After that the block sum is performed as usual [10].

5.2 The multiplication

Here, we only want to consider the multiplication of non-trivial kinds of matrices.
If the `LowRank(`r`)` type is involved, for example AB, and $A = QR$, then

$$AB = (QR)B = Q(RB) = QZ.$$

This means, we multiply the `Dense` type matrix R to B and store the result in Z, which is of `Dense` type. Then, we perform a post-compression of QZ.
To compute $\mathcal{H}$`-matrix` times `Dense`, the matrix of `Dense` type is split into two blocks, so that

$$\begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix} \begin{pmatrix} D_1 \\ D_2 \end{pmatrix} = \begin{pmatrix} H_{11}D_1 + H_{12}D_2 \\ H_{21}D_1 + H_{22}D_2 \end{pmatrix},$$

The result is stored again in `Dense` type.
The remaining kind is the multiplication of two $\mathcal{H}$-matrices. By ensuring the same partitioning of the matrix, the block matrix multiplication is performed as in [10].

5.3 LL^T decomposition of $\mathcal{H}$-matrix

Suppose we are going to compute the Cholesky decomposition of

$$H = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix},$$

where $H_{12} = H_{21}^T$, then:

$$H = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{pmatrix},$$

is computed by

$$\begin{aligned} H_{11} &= L_{11}L_{11}^T, \\ L_{21} &= H_{21}L_{11}^{-1}, \\ H_{22} - L_{21}L_{21}^T &= L_{22}L_{22}^T. \end{aligned}$$

5.4 `Dense` to `LowRank(`r`)`

This algorithm is trying to compress the input matrix $\mathbb{R}^{m\times n}$ to a product of two matrices $\mathbb{R}^{m\times r}$ and $\mathbb{R}^{r\times n}$ so that $r \ll \frac{mn}{m+n}$. The optimal method is the singular value decomposition (SVD), but it requires $\mathcal{O}(mn\min(m,n))$ arithmetic operations. Hence we use truncated rank revealing method which only takes $\mathcal{O}(mnr)$ arithmetic operations and produces the QR decomposition. Other methods like Adaptive Cross Approximation (ACA) [4] can also be useful. However, they have the same order of arithmetic complexity.

5.5 Post-compression of `LowRank(`r`)`

Suppose the matrix is presented as AB, $A \in \mathbb{R}^{m\times r}$, $B \in \mathbb{R}^{r\times n}$, and A is not anymore orthogonal. This post-compression algorithm should find new $\tilde{A}$, $\tilde{B}$ possibly with smaller rank, so that $||AB - \tilde{A}\tilde{B}||_F < \epsilon$. There are several ways to obtain $\tilde{A}$, $\tilde{B}$ [10]. In our algorithm we use the following method:

Algorithm 5 (Post-compression of `LowRank(`r`)`)

1. *Compute QR decomposition of $A = Q_A R_A$,*
2. *Compute $C = R_A B$;*
3. *Compute truncated QR decomposition of $C = Q_C R_C$*
4. *Compute $\tilde{A} = Q_A Q_C$, and assign $\tilde{B} = R_C$.*

5.6 Transformation to $\mathcal{H}$-`matrix`

If the `Dense` matrix has to be transformed into $\mathcal{H}$-`matrix` type, we have to find the corresponding blocks and copy them into $\mathcal{H}$-`matrix` blocks.
If the `LowRank(`r`)` matrix is transformed to $\mathcal{H}$-`matrix` type, then we can construct four blocks of `LowRank(`r`)` type with the same rank. Then each block in the $\mathcal{H}$-`matrix` type has no orthogonal Q anymore. Moreover, it could be the case that the rank is too big. Consider the block (1,1): if $r > \dfrac{b_m b_n}{b_m + b_n}$ then we should transform it to the dense format, otherwise, try to post-compress it.
If the $\mathcal{H}$-`matrix` H_0 with $b_m^{(0)}, b_n^{(0)}$ partitioning is transformed to $\mathcal{H}$-`matrix` H with a different partitioning b_m, b_n ($b_m \neq b_m^{(0)}$ or $b_n \neq b_n^{(0)}$), the algorithm works recursively. Suppose $b_m < b_m^{(0)}$, $b_n < b_n^{(0)}$,

$$H_0 = \begin{pmatrix} H_{11}^{(0)} & H_{12}^{(0)} \\ H_{21}^{(0)} & H_{22}^{(0)} \end{pmatrix},$$

$$H = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix}.$$

Let us define this operation $\text{Split}(H_0, b_m, b_n)$ in the following way. Suppose

$$\text{Split}(H_{11}^{(0)}, b_m, b_n) = \begin{pmatrix} H_{11}^{11} & H_{12}^{11} \\ H_{21}^{11} & H_{22}^{11} \end{pmatrix},$$

$$\text{Split}(H_{12}^{(0)}, b_m, 0) = \begin{pmatrix} H_{11}^{12} \\ H_{21}^{12} \end{pmatrix},$$

$$\text{Split}(H_{21}^{(0)}, 0, b_n) = \begin{pmatrix} H_{11}^{21} & H_{12}^{21} \end{pmatrix},$$

then

$$\text{Split}(H_0, b_m, b_n) = \begin{pmatrix} H_{11}^{11} & \begin{pmatrix} H_{12}^{11} & H_{11}^{12} \end{pmatrix} \\ \begin{pmatrix} H_{21}^{11} \\ H_{11}^{21} \end{pmatrix} & \begin{pmatrix} H_{22}^{11} & H_{21}^{12} \\ H_{12}^{21} & H_{22}^{(0)} \end{pmatrix} \end{pmatrix}.$$

6 Approximate decomposition without $\mathcal{H}$-matrix technique

The main disadvantage of the above described $\mathcal{H}$-matrix technique lies in the high number of post-compressions during Cholesky decomposition. Now we introduce a new approach in order to reduce the number of these numerous post-compressions.

Let us assume the permuted matrix given by the presented clustering algorithm. By the leaf clusters, the columns of the matrix are partitioned into block columns, as suggested by Figure 3. The number of leaves corresponds to the number of block columns.

Let k_1 be the block size, k_2 the number of non-zero rows or columns of the sub-diagonal block and n the dimension of the matrix. The elimination process goes block-column-wise. In case of decomposing the first block-column,

$$\begin{pmatrix} K_{11} & K_{*1}^T \\ K_{*1} & K \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{*1} & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & K - L_{*1}L_{*1}^T \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{*1}^T \\ 0 & I \end{pmatrix}$$

has to be computed, where $K_{*1}^T \in \mathbb{R}^{k_1 \times (n-k_1)}$. In a complete decomposition the sub-diagonal block L_{*1}^T is computed by $L_{*1}^T = L_{11}^{-1} K_{*1}^T$. This result will be approximated by reduced QR-decomposition, so that $L_{*1}^T \approx UV$. This is possible because the sub-diagonal block forms an admissible block. Here, $V \in \mathbb{R}^{r \times (n-k_1)}$ and $U^T U = I$ with r being the approximate rank. This yields

$$\begin{pmatrix} K_{11} & K_{*1}^T \\ K_{*1} & K \end{pmatrix} \approx \begin{pmatrix} L_{11} & 0 \\ V^T U^T & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & K - V^T V \end{pmatrix} \begin{pmatrix} L_{11}^T & UV \\ 0 & I \end{pmatrix}.$$

The block K_{*1}^T is column sparse, i.e., it has only k_2 non-zero columns. This results in a column sparse structure of V. The matrix can be stored as a fully populated matrix together with the indices of the non-zero columns.
The important part of this decomposition is the computation of updates arising from former Schur complements. Assume that the i-th diagonal block is now in the elimination:

$$\left(\begin{array}{c|c} K_{1,1} & K_{*,i}^T \\ \hline K_{*,1} & \begin{array}{c|c} \ddots & \cdots \\ \hline \vdots & \begin{array}{c|c} K_{i-1,i-1} & K_{*,i-1}^T \\ \hline K_{*,i-1} & \begin{array}{c|c} K_{i,i} & K_{*,i}^T \\ \hline K_{*,i} & \ddots \end{array} \end{array} \end{array} \end{array}\right) =$$

$$\left(\begin{array}{c|c} L_{1,1} & 0 \\ \hline V_1^T U_1^T & \begin{array}{c|c} \ddots & 0 \\ \hline \vdots & \begin{array}{c|c} L_{i-1,i-1} & 0 \\ \hline V_{i-1}^T U_{i-1}^T & I \end{array} \end{array} \end{array}\right) S_{i,i} \left(\begin{array}{c|c} L_{1,1} & U_1 V_1 \\ \hline 0 & \begin{array}{c|c} \ddots & \cdots \\ \hline 0 & \begin{array}{c|c} L_{i-1,i-1}^T & U_{i-1} V_{i-1} \\ \hline 0 & I \end{array} \end{array} \end{array}\right),$$

$$S_{i,i} = \left(\begin{array}{c|c} I & 0 \\ \hline 0 & \begin{array}{c|c} \ddots & \cdots \\ \hline \vdots & \begin{array}{c|c} I & 0 \\ \hline 0 & \begin{array}{c|c} K_{i,i} & K_{*,i}^T \\ \hline K_{*,i} & \ddots \end{array} \end{array} \end{array} \end{array}\right) - \begin{pmatrix} 0 \\ V_1 \end{pmatrix} \begin{pmatrix} 0 \\ V_1 \end{pmatrix}^T - \ldots - \begin{pmatrix} 0 \\ \vdots \\ V_{i-1} \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ V_{i-1} \end{pmatrix}^T.$$

We first have to find matrices $V_1, \ldots, V_{i-1}$ that change $K_{i,i}$ and $K_{*,i}^T$. Additional non-zero columns can arise in $K_{*,i}^T$ because of the fill-in. This fill-in is reduced by the norm of columns. If its norm is smaller than the norm of the diagonal block multiplied to a threshold, this column is set to zero.
The computation of the Schur complement in case of complete decomposition amounts to $k_1 k_2^2$ operations, whereas in case of the approximate decomposition it takes $r k_2^2$.

Algorithm 6 *For every block column i, one has to do the following steps:*

1. *Compute updates arising from former Schur complements*
2. *Compute the Cholesky decomposition of the diagonal block $K_{ii} = L_{ii} L_{ii}^T$*
3. *Compute the sub-diagonal block*

$$\begin{array}{ll} k_2 < k_1: & L_{ii} L_{*i}^T = K_{*i}^T \\ k_2 \geq k_1: & \text{with } K_{*i} \approx U_i V_i,\ U_i^T U_i = I \\ & \text{compute } L_{*i}^T = (L_{*i}^T)' U_i^T \\ & \text{by solving } L_{ii} (L_{*i}^T)' U_i^T = V_i^T U_i^T \rightarrow (L_{*i}^T)' = L_{ii}^{-1} V_i^T \end{array}$$

4. Compute a low-rank approximation of L_{*i}^T

$$\begin{aligned} k_2 < k_1: &\quad L_{*i}^T \approx UV \\ k_2 \geq k_1: &\quad L_{*i}^T = (L_{*i}^T)' U_i^T \approx U(VU_i^T) \end{aligned}$$

Within this method, called HSILLT, one will process every block of the matrix once, so that the number of low-rank approximations is rather low now.

7 Complexity estimates

Suppose Mem_n represents the total amount of memory required for storing the Cholesky factors for the HSLLT algorithm. According to our clustering method for 3D problems, the interface cluster has only $\mathcal{O}(n^{2/3})$ degrees of freedom, so $\mathcal{O}(\#S_I) = \mathcal{O}(n^{2/3})$ (see Figure 1) and $\mathcal{O}(\#S_1) = \mathcal{O}(\#S_2) = \mathcal{O}\left(\frac{n-n^{2/3}}{2}\right)$. Thus, a fully populated decomposed block requires

$$Mem_n = n^{4/3} + 2Mem_{\frac{n-n^{2/3}}{2}} \leq n^{4/3} + 2Mem_{n/2}.$$

Assume that $Mem_1 = 1$, then

$$\begin{aligned} Mem_n \leq n^{4/3} + 2Mem_{n/2} \leq n^{4/3} + 2\left(\frac{n}{2}\right)^{4/3} + 4\left(\frac{n}{4}\right)^{4/3} + \ldots = \\ n^{4/3}\left(1 + 2^{-1/3} + \left(2^{-1/3}\right)^2 + \ldots\right) \leq n^{4/3}\frac{1}{1-2^{-1/3}} \leq 5n^{4/3}. \end{aligned}$$

For HSILLT, suppose that the $m \times m$ block associated to the interface degrees of freedom is further divided in several blocks

<table>
<tr><td>D_1</td><td colspan="4">B_1^T</td></tr>
<tr><td rowspan="4">B_1</td><td>D_2</td><td colspan="3">B_2^T</td></tr>
<tr><td rowspan="3">B_2</td><td>D_3</td><td colspan="2">B_3^T</td></tr>
<tr><td rowspan="2">B_3</td><td>D_4</td><td>B_4^T</td></tr>
<tr><td>B_4</td><td>$\ldots$</td></tr>
</table>

.

Each diagonal block D_i is stored as a fully populated matrix and the blocks B_i are approximated by a low-rank matrix with rank at most r. Suppose, that each block D_i is of dimension $k \times k$. With this, the memory requirement for storing the interface block is km words for the diagonal blocks D_i and $mr(1 + \frac{m}{2k})$ words for the sub-diagonal blocks. Assume $k \simeq \sqrt{m}$, then the total memory requirement for storing the interface block is $\mathcal{O}(rm^{3/2})$. Since

$\#S_I = n^{2/3}$, this block needs $\mathcal{O}(rn)$ words of memory, and, in analogy to (11)

$$Mem_n \leq rn + 2Mem_{n/2} \leq rn + 2\left(\frac{rn}{2}\right) + 4\left(\frac{rn}{4}\right) + \ldots \leq rn \log_2 n.$$

Hence the approximated LL^T has an almost linear memory requirement.

The arithmetic complexity of the computation of HSLLT is not linear. We need to compute the Cholesky factorisation for the dense subblocks, so the complexity for the factorisation of the interface block with $\#S_I = n^{2/3}$ degrees of freedom amounts to $\mathcal{O}(n^2)$ arithmetic operations, hence

$$Op_n \leq n^2 + 2Op_{n/2} \leq n^2 + 2\left(\frac{n}{2}\right)^2 + 4\left(\frac{n}{4}\right)^2 + \ldots = \tag{11}$$
$$n^2\left(1 + 2^{-1} + \left(2^{-1}\right)^2 + \ldots\right) \leq 2n^2.$$

In the approximate Cholesky factorisation we have to compute LL^T for the diagonal $k \times k$ blocks and do QR decomposition of the sub-diagonal blocks. It leads to

- $\frac{m}{k}$ Cholesky decompositions of $k \times k$ block: $\mathcal{O}(mk^2)$ arithmetic operations,
- QR/low rank approximation of $(m-k) \times k, \ldots, (m-m+k) \times k$ blocks: $\mathcal{O}(rm^2)$ arithmetic operations,
- update the Schur complement of each block $m^2r + \ldots + (m-m+k)^2r = \mathcal{O}(m^2kr)$ arithmetic operations,

then, in case of $k = \text{const}$, the complexity of the factorisation of one block ($\#S_\text{I} = n^{2/3}$) is $\mathcal{O}(rn^{4/3})$ arithmetic operations. The total arithmetic complexity for HSILLT is

$$Op_n \leq rn^{4/3} + 2Op_{n/2} \leq rn^{4/3} + 2r\left(\frac{n}{2}\right)^{4/3} + 4r\left(\frac{n}{4}\right)^{4/3} + \ldots =$$
$$rn^{4/3}\left(1 + 2^{-1/3} + \left(2^{-1/3}\right)^2 + \ldots\right) \leq rn^{4/3}\frac{1}{1 - 2^{-1/3}} \leq 5rn^{4/3}.$$

If we make a cutoff of non-diagonal entries by threshold and reduce k we can expect that the fill-in will be constant time larger than the system matrix. In this case the arithmetic complexity can be estimated as:

- $\frac{m}{k}$ Cholesky decomposition of $k \times k$ block: $\mathcal{O}(mk^2)$ arithmetic operations,

- QR/low rank approximation of $\frac{m}{k}$ blocks of const$k \times k$ sizes: $\mathcal{O}(rmk)$ arithmetic operations,
- update the Schur complement of each block $\mathcal{O}(rk^2)$ arithmetic operations for each $\frac{m}{k}$ blocks,

hence, again with constant k, it leads to $\mathcal{O}(rn^{2/3})$ arithmetic operations for the factorisation of one block. The total complexity is then

$$Op_n \leq rn^{2/3} + 2Op_{n/2} \leq rn^{2/3} + 2r\left(\frac{n}{2}\right)^{2/3} + 4r\left(\frac{n}{4}\right)^{4/3} + \ldots = rn^{2/3}\left(1 + 2^{1/3} + \left(2^{1/3}\right)^2 + \ldots n^{1/3}\right) \leq 10rn.$$

8 Numerical examples

The three presented algorithms are based on the interface clustering of the degrees of freedom using the geometry and the structure of the sparse matrix (cf. 4). The resulting permutation reduces the number of non-zeros in the Cholesky factor and one obtains a block structure of the matrix. Figure 4 shows these modifications of the structure. The structure of the reordered singular matrix Q and the regularised matrix K are pictured in order to show the influence of additional entries arising from regularisation (cf. 3). The number of non-zeros in the system matrix doubles and the matrix stays sparse.

8.1 Performance rating – exact solver

In Section 4 an exact linear system solver called HSLLT was described. The approach of HSLLT can be compared with super-nodal techniques [12]. They try to find columns with the same sparsity structure in a graph theoretical way by the so called elimination tree. Forming dense block matrices allows the usage of BLAS-3.
For the performance evaluation of our algorithm two freely available solver packages are chosen. One of them is a library called Pardiso[1] (Parallel sparse direct linear solver). It was developed by Olaf Schenk from the University of Basel. The package provides decomposition methods for different kinds of sparse matrices. For our symmetric positive definite test problems we

[1]cf. http://www.computational.unibas.ch/cs/scicomp/software/pardiso/ and provided publications

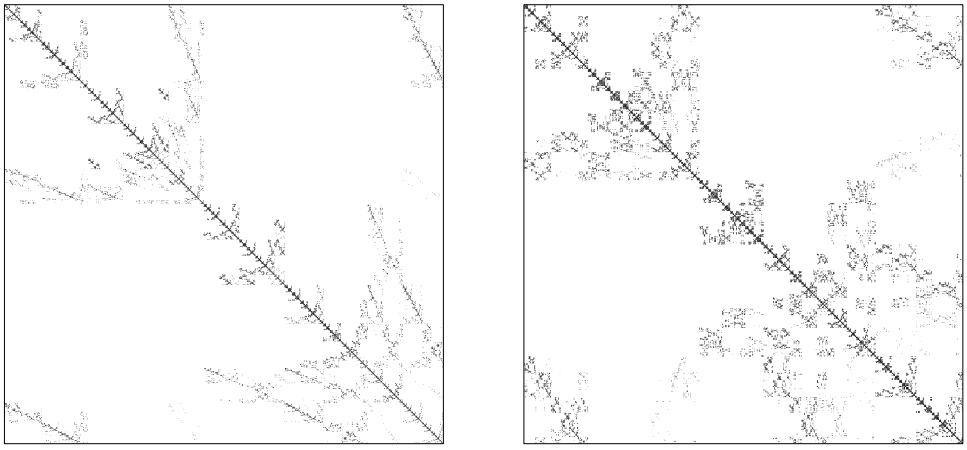

Figure 4: Structure of the reordered singular system matrix Q (left) and the reordered regularised matrix K (right).

selected a sequential left- and right-looking super-nodal Cholesky factorisation. Different graph theoretical methods are available for reordering, e.g. a multiple minimum degree algorithm (MMD) and a nested-dissection reordering provided by the Metis library (G. Karypis). The latter is based on nested-dissection strategies similar to our permutation.
Another solver package is Taucs [2], which comes from the University of Tel Aviv. Sivan Toledo also created one of the leading available solvers. It offers numerous routines for exact decomposition of sparse matrices. They use super-nodal and multi-frontal methods. Several reordering methods provide reduced fill-in. For our comparison, the AMD method by T. Davis and again the Metis library were selected. In a later subsection we return to Taucs and its methods of incomplete factorisation to construct a preconditioner.
In summary, three factorisation methods were appropriate for comparison: HSLLT, the super-nodal method by Pardiso (PLLT) and by Taucs (TLLT). Our test matrices are based on the finite element discretisation of magnetostatic problems. At first we chose simple meshes of a cube consisting of uniformly distributed hexahedrons. An example more representative of real applications is that of a simple magnetic valve. With the described regularisation, all these examples provide symmetric positive definite matrices because of their trivial topology. In more general topologies it is necessary to deal with cohomology aspects to find the bigger kernel of the matrix, which will be done in future work.

[2] cf. http://www.tau.ac.il/∼stoledo/taucs and provided publications

name	k_E	element type
cube1	26460	hexahedrons
cube2	45000	hexahedrons
cube3	70644	hexahedrons
cube4	104544	hexahedrons
cube5	147852	hexahedrons
cube6	201720	hexahedrons
valve	72672	tetrahedrons

Table 1: Models used for the comparison of the different solvers

The reordering algorithms in Pardiso and Taucs are graph theoretical, so that they only need information about the matrix structure. However, HSLLT requires additional information about the geometry of the degrees of freedom. Because of the edge based formulation, the degrees of freedom correspond to the edges of the mesh. An edge will be represented by its midpoint and length.
The three algorithms are compared by the time required for the decomposition of the reordered system matrix. The memory used for storing the triangular sparse decomposition factor is another important aspect. The tests were carried out on an Intel Xeon processor (3.06GHz) and 2GByte of core memory.
In Table 2 the factorisation time and memory requirement of the three exact solvers are compared. Before computing the Cholesky factor, the reordering is done. Because computation times for the reordering are negligible, we only state the time required for the factorisation in Table 2. The columns signed over with Mem(L) include information about the memory needed for storing

		HSLLT		**PLLT**		**TLLT**	
name	N	t_{LL^T}	Mem(L)	t_{LL^T}	Mem(L)	t_{LL^T}	Mem(L)
cube1	26460	12s	154MB	13s	146MB	10.5s	119MB
cube2	45000	36s	332MB	39s	315MB	30s	249MB
cube3	70644	93s	650MB	100s	602MB	74s	477MB
cube4	104544	200s	1128MB	214s	1032MB	155s	812MB
valve	72672	183s	900MB	100s	630MB	80s	500MB

Table 2: Comparison of factorisation time t_{LL^T} and memory for storing the factor Mem(L) for the decomposition methods HSLLT, PLLT and TLLT and different problem sizes.

the lower triangular matrix of the decomposition L.
For the examples listed, one finds comparable calculation times. The main reason lies in the high performance of level 3 BLAS routines, which all methods are using. Thereby calculation speeds of 1GFlop/s are reached. The comparison of required memory shows that HSLLT needs 20-40% higher storage. The reason is inherent in the padding of the sparse structure in order to reach fully populated rows in the sub-diagonal block. For this, more memory due to fill-in than necessary is occupied. The super-nodal approach only combines columns with the same sparsity structure of the factor L without additional storage.
The minimum degree based fill-in reducing ordering was also tested for TLLT and PLLT. We found, that in this case the performance is worse than of HSLLT. This shows the efficiency of nested dissection in comparison to minimum degree strategy. Because this is not the main topic, the table showing this result is not stated here.

8.2 Comparison of approximate decompositions as preconditioner

In Section 5 we introduced two methods providing an approximate decomposition by low-rank approximation. For HLLT we are using interface clustering in order to create the recursive $\mathcal{H}$-matrix structure. The second method HSILLT acts on a block-wise column storage scheme by means of interface clustering. With this approach we benefit from the reduction of fill-in in combination with low-rank approximation.
In order to characterise the performance, we chose an incomplete Cholesky factorisation (ILLT) provided by the Taucs library, which is very fast in case of low accuracy. For this, the sparsity structure of L is reduced by dropping entries so that

$$\widetilde{l_{ij}} = \begin{cases} 0, & a_{ij} = 0 \text{ and } i \neq j \text{ and } l_{ij} < \epsilon_{\text{drop}} ||L_i|| \\ l_{ij}, & \text{otherwise} \end{cases} .$$

Here, a_{ij} is the associated entry of the original matrix, $\varepsilon_{\text{drop}}$ the chosen accuracy and $||L_i||$ the ith row norm of the exact factor.
All three methods HLLT, HSILLT and TILLT provide a preconditioner applicable for iterative methods. The accuracy of such a preconditioner arises in the number of iterations which a preconditioned GMRES method requires to reach a relative residual below 10^{-6}. We will compare the performance of the different methods by factorisation time and memory requirement for the construction of preconditioners with similar accuracy. This means, approximate factorisations yielding the same number of iterations can be compared.

8.2.1 HLLT versus TILLT

In Table 3 we state the performance of both methods for problems of different matrix dimension. Parameter ε of the low-rank approximation differs within $[6 \cdot 10^{-3}, 10^{-3}]$, in order to fix the number of iterations at 30. The maximal block size is 30. The incomplete factorisation TILLT requires the accuracy $\varepsilon_{\text{drop}}$ which is varied within $[3 \cdot 10^{-3}, 6 \cdot 10^{-4}]$.
One can see in Table 3, that TILLT is very fast and needs much less memory than HLLT. A big part of the calculation time in HLLT is used for truncation of blocks in order to reveal the rank after arithmetic operations. Additionally the administration effort of $\mathcal{H}$-matrix operations has to be mentioned. However, the trend for higher dimensions shows that the performance properties approach each other.

		HLLT		**TILLT**	
name	N	t_{LL^T}	Mem(L)	t_{LL^T}	Mem(L)
cube1	26460	20s	54MB	2.9s	22MB
cube2	45000	47s	102MB	7.9s	48MB
cube3	70644	102s	186MB	24s	109MB
cube4	104544	191s	313MB	51s	201MB

Table 3: Performance of HLLT and TILLT under variation of problem size

		HSILLT		**TILLT**	
name	N	t_{LL^T}	Mem(L)	t_{LL^T}	Mem(L)
cube1	26460	2s	20MB	2.9s	22MB
cube2	45000	5s	49MB	7.9s	48MB
cube3	70644	10s	97MB	24s	109MB
cube4	104544	22s	179MB	51s	201MB
cube5	147852	40s	279MB	88s	316MB
cube6	201720	83s	496MB	234s	553MB

Table 4: Performance of HSILLT and TILLT under variation of problem size

8.2.2 HSILLT versus TILLT

The more promising method is given by HSILLT. We fix the number of iterations again at 30. The parameters of the cluster generation now differ from HLLT. We distinguish between maximum cluster sizes of non-interface

and interface leaf clusters. For interface clusters a maximum size of $n_{\min} = 15$ is assumed. In Table 4, the performance is summarised for the cube examples. In Figure 5 one can see that HSILLT is faster than TILLT especially at higher dimensions. Due to low-rank approximation the computational costs of HSILLT increases almost linear with the dimension of the problem.

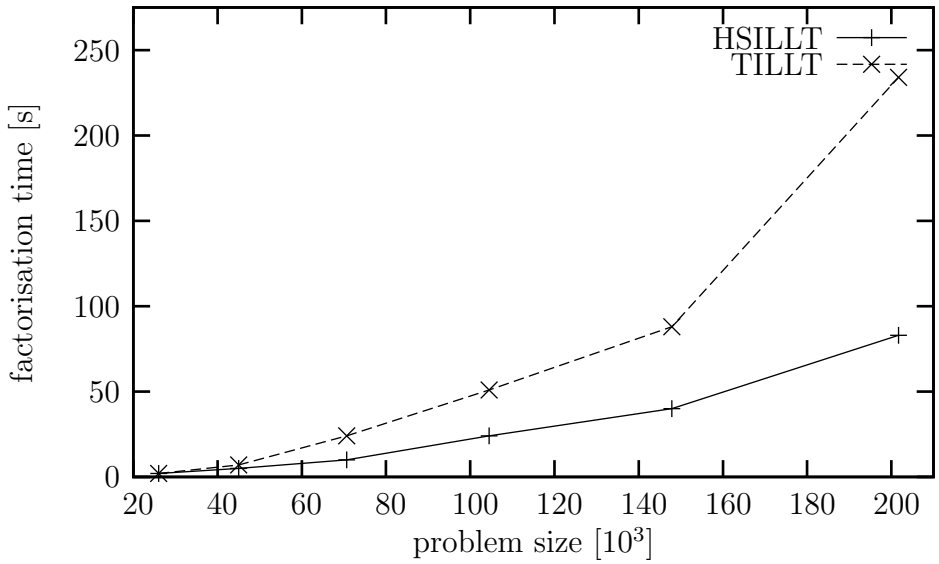

Figure 5: Factorisation time over the different problem sizes for HSILLT and TILLT

Since, the presented algorithm is quite fast for our examples, we consider the magnetic valve problem with a non-uniform geometry. The performance is compared under changing accuracy ϵ. The maximum size of interface clusters is fixed at $n_{min} = 30$. For illustrating this, in Figure 6 a plot of the factorisation time respective the number of iterations is shown. A smaller iteration number indicates a preconditioner with higher accuracy. The comparison between HSILLT and the dashed line of TILLT demonstrates the difference in needed time. Only for high iteration numbers, i.e. low accuracy, TILLT is a little faster.
The memory requirement of HSILLT and TILLT is comparable in the presented examples. For the magnetic valve with the chosen parameters, the memory requirement of HSILLT is higher than that of TILLT. Because of low time requirement some additional considerations for improving the cluster tree can be implemented.
HSILLT is a way creating an approximate (incomplete) factorisation by using

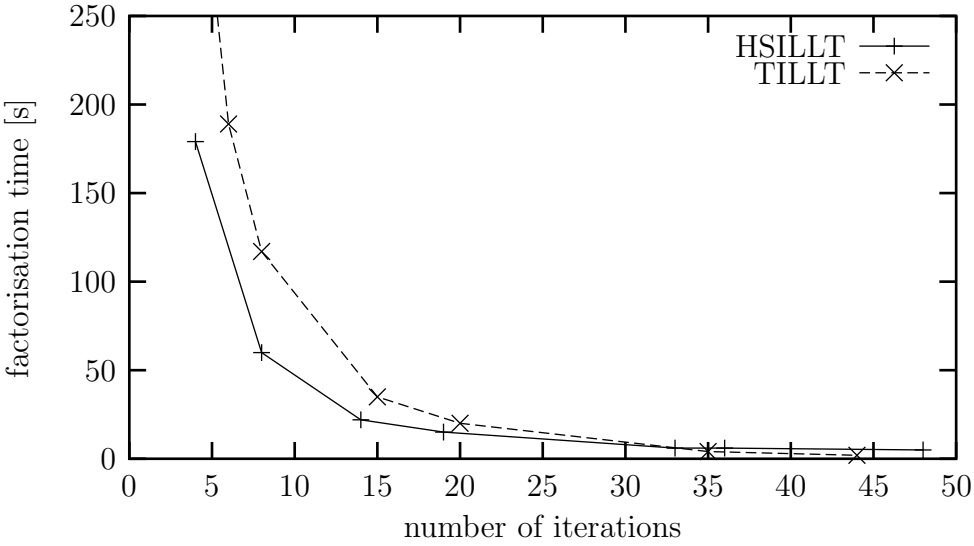

Figure 6: Comparison of factorisation time of HSILLT and TILLT for the magnet valve example

time optimised dense matrix computations. This is the reason of the acceleration in the factorisation time in comparison with TILLT. The low-rank approximation is responsible for the almost linear complexity.

8.3 Conclusion

In this paper we have investigated the solution of linear systems of equations arising from the discretisation of curl-curl-equation. The described regularisation yields a sparse symmetric positive definite system matrix. Under application of hierarchical interface clustering, we analysed an exact decomposition method which is in the stated examples comparable to other solvers. The evaluation of the approximate factorisation HLLT leaded to the result that recursive hierarchical structures do not perform as well as TILLT for the tested examples. The most promising result of our paper is the HSILLT. It allows the calculation of an approximate factorisation by dense matrix computations with almost linear complexity and performs better than the incomplete factorisation provided by Taucs.

References

[1] Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17(4):886–905, 1996.

[2] B. Auchmann, S. Kurz, O. Rain, and S. Russenschuck. Algebraic Properties of BEM-FEM Coupling with Whitney elements. 11th International IGTE Symposium, 2003.

[3] M. Bebendorf. Why approximate *LU* decomposition of finite element discretisations of elliptic operators can be computed with almost linear complexity. *preprint*, 2005.

[4] M. Bebendorf and S. Rjasanow. Adaptive low-rank approximation of collocation matrices. *Computing*, 70(1):1–24, 2003.

[5] A. Bossavit. *Computational Electromagnetism.* Academic Press series in Electromagnetism. Academic Press, 1997.

[6] Alan George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10:345–363, 1973.

[7] Alan George and Joseph W.H. Liu. An automatic nested dissertion algorithm for irregular finite element problems. *SIAM J. Numer. Anal.*, 15:1053–1070, 1978.

[8] Alan George and Joseph W.H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31(1):1–19, 1989.

[9] John R. Gilbert, Gary L. Miller, and Shang-Hua Teng. Geometric mesh partitioning: Implementation and experiments. *SIAM J. Sci. Comput.*, 19(6):2091–2110, 1998.

[10] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[11] W. Hackbusch and B. N. Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic. II. Application to multi-dimensional problems. *Computing*, 64(1):21–47, 2000.

[12] E. G. Ng and B. W. Peyton. Block sparse cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comp.*, 14:1034–1056, 1993.

[13] Sergio Pissanetsky. *Sparse Matrix Technology.* Academic Press, London, 1984.

[14] Y. Saad. *Iterative methods for Sparse Linear Systems.* PWS Publishing, Boston, 1996.

Authors

Ilgis Ibragimov
Department of Mathematics
Saarland University
Postfach 15 11 50
66041 Saarbrücken
Germany
ilgis@num.uni-sb.de

Sergej Rjasanow
Department of Mathematics
Saarland University
Postfach 15 11 50
66041 Saarbrücken
Germany
rjasanow@num.uni-sb.de

Katharina Straube
Robert Bosch GmbH
Postfach 10 60 50
70049 Stuttgart
Germany
katharina.straube@de.bosch.com