Parallel Incomplete Cholesky Preconditioners based on the Non-Overlapping Data Distribution

G. Haase [a]

[a] *Johannes Kepler University of Linz, Inst. of Math., Altenberger Str. 69, A–4040 Linz, Austria*

Abstract

The paper analyses various parallel incomplete factorizations based on the non-overlapping domain decomposition. The general framework is applied to the investigation of the preconditioning step in cg-like methods. Under certain conditions imposed on the finite element mesh, all matrix and vector types given by the special data distribution can be used in the matrix-by-vector multiplications. Not only the well-known domain decomposition preconditioners fit into the concept but also parallelized global incomplete factorizations are feasible. Additionally, those global incomplete factorizations can be used as smoothers in parallel multigrid methods. Numerical results on a parallel machine with distributed memory are presented.

Keywords : Parallel Iterative Solvers, Incomplete Factorization, Preconditioning, Domain Decomposition, Distributed Memory, Finite Element Method.

1 Introduction

A lot of parallel algorithms have been developed for matrix factorizations based on disjoint data distributions, e.g. [4,13,22]. On the other hand, matrix factorizations in case of a non-overlapping, but not disjoint, data distribution are only investigated in [17,18] for special decompositions and a special numbering of the data. In contrast to the work by Notay [17] I use arbitrary domains and unstructured grids.

The non-overlapping Domain Decomposition (DD) data distribution follows closely the ideas of Law [14] also used in connection with preconditioning in [16,9,8]. As technical background a parallel machine with distributed memory is assumed. We will discuss the DD data distribution in the 2D case in detail. The handling of the 3D case is similar.

Let us consider the abstract symmetric, $\mathbb{V}_0$–elliptic and $\mathbb{V}_0$–bounded variational problem

$$\text{Find } \; u \in \mathbb{V}_0 \; : \; a(u,v) \; = < F, v > \; \forall v \in \mathbb{V}_0 \;\; , \tag{1}$$

arising from the weak formulation of a scalar second–order, symmetric and uniformly bounded elliptic boundary value problem (b.v.p.) given in a plane bounded domain $\Omega \subset \mathbb{R}^2$ with a piecewise smooth boundary $\Gamma = \partial\Omega$. Defining the usual linear finite element (f.e.) nodal basis, the f.e. isomorphisms results in a large-scale sparse system

$$K \, \underline{u} \; = \; \underline{f} \tag{2}$$

of finite element equations with the symmetric and positive definite stiffness matrix K. For simplicity, we assume triangular elements in 2D and tetrahedral elements in 3D.

In Sect. 2.1, we introduce the data decomposition with distributed and accumulated vectors according to [14,16,9,8]. In contrast to the known literature, in Sect. 2.2 also matrix-by-vector multiplications with accumulated matrices are under consideration. Those multiplications are only admissible under certain conditions on the f.e. mesh. An abstract framework on that topic can be found in [5]. Block triangular matrices play a special role in the determination of admissible vector types in the multiplications. Therefore, we also investigate the multiplication by factorized matrices where the factorization has to preserve the given matrix pattern.

In Sect. 3, we investigate types of preconditioners of the preconditioning step in cg-like methods in general and distinguish between distributed and accumulated type. It turns out that the DD preconditioners [1,9] are of accumulation type. Now, due to the mesh requirements in Sect. 2, global Incomplete Cholesky (IC) factorization preconditioners are also feasible which are examined in Sect. 4 in detail. To omit the restrictions on the mesh we focus our interest also on IC factorizations with a reduced pattern compared to the original matrix.

Section 5 presents some numerical experiences with the parallelized cg iteration using an IC factorization as preconditioner.

2 The DD Data Distribution and the Parallelized CG

2.1 The DD Data Distribution

First, we subdivide our 2d-domain Ω into p subdomains $\overline{\Omega}_i \ (i = 1, 2, \ldots, p)$, and discretize the subdomains with linear triangular elements. The triangulation is supposed to be conforming in the whole domain Ω. Figure 1 illustrates this for a unit square subdivided into 4 congruent squares.

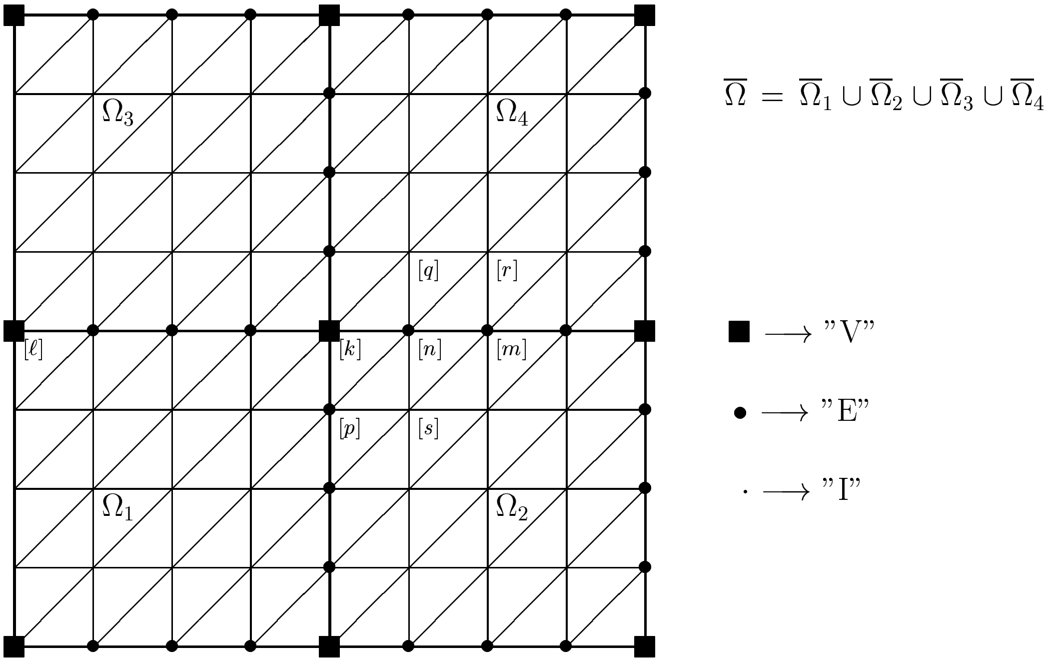


Fig. 1. Domain decomposition and triangularization with node numbers.

Let us introduce 3 classes of nodes: the inner nodes further denoted by the subscript "I", the nodes on the interior of an interface edge (edge nodes) denoted by "E" and the vertex nodes (cross points) "V". If no distinction is necessary, then we call the latter ones coupling nodes on the interfaces indicated by "C". For convenience in the presentation we number first the vertices, then the edge nodes (subsequently on each edge) and last the inner nodes in the subdomains $\Omega_1 \ldots, \Omega_p$ in all vectors and matrices.

According to the p subdomains $\overline{\Omega}_i \ (i = 1, 2, \ldots, p)$, we distribute all matrices and vectors to the p processors $P_i \ (i = 1, 2, \ldots, p)$ of the parallel machine and define A_i as the connectivity matrix of $\overline{\Omega}_i$, i.e. the matrix A_i is the boolean matrix of dimension $N_i \times N$ mapping a global vector $\underline{g} \in \mathbb{R}^N$, into a local vector $\underline{g}_i \in \mathbb{R}^{N_i}$.

Now we will define two types of vectors, the accumulated (type I) and the distributed (type II) vector :

type I : $\underline{\mathfrak{u}}$ and $\underline{\mathfrak{w}}$ are stored in processor P_i ($\hat{=} \overline{\Omega}_i$) as $\underline{\mathfrak{u}}_i = A_i \underline{\mathfrak{u}}$ and $\underline{\mathfrak{w}}_i = A_i \underline{\mathfrak{w}}$, i.e. in each subdomain each node on the interface possesses the full value of its vector component.

type II : $\underline{\mathsf{r}}, \underline{\mathsf{f}}$ are stored in P_i as $\underline{\mathsf{r}}_i, \underline{\mathsf{f}}_i$, so that $\underline{\mathsf{r}} = \sum_{i=1}^{p} A_i^T \underline{\mathsf{r}}_i$ etc. is valid, i.e. the nodes on the interfaces store just a part of the real value.

The matrix K from (2) will be stored in a distributive sense similar to type-II vector so we call it a type-II matrix

$$\mathsf{K} = \sum_{i=1}^{p} A_i^T \mathsf{K}_i A_i \quad , \tag{3}$$

wherein K_i is the stiffness matrix belonging to the subdomain $\overline{\Omega}_i$. Thinking of $\overline{\Omega}_i$ as a finite element, this is exactly the way of constructing f.e. stiffness matrices, therefore the distributed storing of matrix K is the natural one in a f.e. sense. The ordering of the nodes given above and the types of the vectors lead to the following representation of equation (2) :

$$\begin{pmatrix} \mathsf{K}_V & \mathsf{K}_{VE} & \mathsf{K}_{VI} \\ \mathsf{K}_{EV} & \mathsf{K}_E & \mathsf{K}_{EI} \\ \mathsf{K}_{IV} & \mathsf{K}_{IE} & \mathsf{K}_I \end{pmatrix} \cdot \begin{pmatrix} \underline{\mathfrak{u}}_V \\ \underline{\mathfrak{u}}_E \\ \underline{\mathfrak{u}}_I \end{pmatrix} = \begin{pmatrix} \underline{\mathsf{f}}_V \\ \underline{\mathsf{f}}_E \\ \underline{\mathsf{f}}_I \end{pmatrix} . \tag{4}$$

Here K_I is a block diagonal matrix with the entries $\mathsf{K}_{I,i}$. Also K_{IE}, K_{EI}, K_{IV}, K_{VI} are block matrices. If a global accumulation of the matrix K is performed, then we will denote that type-I matrix by $\mathfrak{M}$, and write

$$\mathfrak{M}_i \; = \; A_i \mathfrak{M} A_i^T \quad . \tag{5}$$

Although $\mathsf{K} \equiv \mathfrak{M}$, we have to distinguish between them because of the different local storing. The matrix

$$R \; = \; \sum_{i=1}^{p} A_i^T A_i \tag{6}$$

is a diagonal matrix containing for each node the number of subdomains it belongs to (e.g. according to Fig. 1, $R^{[k]} = 4$, $R^{[n]} = R^{[m]} = R^{[p]} = 2$, $R^{[q]} = 1$).

In the remaining paper we use sub- and superscripts in the following way. $\mathsf{r}_{C,i}^{[n]}$ denotes the n-th component (local or global ordering) of a vector $\underline{\mathsf{r}}$ stored on processor i. The subscript "C" indicates that this part of the vector belongs to the coupling nodes. A similar notation is used for the matrices.

2.2 Basic Operations

The description of the operations represents the results given in [16,9,5]. Especially [5] includes a complete overview to this topic concerning the data distribution given above. Here, we will present just a brief introduction.

2.2.1 Vector Type Changing

It is quite clear that vector addition/subtraction can be done without any communication when all the vectors are of the same type. The change from type-II to a type-I vector, i.e., an accumulation of a vector, requires communication :

$$\underline{\mathfrak{w}}_i \;=\; A_i \sum_{i=1}^{p} A_i^T \underline{\mathfrak{r}}_i \;\; . \tag{7}$$

Changing a type-I vector into a type-II is not unique. One possibility is to divide the value locally by the number of neighbours, i.e.

$$\underline{\mathfrak{r}}_i \;=\; R_i^{-1} \underline{\mathfrak{w}}_i \tag{8}$$

where $R_i = A_i R A_i^T$ denotes those positions of R, defined in (6), which correspond to the proper subvector.

2.2.2 Inner Product

The inner product of different type vectors can be done by summing up the inner products for each region :

$$(\underline{\mathfrak{w}}, \underline{\mathfrak{r}}) \;=\; \underline{\mathfrak{w}}^T \underline{\mathfrak{r}} \;=\; \underline{\mathfrak{w}}^T \sum_{i=1}^{p} A_i^T \underline{\mathfrak{r}}_i \;=\; \sum_{i=1}^{p} (A_i \underline{\mathfrak{w}})^T \underline{\mathfrak{r}}_i \;=\; \sum_{i=1}^{p} (\underline{\mathfrak{w}}_i, \underline{\mathfrak{r}}_i) \tag{9}$$

Any other combination of vectors requires the proper type conversion, occasionally with additional communication.

2.2.3 Matrix-by-Vector Multiplication

In the following, we briefly investigate the matrix-by-vector multiplication with respect to the different matrix and vector types. A detailed description can be found in [5].

(1) Type-II matrix × type-I vector results in a type-II vector.
Indeed, using definition (3) we get :

$$\mathsf{K} \cdot \underline{\mathfrak{w}} \;=\; \sum_{i=1}^{p} A_i^T \mathsf{K}_i A_i \cdot \underline{\mathfrak{w}} \;=\; \sum_{i=1}^{p} A_i^T \underbrace{\mathsf{K}_i \cdot \underline{\mathfrak{w}}_i}_{\underline{\mathfrak{r}}_i} \;=\; \underline{\mathfrak{r}} \tag{10}$$

A following accumulation (7) results in a type-I vector.

(2) Type-II matrix × type-II vector needs a conversion of the vector before the previous multiplication can be applied.

(3) Type-I matrix × type-I vector cannot be done with general type-I matrices $\mathfrak{M}$. Let us have a look at the node n in Fig. 1 and the operation $\underline{\mathfrak{u}} = \mathfrak{M} \cdot \underline{\mathfrak{w}}$. The local multiplication $\underline{\mathfrak{u}}_i = \mathfrak{M}_i \cdot \underline{\mathfrak{w}}_i$ shall result in a type-I vector $\underline{\mathfrak{u}}$ in node n

$$\begin{aligned}
\mathfrak{u}_2^{[n]} &= \mathfrak{M}_2^{[n,n]}\mathfrak{w}_2^{[n]} + \mathfrak{M}_2^{[n,m]}\mathfrak{w}_2^{[m]} + \mathfrak{M}_2^{[n,k]}\mathfrak{w}_2^{[k]} + \mathfrak{M}_2^{[n,p]}\mathfrak{w}_2^{[p]} + \mathfrak{M}_2^{[n,s]}\mathfrak{w}_2^{[s]} \\
\mathfrak{u}_4^{[n]} &= \mathfrak{M}_4^{[n,n]}\mathfrak{w}_4^{[n]} + \mathfrak{M}_4^{[n,m]}\mathfrak{w}_4^{[m]} + \mathfrak{M}_4^{[n,k]}\mathfrak{w}_4^{[k]} + \mathfrak{M}_4^{[n,q]}\mathfrak{w}_4^{[q]} + \mathfrak{M}_4^{[n,r]}\mathfrak{w}_4^{[r]}
\end{aligned}$$

In the equations above, the terms 4 and 5 of the right hand sides differ from each other so that processors 2 and 4 achieve different results instead of a unique one. The reason is the transport of information by the matrix from a processor a to a node additionally belonging to processor b.
Denote by $\omega_i = \{s : x_i \in \overline{\Omega}_s\}$ the processors a node i belongs to, e.g. in Fig. 1, $\omega_n = \{2,4\}$, $\omega_p = \{1,2\}$, $\omega_s = \{2\}$. Then, the transport of information from node j to a node i is only admissible if $\omega_i \subseteq \omega_j$. Representing the matrix entries as directed graph then, e.g., in Fig. 1 the entries $q \to k$, $q \to n$, $s \to n$, $s \to m$, $s \to p$, $p \to k$, $n \to k$, $p \to n$, $n \to p$, $\ell \to k$, $k \to \ell$ are not admissible. Thus, just a matrix of the shape

$$\mathfrak{M} \;=\; \begin{pmatrix} \mathfrak{M}_V & 0 & 0 \\ \mathfrak{M}_{EV} & \mathfrak{M}_E & 0 \\ \mathfrak{M}_{IV} & \mathfrak{M}_{IE} & \mathfrak{M}_I \end{pmatrix} \quad\Longrightarrow\quad \underline{\mathfrak{u}} = \mathfrak{M} \cdot \underline{\mathfrak{w}} \tag{11}$$

with block-diagonal matrices $\mathfrak{M}_I$, $\mathfrak{M}_E$, $\mathfrak{M}_V$ omits such forbidden entries. Whereas the block-diagonality of $\mathfrak{M}_I$ and the block structure of $\mathfrak{M}_{IE}$, $\mathfrak{M}_{IV}$ is guaranteed by the data decomposition, for the remaining three matrices the mesh has to fulfill the requirements

(a) No connection between vertices belonging to different sets of subdomains. This guarantees the block-diagonality of $\mathfrak{M}_V$ (often $\mathfrak{M}_V$ is a diagonal matrix). No connection between vertices and opposite edges, e.g. in Fig. 1, $\ell \not\to p$. This ensures the admissible structure of $\mathfrak{M}_{EV}$ (and $\mathfrak{M}_{VE}$).

(b) No connection between edge nodes belonging to different sets of subdomains. This guarantees the block-diagonality of $\mathfrak{M}_E$.

Requirement (a) can be easily fulfilled if, at least, one node is located on all edges between two vertices and one node lies in the interior of each subdomain. A modification of the given mesh generator can also guarantee requirement 3b, so that, e.g., the edge between nodes p and n is omitted. Figure 2 presents the revised mesh.

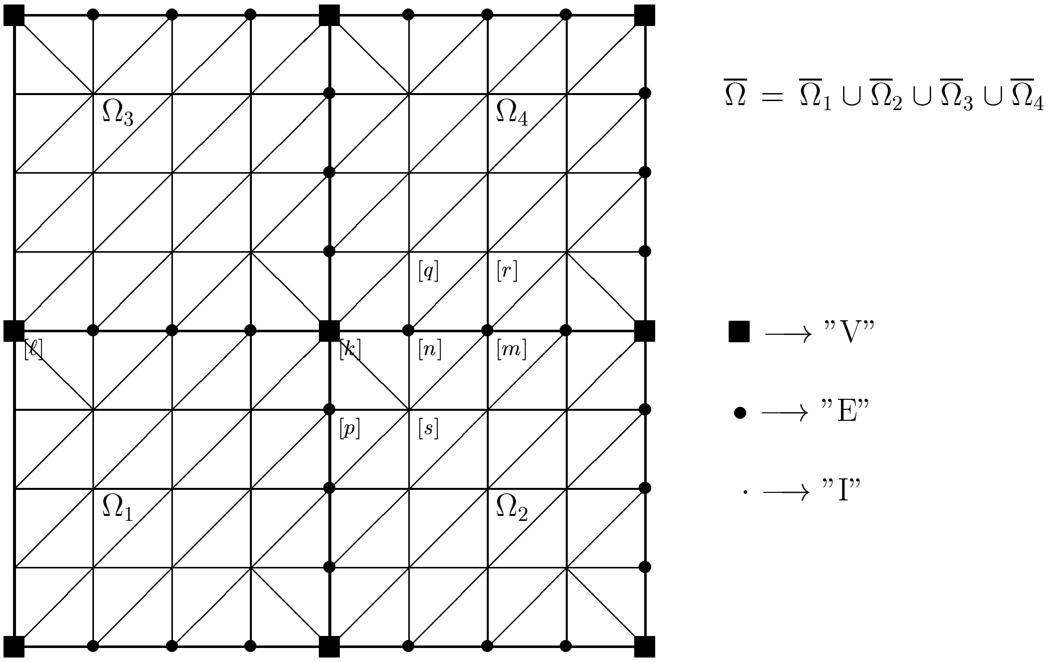


Fig. 2. Domain decomposition and a modified triangularization .

(4) As in the previous point type-I matrix × type-II vector cannot be done with general type-I matrices $\mathfrak{M}$. The result should be a type-II vector. Assuming that the necessary requirements (a) and (b) are fulfilled by the mesh (see Fig. 2) we can focus our interest to the operation $\underline{\mathsf{f}}_I = \mathsf{M}_{IC} \cdot \underline{\mathsf{r}}_C$. Performing this operation locally on processor 4 gives

$$\mathsf{f}_4^{[r]} = \mathfrak{M}_{IC,4}^{[r,n]} \cdot \mathsf{r}_4^{[n]} + \mathfrak{M}_{IC,4}^{[r,m]} \cdot \mathsf{r}_4^{[m]} .$$

Due to the type-II vector property, we miss in the result the entries $\mathfrak{M}_{IC,2}^{[r,n]} \cdot \mathsf{r}_2^{[n]}$ and $\mathfrak{M}_{IC,2}^{[r,m]} \cdot \mathsf{r}_2^{[m]}$. The difference with the previous point is that, now the transport of information from node j to a node i is permitted only if $\omega_i \supseteq \omega_j$. So, just a matrix of the shape

$$\mathfrak{M} = \begin{pmatrix} \mathfrak{M}_V & \mathfrak{M}_{VE} & \mathfrak{M}_{VI} \\ 0 & \mathfrak{M}_E & \mathfrak{M}_{EI} \\ 0 & 0 & \mathfrak{M}_I \end{pmatrix} \quad \Longrightarrow \quad \underline{\mathsf{f}} = \sum_{i=1}^{P} A_i^T \underline{\mathsf{f}}_i = \sum_{i=1}^{P} A_i^T \left(\mathfrak{M}_i \underline{\mathsf{r}}_i\right) \quad (12)$$

with block-diagonal matrices $\mathfrak{M}_I$, $\mathfrak{M}_E$, $\mathfrak{M}_V$ can be used in this type of matrix-by-vector multiplication.

Remark 1 *The above points 3 and 4 can be combined under the assumption that $\mathfrak{M}_V$, $\mathfrak{M}_E$, $\mathfrak{M}_I$ are block-diagonal matrices (guaranteed by the mesh in Fig. 2). Denote with $\mathfrak{M}_L$, $\mathfrak{M}_U$ and $\mathfrak{M}_D$ the strictly lower, upper and diagonal part of $\mathfrak{M}$. Then, we can perform the type I matrix-by-vector multiplication for all types of vectors*

$$\begin{aligned}
\underline{\mathfrak{w}} &= \mathfrak{M} \cdot \underline{\mathfrak{u}} := (\mathfrak{M}_L + \mathfrak{M}_D) \cdot \underline{\mathfrak{u}} + \sum_{i=1}^{P} A_i^T \mathfrak{M}_{U,i} R_i^{-1} \cdot \underline{\mathfrak{u}}_i && \text{(13a)} \\
\underline{\mathfrak{w}} &= \mathfrak{M} \cdot \underline{\mathfrak{r}} := (\mathfrak{M}_L + \mathfrak{M}_D) \sum_{i=1}^{P} A_i^T \cdot \underline{\mathfrak{r}}_i + \sum_{i=1}^{P} A_i^T \mathfrak{M}_{U,i} \cdot \underline{\mathfrak{r}}_i && \text{(13b)} \\
\underline{\mathfrak{f}} &= \mathfrak{M} \cdot \underline{\mathfrak{u}} := R^{-1}(\mathfrak{M}_L + \mathfrak{M}_D) \cdot \underline{\mathfrak{u}} + \mathfrak{M}_U R^{-1} \cdot \underline{\mathfrak{u}} && \text{(13c)} \\
\underline{\mathfrak{f}} &= \mathfrak{M} \cdot \underline{\mathfrak{r}} := R^{-1}(\mathfrak{M}_L + \mathfrak{M}_D) \sum_{i=1}^{P} A_i^T \cdot \underline{\mathfrak{r}}_i + \mathfrak{M}_U \cdot \underline{\mathfrak{r}} \; . && \text{(13d)}
\end{aligned}$$

Note, that in any case the above multiplications require two type conversions, but the choice of the vectors influences directly the amount of communication needed.

If the Matrix $\mathfrak{M}$ has been factorized as $M = \mathfrak{L}^{-1}\mathfrak{U}^{-1}$ with the lower and an upper triangular matrices $\mathfrak{L}^{-1}$ and $\mathfrak{U}^{-1}$ we write

$$\begin{aligned}
\underline{\mathfrak{w}} &= \mathfrak{L}^{-1}\mathfrak{U}^{-1} \cdot \underline{\mathfrak{r}} := \mathfrak{L}^{-1} \sum_{i=1}^{P} A_i^T \mathfrak{U}_i^{-1} \cdot \underline{\mathfrak{r}}_i && \text{(14a)} \\
\underline{\mathfrak{f}} &= \mathfrak{L}^{-1}\mathfrak{U}^{-1} \cdot \underline{\mathfrak{u}} := R^{-1}\mathfrak{L}^{-1} \sum_{i=1}^{P} A_i^T \mathfrak{U}_i^{-1} R^{-1} \cdot \underline{\mathfrak{u}}_i \; . && \text{(14b)}
\end{aligned}$$

The second case, i.e., $M = \mathfrak{U}^{-1}\mathfrak{L}^{-1}$ results in

$$\begin{aligned}
\underline{\mathfrak{w}} &= \mathfrak{U}^{-1}\mathfrak{L}^{-1} \cdot \underline{\mathfrak{r}} := \sum_{i=1}^{P} A_i^T \mathfrak{U}_i^{-1} R_i^{-1} A_i \mathfrak{L}^{-1} \cdot \big(\sum_{j=1}^{P} A_j^T \underline{\mathfrak{r}}_j\big) && \text{(15a)} \\
\underline{\mathfrak{f}} &= \mathfrak{U}^{-1}\mathfrak{L}^{-1} \cdot \underline{\mathfrak{u}} := \mathfrak{U}^{-1} R^{-1} \mathfrak{L}^{-1} \cdot \underline{\mathfrak{u}} \; . && \text{(15b)}
\end{aligned}$$

The equations for the remaining combinations of vector types in (14) and (15) can be easily derived by type conversion. Here, the number and the kind of type conversion is also determined by the way of factoring the matrix $\mathfrak{M}$.

2.3 The Parallelized CG

For completeness of the presentation, the sequential and the parallelized cg algorithm using the data distribution and the relations of the previous Sections 2.1 and 2.2 is given in Fig. 3. Communication is only necessary in the type conversion indicated by $\sum_{j=1}^{P}$ and in the inner products $(\cdot,\cdot)$.

sequential (preconditioned)	parallel
Choose $\underline{u}^0$	Choose $\underline{\mathfrak{u}}^0$
$\underline{r} := \underline{f} - K\cdot\underline{u}^0$	$\underline{\mathsf{r}} := \underline{\mathsf{f}} - \mathsf{K}\cdot\underline{\mathfrak{u}}^0$
$\underline{w} := \underline{r} \qquad (\underline{w} := C^{-1}\cdot\underline{r})$	$\underline{\mathfrak{w}} := \sum\limits_{j=1}^{P} A_j^T\underline{\mathsf{r}}_j$
$\underline{s} := \underline{w}$	$\underline{\mathfrak{s}} := \underline{\mathfrak{w}}$
$\sigma := \sigma_{\text{old}} := \sigma_0 := (\underline{w},\underline{r})$	$\sigma := \sigma_{\text{old}} := \sigma_0 := (\underline{\mathfrak{w}},\underline{\mathsf{r}})$
repeat	repeat
$\quad\underline{v} := K\cdot\underline{s}$	$\quad\underline{\mathsf{v}} := \mathsf{K}\cdot\underline{\mathfrak{s}}$
$\quad\alpha := \sigma/(\underline{s},\underline{v})$	$\quad\alpha := \sigma/(\underline{\mathfrak{s}},\underline{\mathsf{v}})$
$\quad\underline{u} := \underline{u} + \alpha\cdot\underline{s}$	$\quad\underline{\mathfrak{u}} := \underline{\mathfrak{u}} + \alpha\cdot\underline{\mathfrak{s}}$
$\quad\underline{r} := \underline{r} - \alpha\cdot\underline{v}$	$\quad\underline{\mathsf{r}} := \underline{\mathsf{r}} - \alpha\cdot\underline{\mathsf{v}}$
$\quad\underline{w} := \underline{r} \qquad (\underline{w} := C^{-1}\cdot\underline{r})$	$\quad\underline{\mathfrak{w}} := \sum\limits_{j=1}^{P} A_j^T\underline{\mathsf{r}}_j$
$\quad\sigma := (\underline{w},\underline{r})$	$\quad\sigma := (\underline{\mathfrak{w}},\underline{\mathsf{r}})$
$\quad\beta := \sigma/\sigma_{\text{old}}$	$\quad\beta := \sigma/\sigma_{\text{old}}$
$\quad\sigma_{\text{old}} := \sigma$	$\quad\sigma_{\text{old}} := \sigma$
$\quad\underline{s} := \underline{w} + \beta\cdot\underline{s}$	$\quad\underline{\mathfrak{s}} := \underline{\mathfrak{w}} + \beta\cdot\underline{\mathfrak{s}}$
until $\sqrt{\sigma/\sigma_0} <$ accuracy	until $\sqrt{\sigma/\sigma_0} <$ accuracy

Fig. 3. cg for solving Eq. (2)

3 Preconditioners for the Iteration Form

The type conversion $\underline{\mathfrak{w}} = \sum_{j=1}^{P} A_j^T\underline{\mathsf{r}}_j$ in the parallel cg (Fig. 3) is just the type-I identity and equivalent to equation (13b) with the components $\mathfrak{M}_D = \mathfrak{I}$ and $\mathfrak{M}_L = \mathfrak{M}_U = 0$. Thus, we are looking for parallel preconditioners C not requiring much more communication when $\underline{\mathfrak{w}} := C^{-1}\cdot\underline{\mathsf{r}}$ replaces the above type conversion in the parallel cg of Fig. 3.

3.1 Preconditioners of accumulated matrix type

Now we are in the position to choose a preconditioner of accumulated (type-I) matrix type from the equations (13b), (14a) and (15a). We focus our interest on the latter ones, i.e. just on preconditioners written in the factorized form.

Remark 2 *In the symmetric case, the well known ASM-DD-preconditioner [9] using Schur complement preconditioners proposed in [1,20] fits exactly in this scheme presented by equation (14a), although the part for preconditioning the vertex nodes is not a diagonal matrix. The reason is the usual sequential and global handling of the related system of equations.*

3.2 Preconditioners of distributed matrix type

Applying a distributed (type-II) preconditioning matrix $\mathsf{C}^{-1} = \sum_{i=1}^{p} A_i^T \mathsf{C}_i^{-1} A_i$, we need to convert the vectors twice (see Section 2.2). Therefore, the precondition step looks like

$$\underline{\mathfrak{w}} \;=\; \mathsf{C}^{-1} \sum_{j=1}^{P} A_j^T \underline{\mathsf{r}}_j \;:=\; \sum_{i=1}^{P} A_i^T \left(\mathsf{C}_i^{-1} \cdot A_i \sum_{j=1}^{P} A_j^T \underline{\mathsf{r}}_j \right) \quad . \tag{16}$$

The matrix K is not accumulated and so each K_i represents a discretized differential operator with the proper boundary conditions on $\partial\Omega_i$. If $\partial\Omega_i \cap \Gamma_D = \emptyset$ the matrix K_i may become singular. Therefore, we need not to choose C_i in the same way. One possibility is to assemble the stiffness matrix K and set $\mathsf{C}_i = \mathfrak{K}_i$ so that the local preconditioner represents a 2nd order pde in $\widetilde{\Omega}_i$ with homogeneous Dirichlet B.C., where $\widetilde{\Omega}_i$ is the domain Ω_i expanded by the next layer of nodes (elements) over the inner boundary. We end up with an additive Schwarz overlapping preconditioner which is out of scope of this work, for further investigations see [19].

4 Incomplete Cholesky Factorization Preconditioners

The fill-in appearing in the Cholesky factorization of the given accumulated stiffness matrix $\mathfrak{K}$ does not preserve the matrix structure necessary for applying the factorized matrix in a matrix-by-vector product. So we concern with the classic incomplete Cholesky factorization [15] preserving the given pattern of the matrix and take advantage of the block structure of matrix K (4). The incomplete factorization leads to preconditioners of the shape

$\mathfrak{C}^{-1} = \mathfrak{L}^{-1} \cdot \mathfrak{L}^{-T}$ (which is exactly the same matrix structure as in equation (14a)) or $\mathfrak{C}^{-1} = \mathfrak{U}^{-1} \cdot \mathfrak{U}^{-T}$ (15a).
To emphasize the essential points in the following, we will use the different fonts for the types I and II only for those vector and matrix components with subscripts "V" or "E". All remaining parts are written in the standard mathematical font, e.g. $\underline{\mathfrak{r}} = (\underline{\mathfrak{r}}_V, \underline{\mathfrak{r}}_E, \underline{r}_I)^T$. The defect matrix P in the following factorization algorithms contains the difference between the exact factorization and the incomplete one. This matrix will never be calculated.

4.1 The ILL^T-Factorization

To achieve a preconditioner $\mathfrak{C}^{-1} = \mathfrak{L}^{-T} \cdot \mathfrak{L}^{-1}$ applicable in the preconditioning step in Fig. 3) we have to perform an incomplete LL^T-factorization, see Fig. 4. Previous to the factorization we have to change the type of the matrix factorized.

Start	$\mathfrak{K} = \sum\limits_{i=1}^{P} A_i^T \mathsf{K}_i A_i$	
Determine		Why no communication ?
$\mathfrak{L}_V$	$\mathfrak{K}_V = \mathfrak{L}_V \cdot \mathfrak{L}_V^T - P_V$	mesh $\to$ diagonal matrix
$\mathfrak{L}_{EV}$	$\mathfrak{K}_{EV} = \mathfrak{L}_{EV} \cdot \mathfrak{L}_V^T - P_{EV}$	DD, mesh $\to$ block matrices
L_{IV}	$K_{IV} = L_{IV} \cdot \mathfrak{L}_V^T - P_{IV}$	DD, mesh $\to$ block matrices
Modify	$\mathfrak{K}_E := \mathfrak{K}_E - \mathfrak{L}_{EV} \cdot \mathfrak{L}_{EV}^T$	same matrix type
	$K_{EI} := K_{EI} - \mathfrak{L}_{EV} \cdot L_{IV}^T$	same matrix type
	$K_I := K_I - L_{IV} \cdot L_{IV}^T$	same matrix type
Determine		
$\mathfrak{L}_E$	$\mathfrak{K}_E = \mathfrak{L}_E \cdot \mathfrak{L}_E^T - P_E$	DD, mesh $\to$ block matrices
L_{IE}	$K_{IE} = L_{IE} \cdot \mathfrak{L}_E^T - P_{IE}$	DD $\to$ block matrices
Modify	$K_I := K_I - L_{IE} \cdot L_{IE}^T$	same matrix type
Determine		
L_I	$K_I = L_I \cdot L_I^T - P_I$	DD $\to$ block matrices

Fig. 4. Parallelized LL^T-factorization

According to (15a), the application of $\mathfrak{C}^{-1}$ requires two communications in the three vector type conversions. Figure 5 presents the proper parallel algorithm :

I)	$\underline{\mathfrak{r}} := \sum_{i=1}^{P} A_i^T \underline{\mathsf{r}}_i$		
II)	$\underline{\mathfrak{u}}_V := \mathfrak{L}_V^{-1} \underline{\mathfrak{r}}_V$	IV)	$\underline{w}_I := L_I^{-T} \underline{u}_I$
	$\underline{\mathfrak{u}}_E := \mathfrak{L}_E^{-1} (\underline{\mathfrak{r}}_E - \mathfrak{L}_{EV} \underline{\mathfrak{u}}_V)$		$\underline{\mathsf{w}}_E := \mathfrak{L}_E^{-T} (\underline{\mathsf{u}}_E - L_{IE}^T \underline{w}_I)$
	$\underline{u}_I := L_I^{-1} (\underline{r}_I - L_{IE} \underline{\mathfrak{u}}_E - L_{IV} \underline{\mathfrak{u}}_V)$		$\underline{\mathsf{w}}_V := \mathfrak{L}_V^{-T} (\underline{\mathsf{u}}_V - \mathfrak{L}_{EV}^T \underline{\mathsf{w}}_E - L_{IV}^T \underline{w}_I)$
III)	$\underline{\mathsf{u}} := R^{-1} \underline{\mathfrak{u}}$	V)	$\underline{\mathfrak{w}} := \sum_{i=1}^{P} A_i^T \underline{\mathsf{w}}_i$

Fig. 5. ILL^T Preconditioning $\mathfrak{L} \cdot \mathfrak{L}^T \cdot \underline{\mathfrak{w}} = \underline{\mathsf{r}}$

4.2 The IUU^T-Factorization

To achieve a preconditioner $\mathfrak{C}^{-1} = \mathfrak{U}^{-T} \cdot \mathfrak{U}^{-1}$ applicable in the preconditioning step in Fig. 3) we have to perform an incomplete UU^T-factorization, see Fig. 6. During the factorization we have to change the type of the matrix factorized.

Start	K	
Determine		Why no communication ?
U_I	$K_I = U_I \cdot U_I^T - P_I$	DD $\to$ blocks matrices
U_{EI}	$K_{EI} = U_{EI} \cdot U_I^T - P_{EI}$	DD $\to$ blocks matrices
U_{VI}	$K_{VI} = U_{VI} \cdot U_I^T - P_{VI}$	DD $\to$ blocks matrices
Modify	$\mathsf{K}_E := \mathsf{K}_E - U_{EI} \cdot U_{EI}^T$	same matrix type
	$\mathsf{K}_{VE} := \mathsf{K}_{VE} - U_{VI} \cdot U_{EI}^T$	same matrix type
	$\mathsf{K}_V := \mathsf{K}_V - U_{VI} \cdot U_{VI}^T$	same matrix type
Accumulate	$\mathfrak{K}_E$, $\mathfrak{K}_{VE}$	
e.g.	$\mathfrak{K}_{EV} := \sum_{i=1}^{P} A_{E,i}^T \mathsf{K}_{EV,i} A_{V,i}$	—
Determine		
$\mathfrak{U}_E$	$\mathfrak{K}_E = \mathfrak{U}_E \cdot \mathfrak{U}_E^T - P_E$	DD, mesh $\to$ blocks matrices
$\mathfrak{U}_{VE}$	$\mathfrak{K}_{VE} = \mathfrak{U}_{VE} \cdot \mathfrak{U}_E^T - P_{VE}$	DD, mesh $\to$ blocks matrices
Modify	$\mathsf{K}_V := \mathsf{K}_V - \mathfrak{U}_{VE} \cdot R_E^{-1} \cdot \mathfrak{U}_{VE}^T$	same matrix type (15a)
Accumulate	$\mathfrak{K}_V$	—
Determine		
$\mathfrak{U}_V$	$\mathfrak{K}_V = \mathfrak{U}_V \cdot \mathfrak{U}_V^T - P_V$	mesh $\to$ diagonal matrix

Fig. 6. Parallelized IUU^T-factorization

Similar to (14a) the application of $\mathfrak{C}^{-1}$ requires communication in the vector type conversion. Figure 7 presents the proper parallel algorithm :

$$
\begin{array}{ll|ll}
\text{I)} & \underline{u}_I := U_I^{-1}\underline{r}_I & \text{III)} & \underline{\mathfrak{w}}_V := \mathfrak{U}_V^{-T}\underline{\mathfrak{u}}_V \\
& \underline{\mathfrak{u}}_E := \mathfrak{U}_E^{-1}(\underline{\mathfrak{r}}_E - U_{EI}\underline{u}_I) & & \underline{\mathfrak{w}}_E := \mathfrak{U}_E^{-T}(\underline{\mathfrak{u}}_E - \mathfrak{U}_{VE}^T\underline{\mathfrak{w}}_V) \\
& \underline{\mathfrak{u}}_V := \mathfrak{U}_V^{-1}(\underline{\mathfrak{r}}_V - \mathfrak{U}_{VE}\underline{\mathfrak{u}}_E - U_{VI}u_I) & & \underline{w}_I := U_I^{-T}(\underline{u}_I - U_{EI}^T\underline{\mathfrak{w}}_E - U_{VI}^T\underline{\mathfrak{w}}_V) \\
\text{II)} & \underline{\mathfrak{u}} := \sum_{i=1}^{P} A_i^T\underline{\mathfrak{u}}_i & &
\end{array}
$$

Fig. 7. IUU^T Preconditioning $\mathfrak{U}\cdot\mathfrak{U}^T\cdot\underline{\mathfrak{w}} = \underline{\mathfrak{r}}$

4.3 Comparison

The two proposals of the factorization in the previous sections require a different amount of communication in the factorization step and in the application step. The algorithmic work is likely the same when profile minimization routines are used in a preprocessing step. Table 1 summarizes those properties.

	ILL^T : $\mathfrak{L}\cdot\mathfrak{L}^{\mathfrak{T}}$	IUU^T : $\mathfrak{U}\cdot\mathfrak{U}^T$
Factorization start	matrix accumulation	—
Factorization step	—	matrix accumulation
Application step	$2\times$ vector accumulation	$1\times$ vector accumulation

Table 1
Communication in ILL^T and IUU^T

In the IUU^T-factorization step from Fig. 6 the communication is subdivided into communication on edges (3 times in the 2D non-symmetric case) and one communication on the vertex nodes. These communication steps have to be separated in any case, thus it is identical to the matrix conversion in the start step of the ILL^T-factorization in Fig. 6. According to the f.e. discretization, the stiffness matrix K is always distributed stored so that we cannot omit that conversion (see section 2.1).

With respect to the communication we prefer the IUU^T-factorization to define the preconditioner $\mathfrak{C}$.

Remark 3 *If the node numbering in the sequential case represents exactly the numbering introduced by the DD data structures then the numerical behavior of the parallel algorithms in Tables 4-7 correspond exactly to the proper sequential realizations.*

4.4 The Parallel Factorizations in 3D

In 3D we have to handle additionally the nodes on the 2D-interfaces ("Faces") between neighboring domains. These nodes are indicated by the subscript "F", Now, all vectors include 4 components, e.g. $(\underline{u}_V, \underline{u}_E, \underline{u}_F, \underline{u}_I)^T$ and all matrices are 4×4 block matrices with the proper subscripts. If we add to the mesh requirements (a) and (b) similar conditions containing connections between face nodes and connections between face nodes and certain edge nodes, then the algorithms in Figures 4, 5 and 7 are easy to extend in the 3D case.
In the IUU^T-algorithm (Figure 6), after the first modification step, we have to accumulate separately the matrix entries $\mathfrak{K}_F$, $\mathfrak{K}_{EF}$ and $\mathfrak{K}_{VF}$, determine the proper matrices $\mathfrak{U}_F$, $\mathfrak{U}_{EF}$ and $\mathfrak{U}_{VF}$ and then to modify the remaining matrix $\left(\begin{smallmatrix} K_V & K_{VE} \\ K_{EV} & K_E \end{smallmatrix}\right)$. The latter one needs a scaling (type changing) using the diagonal matrix R_F (6) analogously to the last modification in that algorithm.

4.5 Reduced Pattern in the Factorization

The requirements (a) and (b) on the f.e. mesh in Sect. 2.2.3 are easy to realize in the 2D case. In 3D this job becomes much harder. To use the parallelized incomplete Cholesky factorization without fulfilling those requirements on the mesh we need to reduce the pattern of the matrix in the proper way, e.g. in Fig. 1 we have to remove the entries $p \leftrightarrow n$ and similar ones from the directed graph of the matrix. In the same way other sorts of finite elements can be handled. This procedure results in a copy $\widetilde{\mathsf{K}}$ of the original stiffness matrix, but with a reduced pattern.
This matrix $\widetilde{\mathsf{K}}$ will be used only in the incomplete factorizations described in Fig. 4 and Fig. 6. We will call this factorization ILL$^T_{\text{reduced}}$ or IUU$^T_{\text{reduced}}$ depending on the algorithm chosen.

5 Numerical Results

In the numerical experiments the parallelized cg (Fig. 3) with an IUU^T or IUU$^T_{\text{reduced}}$ preconditioner and initial guess $\underline{\mathfrak{u}}^0 = 0$ was used. The cg iteration stopped at a relative accuracy of 10^{-6} measured in the $KC^{-1}K$-norm of the error ($= \sqrt{\sigma}$). In all examples, the error reduction measured in this norm behaved similar to the error reduction in the defect norm. The modified code FEM⊕BEM [7] run on a 16 processor XPLORER-machine by PARSYTEC with 32 MB per processor.

Example 1 *For simplicity, we solve the Poisson equation in the unit square :*

$$\begin{aligned} -\Delta u(x,y) &= 1 \qquad \textit{in } \Omega = (0,1)^2 \ , \\ u(x,y) &= 0 \qquad \textit{on } \Gamma = \partial\Omega \ . \end{aligned}$$

The largest problem on one processor contains 37,121 degrees of freedom.

# processors	# iterations	time for factorization	cg (IUU^T)	one cg step
1	253	0.31 sec	64.4 sec	0.254 sec
4	245	0.08 sec	17.1 sec	0.070 sec
16	202	0.03 sec	4.8 sec	0.024 sec

Table 2
Constant global problem size : 37,121 degrees of freedom

In Table 2, the classical speed-up for 4 and 16 processors reaches 3.8 and 14.6, respectively. It is quite clear that this speed-up will decrease with growing number of processors due to Amdahl‘s law and due to the deterioration of the ratio arithmetic to communicational work (less then 6% of available memory is used in the 16 processor case). The meshes in the subdomains were produced by an automatic parallel mesh generator and therefore the global meshes are different for different numbers of processors. This is the reason for different numbers of cg iterations.

P	#unknowns	# iterations	time for factorization	cg (IUU^T)	one cg step
1	37 121	253	0.31 sec	64.4 sec	0.254 sec
4	147 969	498	0.32 sec	131.4 sec	0.264 sec
16	590 849	845	0.34 sec	238.8 sec	0.277 sec

Table 3
Constant local problem size

The scaled speed-up is defined as $P \cdot \frac{t_{\text{CPU}}(P \text{ proc.})}{t_{\text{CPU}}(1 \text{ proc.})}$ under the assumption of a constant problem size per processor, i.e., the global problem size increases with the number of processors. When examining the 3 time columns in Table 3 the scaled speed-up from 1 to 16 processors achieves 14.6 for the factorization and one cg step. This ensures us that the parallelization for itself works perfect. But, in coincidence with [15], the number of cg iterations in that 2D example grows like the square root of the number of unknowns ($= P \cdot N_{local}$) so that the scaled speed-up for the whole algorithm has to be less then $\sqrt{P}$. Actually, we get a value of 3.7 for 16 processors used ($P = 16$). The parallelized cg with an IUU^T-preconditioning is approximately 40% faster then the diagonal preconditioned one.

Example 2 *As a more challenging example for studying the algorithms and the influence of the reduced pattern in the factorization we consider the more complicated geometry of an electric motor with jumping coefficients in the differential equation. For testing, we consider just one quarter of the domain. Figure 8 presents the different materials and the coarsest mesh used.*

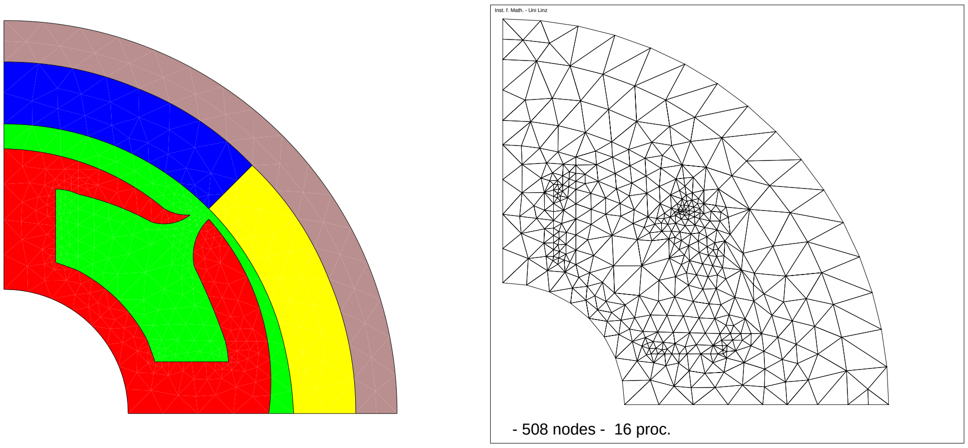


Fig. 8. Material domains and corresponding coarse FEM discretization.

We simplify the variational formulation of the original b.v.p., see [10], into :

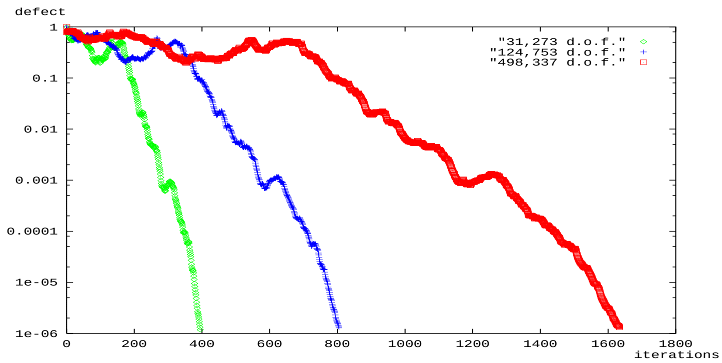
Find the $u = \in \mathbb{V}_0$ *such that for all test functions* $v \in \mathbb{V}_0 = \overset{0}{\mathbb{H}}{}^1(\Omega)$

$$\int\limits_{\Omega} \mu(x,y)\, \nabla^T u\, \nabla v \; dx\, dy = \int\limits_{\Omega} f\, v\; dx\, dy$$

holds, where $\mu(x,y) = \mu_i \;\; \forall (x,y) \in \Omega_i,\; i = 1, 2, \ldots, P$. *The magnitude of the coefficient jumps between subdomain is* 10^3. *The domain* Ω *was decomposed into 16 subdomains with only one material per subdomain.*

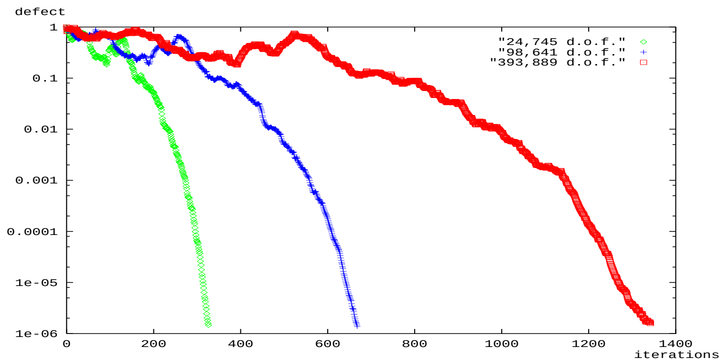
We did several regular refinement steps with the mesh produced by the automatic mesh generator to achieve the meshes used in the calculations. If the automatic mesh generator has to take into account the requirements (a) and (b) then it produces a mesh (Fig. 8, used in Table 4) quite different from the one produced without those conditions (used in Table 5). Therefore the numbers of unknowns in both tables differ significantly. Whereas in the first case the IUU^T preconditioning can be applied we need the $\mathrm{IUU}^T_{\text{reduced}}$ preconditioning in the second case. In Table 5 approximately 5 edges of the whole f.e. connectivity graph are not represented in the reduced pattern of $\widetilde{\mathfrak{K}}$ used in the $\mathrm{IUU}^T_{\text{reduced}}$ preconditioning.

The detailed results for solving the problem above are presented in Tables 4 and 5. In the given example 2, the convergence history of the decfect and the iteration count for IUU^T and $\mathrm{IUU}^T_{\text{reduced}}$ preconditioning behave similar.



# unknowns	1 987	7 861	31 273	124 753	498 337
# cg iterations	95	193	395	805	1635
overall time [sec]	1.65	3.98	15.1	83.9	575.9
mesh generation [sec]	0.31	0.40	0.70	1.86	6.43
factorization [sec]	0.01	0.02	0.05	0.14	0.45
cg solver [sec]	1.33	3.52	14.35	81.9	569.0
one iteration [sec]	0.014	0.018	0.036	0.102	0.348

Table 4
Electrical machine : pcg with IUU^T



# unknowns	1 579	6 229	24 745	98 641	393 889
# cg iterations	79	154	317	650	1342
overall time [sec]	1.23	2.79	10.04	55.9	383.7
mesh generation [sec]	0.31	0.40	0.70	1.72	5.31
factorization [sec]	0.01	0.02	0.05	0.14	0.45
cg solver [sec]	0.94	2.43	9.47	54.5	379.0
one iteration [sec]	0.012	0.016	0.030	0.084	0.282

Table 5
Electrical machine : pcg with IUU$^T_{\text{reduced}}$

Thus, the reduction of the pattern seems to have a negligible influence on the convergence behavior in that example.

Example 3 *As a test example in 3D, the Poisson equation in the unit cube was solved using tetrahedral elements in the discretization. The 3D mesh generator did not fulfill the conditions on the mesh, thus the factorization with the reduced pattern was implemented.*

These experiments have been performed on a 64 processor POWER-GC by PARSYTEC.

N	1 proc.	8 proc.	64 proc.
1305	0.48 (42)	0.83 (43)	1.49 (43)
9265	6.18 (61)	1.94 (62)	2.53 (63)
69729		11.59 (98)	6.07 (98)
540865			32.79 (182)

Table 6
$\mathrm{IUU}^T_{\mathrm{reduced}}$ preconditioned cg in 3D : CPU time (# iterations)

N	1 proc.	8 proc.	64 proc.
1305	0.47 (48)	0.78 (48)	1.58 (48)
9265	4.85 (96)	2.33 (96)	3.61 (96)
69729		14.34 (195)	10.22 (195)
540865			52.18 (394)

Table 7
Diagonal preconditioned cg in 3D : CPU time (# iterations)

As well known from the theory, the iteration counts in Tables 6 and 7 are of the same order, the $\mathrm{IUU}^T_{\mathrm{reduced}}$ preconditioning acts just as a scaling factor (halving the iterations). For more than one processor and a higher number of unknowns, the $\mathrm{IUU}^T_{\mathrm{reduced}}$ preconditioned cg solves the problem 40 % faster than the diagonal preconditioned one. The same holds in the 2D case.

6 Conclusions

The IUU^T- and ILL^T-preconditioners proposed in Sect. 4 are one possibility to code a parallel version of a given solver. There are two opportunities which allow us to realize the parallel version of those factorizations - a f.e. mesh fulfilling the requirements (a) and (b) or the use of a proper reduced pattern in the incomplete factorization. Especially the latter one seems promising in the 3D case.

To improve the preconditioner, variants of the incomplete factorizations with fill-in are feasible - they just have to fulfill the restrictions on the block structure of the matrices. Additionally, a lumping of the defect matrix entries to the main diagonal can be done in the factorization , e.g. MILU or RILU [2,3,21]. Although the acceleration effect of MILU/RILU [21] was observed in the sequential code there was no decrease in the iteration count of the parallel calculations. This behavior seems to be connected with the unfavourable local numbering used in the parallel code (boundary nodes are numbered first) so that in the factorization process a lot of lumping entries are added to the diagonals of the boundary nodes. Here, some special numbering strategies for the internal nodes with respect to the boundary nodes may help, see [17] for a specific discretization. These adapted numbering strategies for unstructured grids have to be investigated in a future work.

The classical speed-up is rather good whereas the scaled speed-up is poor due to the growing of the condition number of the incomplete factorization. So, the parallelized factorizations are well suited as smoothers in a parallelized global multi-grid method [11]. First numerical experiences by Jung using the IUU^T-factorization described as a smoother in a parallelized 3D multi-grid gave a good improvement of the overall solution time [12].

The overall solution times are approximately 4-10 times longer than for the fastest parallel solvers based on pcg with DD-ASM (Additive Schwarz Method) preconditioners [6] or the parallelized multi-grid method [11] in the range of the discretization studied. But in contrast to those multilevel methods we need only one mesh. If just one mesh exists, then the preconditioner proposed accelerates the pcg in comparison to a diagonal preconditioning by 40 %.

In combination with a GMRES-like solver the factorization algorithms given in Sect. 4 are easy to extend for the non-symmetric case. This should work rather well for a weak non-symmetry. In case of a convection dominated problem, a decrease in the efficiency has to be expected, i.e. the stream lines will be cut by the subdomain interfaces. However, a DD according to the stream lines may help. The proper numerical tests will be done in future.

Acknowledgement

I would like to thank Michael Jung for adapting the parallel factorization into a 3D parallel code and for providing the results presented in Tables 6 and 7. I also would like to thank the referees for there suggestions and Yvan Notay for the stimulating discussion.

References

[1] J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring I – IV. *Mathematics of Computation*, 1986, 1987, 1988, 1989. 47, 103–134, 49, 1–16, 51, 415–430, 53, 1–24.

[2] T. C. Chan and H. A. van der Vorst. Approximate and incomplete factorizations. In D. E. Keyes, A. Samed, and V. Venkatakrishnan, editors, *Parallel Numerical Algorithms*, volume 4 of *ICASE/LaRC Interdisciplinary Series in Science and Engineering*, pages 167–202. Kluwer Academic, Dordecht, 1997.

[3] H. C. Elman. Relaxed and stabilized incomplete factorizations for non-self-adjoint linear systems. *BIT*, 29(4):890–915, 1989.

[4] G. Golub and J. M. Ortega. *Scientific Computing: An Introduction with Parallel Computing*. Academic Press, New York, 1993.

[5] U. Groh. Local realization of vector operations on parallel computers. Preprint SPC 94-2, TU Chemnitz, 1994. in German.

[6] G. Haase. Hierarchical extension operators plus smoothing in domain decomposition preconditioners. *Applied Numerical Mathematics*, 1997. accepted for publication.

[7] G. Haase, B. Heise, M. Jung, and M. Kuhn. FEM⊗BEM - a parallel solver for linear and nonlinear coupled FE/BE-equations. DFG-Schwerpunkt "Randelementmethoden", Report 94-16, University Stuttgart, 1994.

[8] G. Haase and U. Langer. The non-overlapping domain decomposition multiplicative Schwarz method. *International Journal of Computer Mathemathics*, 44:223–242, 1992.

[9] G. Haase, U. Langer, and A. Meyer. The approximate Dirichlet decomposition method. part I,II. *Computing*, 47:137–167, 1991.

[10] B. Heise and M. Jung. Parallel solvers for nonlinear elliptic problems based on domain decomposition ideas. *Parallel Computing*, 22:1527–1544, 1997.

[11] M. Jung. On the parallelization of multi-grid methods using a non-overlapping domain decomposition data structure. *Applied Numerical Mathematics*, 1997. accepted for publication.

[12] M. Jung. Parallel multi-level solvers for elliptic boundary value problems in three-dimensional domains. In G. Wittum, editor, *Proceedings of the 5th EMG, 1.–4.10.96, Stuttgart*, ICA-Report. University Stuttgart, 1997.

[13] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings Publishing Company, Redwood City, California, 1994.

[14] K. H. Law. A parallel finite element solution method. *Computer and Structures*, 23(6):845–858, 1989.

[15] J. Meijerink and H. A. van der Vorst. A iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix. *Math.Comp.*, 31:148–162, 1977.

[16] A. Meyer. A parallel preconditioned conjugate gradient method using domain decomposition and inexact solvers on each subdomain. *Computing*, 45:217–234, 1990.

[17] Y. Notay. An efficient parallel discrete PDE solver. *Parallel Computing*, 21:1725–1748, 1995.

[18] Y. Notay. Parallel implementation of preconditioned iterative schemes by means of overlapping decomposition. Techn. Rep. GANMN 95-04, Université Libre de Bruxelles, Brussels, Belgium, 1995.

[19] B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition : Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 1996.

[20] C. H. Tong, T. F. Chan, and C. J. Kuo. Multilevel filtering preconditioners: Extensions to more general elliptic problems. *SIAM J. Sci. Stat. Comput.*, 13:227–242, 1992.

[21] H. A. van der Vorst. High performance preconditioning. *SIAM J. Sci. Stat. Comput.*, 10(6):1174–1185, 1989.

[22] T. Washio and K. Hayami. Parallel block preconditioners based on SSOR and MILU. *Numerical Linear Algebra with Applications*, 1(5):533–553, 1994.