An Incomplete Factorization Preconditioner Based on a Non-Overlapping Domain Decomposition Data Distribution

G. Haase*

December 10, 1996

Abstract

The paper analyzes various parallel matrix-vector multiplications with different matrix and vector types resulting from a non overlapping domain decomposition. Under certain requirements to the f.e. mesh all given matrix and vector types can be used in the multiplication. The general framework is applied to the investigation of the preconditioning step in cg-like methods. 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 global multilevel methods. Numerical results on a SPMD parallel machine are presented.

Keywords : Parallel iterative solvers, Incomplete Factorization, Preconditioning, Domain decomposition,Finite element method.

1 Introduction

The non overlapping domain decomposition (DD) data distribution follows closely the ideas of Law [9] also used in Meyer [11]. The technical background is a parallel machine with distributed memory and links for the communication. As topology of the network we assume a hypercube topology. We will show the DD data distribution in the 2d case, the 3d case is similar.

Lets consider the abstract $\mathbb{V}_0$–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, uniformly 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 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 positive definite stiffness matrix K. For simplicity assume triangular elements in 2d and tetrahedronal elements in 3d.

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

In Sect. 2.1 we introduce the data decomposition with distributed and accumulated vectors according to Law [9], Meyer [11] but whereas therein just distributed matrices are under consideration we focus our interest upon the question why the matrix-vector multiplication is not allowed for accumulated matrices. The general answer was given in Groh [2] but without any further hints. A detailed answer given in Sect. 2.2 includes requirements on the mesh to allow the proper multiplications. Therein block triangular matrices play a special role in the determination of allowed multiplications. So, we also investigate the multiplication by incomplete LU- or UL-factorized matrices where the factorization has to preserve the matrix pattern. Sect. 2.3 describes a parallelized GMRES similar to the cg method in [11].

In Sect. 3 we investigate in general the preconditioners of the preconditioning step in cg-like methods and distinguish between distributed and accumulated type. It turns out that the DD preconditioners [1, 7] are of accumulation type. Now, due to the mesh requirements also global incomplete factorization preconditioners are feasible which are examined in detail in Sect. 4.

Section 5 presents some numerical experiences with the parallelized cg iteration using an incomplete UL-factorization.

2 The DD Data Distribution Used in a Parallelized GMRES

2.1 The DD Data Distribution

First we subdivide our 2d-domain Ω into p subdomains $\overline{\Omega}_i$ $(i = 1, 2, \dots, p)$ and triangulate the subdomains with linear triangular elements. The triangulation has to be conform in the whole domain Ω. Figure 1 illustrates this for an unit square subdivided into 4 congruent squares. Lets 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". When no distinction is necessary we call the latter ones coupling nodes on the interfaces indicated by "C". For convenience in the presentation we number in all vectors and matrices first the vertices, then the edge nodes (subsequently on each edge) and last the inner nodes in the subdomains $\Omega_1 \dots, \Omega_p$.

According to the p subdomains $\overline{\Omega}_i$ $(i = 1, 2, \dots, p)$ we distribute all matrices and and vectors to the p processors P_i $(i = 1, 2, \dots, p)$ 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 have the opportunity to 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 : $\underline{\mathrm{r}}, \underline{\mathrm{f}}$ are stored in P_i as $\underline{\mathrm{r}}_i, \underline{\mathrm{f}}_i$, so that $\underline{\mathrm{r}} = \sum\limits_{i=1}^{p} A_i^T \underline{\mathrm{r}}_i$ etc. is valid, i.e. the nodes on the interfaces posses store just a part of the real value.

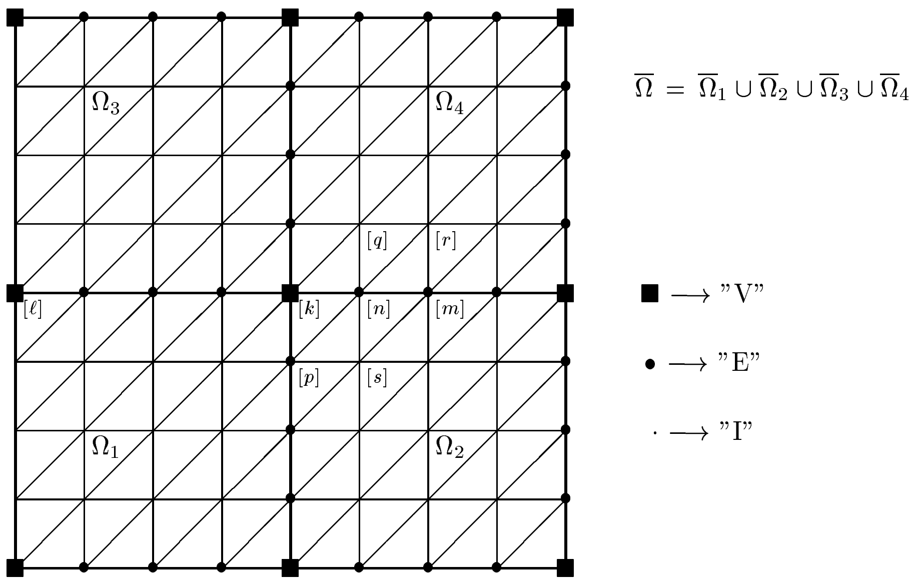

Ω_3
Ω_4
$\overline{\Omega} = \overline{\Omega}_1 \cup \overline{\Omega}_2 \cup \overline{\Omega}_3 \cup \overline{\Omega}_4$
$[q]$
$[r]$
$[\ell]$
$[k]$
$[n]$
$[m]$
$[p]$
$[s]$
Ω_1
Ω_2
■ ⟶ "V"
● ⟶ "E"
· ⟶ "I"

Figure 1: Domain decomposition and triangularization with node numbers.

The matrix K from (2) will be stored (and also accumulated) 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 an f.e. element this exactly the way of constructing f.e. stiffness matrices, so 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 presentation of equation (2) :

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

Here K_I is a block diagonal matrix with the entries $\mathsf{K}_{I,i}$. Also K_{IC}, K_{CI}, K_{IV}, K_{VI} are block matrices. In case when a global accumulation of the matrix K is performed 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. $\mathsf{v}^{[n]}_{C,i}$ denotes the n-th component (local or global ordering) of a vector $\underline{\mathfrak{r}}$ stored on processor i. The subscript "C" indicates that this part of the vector belongs to the edge nodes. A similar notation will be used for the matrices.

2.2 Basic Operations

The description of the operations represents the results given in [11, 7, 2]. Especially the latter one gives a complete overview to this topic concerning with the data distribution given above. Here, we will give 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 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 opportunity is to divide locally the value by the number of neighbours, i.e.

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

with R defined in (6).

2.2.2 Inner Product.

The inner product of different type vectors can be done by just sum up one real number :

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

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

2.2.3 Matrix-Vector Multiplication.

In the following we briefly investigate the matrix vector multiplication with respect to the different matrix and vector types. A detailed description can be found in [2].

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

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

   Performing the sum (i.e. a communication) results in type-I vector.

2. Type-II matrix $\times$ type-II vector needs a conversion of the vector before the previous multiplication can be applied.

3. Type-I matrix $\times$ type-I vector cannot be done with general type-I matrices $\mathfrak{M}$.
   Lets 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 above equations the terms 4 and 5 of the right hand sides differ so that the processors 2 and 4 achieve different results instead of the unique one. The reason is the transport of information by the matrix from a processor a to a node also belonging to processor b or in other words the transport of information from node i to a node j is only allowed when the set of processors node j belongs to is a subset of the processors owning node i. When representing the matrix entries as a directed graph that means e.g. that 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 allowed.
   So, 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 $\mathfrak{M}_I$ is guaranteed by the data decomposition, for the remaining two matrices the mesh has to fulfill the requirements

   (a) No connection between vertices belonging to different sets of subdomains.

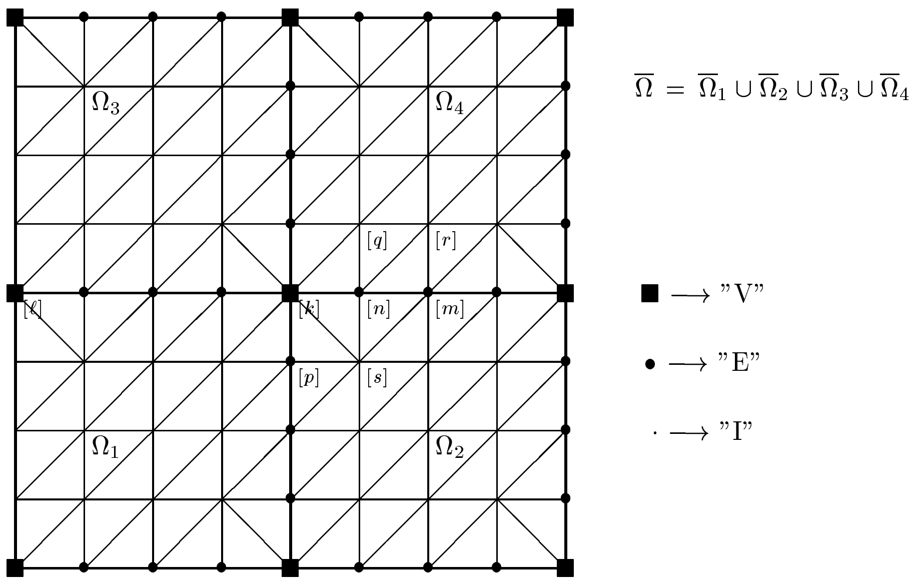
   (b) No connection between edge nodes belonging to different sets of subdomains

   Requirement 3a can be easily fulfilled when at least one node is located on all edges between two vertices. A modification of the given mesh generator can also guarantee requirement 3b to omit e.g. the edge between nodes p and n. Figure 2 present the revised mesh.

4. As in the previous point Type-I matrix $\times$ 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 3a and 3b are fulfilled by the mesh (see Fig. 2) we can focus our interest to the operation $\underline{\mathfrak{f}}_I = \mathsf{M}_{IC} \cdot \underline{\mathfrak{r}}_C$. Performing this operation locally on processor 4 gives us

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

   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 $\mathfrak{M}_{IC,2}^{[r,m]} \cdot \mathsf{r}_2^{[m]}$. In difference to the previous point now the



$\overline{\Omega} = \overline{\Omega}_1 \cup \overline{\Omega}_2 \cup \overline{\Omega}_3 \cup \overline{\Omega}_4$
Ω_3
Ω_4
Ω_1
Ω_2
$[q]$
$[r]$
$[\ell]$
$[k]$
$[n]$
$[m]$
$[p]$
$[s]$
■ ⟶ "V"
● ⟶ "E"
· ⟶ "I"

Figure 2: Domain decomposition and a modified triangularization .

transport of information from node i to a node j is only allowed when the set of processors node i belongs to is a subset of the processors owning node j, e.g. $n \to k$ and $q \to k$ are allowed but not the opposite direction. So, just a matrix of the shape

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

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

Remark : 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 be 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 $\mathfrak{M}$ we can perform the type I matrix-vector multiplication for all types of vectors

$$\underline{\mathfrak{w}} = \mathfrak{M} \cdot \underline{\mathfrak{u}} \;:=\; (\mathfrak{M}_L + \mathfrak{M}_D) \cdot \underline{\mathfrak{u}} + \sum_{i=1}^{P} A_i^T \mathfrak{M}_{U,i} R_i^{-1} \cdot \underline{\mathfrak{u}}_i \tag{13a}$$

$$\underline{\mathfrak{w}} = \mathfrak{M} \cdot \underline{\mathfrak{r}} \;:=\; (\mathfrak{M}_L + \mathfrak{M}_D) \sum_{i=1}^{P} A_i^T \cdot \underline{\mathfrak{r}}_i + \sum_{i=1}^{P} A_i^T \mathfrak{M}_{U,i} \cdot \underline{\mathfrak{r}}_i \tag{13b}$$

$$\underline{\mathfrak{f}} = \mathfrak{M} \cdot \underline{\mathfrak{u}} \;:=\; R^{-1} (\mathfrak{M}_L + \mathfrak{M}_D) \cdot \underline{\mathfrak{u}} + \mathfrak{M}_U R^{-1} \cdot \underline{\mathfrak{u}} \tag{13c}$$

$$\underline{\mathfrak{f}} = \mathfrak{M} \cdot \underline{\mathfrak{r}} \;:=\; R^{-1} (\mathfrak{M}_L + \mathfrak{M}_D) \sum_{i=1}^{P} A_i^T \cdot \underline{\mathfrak{r}}_i + \mathfrak{M}_U \cdot \underline{\mathfrak{r}} \;. \tag{13d}$$

Please 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.
Factorizing the Matrix $\mathfrak{M}$ into a lower and an upper triangular matrix $\mathfrak{L}^{-1}$ and $\mathfrak{U}^{-1}$ can be done in two ways. First we can write

$$\underline{\mathfrak{w}} = \mathfrak{L}^{-1}\mathfrak{U}^{-1}\cdot\underline{\mathfrak{r}} \;:=\; \mathfrak{L}^{-1}\sum_{i=1}^{P} A_i^T\mathfrak{U}_i^{-1}\cdot\underline{\mathfrak{r}}_i \tag{14a}$$

$$\underline{\mathfrak{f}} = \mathfrak{L}^{-1}\mathfrak{U}^{-1}\cdot\underline{\mathfrak{u}} \;:=\; R^{-1}\mathfrak{L}^{-1}\sum_{i=1}^{P} A_i^T\mathfrak{U}_i^{-1}R^{-1}\cdot\underline{\mathfrak{u}}_i \ , \tag{14b}$$

the second case is

$$\underline{\mathfrak{w}} = \mathfrak{U}^{-1}\mathfrak{L}^{-1}\cdot\underline{\mathfrak{r}} \;:=\; \sum_{i=1}^{P} A_i^T\mathfrak{U}_i^{-1}R_i^{-1}A_i\mathfrak{L}^{-1}\cdot\Big(\sum_{j=1}^{P} A_j^T\underline{\mathfrak{r}}_j\Big) \tag{15a}$$

$$\underline{\mathfrak{f}} = \mathfrak{U}^{-1}\mathfrak{L}^{-1}\cdot\underline{\mathfrak{u}} \;:=\; \mathfrak{U}^{-1}R^{-1}\mathfrak{L}^{-1}\cdot\underline{\mathfrak{u}} \ . \tag{15b}$$

The remaining equations can be easily derived by type conversion. Here, the number and the kind of type conversion is also determined by the way of factorizing the matrix $\mathfrak{M}$.

2.3 The Parallelized GMRES

Using the data distribution and the relations of the previous sections 2.1 and 2.2 we are in the position to write down a parallel algorithm for the GMRES with the same amount of communication as in the parallelized cg, i.e. communication is necessary just in the type conversion indicated by $\sum_{j=1}^{P}$ and in the inner product $(\cdot,\cdot)$. Fig. 3 illustrates the parallelized algorithm.

```
Choose $\underline{\mathfrak{u}}^0$
$\underline{\mathsf{r}}$ := $\underline{\mathsf{f}} - \mathsf{K}\cdot\underline{\mathfrak{u}}^0$
$\underline{\mathfrak{w}}^1$ := $\sum_{j=1}^{P} A_j^T \underline{\mathsf{r}}_j$
$z_1$ := $(\underline{\mathfrak{w}}^1, \underline{\mathsf{r}})$
$k$ := $0$
repeat  $k := k+1$
        $\underline{\mathsf{r}}$ := $\mathsf{K}\cdot\underline{\mathfrak{w}}^k$
        for $i := 1$ to $k$ do
                $h_{i,k}$ := $(\underline{\mathfrak{w}}^i, \underline{\mathsf{r}})$
                $\underline{\mathsf{r}}$ := $\underline{\mathsf{r}} - h_{i,k}\cdot R^{-1}\cdot\underline{\mathfrak{w}}^i$
        end
        $\underline{\mathfrak{w}}^{k+1}$ := $\sum_{j=1}^{P} A_j^T \underline{\mathsf{r}}_j$
        $h_{k+1,k}$ := $(\underline{\mathfrak{w}}^{k+1}, \underline{\mathsf{r}})$
        $\underline{\mathfrak{w}}^{k+1}$ := $\underline{\mathfrak{w}}^{k+1}/h_{k+1,k}$
        for $i := 1$ to $k-1$ do  $\begin{pmatrix} h_{i,k} \\ h_{i+1,k} \end{pmatrix} := \begin{pmatrix} c_{i+1} & s_{i+1} \\ s_{i+1} & -c_{i+1} \end{pmatrix} \cdot \begin{pmatrix} h_{i,k} \\ h_{i+1,k} \end{pmatrix}$
        $\alpha$ := $\sqrt{h_{k,k}^2 + h_{k+1,k}^2}$
        $s_{k+1}$ := $h_{k+1,k}/\alpha$ ; $c_{k+1}$ := $h_{k,k}/\alpha$ ; $h_{k,k}$ := $\alpha$
        $z_{k+1}$ := $s_{k+1} z_k$ ; $z_k$ := $c_{k+1} z_k$
until  $|z_{k+1}| < \mathit{tolerance}$
$y_k$ := $z_k/h_{k,k}$
for $i := k-1$ down to $1$ do  $y_i$ := $(z_i - \sum_{j=i+1}^{k} h_{i,j} y_j)/h_{i,i}$
$\underline{\mathfrak{u}}^k$ := $\underline{\mathfrak{u}}^0 + \sum_{i=1}^{k} y_i \cdot \underline{\mathfrak{w}}^j$
```

Figure 3: Parallelized GMRES

3 Preconditioners for the Iteration Form

The type conversion $\underline{\mathfrak{w}} = \sum_{j=1}^{P} A_j^T \underline{\mathfrak{r}}_j$ in the GMRES (Fig. 3) is just the type-I identity and equivalent to equation (13b) with the components $\mathfrak{M}_L = \mathfrak{M}_U = 0$ and $\mathfrak{M}_D = \mathfrak{I}$. So, we are looking for preconditioners not requiring much more then the above type conversion from a type-II to a type-I vector.

3.1 Type-I Preconditioners

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

Remark : In the symmetric case the well known ASM-DD-preconditioner [7] using Schur complement preconditioners proposed in [1, 13] fits exactly in this scheme presented by equation (14a), although the part concerning with the vertex nodes is not a diagonal matrix. The reason is the usual serial and global handling of the relating system of equations.

3.2 Type-II Preconditioners

Applying a type-II preconditioning matrix $\mathsf{C}^{-1} = \sum\limits_{i=1}^{p} A_i^T \mathsf{C}_i^{-1} A_i$ we are in need to convert the vectors twice (see section 2.2) so that the precondition step looks like

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

The matrix K is not accumulated and so the submatrices K_i represent a 2nd order pde in Ω_i with homogeneous Neumann B.C. on the inner boundaries $\partial\Omega_i \setminus \partial\Omega$. If $\partial\Omega_i \cap \Gamma_D = \emptyset$ the matrix K_i is singular. Therefore we cannot choose C_i in the same way. One opportunity is to assemble the stiffness Matrix K and set $\mathsf{C}_i = \mathfrak{K}_{\widetilde{i}}$ so that the local preconditioner represents a 2nd order pde in Ω_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. So we end up with an additive Schwarz overlapping preconditioner which is out of scope of this work, for further investigations see [12].

4 Incomplete Factorization Preconditioners

The choices $\mathfrak{C}^{-1} = \mathfrak{L}^{-1} \cdot \mathfrak{U}^{-1}$ (which is exactly the same matrix structure as in equation 14a) or $\mathfrak{C}^{-1} = \mathfrak{U}^{-1} \cdot \mathfrak{L}^{-1}$ (15a) indicate factorization $\mathfrak{U} \cdot \mathfrak{L}$ or $\mathfrak{L} \cdot \mathfrak{U}$ of the given accumulated stiffness matrix $\mathfrak{K}$ but the fill-in does not fit into our DD data concept and would scale down the problem size handled in parallel. So we concern with the classic ILU factorization [10] preserving the given pattern of the matrix and take advantage of the block structure of matrix K (4).
To present the following more clearly we will use the notation for vector and matrix types I and II only for those components which really depend on the type. All remaining parts are written in the standard mathematical font, e.g. $\underline{\mathfrak{r}} = (\underline{\mathfrak{r}}_V, \underline{\mathfrak{r}}_E, \underline{r}_I)^T$.

4.1 The ILU-Factorization

To achieve a preconditioner $\mathfrak{C}^{-1} = \mathfrak{U}^{-1} \cdot \mathfrak{L}^{-1}$ we have to perform an incomplete LU-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 parallel Γ
$\mathfrak{L}_V, \mathfrak{U}_V$	$\mathfrak{K}_V \;=\; \mathfrak{L}_V \cdot \mathfrak{U}_V$	mesh → diagonal matrix
$\mathfrak{L}_{EV}$	$\mathfrak{K}_{EV} \;=\; \mathfrak{L}_{EV} \cdot \mathfrak{U}_V$	DD, mesh → block matrices
$\mathfrak{L}_{IV}$	$\mathfrak{K}_{IV} \;=\; \mathfrak{L}_{IV} \cdot \mathfrak{U}_V$	DD, mesh → block matrices
$\mathfrak{U}_{VE}$	$\mathfrak{K}_{VE} \;=\; \mathfrak{L}_V \cdot \mathfrak{U}_{VE}$	DD, mesh → block matrices
$\mathfrak{U}_{VI}$	$\mathfrak{K}_{VI} \;=\; \mathfrak{L}_V \cdot \mathfrak{U}_{VI}$	DD, mesh → block matrices
Modify		
	$\mathfrak{K}_E \;:=\; \mathfrak{K}_E - \mathfrak{L}_{EV} \cdot \mathfrak{U}_{VE}$	same matrix type
	$K_{EI} \;:=\; K_{EI} - \mathfrak{L}_{EV} \cdot U_{VI}$	same matrix type
	$K_{IE} \;:=\; K_{IE} - L_{IV} \cdot \mathfrak{U}_{VE}$	same matrix type
	$K_I \;:=\; K_I - L_{IV} \cdot U_{VI}$	same matrix type
Determine		
$\mathfrak{L}_E, \mathfrak{U}_E$	$\mathfrak{K}_E \;=\; \mathfrak{L}_E \cdot \mathfrak{U}_E$	DD, mesh → block matrices
L_{IE}	$K_{IE} \;=\; L_{IE} \cdot \mathfrak{U}_E$	DD → block matrices
U_{EI}	$K_{EI} \;=\; \mathfrak{L}_E \cdot U_{EI}$	DD → block matrices
Modify		
	$K_I \;:=\; K_I - L_{IE} \cdot U_{EI}$	same matrix type
Determine		
$\mathfrak{L}_I, \mathfrak{U}_I$	$\mathfrak{K}_I \;=\; \mathfrak{L}_I \cdot \mathfrak{U}_I$	DD → block matrices

Figure 4: Parallelized LU-factorization

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

$$
\begin{array}{rl|rl}
\text{I)} & \underline{\mathfrak{r}} := \sum\limits_{i=1}^{P} A_i^T \underline{\mathsf{r}}_i & & \\
\text{I)} & \underline{\mathfrak{u}}_V := \mathfrak{L}_V^{-1} \underline{\mathfrak{r}}_V & \text{IV)} & \underline{w}_I := U_I^{-1} \underline{u}_I \\
 & \underline{\mathfrak{u}}_E := \mathfrak{L}_E^{-1} (\underline{\mathfrak{r}}_E - \mathfrak{L}_{EV} \underline{\mathfrak{u}}_V) & & \underline{\mathsf{w}}_E := \mathfrak{U}_E^{-1} (\underline{\mathsf{u}}_E - U_{EI} \underline{w}_I) \\
 & \underline{u}_I := L_I^{-1} (\underline{r}_I - L_{IE} \underline{\mathfrak{u}}_E - L_{IV} \underline{\mathfrak{u}}_V) & & \underline{\mathsf{w}}_V := \mathfrak{U}_V^{-1} (\underline{\mathsf{u}}_V - \mathfrak{U}_{VE} \underline{\mathsf{w}}_E - U_{VI} \underline{w}_I) \\
\text{III)} & \underline{\mathsf{u}} := R^{-1} \underline{\mathfrak{u}} & \text{V)} & \underline{\mathfrak{w}} := \sum\limits_{i=1}^{P} A_i^T \underline{\mathsf{w}}_i
\end{array}
$$

Figure 5: ILU Preconditioning $\mathfrak{L} \cdot \mathfrak{U} \cdot \underline{\mathfrak{w}} = \underline{\mathfrak{r}}$

4.2 The IUL-Factorization

To achieve a preconditioner $\mathfrak{C}^{-1} = \mathfrak{L}^{-1} \cdot \mathfrak{U}^{-1}$ we have to perform an incomplete UL-factorization, see Fig. 6. During the factorization we change the type of the matrix factorized.

Similar to (14a) the application of $\mathfrak{C}^{-1}$ requires communication in the vector

Start	K	
Determine		Why parallel Γ
U_I, L_I	$K_I \ = \ U_I \cdot L_I$	DD $\rightarrow$ blocks matrices
L_{IE}	$K_{IE} \ = \ U_I \cdot L_{IE}$	DD $\rightarrow$ blocks matrices
L_{IV}	$K_{IV} \ = \ U_I \cdot L_{IV}$	DD $\rightarrow$ blocks matrices
U_{EI}	$K_{EI} \ = \ U_{EI} \cdot L_I$	DD $\rightarrow$ blocks matrices
U_{VI}	$K_{VI} \ = \ U_{VI} \cdot L_I$	DD $\rightarrow$ blocks matrices
Modify		
	$\mathsf{K}_E \ := \ \mathsf{K}_E - U_{EI} \cdot L_{IE}$	same matrix type
	$\mathsf{K}_{EV} \ := \ \mathsf{K}_{EV} - U_{EI} \cdot L_{IV}$	same matrix type
	$\mathsf{K}_{VE} \ := \ \mathsf{K}_{VE} - U_{VI} \cdot L_{IE}$	same matrix type
	$\mathsf{K}_V \ := \ \mathsf{K}_V - U_{VI} \cdot L_{IV}$	same matrix type
Accumulate	$\mathfrak{K}_E$, $\mathfrak{K}_{EV}$, $\mathfrak{K}_{VE}$	
e.g.	$\mathfrak{K}_{EV} \ := \ \sum\limits_{i=1}^{P} A_{E,i}^T \mathsf{K}_{EV,i} A_{V,i}$	—
Determine		
$\mathfrak{U}_E$, $\mathfrak{L}_E$	$\mathfrak{K}_E = \mathfrak{U}_E \cdot \mathfrak{L}_E$	DD, mesh $\rightarrow$ blocks matrices
$\mathfrak{U}_{VE}$	$\mathfrak{K}_{VE} = \mathfrak{U}_{VE} \cdot \mathfrak{L}_E$	DD, mesh $\rightarrow$ blocks matrices
$\mathfrak{L}_{EV}$	$\mathfrak{K}_{EV} = \mathfrak{U}_E \cdot \mathfrak{L}_{EV}$	DD, mesh $\rightarrow$ blocks matrices
Modify		
	$\mathsf{K}_V \ := \ \mathsf{K}_V - \mathfrak{U}_{VE} \cdot R_E^{-1} \cdot \mathfrak{L}_{EV}$	same matrix type (15a)
Accumulate	$\mathfrak{K}_V$	—
Determine		
$\mathfrak{U}_V$, $\mathfrak{L}_V$	$\mathfrak{K}_V = \mathfrak{U}_V \cdot \mathfrak{L}_V$	mesh $\rightarrow$ diagonal matrix

Figure 6: Parallelized UL-factorization

type conversion. Figure 7 presents the proper parallel algorithm :

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

Figure 7: IUL Preconditioning $\mathfrak{U} \cdot \mathfrak{L} \cdot \underline{\mathfrak{w}} = \underline{\mathfrak{r}}$

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 whereas the algorithmic work is nearly the same. The Table 1 summarizes the properties.

In the IUL-factorization step in Fig. 6 the communication is subdivided into communication upon edges (3 times in the 2D non-symmetric case) and one communication on the vertex nodes. These communication steps have to be

	ILU : $\mathfrak{L} \cdot \mathfrak{U}$	IUL : $\mathfrak{U} \cdot \mathfrak{L}$
Factorization start	matrix accumulation	—
Factorization step	—	matrix accumulation
Application step	$2 \times$ vector accumulation	$1 \times$ vector accumulation

Table 1: Communication in ILU and IUL

separated in any case and so it is identical to the matrix conversion in the start step of the ILU-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 IUL-factorization to define the preconditioner $\mathfrak{C}$.

5 Numerical Results

Due to the restriction of the given code to symmetric problems in the numerical experiments the parallelized cg-method with an incomplete Cholesky preconditioning, similar to the IUL one (7), was used. The cg iteration stops at a relative accuracy of 10^{-6} measured in the $KC^{-1}K$-norm of the error. The modified code FEM⊗BEM [6] run an a 16 processor XPLORER-machine by PARSYTEC with 32 MB per processor.

Test example : For simplicity we solve the Laplace equation in the unit square

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

The largest problem feasible with the code above on one processor contains 37.121 degrees of freedom. Please note, that the nodes belonging to the Dirichlet boundary are not counted.

In Table 2, the classical speed up for 4 and 16 processors reaches 3.8 and 14.6,

# processors	# iterations	time for factorization	time for cg	time for 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

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 fact that in the 16 processor case less then 6% of available memory is used and so the relation between communicational to arithmetical work becomes worse. The meshes in the subdomains were produced by an automatic mesh generator and therefore the global meshes are different for different numbers of processors. This is the reason for different numbers of cg iterations.

In principle the scaled speed up, i.e. the global problem size increases with the number of processors, should be much better then the classical speed up.

P	#unknowns	# iterations	time for factorization	cg	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

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 Gustafsson [3] the number of cg iterations 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 .

Nevertheless, using a fast domain decomposition preconditioner in the cg [4] (Algorithm 1b with BPS and hierarchical extension plus smoothing therein) we may solve the problem much faster,e.g. it takes 16 processors with 590 849 unknowns just 54 seconds (see Table 10). This special preconditioner requires a multilevel structure of the grid whereas the ICC preconditioning uses just one grid. So, the preconditioning described in Sect. 4 can be used for parallel computations using just one grid. In case of an AMG technique used in the DD-preconditioner the result of the comparison depends strongly on the concrete AMG implementation.

The ICC may serve as a smoother in a parallelized global multi-grid method [8], the same holds for the IUL algorithm in Sect. 4.2 .

6 Conclusions

The IUL- and ILU-preconditioners proposed in Sect. 4 are one opportunity to code a parallel version of a solver given and work also for multiply degrees of freedom per node. The only restrictions are the requirements 3a and 3b to the mesh which have to be extended in a similar way in the 3D case.

The classical speedup is rather good whereas due to the growing of the condition number of the incomplete factorization the scale up is poor. So, the parallelized factorizations are well suited for use as a smoother in a parallelized global multi-grid method [8]. The overall solution times are approximately 4-10 times longer then for the fastest parallel solvers based on pcg with DD-ASM (Additive Schwarz Method) preconditioners [4] or the parallelized multi-grid method [8]. But in contrast to those multi level methods we need only one mesh.

In the non symmetric case of matrix K we have to distinguish between strong and weak non-symmetry. In case of a convection dominated problem the numbering proposed in Sect. 2.1 does not represent the necessary numbering along the stream lines so that a decrease in the efficiency has to be expected, i.e. the stream lines will be cut by the domain interfaces. On the other hand when the problem is not convection dominated but non symmetric the preconditioners proposed will work good as precondioners in the parallelized GMRES. The proper test will be done in future.

A Detailed Numerical Results

The abbreviation ICC denotes the IUL-factorization from section 4.2 and will be used as a preconditioner in the pcg in Tables 4 – 7. In the remaining tables a DD-ASM (Additive Schwarz Method) will be used as preconditioner which components are a BPS [1] and a hierarchical extension with smoothing in a sophisticated implementation [4, 5].

As a challenging example we calculated the magnetic potential in an electrical machine. The results are presented in Tables 7 and 11.

level	1	2	3	4	5	6	7
# unknowns	14	45	161	609	2 369	9 345	37 121
# cg iterations	3	8	15	30	61	124	253
overall time [sec]	0.01	0.02	0.04	0.17	1.16	8.83	69.1
mesh generation [sec]	0.01	0.01	0.03	0.08	0.29	1.11	4.44
factorization [sec]					0.01	0.07	0.31
cg solver [sec]	0.01	0.01	0.01	0.09	0.86	7.65	64.35
one iteration [sec]					0.014	0.062	0.254

Table 4: Model example : 1x1_1.bsp; pcg with ICC

level	1	2	3	4	5	6	7
# unknowns	45	161	609	2 369	9 345	37 121	147 969
# cg iterations	8	15	29	60	120	245	498
overall time [sec]	0.06	0.10	0.19	0.55	2.69	18.32	136.18
mesh generation [sec]	0.03	0.04	0.05	0.11	0.32	1.15	4.49
factorization [sec]					0.02	0.08	0.32
cg solver [sec]	0.03	0.06	0.14	0.44	2.35	17.09	131.4
one iteration [sec]					0.02	0.07	0.264

Table 5: Model example : 2x2_1.bsp; pcg with ICC

level	1	2	3	4	5	6	7
# unknowns	161	609	2 369	9 345	37 121	147 969	590 849
# cg iterations	11	24	48	99	202	419	845
overall time [sec]	0.27	0.41	0.70	1.38	5.30	32.9	238.8
mesh generation [sec]	0.15	0.16	0.17	0.23	0.44	1.29	4.65
factorization [sec]				0.01	0.03	0.09	0.34
cg solver [sec]	0.12	0.25	0.53	1.14	4.83	31.5	233.8
one iteration [sec]	0.011	0.010	0.011	0.011	0.024	0.075	0.277

Table 6: Model example : 4x4_1.bsp; pcg with ICC

level	1	2	3	4	5	6
# unknowns	508	1 987	7 861	31 273	124 753	498 337
# cg iterations	44	95	193	395	805	1635
overall time [sec]	0.85	1.65	3.98	15.1	83.9	575.9
mesh generation [sec]	0.28	0.31	0.40	0.70	1.86	6.43
factorization [sec]	0.01	0.01	0.02	0.05	0.14	0.45
cg solver [sec]	0.56	1.33	3.52	14.35	81.9	569.0
one iteration [sec]	0.011	0.014	0.018	0.036	0.102	0.348

Table 7: Electrical machine : mb16.bsp; pcg with ICC

level	1	2	3	4	5	6	7
# unknowns	14	45	161	609	2 369	9 345	37 121
# cg iterations	1	4	4	4	4	4	4
cg solver [sec]			0.02	0.09	0.46	2.07	9.00
one iteration [sec]				0.018	0.115	0.414	1.80

Table 8: Model example : 1x1_1.bsp; pcg with DD-ASM-Alg1b (BPS,1g)

level	1	2	3	4	5	6	7
# unknowns	45	161	609	2 369	9 345	37 121	147 969
# cg iterations	3	10	12	14	16	17	19
cg solver [sec]	0.01	0.04	0.10	0.34	1.61	7.53	35.7
one iteration [sec]				0.023	0.095	0.418	1.79

Table 9: Model example : 2x2_1.bsp; pcg with DD-ASM-Alg1b (BPS,1g)

level	1	2	3	4	5	6	7
# unknowns	161	609	2 369	9 345	37 121	147 969	590 849
# cg iterations	10	17	19	22	24	27	29
cg solver [sec]	0.08	0.17	0.26	0.64	2.51	12.00	53.96
one iteration [sec]				0.028	0.100	0.429	1.80

Table 10: Model example : 4x4_1.bsp; pcg with DD-ASM-Alg1b (BPS,1g)

level	1	2	3	4	5	6
# unknowns	508	1 987	7 861	31 273	124 753	498 337
# cg iterations	19	34	36	37	38	38
cg solver [sec]	0.51	1.23	2.12	5.73	21.17	84.64
one iteration [sec]	0.026	0.035	0.057	0.151	0.543	2.23

Table 11: Electrical machine : m16.bsp; pcg with DD-ASM-Alg1b (BPX,1g)

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] Ulrich Groh. Local realization of vector operations on parallel computers. Preprint SPC 94-2, TU Chemnitz, 1994. in german.

[3] I. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.

[4] Gundolf Haase. Hierarchical extension operators plus smoothing in domain decomposition preconditioners. *Applied Numerical Mathematics*, 20:1–20, 1996.

[5] Gundolf Haase. Multilevel extension techniques in domain decomposition preconditioners. Institutsbericht Nr. 508, University Linz, Institute of Mathematics, November 1996. submitted for publication in the proceedings of the "The Ninth International Conference on Domain Decomposition", Bergen.

[6] Gundolf Haase, Bodo Heise, Michael Jung, and Michael Kuhn. FEM⚭BEM - a parallel solver for linear and nonlinear coupled FE/BE-equations. DFG-Schwerpunkt "Randelementmethoden", Report 94-16, University Stuttgart, 1994.

[7] Gundolf Haase, Ulrich Langer, and Arnd Meyer. The approximate Dirichlet decomposition method. part I,II. *Computing*, 47:137–167, 1991.

[8] Michael Jung. On the parallelization of multi-grid methods using a non-overlapping domain decomposition data structure. *Applied Numerical Mathematics*, 20, 1996.

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

[10] T. Manteuffel. A incomplete factorization technique for positive definite linear systems. *Math.Comp.*, 34:473–497, 1980.

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

[12] Barry Smith, Petter Bjørstad, and William Gropp. *Domain Decomposition : Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 1996.

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