New Matrix-by-Vector Multiplications Based on a Nonoverlapping Domain Decomposition Data Distribution

Gundolf Haase

University Linz, Inst. of Math., Altenbergerstr. 69, A–4040 Linz, Austria

Abstract. The nonoverlapping domain decomposition (DD) method allows to store vectors as accumulated or as distributed (each process stores a part of that value) vectors. The case of distributed stored matrices is fully investigated, whereas the use of accumulated matrices is not customary in the DD community. This paper is concerned with the accumulated matrices and investigates conditions under which matrix-by-vector multiplications can be realized. Especially, we derive restrictions on the matrix shape and the finite element (FE) mesh. As a new result, the special structure of these admissible accumulated matrices leads directly to global incomplete factorizations as preconditioners in CG-like methods. Also, the well-known DD preconditioners fit into the general framework of the matrix-by-vector operations presented.

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

1 Introduction

Let us consider an abstract elliptic and bounded variational problem arising from the weak formulation of a scalar second–order, uniformly elliptic boundary value problem (BVP) given in a plane bounded domain $\Omega \subset \mathbb{R}^2$ with a piecewise smooth boundary $\Gamma = \partial\Omega$. Defining the usual linear FE nodal basis the FE isomorphism results in a large-scale sparse system

$$K\,\underline{u} \;=\; \underline{f} \tag{1}$$

of equations with the positive definite stiffness matrix K. For simplicity, we assume triangular elements in 2d and tetrahedrons in 3d. We focus our interest on the 2d case; the 3d case is similar.

The nonoverlapping domain decomposition (DD) data distribution following the ideas in [7,8] allows to store vectors as accumulated (each process stores the full value) or as distributed (each process stores a part of that value) vectors. The same storage classes exist for matrices. The case of distributed stored matrices is fully investigated (see [7,8]). So, we focus our interest on the question why the matrix-by-vector multiplication is not allowed for accumulated matrices. The general answer was given in Groh [2] but without any further investigations. A detailed answer, given in Sect. 2.2, includes requirements on mesh and matrix shape to allow the proper multiplications. 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 [1,5] are of accumulated type. Now, due to the assumptions on the mesh also global incomplete factorization preconditioners are feasible which are examined in detail in Sect. 4. Some numerical experiences are presented in Sect. 5.

2 The Matrix-by-Vector Multiplication in the DD

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 discretize 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.

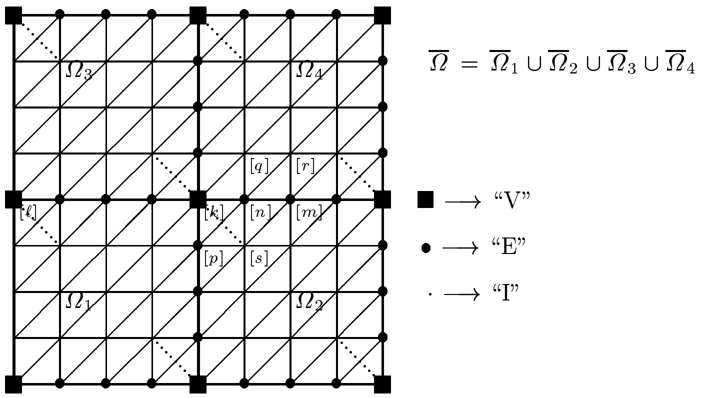

Fig. 1. Domain decomposition, initial (/) and modified ($\ddots$) triangulation.

Let us introduce 3 classes of nodes: the inner nodes further denoted by the subscript "I", the nodes on the interior of a coupling edge (edge nodes) denoted by "E" and the vertex nodes (cross points) "V". If no distinction is necessary then the latter are called coupling nodes indicated by "C". The unknowns are ordered such that we number 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 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 :

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 every interface node possesses the full value of its vector components.

II : $\underline{\mathsf{r}}, \underline{\mathsf{f}}$ are stored in P_i as $\underline{\mathsf{r}}_i, \underline{\mathsf{f}}_i$, so that $\underline{\mathsf{r}} = \sum\limits_{i=1}^{p} A_i^T \underline{\mathsf{r}}_i$ etc. is valid, i.e., the nodes on the interfaces store just a part of the actual value which will be recovered by accumulation.

The matrix K (1) will be stored in a distributive sense similar to a type-II vector. We call it a type-II matrix $\mathsf{K} = \sum\limits_{i=1}^{p} A_i^T \mathsf{K}_i A_i$, where K_i is the stiffness matrix belonging to the subdomain $\overline{\Omega}_i$. Looking at $\overline{\Omega}_i$ as a macro finite element its element stiffness matrix K_i is identical to the proper local part of the distributed stored matrix K. The ordering of the nodes given above and the types of the vectors lead to the following presentation of equation (1) :

$$\begin{pmatrix} \mathsf{K}_V & \mathsf{K}_{VE} & \mathsf{K}_{VI} \\ \mathsf{K}_{EV} & \mathsf{K}_E & \mathsf{K}_{EI} \\ \mathsf{K}_{IV} & \mathsf{K}_{IE} & \mathsf{K}_I \end{pmatrix} \cdot \begin{pmatrix} \underline{\mathfrak{u}}_V \\ \underline{\mathfrak{u}}_E \\ \underline{\mathfrak{u}}_I \end{pmatrix} = \begin{pmatrix} \underline{\mathsf{f}}_V \\ \underline{\mathsf{f}}_E \\ \underline{\mathsf{f}}_I \end{pmatrix} . \tag{2}$$

Here K_I is a block diagonal matrix with the entries $\mathsf{K}_{I,i}$. K_{IC}, K_{CI}, K_{IV}, K_{VI} are block matrices, too. If a global accumulation of the matrix K is performed then we will denote that type-I matrix by $\mathfrak{M}$ and write $\mathfrak{M}_i = A_i \mathfrak{M} A_i^T$. Although $\mathsf{K} \equiv \mathfrak{M}$ holds we have to distinguish between both because of the different local storing, i.e., $\mathsf{K}_\mathsf{i} \neq \mathfrak{M}_\mathsf{i}$. The entries of the diagonal matrix $R = \sum\limits_{i=1}^{p} A_i^T A_i$ are equal to the number of subdomains a node belongs to (e.g., according to Fig. 1 $R^{[k]} = 4$, $R^{[n]} = R^{[m]} = R^{[p]} = 2$, $R^{[q]} = 1$).

The change from type II to a type I vector requires communication :

$$\underline{\mathfrak{w}}_i = A_i \sum_{i=1}^{p} A_i^T \underline{\mathsf{r}}_i . \tag{3}$$

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., $\underline{\mathsf{r}}_i = R^{-1} \underline{\mathfrak{w}}_i$ with R defined in the paragraph above.

2.2 Matrix-by-Vector Operations

In the following we investigate briefly the matrix-by-vector multiplication with respect to accumulated matrices and different vector types. Sub- and superscripts are used in the following way : $\mathsf{v}_{C,i}^{[n]}$ denotes the n-th component (local or global ordering) of a vector $\underline{\mathsf{v}}$ stored on processor i. The subscript "C" indicates that this part of the vector belongs to the coupling nodes. A similar notation will be used for the matrices.

1. Type-I matrix $\times$ type-I vector cannot be realized with general type-I matrices $\mathfrak{M}$. Let us have a look at the node n in Fig. 1 and the operation $\mathfrak{M} \cdot \underline{\mathfrak{w}}$. The local multiplications $\underline{u}_i = \mathfrak{M}_i \cdot \underline{\mathfrak{w}}_i$ $(i = 2, 4)$ for node n result in

$$\begin{aligned} u_2^{[n]} &= \mathfrak{M}_2^{[n,n]}\mathfrak{w}_2^{[n]} + \mathfrak{M}_2^{[n,m]}\mathfrak{w}_2^{[m]} + \mathfrak{M}_2^{[n,k]}\mathfrak{w}_2^{[k]} + \mathfrak{M}_2^{[n,p]}\mathfrak{w}_2^{[p]} + \mathfrak{M}_2^{[n,s]}\mathfrak{w}_2^{[s]} \\ u_4^{[n]} &= \mathfrak{M}_4^{[n,n]}\mathfrak{w}_4^{[n]} + \mathfrak{M}_4^{[n,m]}\mathfrak{w}_4^{[m]} + \mathfrak{M}_4^{[n,k]}\mathfrak{w}_4^{[k]} + \mathfrak{M}_4^{[n,q]}\mathfrak{w}_4^{[q]} + \mathfrak{M}_4^{[n,r]}\mathfrak{w}_4^{[r]} \ . \end{aligned}$$

In the equations above, the terms 4 and 5 of the right-hand sides differ. The processors 2 and 4 achieve different results, instead of the unique one, since processor 2 uses data not available to processor 4 and vice versa. Denoting the set of processors a node j belongs to by ω_j we can ensure identical results in the equations above only if we use nodes j fulfilling $\omega_n \subseteqq \omega_j$ $(j \in \{n, k, m\})$ in the multiplication. If the matrix is represented by a directed graph then certain edges of that graph are not allowed. For example, 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. So, only a matrix of the shape

$$\mathfrak{M} \ = \begin{pmatrix} \mathfrak{M}_V & 0 & 0 \\ \mathfrak{M}_{EV} & \mathfrak{M}_E & 0 \\ \mathfrak{M}_{IV} & \mathfrak{M}_{IE} & \mathfrak{M}_I \end{pmatrix} \quad \Longrightarrow \quad \underline{\mathfrak{u}} = \mathfrak{M} \cdot \underline{\mathfrak{w}} \tag{4}$$

with block-diagonal matrices $\mathfrak{M}_I$, $\mathfrak{M}_E$, $\mathfrak{M}_V$ ensures coherent data stored as a type-I vector. While 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 edges belonging to different sets of subdomains.
   Requirement 1a can be easily fulfilled if at least one node is located on all edges between two vertices. A modification of the given mesh generator can also guarantee requirement 1b, e.g., if the edge between nodes p and n is omitted. The dotted lines in Figure 1 represent the necessary changes in the mesh.
2. As in the previous point, Type-I matrix $\times$ type-II vector cannot be performed by general type-I matrices $\mathfrak{M}$, even if the necessary requirements 1a and 1b are fulfilled by a modified mesh (see Fig. 2). Calculating $\mathsf{M}_{IC} \cdot \underline{\mathfrak{r}}_C$ locally on processor 4 results in

$$f_4^{[r]} \ = \ \mathfrak{M}_{IC,4}^{[r,n]} \cdot \mathsf{r}_4^{[n]} + \mathfrak{M}_{IC,4}^{[r,m]} \cdot \mathsf{r}_4^{[m]} \ .$$

The interior node r should always store the full value, but from a global view the entries $\mathfrak{M}_{IC,2}^{[r,n]} \cdot \mathsf{r}_2^{[n]}$ and $\mathfrak{M}_{IC,2}^{[r,m]} \cdot \mathsf{r}_2^{[m]}$ are missing. So, this operation is not valid. Now, using the sets ω_j from the previous point we have to ensure $\omega_r \supseteqq \omega_j$ $(j \notin \{n, m\})$ to perform the multiplication under consideration, e.g., the entries $\mathfrak{M}^{[k,n]}$ and $\mathfrak{M}^{[k,q]}$ are allowed but not the transposed ones. So, just a matrix of the shape

$$\mathfrak{M} \ = \begin{pmatrix} \mathfrak{M}_V & \mathfrak{M}_{VE} & \mathfrak{M}_{VI} \\ 0 & \mathfrak{M}_E & \mathfrak{M}_{EI} \\ 0 & 0 & \mathfrak{M}_I \end{pmatrix} \quad \Longrightarrow \quad \underline{\mathsf{f}} \ = \sum_{i=1}^{P} A_i^T \underline{\mathsf{f}}_i \ = \sum_{i=1}^{P} A_i^T \left(\mathfrak{M}_i \underline{\mathfrak{r}}_i\right) \tag{5}$$

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

Remark : The factorization of a matrix $\mathfrak{M}$ into a lower and an upper triangular matrix $\mathfrak{L}^{-1}$ and $\mathfrak{U}^{-1}$ can be realized in two ways. Applying it in the preconditioning step $\underline{\mathfrak{w}} = C^{-1}\underline{\mathfrak{r}}$ of CG-like methods results in :

$$\underline{\mathfrak{w}} = \mathfrak{L}^{-1}\mathfrak{U}^{-1} \cdot \underline{\mathfrak{r}} := \mathfrak{L}^{-1} \sum_{i=1}^{P} A_i^T \mathfrak{U}_i^{-1} \cdot \underline{\mathfrak{r}}_i \tag{6a}$$

$$\underline{\mathfrak{w}} = \mathfrak{U}^{-1}\mathfrak{L}^{-1} \cdot \underline{\mathfrak{r}} := \sum_{i=1}^{P} A_i^T \mathfrak{U}_i^{-1} R_i^{-1} A_i \mathfrak{L}^{-1} \cdot (\sum_{j=1}^{P} A_j^T \underline{\mathfrak{r}}_j) \tag{6b}$$

The remaining equations can be easily derived by type conversion.

3 Preconditioners for the Iteration Form

3.1 Type-I Preconditioners

We are in the position to choose a factorized type-I preconditioner from the equations (6a) and (6b). To minimize the required communication, the first one will be investigated in the remaining sections.

Remark : In the symmetric case, the well known ASM-DD-preconditioner [5] fits exactly in the scheme presented by equation (6a). Therein, Schur complement preconditioners proposed in [1,10] are used in which the vertex nodes do not form a block-diagonal matrix. This is no contradiction to the requirements on accumulated matrices because of the usual serial and global handling of the related system of equations.

3.2 Type-II Preconditioners

Applying a type-II preconditioning matrix $\mathsf{C}^{-1} = \sum_{i=1}^{p} A_i^T \mathsf{C}_i^{-1} A_i$ we are in need to convert the vectors twice so that the preconditioning step looks like

$$\underline{\mathfrak{w}} = \mathsf{C}^{-1} \sum_{j=1}^{P} A_j^T \underline{\mathfrak{r}}_j := \sum_{i=1}^{P} A_i^T \left(\mathsf{C}_i^{-1} \cdot A_i \sum_{j=1}^{P} A_j^T \underline{\mathfrak{r}}_j \right) . \tag{7}$$

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 $\partial\Omega_i \backslash \partial\Omega$. If $\partial\Omega_i \cap \Gamma_D = \emptyset$ the matrix K_i may become singular. Therefore we cannot choose C_i in the same way. One opportunity is to assemble the stiffness Matrix K and set $\mathsf{C}_i = \mathfrak{K}_i$ so that the local preconditioner represents a 2nd order PDE in $\widetilde{\Omega}_i$ with homogeneous Dirichlet B.C. on $\partial\widetilde{\Omega}_i$, 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 [9].

4 An Incomplete Factorization Preconditioner

The choice $\mathfrak{C}^{-1} = \mathfrak{L}^{-1} \cdot \mathfrak{U}^{-1}$ (6a) indicates a factorization $\mathfrak{U} \cdot \mathfrak{L}$ 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 are concerned with the classic ILU factorization [3] preserving the given pattern of the matrix and take advantage of the block structure of matrix K (2).
Note that the matrix has to be accumulated at first during the factorization.

Start	K	
Determine		Why parallel ?
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.	$\mathfrak{K}_{EV} := \sum_{i=1}^{P} A_{E,i}^T \mathsf{K}_{EV,i} A_{V,i}$	—
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 (6b)
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

Fig. 2. Parallelized IUL-factorization

Similar to (6a) the application of $\mathfrak{C}^{-1}$ requires communication within the vector type conversion. Figure 3 presents the proper parallel algorithm.

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

Fig. 3. IUL Preconditioning $\mathfrak{U} \cdot \mathfrak{L} \cdot \underline{\mathfrak{w}} = \underline{\mathfrak{r}}$

5 Numerical Results

Due to the restriction of the given code to symmetric problems a parallelized CG-method with an incomplete Cholesky preconditioning, similar to (3), was used in the numerical experiments. The CG iteration stops at a relative accuracy of 10^{-6} measured in the $KC^{-1}K$-norm of the error. The code runs on a 16 processor XPLORER-machine by PARSYTEC with 32 MB per processor. For simplicity we solve the Laplace equation $-\triangle u(x,y) = 1$ with homogeneous Dirichlet B.C. in the unit square as a test example.

Table 1. Constant global problem size : 37.121 degrees of freedom

# processors	# iterations	factorization	CG	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

In Table 1, 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. Moreover, the relation between communication and arithmetic work becomes rather poor due to the fact that in the 16 processor case less then 6% of available memory is used. With respect to the requirements on the mesh, the automatic mesh generator produced different global meshes for different numbers of subdomains. This results in different iteration counts.

Table 2. Constant local problem size

P	#unknowns	# iterations	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

In principle, the scaled speed up should be much better than the classical speed up. Examining the 3 time columns in Table 2 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. 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 have got a value of 3.7 .

6 Conclusions

The IUL-preconditioners proposed in Sect. 4 is one opportunity to code a parallel version of a given solver and works also for multiple degrees of freedom per node. The only restrictions are the requirements 1a and 1b on the mesh which have to be extended in a similar way in the 3D case.

The classical speedup is rather good whereas the scale up is poor due to the growing of the condition number with the number of unknowns. So, the parallelized factorizations are also well suited for use as a smoother in a parallelized global multi-grid method [6]. The overall solution times are approximately 4-10 times longer than for the fastest parallel solvers based on a CG with DD-ASM (Additive Schwarz Method) preconditioners [4] or the parallelized multi-grid method [6]. But in contrast to those multi-level methods we need only one mesh.

In the nonsymmetric case we have to distinguish between strong and weak nonsymmetry. In case of a convection dominated problem the numbering proposed in Sect. 2.1 does not represent the necessary numbering along the stream lines. A decrease in the numerical efficiency has to be expected caused by cutting the stream lines on domain interfaces. On the other hand if the problem is not convection dominated but nonsymmetric then the preconditioners under consideration can be used in the parallelized GMRES. The proper tests will be done in the future.

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. U. 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. G. Haase. Hierarchical extension operators plus smoothing in domain decomposition preconditioners. *Applied Numerical Mathematics*, 23(3):327–346, May 1997.
5. G. Haase, U. Langer, and A. Meyer. The approximate Dirichlet decomposition method. part I,II. *Computing*, 47:137–167, 1991.
6. M. Jung. On the parallelization of multi-grid methods using a non-overlapping domain decomposition data structure. *Applied Numerical Mathematics*, 23(1), 1997.
7. K. H. Law. A parallel finite element solution method. *Computer and Structures*, 23(6):845–858, 1989.
8. A. Meyer. A parallel preconditioned conjugate gradient method using domain decomposition and inexact solvers on each subdomain. *Computing*, 45:217–234, 1990.
9. B. Smith, P. B. rstad, and W. Gropp. *Domain Decomposition : Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 1996.
10. 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.