Technische Universität Chemnitz-Zwickau

DFG-Forschergruppe "SPC" · Fakultät für Mathematik

Michael Jung

Parallelization of multi-grid methods based on domain decomposition ideas

Abstract

In the paper, the parallelization of multi-grid methods for solving second-order elliptic boundary value problems in two-dimensional domains is discussed. The parallelization strategy is based on a non-overlapping domain decomposition data structure such that the algorithm is well-suited for an implementation on a parallel machine with MIMD architecture. For getting an algorithm with a good parallel performance it is necessary to have as few communication as possible between the processors. In our implementation, communication is only needed within the smoothing procedures and the coarse-grid solver. The interpolation and restriction procedures can be performed without any communication.

New variants of smoothers of Gauss-Seidel type having the same communication cost as Jacobi smoothers are presented. For solving the coarse-grid systems iterative methods are proposed that are applied to the corresponding Schur complement system.

Three numerical examples, namely a Poisson equation, a magnetic field problem, and a plane linear elasticity problem, demonstrate the efficiency of the parallel multi-grid algorithm.

Key words: Multi-grid methods, Parallel algorithms, Finite element discretizations

AMS(MOS) subject classification: 65N55, 65Y05, 65N30

Preprint-Reihe der Chemnitzer DFG-Forschergruppe "Scientific Parallel Computing"

SPC 95_27 **November 1995**

1 Introduction

The numerical simulation of complex field problems requires highly efficient algorithms as well as computers with a high CPU power and a large storage capacity. The algorithms used are often based on finite element (FE) discretizations. Since the FE method is closely connected with the domain decomposition (DD) idea, it is very natural to implement the algorithms on multiple instruction multiple data (MIMD) parallel computers with a message passing communication handling. Hereby, the domain Ω in which the field problem is considered will be decomposed into subdomains Ω_i, and the data associated with the subdomains $\bar{\Omega}_i$ are stored on processor P_i. Using such a data distribution, the generation of the FE triangulation (see, e.g., [14]) and the generation of the FE stiffness matrix as well as the load vector can be performed in parallel very well.

The FE discretization leads in general to a large scale system of (non-)linear algebraic equations. The non-linear systems can be solved by utilizing a linearization technique, e.g. a Newton method, such that the efficient solution of large scale linear systems of FE equations is a basic problem. We know that multi-grid (MG) methods are among the most efficient iterative solvers on sequential computers (see, e.g., [19, 21]). Therefore, it is of interest to implement MG methods also on parallel computers in order to get a very fast parallel solver. This topic is discussed extensively in [24] (see also the literature cited therein).

Within a MG algorithm implemented on a parallel machine one has to choose the smoothing procedures and the coarse-grid solver in such a way that the data exchange between the processors, i.e. the communication cost, can be kept as small as possible. Using the data distribution described above, the restriction and the interpolation procedures do not need any communication. Damped point-wise Jacobi smoothers and damped inexact block Jacobi smoothers are described in [4]. Both smoothers can be parallelized very well. The inexact block Jacobi smoothers use point-wise Gauss-Seidel or ILU methods for solving the corresponding block systems approximately. In [30] a smoother of Gauss-Seidel type is discussed, where a block Gauss-Seidel iteration is performed on each processor (subdomain). In this way the processors of two neighbouring subdomains produce different values for the unknowns corresponding to the common part of the subdomain boundary. Therefore, after each smoothing step it is necessary to equalize these values, i.e. an exchange of this data between the corresponding processors must be performed. A parallel SOR (PSOR) smoother is proposed in [35]. Here, the set of mesh points is partitioned into p disjoint sets $\Omega_{h,i}$ which are further divided into three disjoint subsets $\Omega_{h,i}^{(k)}$, $k = 1, 2, 3$, where $\Omega_{h,i}^{(1)}$ and $\Omega_{h,i}^{(3)}$ contain the mesh points of $\Omega_{h,i}$ that are coupled to the mesh points of $\Omega_{h,j}$ with $j < i$ and $j > i$, respectively. First, one SOR sweep applied to the subproblems on $\Omega_{h,i}^{(1)}$, $i = 1, 2, \ldots, p$, is performed in parallel. After a data exchange concerning the mesh points in $\Omega_{h,i}^{(1)}$, one SOR sweep is carried out in parallel for the subproblems on $\Omega_{h,i}^{(2)}$ and $\Omega_{h,i}^{(3)}$. In a last step, the new data corresponding to the mesh points in $\Omega_{h,i}^{(3)}$ are sent to the neighbouring processors. Other possibilities for parallel smoothers are ILU smoothers [5], damped coupled alternating line Gauss-Seidel smoothers [27], smoothers of Chebyshev type [32], and Gauss-Seidel smoothers which use a multicolouring technique (see, e.g., [1]).

In the present paper we discuss the implementation of point-wise Gauss-Seidel smoothers on MIMD computers. These smoothers use a special ordering of the nodes which is typical for algorithms based on DD ideas (for more details see Subsection 3.3.1 and [18]). The most important feature of this implementation is the fact that we need the same communication cost as for a point-wise Jacobi smoother.

The second main problem for getting an efficient parallel MG code is the appropriate choice of the coarse-grid solver. One possibility is a direct solver which runs on one processor called processor P_0. Here, each processor has to send its part of the defect vector to the processor P_0. Then, the coarse-grid system will be solved on processor P_0 and the parts of the solution vector will be sent back to the corresponding processors. For getting a good parallel performance, the MG algorithm is sometimes organized in such a way that on the lower levels a smaller number of processors is involved in the computation process. This leads to a good balance between processing and communication time on each level, but within the restriction and the interpolation procedures communication is necessary [4]. Another possibility for solving the coarse-grid system is the application of an iterative method. This is more attractive in the case of a relatively fine coarse-grid system. In our implementation we want to use a preconditioned Schur complement conjugate gradient method which involves all processors. This is caused by our discretization strategy, where the system of FE equations corresponding to the coarsest mesh is also distributed to all processors. An analogous idea, a dual Schur complement algorithm, for solving the coarse-grid system is discussed in [28].

The present paper is organized as follows: In Section 2 we introduce the considered second-order elliptic boundary value problems in two-dimensional domains and some notation. Section 3 is devoted to the implementation of the MG algorithm on MIMD computers. The construction of the hierarchy of the FE triangulations and the DD data structure used are discussed briefly. The essential part of this paper is the presentation of the parallel implementation of point-wise Gauss-Seidel smoothers which require the same communication cost as point-wise Jacobi smoothers. Furthermore, we show that the interpolation and restriction procedures within the MG algorithm can be performed without any communication, and we analyse the communication cost for different kinds of MG cycles. In Section 4 three numerical examples, a Poisson equation, a magnetic field problem, and a plane linear elasticity problem, are presented. We discuss the parallel performance of the MG method on two kinds of parallel machines, namely on a GC/PP-128 machine and on a GCel-192 system.

2 Finite element discretization of the boundary value problems and remarks on multi-grid algorithms

In this paper we consider second-order elliptic boundary value problems (BVP) in two-dimensional domains. The weak formulation of such a problem can be written in the form:

$$\text{Find } u \in \mathcal{V}_0 \text{ such that} \quad a(u,v) = \langle F, v\rangle \quad \text{for all } v \in \mathcal{V}_0. \tag{1}$$

The space $\mathcal{V}_0$ is a subspace of the Sobolev space $[H^1(\Omega)]^s$ defined by the Dirichlet boundary conditions on $\Gamma_D \subset \partial\Omega$ (meas $\Gamma_D > 0$) which are supposed to be homogenized here. If it is not stated otherwise, as e.g. in Subsection 4.3, the case $s = 1$ is considered.

We suppose that the bilinear form $a(.,.)$ is symmetric, $\mathcal{V}_0$-elliptic, and $\mathcal{V}_0$-bounded. In the right-hand side of the BVP (1), $\langle .,.\rangle : \mathcal{V}_0^* \times \mathcal{V}_0 \rightarrow \mathbb{R}^1$ is the duality pairing, $\mathcal{V}_0^*$ denotes the dual space to $\mathcal{V}_0$, and $F \in \mathcal{V}_0^*$ is a linear and bounded functional on $\mathcal{V}_0$. Examples for such BVPs are presented in Section 4.

Since the parallel MG methods described in Section 3 are based on DD ideas, the discretization process of problem (1) starts with a decomposition of the domain Ω into non-overlapping subdomains Ω_i, i.e.

$$\bar{\Omega} = \bigcup_{i=1}^{p} \bar{\Omega}_i\,, \quad \Omega_i \cap \Omega_{i'} = \emptyset \quad \text{for} \quad i \neq i'\,. \tag{2}$$

In each subdomain Ω_i we generate a sequence of nested FE triangulations $\mathcal{T}_q$, $q = 1, 2, \ldots, l$, consisting of triangular elements. The mesh generation algorithm is realized in such a way that it produces an admissible FE triangulation of the whole domain Ω. A more detailed description of the mesh generation is given in Subsection 3.1.

Corresponding to each triangulation, the FE subspaces $V_q \subset \mathcal{V}_0$, $q = 1, 2, \ldots, l$, are defined by

$$V_q = \mathrm{span}\{\varphi_{q,m}\,,\ m = 1, 2, \ldots, N_q\}\,, \tag{3}$$

where N_q denotes the number of nodes in $\Omega \cup \Gamma_N$ ($\Gamma_N = \partial\Omega \setminus \bar{\Gamma}_D$). The functions $\varphi_{q,m}$ define the usual nodal basis of piecewise linear functions.

The sequence of FE subspaces V_q results in a sequence of FE schemes:

$$\text{Find } u_q \in V_q \text{ such that} \quad a(u_q, v_q) = \langle F, v_q \rangle \quad \text{for all } v_q \in V_q \tag{4}$$

approximating problem (1) on the triangulations $\mathcal{T}_q$, $q = 1, 2, \ldots, l$. These problems can be written in matrix representation, i.e.:

$$\text{Find } \underline{u}_q \in \mathbb{R}^{N_q} \quad \text{such that} \quad K_q \underline{u}_q = \underline{f}_q \tag{5}$$

with $u_q = (\varphi_{q,1}\ \ \varphi_{q,2}\ \ \ldots\ \ \varphi_{q,N_q})\,\underline{u}_q$, the stiffness matrices $K_q = [a(\varphi_{q,n}, \varphi_{q,m})]_{m,n=1}^{N_q}$, and the load vectors $\underline{f}_q = [\langle F, \varphi_{q,m} \rangle]_{m=1}^{N_q}$.

We want to apply a MG algorithm for solving the systems of algebraic FE equations (5). Within this algorithm, the mesh $\mathcal{T}_{q_0}$, $q_0 \geq 1$, will be used as the coarsest mesh such that we get a $(l - q_0 + 1)$–grid algorithm. One iteration step (cycle) of this algorithm is performed in the following way.

One cycle of a $(l - q_0 + 1)$–grid algorithm

Let the kth iterate $\underline{u}_l^{(k,0)}$ be given.

1. Pre-smoothing
$$\underline{u}_l^{(k,1)} = G_l^V(\nu_{l,1}, K_l, \underline{f}_l, \underline{u}_l^{(k,0)})\,.$$

2. Coarse-grid correction

   (a) Compute the defect
   $$\underline{d}_l^{(k)} = \underline{f}_l - K_l \underline{u}_l^{(k,1)}\,.$$

   (b) Restrict the defect to the $(l-1)$–level mesh
   $$\underline{d}_{l-1}^{(k)} = I_l^{l-1} \underline{d}_l^{(k)}\,.$$

   (c) Solve the coarse-grid system
   $$K_{l-1} \underline{w}_{l-1}^{(k)} = \underline{d}_{l-1}^{(k)}$$
   by means of μ_{l-1} iteration steps of a $(l - q_0)$–grid algorithm (with the initial guess 0) if $l - 1 > q_0$, otherwise by means of a preconditioned conjugate gradient algorithm $\Longrightarrow$ $\underline{\widetilde{w}}_{l-1}^{(k)}$.

(d) Interpolate $\underline{\widetilde{w}}_{l-1}^{(k)}$ onto the finer mesh $\mathcal{T}_l$ and correct the approximate $\underline{u}_l^{(k,1)}$

$$\underline{u}_l^{(k,2)} = \underline{u}_l^{(k,1)} + I_{l-1}^l \underline{\widetilde{w}}_{l-1}^{(k)} .$$

3. Post-smoothing

$$\underline{u}_l^{(k,3)} = G_l^N(\nu_{l,2}, K_l, \underline{f}_l, \underline{u}_l^{(k,2)}) .$$

Set $\underline{u}_l^{(k+1,0)} = \underline{u}_l^{(k,3)}$.

In dependence on the choice of the parameters $\nu_{q,1}$, $\nu_{q,2}$, and μ_{q-1}, $q = l, l-1, \ldots, q_0+1$, we get the V-cycle ($\nu_{q,1} = \nu_1$, $\nu_{q,2} = \nu_2$, $\mu_{q-1} = 1$), the W-cycle ($\nu_{q,1} = \nu_1$, $\nu_{q,2} = \nu_2$, $\mu_{q-1} = 2$), and the generalized V-cycle (gV-cycle, $\nu_{q-1,1} = 2\nu_{q,1}$, $\nu_{q-1,2} = 2\nu_{q,2}$, $\mu_{q-1} = 1$). We can also perform the F-cycle, a cycling scheme between the V-cycle and the W-cycle. The F-cycle is defined as follows: For $l = q_0 + 1$ the F- and the V-cycle are identical, for $l \geq q_0 + 2$ first a F-cycle and then a V-cycle are performed for solving the $(l-1)$–level coarse-grid system.

The application of the various MG cycles leads to MG algorithms with different convergence properties. If the solution u of the BVP (1) is regular, i.e. $u \in H^2(\Omega)$, then the convergence rate of the MG V-cycle is independent of the discretization parameter. This convergence result is proved for MG algorithms with different kinds of smoothing procedures, e.g. in [6] with Richardson smoothers, in [3, 7] with general symmetric smoothers, and in [8] with smoothers based on subspace decompositions (point, line, and block versions of Jacobi and Gauss-Seidel iteration). For non-regular solutions u, i.e. $u \in H^{1+\alpha}(\Omega)$ with $0 < \alpha < 1$, the convergence factor of the MG V-cycle approaches one as $1 - \mathcal{O}(l^{(\alpha-1)/\alpha})$ [7, 8]. An analogous result is proved without any assumption on the regularity of the solution u for MG algorithms with Jacobi smoothers in [10] and with SOR smoothers in [34]. In [7, 8] it is shown that the convergence rate of the generalized V-cycle does not depend on the discretization parameter even in the case of a non-regular solution. The same statement is true for the W-cycle [3, 19]. The convergence rate of the F-cycle goes to one as $1 - \mathcal{O}(l^{-(1-\alpha)^2/\alpha})$, i.e. the F-cycle has an asymptotically better convergence factor than the V-cycle (see, e.g., [23]).

3 Parallel multi-grid algorithm

In this Section we describe an implementation of a MG algorithm on MIMD parallel computers with message-passing. The basis of the parallelization strategy is a non-overlapping DD data structure which will be discussed in Subsection 3.2. This data structure allows to perform the MG interpolation and restriction procedures without any communication such that data exchange is needed within the smoothers and the coarse-grid solver only. We present in Subsection 3.3.1 Gauss-Seidel type smoothers which require the same communication cost as Jacobi smoothers but yield in general more robust MG algorithms (see also Section 4). For solving the systems of algebraic FE equations on the coarsest grid, a parallel version of the preconditioned conjugate gradient (PCG) method applied to the corresponding Schur complement system is used, see Subsection 3.5.

3.1 Hierarchical triangulations

We assume that the decomposition (2) of the domain Ω into p non-overlapping subdomains Ω_i is given. Since we want to use MG methods we have to generate a sequence of nested triangular meshes $\mathcal{T}_q$, $q = 1, 2, \ldots, l$. The construction of the coarsest mesh $\mathcal{T}_1$ starts with a decomposition of the coupling boundary $\Gamma_C = \bigcup_{i=1}^{p} \partial\Omega_i$ into so-called basic lines $\Gamma_{C,j}$ ($j = 1, 2, \ldots, j_C$) with $\Gamma_C = \bigcup_{j=1}^{j_C} \bar{\Gamma}_{C,j}$, $\Gamma_{C,j} \cap \Gamma_{C,j'} = \emptyset$ for $j \neq j'$. We suppose that a basic line is either a part of the intersection of a subdomain boundary with the boundary of the domain Ω, $\Gamma_{C,j} \subset \partial\Omega_i \cap \partial\Omega$, or a part of the intersection of the boundaries of two neighbouring subdomains, $\Gamma_{C,j} \subset \partial\Omega_i \cap \partial\Omega_{i'}$ ($i \neq i'$), see also Figure 1.

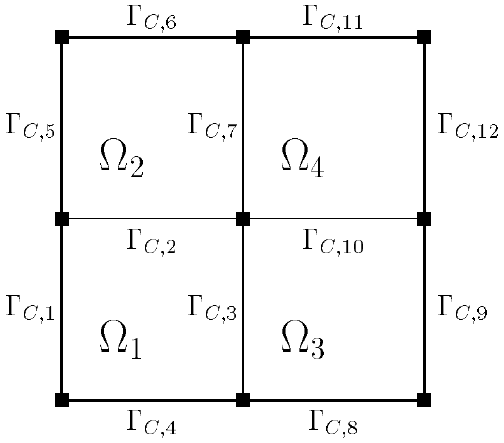


Figure 1: Decomposition of the coupling boundary into basic lines

For each basic line $\Gamma_{C,j}$, $j = 1, 2, \ldots, j_C$, we fix the number of nodes which are to be generated in the triangulation $\mathcal{T}_1$ as well as the method of distribution of the nodes (e.g. equidistant distribution, refinement in direction of the starting or the ending point of the basic line). On the basis of this distribution of nodes, the program PARMESH [14] (see also [15]) generates in parallel FE triangulations of the subdomains Ω_i which result in an admissible FE triangulation of the whole domain Ω. In the present version of this mesh generator the basic lines are assumed to be straight lines, arcs of a circle, or parabolas, respectively. The finer triangulations $\mathcal{T}_q$, $q = 2, 3, \ldots, l$, are obtained by a successive refinement process, i.e. all triangles of the triangulation $\mathcal{T}_{q-1}$ are divided into four smaller subtriangles. This refinement process can be performed in parallel without any communication (see, e.g., [15]).

In each triangulation the nodes are classified into three groups: the *cross-points* (vertices), i.e. the starting and the ending points of the basic lines (marked by the symbol $\blacksquare$ in Figure 1), the *edge coupling nodes*, i.e. the nodes which are generated on the basic lines, and the *inner nodes*.

For the global numbering of the nodes in each triangulation the following order is used: cross-points, edge coupling nodes on $\Gamma_{C,1}$, edge coupling nodes on $\Gamma_{C,2}$, ... , edge coupling nodes on Γ_{C,j_C}, inner nodes of Ω_1, inner nodes of Ω_2, ... , inner nodes of Ω_p. An analogous local numbering of the nodes is employed on each processor.

3.2 Non-overlapping domain decomposition data structure

The numbering of the nodes described in Subsection 3.1 induces the following block structure of the systems (5) of algebraic FE equations

$$\begin{pmatrix} K_{q,V} & K_{q,VE} & K_{q,VI} \\ K_{q,EV} & K_{q,E} & K_{q,EI} \\ K_{q,IV} & K_{q,IE} & K_{q,I} \end{pmatrix} \begin{pmatrix} \underline{u}_{q,V} \\ \underline{u}_{q,E} \\ \underline{u}_{q,I} \end{pmatrix} = \begin{pmatrix} \underline{f}_{q,V} \\ \underline{f}_{q,E} \\ \underline{f}_{q,I} \end{pmatrix}. \tag{6}$$

Here, the indices "V", "E", and "I" correspond to the cross-points (vertices), the edge coupling nodes, and the inner nodes, respectively. The stiffness matrices K_q and the load vectors $\underline{f}_q$ can be represented as sums of super-element (subdomain) stiffness matrices $K_{q,i}$ and super-element load vectors $\underline{f}_{q,i}$. With the $(N_{q,i} \times N_q)$ Boolean matrices $A_{q,i}$ mapping some global vector $\underline{v}_q \in \mathbb{R}^{N_q}$ of nodal variables into the super-element vector $\underline{v}_{q,i} \in \mathbb{R}^{N_{q,i}}$ of variables associated with the subdomain $\bar{\Omega}_i$ only, we get

$$K_q = \sum_{i=1}^{p} A_{q,i}^T K_{q,i} A_{q,i} \qquad \text{and} \qquad \underline{f}_q = \sum_{i=1}^{p} A_{q,i}^T \underline{f}_{q,i}\,. \tag{7}$$

The matrices $K_{q,i}$ and the vectors $\underline{f}_{q,i}$ have the same block structure as given in (6). On each processor P_i only the corresponding super-element stiffness matrix $K_{q,i}$ and the super-element load vector $\underline{f}_{q,i}$ are stored. A consequence of this storage is the following: If we need the elements of the matrix $K_{q,V}$ and the vector $\underline{f}_{q,V}$ or the elements of the matrix $K_{q,E}$ and the vector $\underline{f}_{q,E}$ on a processor we have to perform a summation over cross-points or over edge coupling nodes, respectively. In order to get the elements of the matrix $K_{q,VE}$ also a summation over the edge coupling nodes is necessary. Since each inner node belongs to one processor only, the processor P_i has the full information about the elements of the matrix $K_{q,I,i}$ and the vector $\underline{f}_{q,I,i}$. Figure 2 shows the triangulation of a square, and Figure 3 illustrates the structure of the corresponding matrices $A_{q,i}^T K_{q,i} A_{q,i}$, $i = 1, \dots, 4$. In Figure 4 the part

$$K_{q,C} = \begin{pmatrix} K_{q,V} & K_{q,VE} \\ K_{q,EV} & K_{q,E} \end{pmatrix} \tag{8}$$

of the corresponding matrix K_q is presented. As mentioned above, there are elements of the matrix K_q for which the real value is known on a processor only after a summation over the cross-points or the edge coupling nodes, respectively. This elements are marked with the grey colour. All other elements of the matrix K_q are stored on exactly one processor.

For the implementation of parallel solvers it is convenient to introduce two types of distribution of vectors to the processors P_i (see, e.g., [17, 18]). A vector $\underline{v}_q$ is said to be of *overlapping type* if $\underline{v}_q$ is stored on processor P_i as $\underline{v}_{q,i} = A_{q,i}\underline{v}_q$. A vector $\underline{f}_q$ of *adding type* is stored on processor P_i as $\underline{f}_{q,i}$ such that $\underline{f}_q = \sum_{i=1}^{p} A_{q,i}^T \underline{f}_{q,i}$. For example, in the MG algorithm which we describe in Section 2, the load vector $\underline{f}_l$ and the defect vector $\underline{d}_l^{(k)}$ are of adding type whereas the vectors of the approximate solutions $\underline{u}_l^{(k,\cdot)}$ and the correction vector $\underline{\tilde{w}}_{l-1}^{(k)}$ are of overlapping type.

Within the smoothing procedures and within the coarse-grid solver used in the MG algorithm it is necessary to convert a vector of adding type into a vector of overlapping type (see Subsection 3.3 and Subsection 3.5). This type conversion requires a data exchange of the order $\mathcal{O}(N_q^{0.5})$ between the processors. Here, the communication is realized via a (virtual) hypercube topology and direct links (under PARIX) between processors containing adjacent subdomains. A library of standard communication routines [16] is used. The data exchange for the type conversion of a vector is split into two steps. In the first one, the so-called *cross-point communication*, the components of the vector which correspond to the cross-points are accumulated. This accumulation is realized via forming a cube sum. In the second step the components of the vector corresponding to the edge coupling nodes are accumulated. For the nodes on each part $\Gamma_{C,j}$ of the coupling boundary we have only a data exchange between two neighbouring subdomains, i.e. only a local communication which will be called *communication over the edge coupling nodes.* A more detailed discussion of the communication procedures can be found in [2].

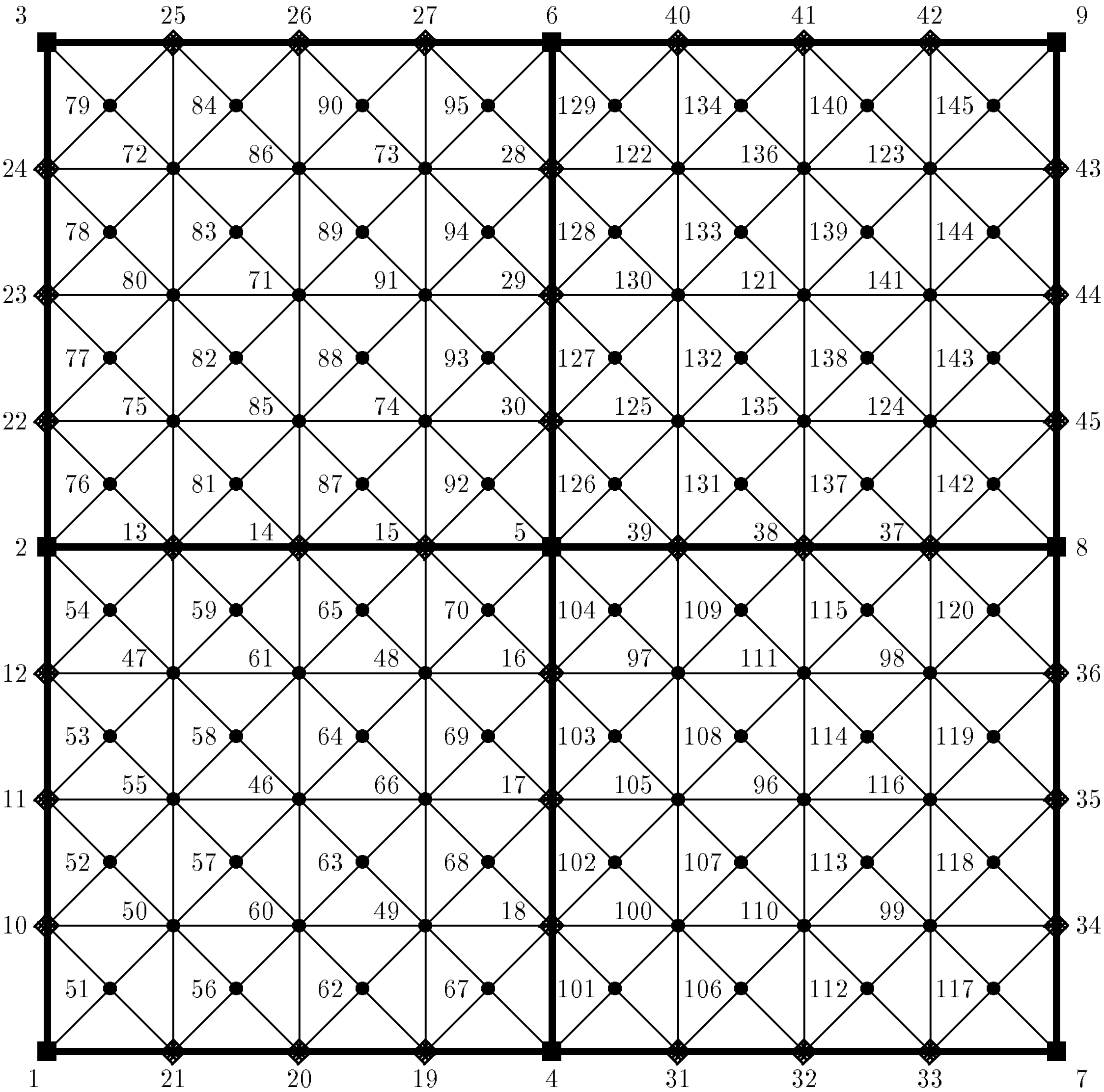


Figure 2: A triangulation of a square

3.3 Smoothing procedures

The implementation of Gauss-Seidel type smoothers and Jacobi smoothers is discussed in this Subsection. Both smoothers are based on the block structure (6) of the systems of algebraic FE equations and need the same communication cost, i.e., they need in each

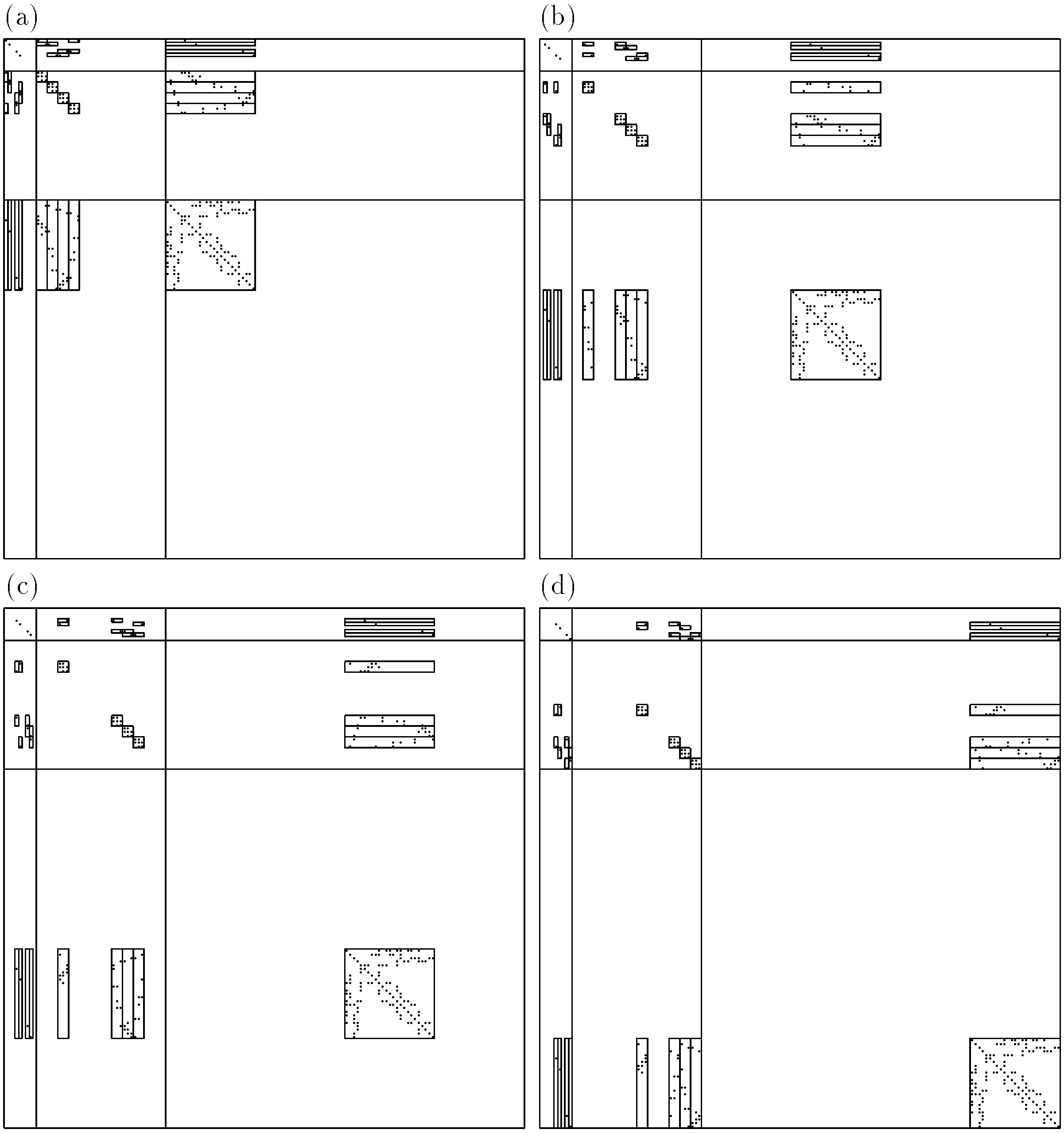


Figure 3: The matrices $A_{q,i}^T K_{q,i} A_{q,i}$ ((a) for $i = 1$, (b) for $i = 2$, (c) for $i = 3$, and (d) for $i = 4$) corresponding to the triangulation in Figure 2

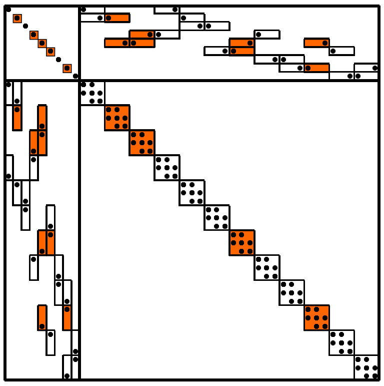


Figure 4: The left upper part $K_{q,C}$ of the matrix K_q

smoothing step the communication cost for the conversion of a vector of adding type into a vector of overlapping type.

To keep the explanation as simple as possible, we describe the smoothers for MG algorithms which are applied to BVPs with a scalar solution u.

In the following it is supposed that at least one edge coupling node is generated on each part $\Gamma_{C,j}$ ($j = 1, 2, \ldots, j_C$) of the coupling boundary and that there exists no edge in the triangulation $\mathcal{T}_1$ connecting nodes on two different parts $\Gamma_{C,j}$ and $\Gamma_{C,j'}$. Under these assumptions the matrices $K_{q,V}$ are diagonal matrices and the matrices $K_{q,E}$ are block-diagonal matrices with tridiagonal blocks. Using the numbering of the nodes described in Subsection 3.2, the matrices $K_{q,I}$ are also block-diagonal matrices with the blocks $K_{q,I,i}$ (see also Figure 3). The blocks $K_{q,I,i}$ have no special structure.

3.3.1 Smoothers of Gauss-Seidel type

One iteration step of a point-wise Gauss-Seidel iteration is defined as follows:

Let the kth iterate $\underline{u}_q^{(k)}$ be given. The new approximate solution $\underline{u}_q^{(k+1)}$ will be computed in the following way:

$$K_{q,V}\underline{u}_{q,V}^{(k+1)} = \underline{f}_{q,V} - K_{q,VE}\underline{u}_{q,E}^{(k)} - K_{q,VI}\underline{u}_{q,I}^{(k)} \tag{9}$$

$$(D_{q,E} + L_{q,E})\,\underline{u}_{q,E}^{(k+1)} = \underline{f}_{q,E} - K_{q,EV}\underline{u}_{q,V}^{(k+1)} - U_{q,E}\underline{u}_{q,E}^{(k)} - K_{q,EI}\underline{u}_{q,I}^{(k)} \tag{10}$$

$$(D_{q,I} + L_{q,I})\,\underline{u}_{q,I}^{(k+1)} = \underline{f}_{q,I} - K_{q,IV}\underline{u}_{q,V}^{(k+1)} - K_{q,IE}\underline{u}_{q,E}^{(k+1)} - U_{q,I}\underline{u}_{q,I}^{(k)}. \tag{11}$$

Here, $L_{q,*}$, $D_{q,*}$, and $U_{q,*}$ ($*$ stands for E and I, respectively) are a strict lower triangular matrix, a diagonal matrix, and a strict upper triangular matrix, respectively, with $K_{q,*} = L_{q,*} + D_{q,*} + U_{q,*}$.

The block structure of the matrices $K_{q,E}$ and $K_{q,I}$ allows a decomposition of the system of algebraic equations (10) into j_C decoupled systems of algebraic equations

$$(D_{q,E,j} + L_{q,E,j})\,\underline{u}_{q,E,j}^{(k+1)} = \underline{f}_{q,E,j} - K_{q,EV,j}\underline{u}_{q,V,j}^{(k+1)} - U_{q,E,j}\underline{u}_{q,E,j}^{(k)} - K_{q,EI,j}\underline{u}_{q,I}^{(k)}, \tag{12}$$

$j = 1, 2, \ldots, j_C$, and a decomposition of the system of equations (11) into p decoupled systems

$$(D_{q,I,i} + L_{q,I,i})\,\underline{u}_{q,I,i}^{(k+1)} = \underline{f}_{q,I,i} - K_{q,IV,i}\underline{u}_{q,V,i}^{(k+1)} - K_{q,IE,i}\underline{u}_{q,E,i}^{(k+1)} - U_{q,I,i}\underline{u}_{q,I,i}^{(k)}, \tag{13}$$

$i = 1, 2, \ldots, p$.

We want to solve the systems of equations (9) – (11) in parallel. The aim is that on each processor P_i only those components of the vectors $\underline{u}_{q,V}^{(k+1)}$, $\underline{u}_{q,E}^{(k+1)}$, and $\underline{u}_{q,I}^{(k+1)}$ are determined, which are associated with the subdomain $\bar{\Omega}_i$. For realizing this aim we need on each processor P_i those parts of the matrices $K_{q,V}$ and $K_{q,E}$ in assembled form which correspond to the nodes in $\bar{\Omega}_i$. As mentioned in Subsection 3.2, this assembly requires communication. Since the diagonal matrix $K_{q,V}$ and the diagonal parts as well as the subdiagonal parts of the symmetric tridiagonal blocks of the matrix $K_{q,E}$ can be handled as vectors of adding type, we have to perform once the cross-point communication which is necessary in the first step of the type conversion of a vector, and twice we need the communication over the edge coupling nodes (see also Subsection 3.2). It is clear that this assembly must be performed for all levels q only once before starting the MG algorithm.

Within each smoothing step the following operations are performed: First on each processor P_i those components of the right-hand side of the system (9) are computed which are associated with the subdomain $\bar{\Omega}_i$. There are representations of the matrices $K_{q,VE}$, $K_{q,VI}$, and the vector $\underline{f}_{q,V}$ which are analogous to (7). Hence, the right-hand side of system (9) is a vector of adding type. After a type conversion into a vector of overlapping type, each processor P_i can determine the components of the vector $\underline{u}_{q,V}^{(k+1)}$ corresponding to $\bar{\Omega}_i$.

Then we have to solve the system of algebraic equations (10). Solving this system of equations in parallel means that on processor P_i those systems (12) are solved which correspond to the parts $\Gamma_{C,j}$, $j \in \mathcal{J}_{C,i}$, of the subdomain boundary $\partial\Omega_i = \cup_{j\in\mathcal{J}_{C,i}}\Gamma_{C,j}$. We compute the vectors

$$\underline{r}_{q,E,j} = \underline{f}_{q,E,j} - K_{q,EV,j}\underline{u}_{q,V,j}^{(k+1)} - K_{q,EI,j}\underline{u}_{q,I,i}^{(k)} ,$$

$j \in \mathcal{J}_{C,i}$, on each processor P_i and convert these vectors $\underline{r}_{q,E,j}$ into vectors $\bar{\underline{r}}_{q,E,j}$ of overlapping type. After this type conversion the systems of equations

$$(D_{q,E,j} + L_{q,E,j})\,\underline{u}_{q,E,j}^{(k+1)} = \bar{\underline{r}}_{q,E,j} - U_{q,E,j}\underline{u}_{q,E,j}^{(k)} ,$$

$j \in \mathcal{J}_{C,i}$, can be solved on processor P_i without any communication.

Each processor P_i has the full information about the right-hand side and the matrix $D_{q,I,i} + L_{q,I,i}$ of the corresponding system of equations (13). Therefore, these systems of equations can be solved in parallel without any communication.

Consequently, in each smoothing step we have only the communication cost which is necessary for the conversion of one vector of adding type into a vector of overlapping type. Of course, we have a small overhead of computational work within this parallel implementation since most components of the vectors $\underline{u}_{q,V}^{(k+1)}$ and $\underline{u}_{q,E}^{(k+1)}$ are computed on two or more processors. But this overhead is only of lower order, i.e. of order $\mathcal{O}(N_q^{0.5})$.

Remark 3.1

(i) Using the same ideas as described above we can define a Gauss-Seidel type smoother which works in reverse order.

(ii) If equation (10) is replaced by

$$K_{q,E}\underline{u}_{q,E}^{(k+1)} = \underline{f}_{q,E} - K_{q,EV}\underline{u}_{q,V}^{(k+1)} - K_{q,EI}\underline{u}_{q,I}^{(k)} \tag{14}$$

we get another variant of a Gauss-Seidel type smoother. System (14) decomposes into j_C decoupled systems of equations

$$K_{q,E,j}\underline{u}_{q,E,j}^{(k+1)} = \underline{f}_{q,E,j} - K_{q,EV,j}\underline{u}_{q,V,j}^{(k+1)} - K_{q,EI,j}\underline{u}_{q,I}^{(k)} ,$$

with tridiagonal matrices $K_{q,E,j}$, $j = 1, 2, \ldots j_C$. For solving these systems of equations, a standard Gauss algorithm for tridiagonal matrices (see, e.g., [29]) will be used. We compare MG algorithms with this smoother and the point-wise Gauss-Seidel smoother in Subsections 4.2 and 4.3.

(iii) If there exist edges in the triangulation $\mathcal{T}_1$ connecting nodes on two different parts $\Gamma_{C,j}$ and $\Gamma_{C,j'}$ of the coupling boundary, then the system (10) can not be decomposed into j_C decoupled systems of algebraic FE equations. In this case we replace equation (10) by the following equations

$$
\begin{aligned}
\underline{r}_{q,E}^{(k+1)} &= \underline{f}_{q,E} - K_{q,EV}\underline{u}_{q,V}^{(k+1)} - K_{q,E}\underline{u}_{q,E}^{(k)} - K_{q,EI}\underline{u}_{q,I}^{(k)} \\
(D_{q,E} + L'_{q,E})\,\underline{w}_{q,E}^{(k+1)} &= \underline{r}_{q,E}^{(k+1)} \\
\underline{u}_{q,E}^{(k+1)} &= \underline{u}_{q,E}^{(k)} + \underline{w}_{q,E}^{(k+1)}\,.
\end{aligned}
\tag{15}
$$

We get the matrix $(D_{q,E} + L'_{q,E})$ from the matrix $(D_{q,E} + L_{q,E})$ by omitting the non-zero elements which result from edges connecting nodes on two different parts of the coupling boundary. Therefore, system (15) can be solved in parallel in the same way as described above for the system (10).

(iv) If the solution u of the BVP (1) is a vector function, i.e. $u \in [H^1(\Omega)]^s$ with $s > 1$, then the smoother is modified. We replace all operations with the elements of the matrix K_q by block operations with the corresponding $(s \times s)$ blocks.

3.3.2 Smoother of Jacobi type

Using again the block structure (6) of the systems of algebraic FE equations, one iteration step of a smoother of Jacobi type can be written in the form:

$$
\begin{aligned}
K_{q,V}\underline{u}_{q,V}^{(k+1)} &= (1-\omega)K_{q,V}\underline{u}_{q,V}^{(k)} - \omega\,(\underline{f}_{q,V} - K_{q,VE}\underline{u}_{q,E}^{(k)} - K_{q,VI}\underline{u}_{q,I}^{(k)}) \\
D_{q,E}\underline{u}_{q,E}^{(k+1)} &= (1-\omega)D_{q,E}\underline{u}_{q,E}^{(k)} - \omega\,(\underline{f}_{q,E} - K_{q,EV}\underline{u}_{q,V}^{(k)} - (L_{q,E}+U_{q,E})\underline{u}_{q,E}^{(k)} - K_{q,EI}\underline{u}_{q,I}^{(k)}) \\
D_{q,I}\underline{u}_{q,I}^{(k+1)} &= (1-\omega)D_{q,I}\underline{u}_{q,I}^{(k)} - \omega\,(\underline{f}_{q,I} - K_{q,IV}\underline{u}_{q,V}^{(k)} - K_{q,IE}\underline{u}_{q,E}^{(k)} - (L_{q,I}+U_{q,I})\underline{u}_{q,I}^{(k)})\,.
\end{aligned}
$$

The smoothing parameter ω has to be chosen in an appropriate way (see, e.g., [31] and the numerical results in Section 4).

If we assemble the matrices $K_{q,V}$ and $D_{q,E}$ before starting the smoothing procedure on the level q, we can implement the Jacobi smoother in a manner analogous to the Gauss-Seidel smoother. Therefore, we have for both smoothers the same communication cost within each smoothing step. Using our non-overlapping DD data structure the described implementations are the best possibilities with respect to the communication cost.

3.4 Interpolation and restriction procedures

Since the BVP (1) is discretized with piecewise linear trial functions, linear interpolation is used within the MG algorithm. The interpolation operator I_{q-1}^{q} maps a vector $\underline{\tilde{w}}_{q-1} \in \mathbb{R}^{N_{q-1}}$ onto a vector $\underline{\tilde{w}}_q \in \mathbb{R}^{N_q}$. All components of the vector $\underline{\tilde{w}}_q$, which are associated with nodes of the triangulation $\mathcal{T}_{q-1}$, are equal to the corresponding components of the vector $\underline{\tilde{w}}_{q-1}$. The components of the vector $\underline{\tilde{w}}_q$ corresponding to the new nodes in the triangulation $\mathcal{T}_q$ are defined as the mean value of those components of the vector $\underline{\tilde{w}}_{q-1}$ which are associated with the father nodes. Since the father nodes of a new node in $\bar{\Omega}_i$ also belong to $\bar{\Omega}_i$ and since the correction vectors $\underline{\tilde{w}}_{q-1}^{(k)}$, $\underline{\tilde{w}}_q^{(k)}$ are stored as vectors of overlapping type (see Subsection 3.2), the interpolation procedure can be performed in parallel without any communication. On each processor P_i we carry out the local interpolation procedure $\underline{\tilde{w}}_{q,i} = I_{q-1,i}^{q}\underline{\tilde{w}}_{q-1,i}$, where the operators $I_{q-1,i}^{q}$, $i = 1, 2, \ldots, p$, are defined in the same way as the operator I_{q-1}^{q}. From the obvious relations

$$
\underline{\tilde{w}}_{q,i} = I_{q-1,i}^{q} A_{q-1,i}\underline{\tilde{w}}_{q-1} \qquad \text{and} \qquad \underline{\tilde{w}}_{q,i} = A_{q,i} I_{q-1}^{q}\underline{\tilde{w}}_{q-1}
$$

we get

$$I_{q-1,i}^{q} A_{q-1,i} = A_{q,i} I_{q-1}^{q} \qquad \text{and} \qquad A_{q-1,i}^{T} (I_{q-1,i}^{q})^{T} = (I_{q-1}^{q})^{T} A_{q,i}^{T} \,. \tag{16}$$

As usual, the restriction operator I_q^{q-1} is defined by $I_q^{q-1} = (I_{q-1}^q)^T$. For the parallel implementation of the restriction procedure we introduce the local restriction operators $I_{q,i}^{q-1}$ as $I_{q,i}^{q-1} = (I_{q-1,i}^q)^T$. Then the operations

$$\underline{d}_{q-1,i} = I_{q,i}^{q-1} \underline{d}_{q,i} \tag{17}$$

are performed simultaneously on the processors P_i. Since the defect vectors $\underline{d}_q$ and $\underline{d}_{q-1}$ are stored as vectors of adding type, we obtain by using relation (16)

$$\underline{d}_{q-1} = \sum_{i=1}^{p} A_{q-1,i}^{T} \underline{d}_{q-1,i} = \sum_{i=1}^{p} A_{q-1,i}^{T} I_{q,i}^{q-1} \underline{d}_{q,i} = \sum_{i=1}^{p} I_{q}^{q-1} A_{q,i}^{T} \underline{d}_{q,i} = I_{q}^{q-1} \sum_{i=1}^{p} A_{q,i}^{T} \underline{d}_{q,i} = I_{q}^{q-1} \underline{d}_{q} \,,$$

i.e. the simultaneous realization of (17) is justified.

3.5 Coarse-grid solvers

We use parallelized PCG methods applied to the corresponding Schur complement system

$$(K_{q_0,C} - K_{q_0,CI} K_{q_0,I}^{-1} K_{q_0,IC})\, \underline{u}_{q_0,C} = \underline{f}_{q_0,C} - K_{q_0,CI} K_{q_0,I}^{-1} \underline{f}_{q_0,I}$$

as coarse-grid solvers. The matrix $K_{q_0,C}$ is defined in (8) with $q = q_0$ and $K_{q_0,CI} = K_{q_0,IC}^T = (K_{q_0,IV} \quad K_{q_0,IE})^T$. Within the coarse-grid solver, communication is required during the computation of the two scalar products and in the preconditioner, whereas all other operations are completely parallel. A non-standard formulation of the PCG method which minimizes the communication between the processors (see [25, 26]) is used. The advantage of this formulation is the fact that the two scalar products per iteration step can be computed immediately one after another. Thus, both values can be sent together and hence communication (start–up time) is reduced. The matrix by vector multiplication with the Schur complement matrix $(K_{q_0,C} - K_{q_0,CI} K_{q_0,I}^{-1} K_{q_0,IC})$ makes use of a Cholesky factorization of the matrices $K_{q_0,I,i}$, $i = 1, 2, \ldots, p$.

There are some possibilities for defining the preconditioner, e.g., we can choose the diagonal part of the matrix K_C, BPX-preconditioners with a global cross-point system [11, 33], or BPS-preconditioners using ideas of Dryja [13] on the coupling boundaries and a global cross-point system (see also [9]). In the case of the BPX and the BPS preconditioners we need a hierarchy of partitions of the coupling boundary. In general, such a hierarchy does not exist for the mesh $\mathcal{T}_{q_0}$. Therefore, we generate a sequence of nested auxiliary meshes down to a mesh which contains the cross-points only. After a mapping between the original mesh on the coupling boundary and the finest auxiliary mesh we perform the BPX- or BPS-algorithm on the sequence of auxiliary meshes.

The preconditioners require communication in the order of the communication cost for the type conversion of a vector on level q_0.

3.6 Analysis of the cost of arithmetical work and the communication cost

Now we want to analyse the cost of arithmetical work and the communication cost for one MG cycle. This cost is given in Table 1. Therein, $W_{q,s}$ denotes the cost of arithmetical work for one smoothing step, $\nu = \nu_1 + \nu_2$, $W_{q,T}$ describes the cost of arithmetical work for the computation of the defect, the restriction procedure, and the interpolation procedure on level q, and W_{q_0} stands for the cost of arithmetical work for solving the coarse-grid system. The communication cost for one smoothing step is denoted by $C_{q,s}$, $q > q_0$, and C_{q_0} stands for the communication cost within the coarse-grid solver. The communication cost $C_{q,s}$ is the cost which we need for the type conversion of a vector (see also Subsection 3.3). The communication cost C_{q_0} depends on the number of iterations of the coarse-grid solver. In each of these iteration steps we have communication in the order of the communication cost for one type conversion of a vector.

	Cost of arithmetical work	Communication cost
V	$\sum_{q=q_0+1}^{l} [\nu W_{q,s} + W_{q,T}] + W_{q_0}$	$\sum_{q=q_0+1}^{l} \nu C_{q,s} + C_{q_0}$
F	$\sum_{q=q_0+1}^{l} (l-q+1)[\nu W_{q,s} + W_{q,T}] + (l-q_0)W_{q_0}$	$\sum_{q=q_0+1}^{l} (l-q+1)\nu C_{q,s} + (l-q_0)C_{q_0}$
gV	$\sum_{q=q_0+1}^{l} [2^{l-q}\nu W_{q,s} + W_{q,T}] + W_{q_0}$	$\sum_{q=q_0+1}^{l} 2^{l-q}\nu C_{q,s} + C_{q_0}$
W	$\sum_{q=q_0+1}^{l} 2^{l-q}[\nu W_{q,s} + W_{q,T}] + 2^{l-q_0-1}W_{q_0}$	$\sum_{q=q_0+1}^{l} 2^{l-q}\nu C_{q,s} + 2^{l-q_0-1}C_{q_0}$

Table 1: Cost of arithmetical work and communication cost for the different MG cycles

Table 1 shows that the V-cycle is the cheapest variant with respect to the cost of arithmetical work and the communication cost. But this cycle has the worst convergence rate. The question arises which cycle will lead to the fastest MG algorithm for getting an approximate solution of the system of equations (5) with a certain accuracy. The answer can be different for implementations on sequential and parallel computers. On parallel machines it is important to have as few communication steps as possible. Therefore, the V- or the F-cycle will give the fastest MG algorithm in many applications (see also Section 4).

4 Numerical results

In this Section, we present three examples which will show the efficiency of the parallel MG algorithm. In order to compare the performance on different multiprocessor systems, we carried out the computations on a GC/PP-128 machine and on a GCel-192 system. The first one is provided with 128 processors of the type PowerPC-601 installed at 64 nodes in a 2D-grid topology. The multiprocessor system GCel-192 works with 192 processors of the type T805. On the GC/PP-128 machine each processor has a memory of 16 MByte and on the other one a memory of 4 MByte.

Our algorithms are implemented in the program FEM⊗BEM [15]. All communication between the processors is realized via a (virtual) hypercube topology using a library of standard communication routines [16].

4.1 Poisson equation

As first example we consider the BVP (1) with

$$a(u,v) = \int_\Omega \nabla^T v\, \nabla u\, dx \qquad \text{and} \qquad \langle F, v\rangle = \int_\Omega 1\, v\, dx$$

in the domain $\Omega = (0,1) \times (0,1)$. The space $\mathcal{V}_0$ is defined by $\mathcal{V}_0 = \{v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega\}$.

The domain Ω is divided into p ($p = 4$, 16, or 64) congruent subdomains. Figure 2 in Subsection 3.2 shows a triangulation of the domain Ω and the decomposition into four subdomains.

In Table 3 we show the performance of the parallel MG algorithm on different numbers of processors and on different multiprocessor systems. The triangulation $\mathcal{T}_{q^*}$ used as coarsest mesh $\mathcal{T}_{q_0}$ within the MG algorithm is obtained by dividing all triangles of the triangulation from Figure 2 into four congruent subtriangles. In this way we have as coarsest mesh $\mathcal{T}_{q_0}$ a triangulation with 545 nodes and 1024 elements. The finer triangulations $\mathcal{T}_q$, $q = q^*+1, \ldots, l$, are constructed by a successive refinement process in which each triangle of the triangulation $\mathcal{T}_{q-1}$ is divided into four congruent subtriangles. Table 2 contains the number of nodes N_q in the triangulations $\mathcal{T}_q$, $q = q^*-2, q^*-1, q^*, \ldots, l$.

level q	q^*-2	q^*-1	q^*	q^*+1	q^*+2	q^*+3	q^*+4	q^*+5	q^*+6
N_q	41	145	545	2113	8321	33025	131585	525313	2099201

Table 2: Number of nodes in the triangulations $\mathcal{T}_q$, $q = q^*-2, q^*-1, \ldots, q^*+6$

In Table 3 we give the number $l_u = q-q^*+1$ of levels used, the number of MG iterations #it, and the total time needed for solving the system of finite element equations on level $l_f = l_u + q^* - 1$. This total time includes the time for the communication, i.e. the time for input, output and waiting of the processors, as well as the time for the processing. The numbers in parentheses are the percentage of the communication time which we measure during the program run. Because of the uniform distribution of the subdomains to the processors all processors have the same load.

Within the MG algorithm two pre- and two post-smoothing steps of the Gauss-Seidel smoother forward described in Subsection 3.3.1 are used. For solving the coarse-grid system, a Schur complement PCG method with a BPS preconditioner is applied, where a coarse-grid solution with a relative accuracy of $\varepsilon_{PCG,q^*} = 0.05$ is computed. In the case of the computation on one processor, the coarse-grid solver is a direct solver based on the Cholesky factorization. The coarse-grid systems have the same size for the computations on 1, 4, 16, or 64 processors, but the number of unknowns of the corresponding Schur complement system is growing with the number of processors, and the size of the matrices $K_{q^*,I,i}$ is decreasing. Therefore, the coarse-grid solvers differ for computations on different numbers of processors. The MG algorithm is started with the initial guess $\underline{u}_{l_f}^{(0,0)} = 0$, and it is terminated when a relative defect of 10^{-6} is achieved. We observe a very fast convergence of the V-cycle such that we do not test the other MG cycles for this example.

The scaled efficiency $S(p_1,p_2)$, $p_1 < p_2$, given in Table 3 is defined by

$$S(p_1,p_2) = \frac{p_1}{p_2} \cdot \frac{T(p_1)}{N(p_1)} \cdot \frac{N(p_2)}{T(p_2)},$$

where $T(p_1)$ ($T(p_2)$) denotes the total time on p_1 (p_2) processors, and $N(p_1)$ ($N(p_2)$) is the total number of unknowns in the computation. We compute the scaled efficiency by using the last line of each column in the Table 3. In the one processor case, we have 5 MG iterations for $l_u = 2$, but 6 MG iterations are needed for $l_u = 3$, $l_u = 4$, and $l_u = 5$ in the cases of 4, 16, and 64 processors, respectively. Therefore, the scaled efficiency is referred to 9.9 sec which would be the time for 6 MG on one processor.

l_u	#it	GCel-192				GC/PP-128			
		Total time [sec] (communication) [%]				Total time [sec] (communication) [%]			
		1 proc.	4 proc.	16 proc.	64 proc.	1 proc.	4 proc.	16 proc.	64 proc.
2	5	8.25	5.41 (3)	1.57 (22)	2.59 (48)	0.19	0.29 (58)	0.52 (90)	1.16 (95)
3	6		12.80 (2)	4.45 (12)	4.48 (47)	0.96	0.63 (46)	0.86 (85)	2.01 (86)
4	6			14.02 (5)	7.54 (35)	3.99	1.55 (24)	1.33 (70)	2.33 (92)
5	6				17.62 (18)		5.00 (10)	2.62 (43)	3.05 (85)
6	6							7.27 (19)	4.59 (66)
7	6								9.51 (37)
$S(1,p)$			0.76	0.69	0.55		0.79	0.54	0.42
$S(p,4p)$		0.76	0.90	0.79		0.79	0.69	0.76	

Table 3: The performance of the parallel MG algorithm on different numbers of processors and different machines

On 1, 4, or 16 processors it is possible to use a coarser triangulation $\mathcal{T}_{q_0}$ than the triangulation $\mathcal{T}_{q^*}$ as coarsest mesh within the MG algorithm. Of course, we can also consider a finer triangulation as coarsest mesh for the MG iteration. In Table 4 the best possible MG algorithm for solving the problem with N unknowns are given. Table 4 contains the total time and the number of levels l_u used. The coarse-grid solution is computed with a relative accuracy of $2^{q^*-q_0}\varepsilon_{PCG,q^*}$.

N	GCel-192				GC/PP-128			
	Total time [sec] / levels				Total time [sec] / levels			
	1 proc.	4 proc.	16 proc.	64 proc.	1 proc.	4 proc.	16 proc.	64 proc.
2113	8.25 / 2	3.49 / 4	1.46 / 3	2.59 / 2	0.18 / 3	0.29 / 2	0.52 / 2	1.16 / 2
8321		12.71 / 5	4.36 / 4	3.59 / 2	0.96 / 3	0.61 / 4	0.73 / 2	1.33 / 2
33025			13.93 / 5	7.41 / 3	3.94 / 6	1.55 / 4	1.31 / 3	1.65 / 2
131585				17.49 / 4		4.73 / 4	2.60 / 4	2.23 / 3
525313							6.77 / 4	4.16 / 3
2099201								9.13 / 5

Table 4: The best possible MG algorithms for solving a system of equation with N unknowns

Table 4 shows that we get the best MG algorithms on the GC/PP-128 machine if we use relatively fine coarse-grid systems. This is caused by the high processing power and the relatively slow communication performance of this machine. From Table 3 we can see that the arithmetical work runs about 30 times faster on the GC/PP-128 machines than on the GCel-192 system, but the communication is only slightly faster.

Next, we investigate the influence of different smoothers on the convergence rate. Here, we give only the results which are obtained on the GC/PP-128 machine with 16 processors. In Table 5, we summarize the numbers of iterations and the total time [sec] needed by the application of the different smoothers. Two pre- and two post-smoothing steps of the Gauss-Seidel forward method are denoted by GS(2f,2f); GS(fb,fb) is the application of one Gauss-Seidel step forward and one Gauss-Seidel step backward in the pre-smoothing as well as in the post-smoothing. The other notations are to be understood in an analogous way. All other components of the MG algorithm are the same as in the experiments given in Table 3. Table 5 shows that we get the best convergence if only the Gauss-Seidel smoother forward or the Gauss-Seidel smoother backward are applied.

	GS(2f,2f)		GS(2b,2b)		GS(2f,2b)		GS(2b,2f)		GS(fb,fb)		GS(bf,bf)	
l_u	#it	time	#it	time	#it	time	#it	time	#it	time	#it	time
2	5	0.52	5	0.52	6	0.63	7	0.73	7	0.74	7	0.73
3	6	0.86	6	0.86	6	0.85	7	1.00	8	1.14	8	1.12
4	6	1.33	6	1.31	6	1.31	8	1.72	8	1.72	8	1.70
5	6	2.62	6	2.77	7	3.13	8	3.56	8	3.55	8	3.55
6	6	7.27	6	7.30	7	8.98	8	10.22	8	10.27	8	10.16

Table 5: Comparison of the different smoothers

4.2 Magnetic field problem

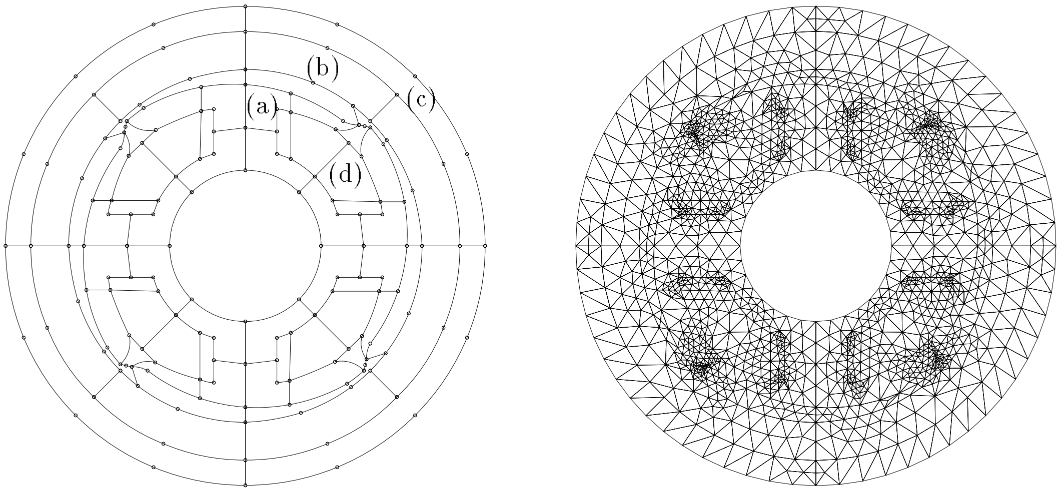
We can describe a non-linear stationary magnetic field problem by means of the BVP (1). Here, the bilinear form $a(.,.)$ and the right-hand side $\langle F, . \rangle$ are defined in the following way:

$$a(u,v) = \int_\Omega \nu(x, |\nabla u|)\, \nabla^T v \nabla u \, dx \quad \text{and} \quad \langle F, v\rangle = \int_\Omega (Sv - H_{0,x_2}\frac{\partial v}{\partial x_1} + H_{0,x_1}\frac{\partial v}{\partial x_2})\, dx\,.$$

The solution u is the x_3-component of the vector potential $\vec{A}$. The x_3-component of the current density is represented by S, and the vector $\vec{H} = (H_{0,x_1}, H_{0,x_2}, 0)^T$ describes the magnetization of permanent magnets. The function $\nu(x, |\nabla u|)$ is constant for not ferromagnetic materials (e.g. copper, air, vacuum), i.e. $\nu(x, |\nabla u|) = \mu_0^{-1}\mu_r^{-1}$ with the absolute permeability μ_0 and the relative permeability μ_r. Properties of the function ν for ferromagnetic materials are formulated in [20].

The non-linearity is handled by means of a nested Newton algorithm, where in each Newton step the parallel MG algorithm is used for solving the corresponding linear problem (see [20]). In the present paper we neglect the non-linearity and assume that the function $\nu(x, |\nabla u|) = \nu(x)$ is constant within each material. In this way we demonstrate the applicability of our MG algorithm to problems with jumps in the coefficient function.

A direct current motor which is excited by permanent magnets serves as test problem. Figure 5 shows the cross-section of this motor and its decomposition into 64 subdomains. Furthermore, the coarsest mesh $\mathcal{T}_{q_0} = \mathcal{T}_1$ (1456 nodes) used within the MG algorithm is given in this figure.



(a) iron $\mu_r = 1353$, (b) permanent magnet $\mu_r = 1.15$,
(c) sheet-metal shell $\mu_r = 1687$, (d) air $\mu_r = 1$

Figure 5: Cross-section of the motor, decomposition into subdomains, and the coarsest mesh

In order to demonstrate the scalability of our MG algorithm we consider an additional test problem, a quarter of the original motor, such that we are in the position to compare problems on 16 and 64 processors.

Again we perform the calculations on the two types of parallel computers. The ingredients of the MG algorithm are the same as described in Subsection 4.1 for the Poisson equation. The computations on 2, 3, ..., 6 levels are summarized in Table 6. The scaled efficiency S in the last line is defined in Subsection 4.1. We compare the application of the BPX and BPS preconditioners within the Schur complement conjugate gradient (SCCG) method for solving the coarse-grid system. In both cases the coarse-grid solution is computed with a relative accuracy $\varepsilon_{PCG} = 10^{-1}$.

#it	GCel-192				GC/PP-128			
	Total time [sec] (communication) [%]				Total time [sec] (communication) [%]			
	BPX-SCCG		BPS-SCCG		BPX-SCCG		BPS-SCCG	
	16 proc.	64 proc.	16 proc.	64 proc.	16 proc.	64 proc.	16 proc.	64 proc.
7	4.59 (35)	18.63 (45)	4.16 (22)	14.62 (26)	1.85 (92)	5.21 (93)	1.18 (87)	3.03 (88)
9	9.57 (30)	29.34 (43)	9.18 (20)	24.16 (27)	2.72 (89)	7.79 (92)	1.92 (84)	4.97 (89)
11	27.62 (16)	53.43 (33)	27.16 (13)	47.19 (22)	4.16 (82)	11.27 (90)	3.21 (77)	7.85 (85)
14	111.18 (7)	145.36 (18)	110.55 (6)	136.88 (12)	8.09 (64)	18.42 (80)	6.83 (57)	13.90 (74)
15					18.47 (36)	30.99 (59)	17.13 (31)	26.11 (52)
S		0.77		0.81		0.60		0.66

Table 6: Comparison of the performance of the parallel MG algorithm on different numbers of processors and different machines

The solution u of this BVP belongs only to a space $H^{1+\alpha}(\Omega)$ with $0 < \alpha < 1$. This leads to the growing number of iterations of the MG algorithm with the V-cycle (see also

Section 2). Table 6 shows that the MG algorithm with the BPS-SCCG coarse-grid solver is slightly faster than the MG algorithm which uses the BPX-SCCG solver. The BPS-SCCG solver needs more iterations than the BPX-SCCG but the communication cost within one iteration step is lower. Therefore, the application of the BPS-SCCG seems to be more favourable.

In Figure 6 we present two bar graphs for the test problem with 16 processors on the two parallel machines. The bar graphs indicate the time proportion between communication and processing. Each bar shows the input time including waiting (left, grey), the output time including waiting (middle, black), and the processing time (right, white), in relation to the total time for each of the 16 processors. Obviously, the processing time differs for different load caused by the decomposition of the domain. Therefore, the processors with less load have to wait for the others. The percentage of the communication time given in Table 6 we get by the relation

$$1 - \frac{\max\limits_{i=1,2,\ldots,p} \{\text{processing time on processor} P_i\}}{\text{total time}}.$$

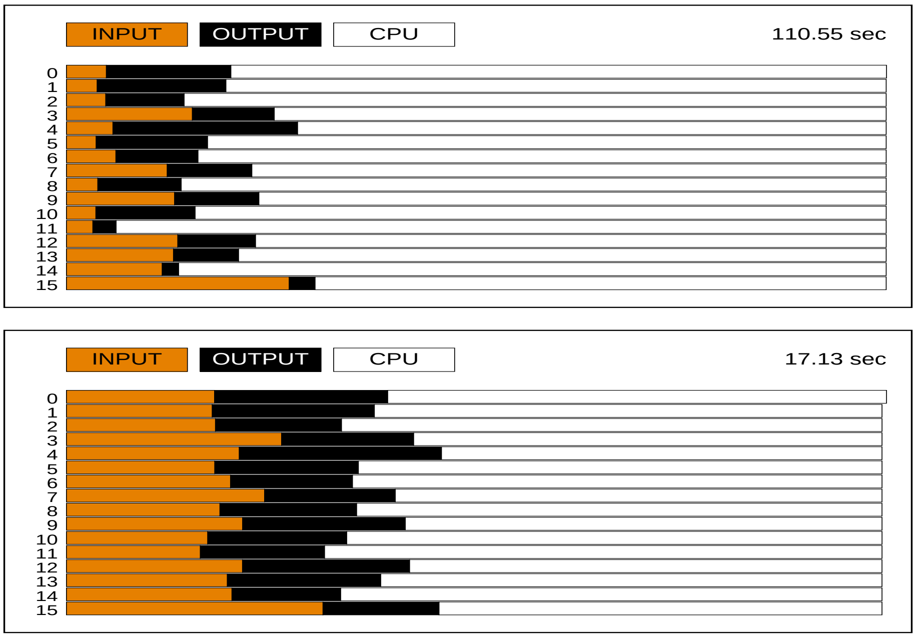


Figure 6: Communication and processing time (top: GCel-192, bottom: GC/PP-128)

From Section 2 we know that the F-cycle, the generalized V-cycle, and the W-cycle, respectively, will produce a MG algorithm which needs fewer iterations than that with the V-cycle. Of course, the application of these MG cycles leads to more arithmetical work within each iteration step. In Table 7 we compare the application of the different MG cycles for the computation on 16 processors. We see that the V-cycle gives the fastest MG algorithm except for the problem with 6 levels, where the F-cycle is faster.

Next, we compare the influence of different smoothers on the convergence rate (see Table 8) of the MG algorithm with $l_u = 2, 3, \ldots, 6$ levels.. In Table 8 we use the same notation as in Table 6. Within the MG algorithm we apply additionally the Jacobi smoother with $\omega = 0.71$ (J(2,2) means two pre- and two post-smoothing steps) and the Gauss-Seidel type smoother described in Remark 3.1 (ii). This smoother is denoted by $\mathrm{GS_b}$(2f,2f). Again the Gauss-Seidel smoother forward leads to the best MG algorithm.

l_u	GCel-192								GC/PP-128							
	V-cycle		gV-cycle		F-cycle		W-cycle		V-cycle		gV-cycle		F-cycle		W-cycle	
	#it	time	#it	time	#it	time	#it	time	#it	time	#it	time	#it	time	#it	time
2	7	4.16	7	4.16	7	4.16	7	4.16	7	1.18	7	1.18	7	1.18	7	1.18
3	9	9.18	8	9.28	7	11.04	7	11.04	9	1.92	8	1.98	7	2.55	7	2.55
4	11	27.16	9	29.21	8	31.22	8	35.26	11	3.21	9	3.90	8	4.88	8	5.99
5	14	110.55	9	100.30	8	92.54	8	111.61	14	6.83	9	8.20	8	8.37	8	12.98
6									15	17.13	9	20.73	8	16.78	8	30.27

Table 7: Comparison of the different MG cycles

GS_b(2f,2f)		GS(2f,2f)		GS(2b,2b)		GS(2f,2b)		GS(2b,2f)		GS(fb,fb)		GS(bf,bf)		J(2,2)	
#it	time	#it	time	#it	time	#it	time	#it	time	#it	time	#it	time	#it	time
7	1.17	7	1.18	7	1.21	9	1.52	8	1.36	9	1.51	11	1.90	22	2.20
9	1.91	9	1.92	9	1.98	10	2.21	10	2.19	12	2.59	12	2.63	16	4.14
11	3.18	11	3.21	12	3.68	12	3.62	13	3.88	16	4.73	15	4.49	21	7.71
14	6.84	14	6.83	15	7.86	15	7.63	16	8.03	19	9.54	18	9.04	27	16.32
15	17.04	15	17.13	17	21.30	17	20.14	17	20.09	20	23.53	19	22.50	30	39.72

Table 8: Comparison of the different smoothers

In Table 7 we see that the convergence rate of the MG V-cycle is relatively slow. We know that MG algorithms can be used for defining preconditioners of the PCG method implicitly (see, e.q., [22]). Table 9 shows results obtained by means of a parallel PCG method applied to the systems of algebraic FE equation (5). Within the PCG method the preconditioner is defined by one MG V-cycle with two Gauss-Seidel steps backward in the pre-smoothing and two Gauss-Seidel steps forward in the post-smoothing as well as the BPS-SCCG method as coarse-grid solver. The resulting algorithm is called MG(1)-PCG method.

#it	GCel-192				GC/PP-128			
	Total time [sec] (communication) [%]				Total time [sec] (communication) [%]			
	BPX-SCG		BPS-SCG		BPX-SCG		BPS-SCG	
	16 proc.	64 proc.	16 proc.	64 proc.	16 proc.	64 proc.	16 proc.	64 proc.
6	4.49 (35)	17.94 (44)	3.81 (22)	13.06 (25)	1.80 (91)	4.97 (93)	1.05 (87)	2.65 (88)
7	8.40 (27)	24.90 (41)	7.76 (19)	19.50 (26)	2.29 (89)	6.51 (92)	1.52 (83)	3.99 (88)
8	23.37 (14)	42.24 (30)	22.41 (11)	35.90 (20)	3.26 (81)	8.42 (89)	2.35 (74)	5.62 (84)
8	76.29 (6)	95.29 (16)	75.48 (5)	88.85 (10)	5.12 (59)	10.97 (78)	4.28 (52)	8.17 (71)
8					11.69 (31)	18.28 (54)	10.86 (26)	15.49 (46)
S		0.81		0.86		0.65		0.71

Table 9: Comparison of the performance of the parallel MG(1)-PCG algorithm on different numbers of processors and different machines

4.3 Plane linear elasticity problem

In this Subsection we want to compute the displacement field $u = (u_1, u_2)^T \in \mathcal{V}_0$ ($\mathcal{V}_0 = \{u \in [H^1(\Omega)]^2 \,:\, u = 0 \text{ on } \Gamma_D\}$) which is caused by a surface traction $g_N = (g_{N,1}, g_{N,2})^T$. The displacement field u is the solution of the BVP (1), where

$$a(u,v) = \int\limits_{\Omega} e^T(v)\,\sigma(u)\,dx \qquad \text{and} \qquad \langle F, v\rangle = \int\limits_{\Gamma_N} v^T g_N\,ds$$

with $e(v) = (\varepsilon_{11}(v), \varepsilon_{22}(v), 2\varepsilon_{12}(v))^T$, $\sigma(u) = (\sigma_{11}(u), \sigma_{22}(u), \sigma_{12}(u))^T$, and $\sigma(u) = De(u)$. The components ε_{ij}, $i,j = 1,2$, of the strain tensor are defined by

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \quad \text{and} \quad D = \frac{E}{1-\nu^2}\begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2}\end{pmatrix}$$

for the state of plane stress. Here, E denotes the Young's elasticity modulus, and ν is the Poisson's ratio.

In our example we have $E = 1000.$ and $\nu = 0.3$. The domain Ω is shown in Figure 7 (see also [12]).

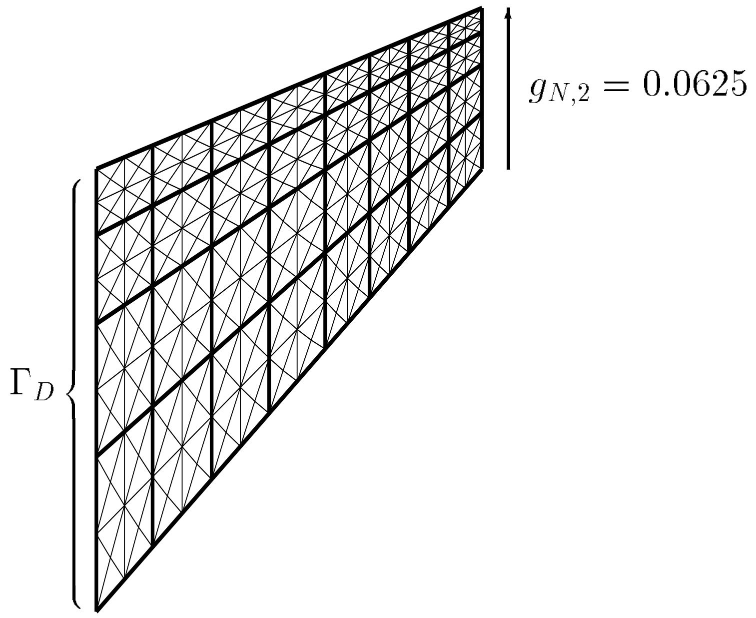


Figure 7: The decomposition of the domain Ω into 32 subdomains and a triangulation

We want to analyse the scalability of the parallel MG algorithm. In our computations the discretized BVP is solved on 1, 2, 8, and 32 processors. Figure 7 shows the decomposition of the domain Ω into 32 subdomains. The decomposition into 2 or 8 subdomains is made in an analogous way. The coarsest triangulation $\mathcal{T}_{q_0}$ used within the MG algorithm is also illustrated in Figure 7. This triangulation contains 512 elements and 281 nodes. Table 10 contains the number of nodes N_q in the triangulations $\mathcal{T}_q$, $q = q_0, q_0+1, \ldots, l$.

Since on each processor the same number of nodes is stored all processors have the same load.

In Table 11 we give the number $l_u = q - q_0 + 1$ of levels used, the number of MG iterations #it, and the total time needed for solving the system of finite element equations

level q	q_0	q_0+1	q_0+2	q_0+3	q_0+4	q_0+5
N_q	281	1073	4193	16577	65921	262913

Table 10: Number of nodes in the triangulations $\mathcal{T}_q$, $q = q_0, q_0+1, \ldots, q_0+5$

on level $l_f = l_u + q_0 - 1$. This total time includes the time for the communication, i.e. the time for input, output and waiting of the processors, as well as the time for processing. The numbers in parentheses are the percentage of the communication time which we measure during the program run. The scaled efficiency given in the last two lines of Table 11 is defined in Subsection 4.1.

Within the MG algorithm two pre- and two post-smoothing steps of the Gauss-Seidel smoother forward described in Subsection 3.3.1 are used. Since the solution u of the BVP (1) is a vector function, the Gauss-Seidel smoother which uses operations with (2×2) blocks (see Remark 3.1 (iv)) is applied. The coarse-grid solver is a Schur complement PCG method with the BPS preconditioner, and the coarse-grid solutions are computed with a relative accuracy of $\varepsilon_{PCG,q_0} = 0.05$. In the case of the computation on one processor, the coarse-grid solver is a direct solver based on the Cholesky factorization. The coarse-grid systems have the same size for the computations on 1, 2, 8, or 32 processors, but the number of unknowns of the corresponding Schur complement system is growing with the number of processors, and the size of the matrices $K_{q_0,I.i}$ is decreasing. Therefore, the coarse-grid solvers differ for computations on different numbers of processors. The MG algorithm (V-cycle) is started with the initial guess $\underline{u}_{l_f}^{(0,0)} = 0$, and it is terminated when a relative defect of 10^{-6} is achieved.

l_u	#it	GCel-192				GC/PP-128			
		Total time [sec] (communication) [%]				Total time [sec] (communication) [%]			
		1 proc.	2 proc.	8 proc.	32 proc.	1 proc.	2 proc.	8 proc.	32 proc.
2	16	41.64	76.21 (1)	15.14 (10)	15.44 (27)	1.41	2.30 (25)	2.30 (78)	4.11 (90)
3	20		106.38 (1)	38.74 (6)	26.97 (26)	5.86	5.24 (16)	3.90 (67)	6.43 (89)
4	22			124.74 (2)	53.26 (18)		15.63 (7)	7.71 (43)	9.02 (83)
5	23				143.64 (8)			21.23 (19)	13.94 (65)
6	23								28.09 (37)
$S(1,p)$			0.76	0.64	0.56		0.74	0.54	0.41
$S(p_1,p_2)$		0.76	0.84	0.86		0.79	0.73	0.75	

Table 11: The performance of the parallel MG algorithm on different numbers of processors and different machines

Table 11 shows that the scalability on the GCel-192 system is slightly better than on the GC/PP-128 machine. From this table we can see that the convergence rate of the MG V-cycle is relatively slow. In order to get an algorithm with a better convergence we can employ other MG cycles or we can use one MG V-cycle for defining a preconditioner in the PCG method. In this case we use two Gauss-Seidel steps backward in the pre-smoothing and two Gauss-Seidel steps forward in the post-smoothing. The resulting PCG method is called MG(1)-PCG method. Both possibilities for getting algorithms with better convergence properties are discussed in the following. At first, we investigate the convergence properties of the generalized V-cycle, the F-cycle, and the W-cycle. Table 12 shows that these cycles give a MG algorithm with a faster convergence than that with the V-cycle, but the total time needed on both multiprocessor systems is higher than that of the V-cycle. The computations for Table 12 are performed on 32 processors.

l	GCel-192								GC/PP-128							
	V-cycle		gV-cycle		F-cycle		W-cycle		V-cycle		gV-cycle		F-cycle		W-cycle	
	#it	time	#it	time	#it	time	#it	time	#it	time	#it	time	#it	time	#it	time
2	16	15.44	16	15.44	16	15.44	16	15.44	16	4.11	16	4.11	16	4.11	16	4.11
3	20	26.97	18	26.62	18	40.25	18	40.25	20	6.43	18	6.61	18	9.89	18	9.89
4	22	53.26	18	55.72	18	82.13	18	97.37	22	9.02	18	10.74	18	16.68	18	20.66
5	23	143.64	18	161.93	18	191.19	18	261.44	23	13.94	18	21.22	18	26.76	18	44.24
6									23	28.19	18	48.57	18	47.47	18	98.31

Table 12: Comparison of the different MG cycles

In Table 13 we summarize some results concerning the MG(1)-PCG method. Comparing Tables 11 and 13 we can see that the MG(1)-PCG algorithm converges faster than the MG algorithm. Furthermore, the MG(1)-PCG method has a better scalability.

l_u	#it	GCel-192				GC/PP-128			
		Total time [sec] (communication) [%]				Total time [sec] (communication) [%]			
		1 proc.	2 proc.	8 proc.	32 proc.	1 proc.	2 proc.	8 proc.	32 proc.
2	8	35.73	49.55 (1)	8.30 (8)	7.92 (26)	0.97	1.23 (32)	1.06 (77)	2.07 (90)
3	9		110.20 (1)	23.68 (4)	13.30 (23)	3.19	2.50 (14)	1.67 (64)	2.87 (88)
4	9			80.80 (1)	28.35 (14)	11.96	7.10 (6)	3.25 (37)	3.67 (81)
5	9				86.42 (6)			9.07 (16)	5.64 (62)
6	9								11.97 (34)
$S(1,p)$			0.63	0.85	0.79		0.89	0.69	0.52
$S(p_1,p_2)$		0.63	1.34	0.93		0.89	0.78	0.75	

Table 13: The performance of the parallel MG(1)-PCG algorithm on different numbers of processors and different machines

Finally, we analyse the influence of different smoothers on the convergence rate of the V-cycle. In Table 14 we use the notation introduced in Table 5 (Subsection 4.1) and Table 8 (Subsection 4.2) . Since we have in this example a vector function u as solution of the BVP, we apply the block version of the smoothers (see Remark 3.1 (iv)). The smoothing parameter ω for the Jacobi smoother is 0.96. We determined this parameter experimentally. All results given in Table 14 are obtained on 32 processors. Obviously, we get the best algorithm if we use only the Gauss-Seidel smoother forward or the Gauss-Seidel smoother backward. Furthermore, we see that the smoother $\mathrm{GS_b}$(2f,2f) described in Remark 3.1 (ii) gives a slightly faster MG algorithm than the application of the smoother GS(2f,2f).

Further numerical examples can be found in [20], where the parallel MG algorithm is also compared with parallel DD PCG methods.

l	GS_b(2f,2f)		GS(2f,2f)		GS(2b,2b)		GS(2f,2b)		GS(2b,2f)		GS(fb,fb)		J(2,2)	
	#it	time	#it	time	#it	time	#it	time	#it	time	#it	time	#it	time
2	16	4.07	16	4.11	18	4.62	22	5.64	21	5.46	27	6.96	32	9.55
3	20	6.36	20	6.43	21	6.53	24	7.54	24	7.62	31	9.72	39	15.42
4	22	8.90	22	9.02	22	8.57	25	9.98	26	10.40	31	12.28	42	21.75
5	23	13.72	23	13.94	23	13.33	26	15.38	27	15.95	32	18.82	44	32.51
6	23	28.17	23	29.72	23	27.50	26	31.35	27	32.62	32	38.44	44	57.23

Table 14: Comparison of the different smoothers

5 Conclusions

Based on a non-overlapping DD data structure a uniform concept for the parallel generation of the hierarchy of FE triangulations, the generation of the systems of FE equations, and the solution algorithm is presented.

The data structure used allows to parallelize Jacobi smoothers and point-wise Gauss–Seidel smoothers very well. Hereby, the Gauss–Seidel smoother makes use of the block structure of the system of FE equations induced by the classification of the nodes into cross-points, edge coupling nodes, and inner nodes. This idea can be extended easily to the 3D case, where we have additionally face coupling nodes.

As coarse-grid solvers within the MG algorithm, PCG methods applied to the Schur complement system are proposed. These solvers benefit by the good convergence properties of well-known DD algorithms.

The parallel MG algorithm is implemented on two kinds of MIMD computers, namely on a GC/PP-128 machine and on a GCel-192 system. Numerical examples demonstrate the performance of the MG algorithm on both multiprocessor systems. The computations show that one should distribute the data to the processors in such a way that the storage capacity of the processors is utilized fully. Then we get the best proportion between processing and communication time.

References

[1] M. Alef. Concepts for efficient multigrid implementation on SUPRENUM-like architectures. *Parallel Comput.*, 17(1):1–16, 1991.

[2] Th. Apel, G. Haase, A. Meyer, and M. Pester. Parallel solution of finite element equation systems: efficient inter-processor communication. Preprint SPC 95_5, Technische Universität Chemnitz-Zwickau, Fakultät für Mathematik, 1995.

[3] R. E. Bank and C. C. Douglas. Sharp estimates for multigrid rates of convergence with general smoothing and acceleration. *SIAM J. Numer. Anal.*, 22(4):617–633, 1985.

[4] P. Bastian. *Parallele adaptive Mehrgitterverfahren.* PhD thesis, Universität Heidelberg, 1994.

[5] P. Bastian and G. Horton. Parallelization of robust multigrid methods: ILU factorization and frequency decomposition method. *SIAM J. Sci. Stat. Comput.*, 12(6):1457–1470, 1991.

[6] D. Braess and W. Hackbusch. A new convergence proof for multigrid methods including the V–cycle. *SIAM J. Numer. Anal.*, 20:967–975, 1983.

[7] J. H. Bramble and J. E. Pasciak. New convergence estimates for multigrid algorithms. *Math. Comput.*, 49:311–329, 1987.

[8] J. H. Bramble and J. E. Pasciak. The analysis of smoothers for multigrid algorithms. *Math. Comput.*, 58:467–488, 1992.

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

[10] J. H. Bramble, J. E. Pasciak, J. Wang, and J. Xu. Convergence estimates for multigrid algorithms without regularity assumptions. *Math. Comput.*, 57:23–45, 1991.

[11] J. H. Bramble, J. E. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Math. Comput.*, 55:1–22, 1990.

[12] R. D. Cook. Ways to improve the bending response of finite elements. *Int. J. Numer. Methods Eng.*, 11:1029–1039, 1977.

[13] M. Dryja. A capacitance matrix method for Dirichlet problems on polygonal regions. *Numer. Math.*, 39(1):51–64, 1982.

[14] G. Globisch. PARMESH - a parallel mesh generator. *Parallel Comput.*, 21(3):509–524, 1995.

[15] G. Haase, B. Heise, M. Jung, and M. Kuhn. FEM$\otimes$BEM – A parallel solver for linear and nonlinear coupled FE/BE–equations. Report 96–16, DFG–Schwerpunkt Randelementmethoden, 1994.

[16] G. Haase, Th. Hommel, A. Meyer, and M. Pester. Bibliotheken zur Entwicklung paralleler Algorithmen (3rd edition). Preprint SPC 95_20, Technische Universität Chemnitz-Zwickau, Fakultät für Mathematik, 1995.

[17] G. Haase, U. Langer, and A. Meyer. The approximate Dirichlet domain decomposition method. Part I: An algebraic approach. Part II: Applications to 2nd-order elliptic boundary value problems. *Computing*, 47:137–151 (Part I), 153–167 (Part II), 1991.

[18] G. Haase, U. Langer, and A. Meyer. Parallelisierung und Vorkonditionierung des CG-Verfahrens durch Gebietszerlegung. In G. Bader, R. Rannacher, and G. Wittum, editors, *Parallele Algorithmen auf Transputersystemen*, Teubner-Scripten zur Numerik III, pages 80–116. Teubner–Verlag Stuttgart, 1992. Tagungsbericht der GAMM-Tagung, 31. Mai - 1. Juni 1991, Heidelberg.

[19] W. Hackbusch. *Multi-Grid Methods and Applications*, volume 4 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1985.

[20] B. Heise and M. Jung. Comparison of parallel solvers for nonlinear elliptic problems based on domain decomposition ideas. Institutsbericht Nr. 494, Johannes Kepler Universität Linz, Institut für Mathematik, 1995.

[21] M. Jung and U. Langer. Applications of multilevel methods to practical problems. *Surv. Math. Ind.*, 1:217–257, 1991.

[22] M. Jung, U. Langer, A. Meyer, W. Queck, and M. Schneider. Multigrid preconditioners and their applications. In G. Telschow, editor, *Third Multigrid Seminar, Biesenthal 1988*, pages 11–52, Berlin, 1989. Karl-Weierstraß-Institut. Report R–MATH–03/89.

[23] J. Mandel and S. V. Parter. On the multigrid F-cycle. *Appl. Math. Comput.*, 37:19–36, 1990.

[24] O. A. McBryan, P. O. Frederickson, J. Linden, A. Schüller, K. Solchenbach, K. Stüben, C. A. Thole, and U. Trottenberg. Multigrid methods on parallel computers – a survey of recent developments. *IMPACT Comput. Sci. Eng.*, 3:1–75, 1991.

[25] M. Meisel and A. Meyer. Implementierung eines parallelen Schur–Komplement CG–Verfahrens in das Programmpaket FEAP. Preprint SPC 95_2, Technische Universität Chemnitz-Zwickau, Fakultät für Mathematik, 1995.

[26] M. Meisel and A. Meyer. Kommunikationstechnologien beim parallelen vorkonditionierten Schur–Komplement CG–Verfahren. Preprint SPC 95_19, Technische Universität Chemnitz-Zwickau, Fakultät für Mathematik, 1995.

[27] K. Oosterlee. The convergence of parallel multiblock multigrid methods. Arbeitspapiere der GMD 850, Gesellschaft für Mathematik und Datenverarbeitung mbH, Sankt Augustin, 1994.

[28] F.-X. Roux and D. Tromeur-Dervout. Parallelization of a multigrid solver via a domain decomposition method. In D. E. Keyes and J. Xu, editors, *Domain Decomposition Methods in Sientific and Engineering Computing*, pages 439–444. The Pennsylvania State University, 1994. Proceedings of the 7th International Conference on Domain Decomposition, October 27–30, 1993.

[29] A. A. Samarskij and E. S. Nikolajev. *Numerical Methods for Grid Equations. Vol. I: Direct Methods*. Birkhäuser, Basel Boston Berlin, 1989.

[30] F. Schieweck. A parallel multigrid algorithm for solving the Navier-Stokes equations. *IMPACT Comput. Sci. Eng.*, 5:345–378, 1993.

[31] K. Stüben and U. Trottenberg. Multigrid methods: Fundamental algorithms, model problem analysis and applications. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods, Proceedings of the Conference held at Köln–Porz, November 23–27, 1981*, volume 960 of *Lecture Notes in Mathematics*, pages 1–176, Berlin–Heidelberg–New York, 1982. Springer–Verlag.

[32] M. Theß. Untersuchungen zur Parallelisierung von Mehrgitterverfahren für elliptische Randwertprobleme in dreidimensionalen Gebieten. Diplomarbeit, Technische Universität Chemnitz-Zwickau, Fachbereich Mathematik, 1994.

[33] C. H. Tong, T. F. Chan, and C. C. J. Kuo. A domain decomposition preconditioner based on a change to a multilevel nodal basis. *SIAM J. Sci. Stat. Comput.*, 12(6):1486–1495, 1991.

[34] J. Wang. Convergence analysis without regularity assumptions for multigrid algorithms based on SOR smoothing. *SIAM J. Numer. Anal.*, 29(4):987–1001, 1992.

[35] D. Xie. New parallel SOR methods by domain partitioning. *SIAM J. Sci. Comput.* (to appear).

Author's address:

Dr. rer. nat. Michael Jung
Technische Universität Chemnitz-Zwickau
Fakultät für Mathematik
D – 09107 Chemnitz

e-mail: dr.michael.jung@mathematik.tu-chemnitz.de

Other titles in the SPC series:

95_13 A. Meyer, D. Michael. Some remarks on the simulation of elasto-plastic problems on parallel computers. March 1995

95_14 B. Heinrich, S. Nicaise, B. Weber. Elliptic interface problems in axisymmetric domains. Part I: Singular functions of non-tensorial type. April 1995

95_15 B. Heinrich, B. Lang, B. Weber. Parallel computation of Fourier-finite-element approximations and some experiments. May 1995

95_16 W. Rath. Canonical forms for linear descriptor systems with variable coefficients. May 1995

95_17 C. He, A. J. Laub, V. Mehrmann. Placing plenty of poles is pretty preposterous. May 1995

95_18 J. J. Hench, C. He, V. Kučera, V. Mehrmann. Dampening controllers via a Riccati equation approach. May 1995

95_19 M. Meisel, A. Meyer. Kommunikationstechnologien beim parallelen vorkonditionierten Schur-Komplement CG-Verfahren. Juni 1995

95_20 G. Haase, T. Hommel, A. Meyer and M. Pester. Bibliotheken zur Entwicklung paralleler Algorithmen. Juni 1995.

95_21 A. Vogel. Solvers for Lamé equations with Poisson ratio near 0.5. June 1995.

95_22 P. Benner, A. J. Laub, V. Mehrmann. A collection of benchmark examples for the numerical solution of algebraic Riccati equations I: Continuous-time case. October 1995.

95_23 P. Benner, A. J. Laub, V. Mehrmann. A collection of benchmark examples for the numerical solution of algebraic Riccati equations II: Discrete-time case. to appear: December 1995.

95_24 P. Benner, R. Byers. Newton's method with exact line search for solving the algebraic Riccati equation. October 1995.

95_25 P. Kunkel, V. Mehrmann. Local and Global Invariants of Linear Differential-Algebraic Equations and their Relation. July 1995.

95_26 C. Israel. NETGEN69 - Ein hierarchischer paralleler Netzgenerator. August 1995.

95_27 M. Jung. Parallelization of multigrid methods based on domain decomposition ideas. September 1995.

95_28 P. Benner, H. Faßbender. A restarted symplectic Lanczos method for the Hamiltonian eigenvalue problem. October 1995.

95_29 G. Windisch. Exact discretizations of two-point boundary value problems. October 1995.

95_30 S. V. Nepomnyaschikh. Domain decomposition and multilevel techniques for preconditioning operators. November 1995.

95_31 H. Matthes. Parallel preconditioners for plate problems. November 1995.

Some papers can be accessed via anonymous ftp from server `ftp.tu-chemnitz.de`, directory `pub/Local/mathematik/SPC`. (Note the capital L in Local!)
The complete list of current and former preprints is available via
`http://www.tu-chemnitz.de/~pester/sfb/spc95pr.html`.