Parallel Multigrid 3D Maxwell Solvers [1]

Gundolf Haase [a] Michael Kuhn [a] Ulrich Langer [a]

[a] *Johannes Kepler University of Linz,*
Institute of Analysis and Computational Mathematics,
Altenberger Str. 69, A–4040 Linz, Austria

Abstract

3D magnetic field problems are challenging not only because of interesting applications in the industry but also from the mathematical point of view. In the magnetostatic case, our Maxwell solver is based on a regularized mixed variational formulation of the Maxwell equations in $\mathrm{H}_0(\mathbf{curl}) \times \mathrm{H}_0^1(\Omega)$ and their discretization by the Nédélec and Lagrange finite elements. Eliminating the Lagrange multiplier from the mixed finite element equations, we arrive at a symmetric and positive definite (spd) problem that can be solved by some parallel multigrid preconditioned conjugate gradient (pcg) method. More precisely, this pcg solver contains a standard scaled Laplace multigrid regularizator in the regularization part and a special multigrid preconditioner for the regularized Nédélec finite element equations that we want to solve. The parallelization of the pcg algorithm, the Laplace multigrid regularizator and the multigrid preconditioner are based on a unified domain decomposition data distribution concept. We present the results of some numerical experiments on a parallel machine with distributed memory that show the high efficiency of our approach for real-life applications.

Keywords : 3D Maxwell Equations, Magnetostatics, Finite Element Method, Domain Decomposition Data Distribution, Parallel Iterative Solvers, Parallel Multigrid Preconditioners, Distributed Memory.

1 Introduction

The computer simulation of 3D technical magnetic field problems often requires powerful finite element solvers due to the geometrical and physical complexity. Moreover, advanced problems such as nonlinear or/and transient

[1] This research has been supported by the Austrian Science Fund FWF within the SFB "Numerical and Symbolic Scientific Computing" under the grant SFB F013

problems, as well as coupled magneto-mechanical field problems and optimal design problems require the solution of linearized problems several times (see, e.g. [16]). The availability of fast and efficient solvers is crucial especially in such cases.

The most efficient solvers for finite element equations are certainly multigrid, or multilevel methods. Typically, the multigrid convergence rate is independent of the mesh size parameter, and the arithmetical complexity grows linearly with the number of unknowns. However, the standard multigrid algorithms fail for the Maxwell finite element equations in the sense that the convergence rate deteriorates as the mesh size decreases. To overcome this drawback, R. Hiptmair proposed to modify the smoothing iteration by adding a smoothing step in the discrete potential space [11]. Similarly, D. Arnold, R. Falk and R. Winther suggested a special block smoother that has the same effect [1]. In [2], one can find some implementation issues for the multigrid algorithm proposed by R. Hiptmair and numerical results for the eddy current problem that gives an additional L_2–term after the time discretization.

The parallelization of these or, more precisely, of appropriately modified multigrid solvers is certainly the only feasible way to enhance the efficiency of these algorithms. However, the parallelization is not straightforward due to the peculiarities of the multigrid methods for the Maxwell equations. In this paper, we propose a unified approach to the parallelization of multigrid methods based on a data distribution that is known from the non-overlapping domain decomposition.

In order to develop a basic parallel Maxwell solver that can be used for more advanced problems as basic module, it is sufficient to consider the magnetostatic case. In the magnetostatic case, the Maxwell equations can be reduced to the curl-curl–equation that is not uniquely solvable because of the large kernel of the curl-operator (gradient fields). In practice, a gauging condition is imposed in order to pick out a unique solution. The so-called Coulomb gauging aims at a divergence-free solution (vector potential). The weak formulation of the curl-curl–equation and the gauging condition together with a clever regularization leads to a regularized mixed variational formulation of the magnetostatic Maxwell equations in $\mathrm{H}_0(\mathbf{curl}) \times \mathrm{H}_0^1(\Omega)$ that has a unique solution. The discretization by the Nédélec and Lagrange finite elements results in large, sparse, symmetric, but indefinite system of finite element equations. Eliminating the Lagrange multiplier from the mixed finite element equations, we arrive at a symmetric and positive definite (spd) problem that can be solved by some parallel multigrid preconditioned conjugate gradient (pcg) method. More precisely, this pcg solver contains a standard scaled Laplace multigrid regularizator in the regularization part and a special multigrid preconditioner for the regularized Nédélec finite element equations that we want to solve. The parallelization of the pcg algorithm, the Laplace multigrid regularizator

and the multigrid preconditioner are based on a unified domain decomposition data distribution concept mentioned above. From the parallelization point of view, we prefer Hiptmair's multigrid method with some modifications for the construction of the special multigrid preconditioner.

The rest of the paper is organized as follows. In Section 2, we recall the mixed variational formulation of the magnetostatic Maxwell equation, its regularizations, and its finite element discretization. Further, we present the multigrid preconditioned conjugate gradient method for solving the spd reduced problem. Section 3 provides the data distribution concept and the parallelization of the basic algorithms, namely the pcg and the multigrid algorithms. In Section 4, we discuss the parallelization peculiarities that come along with the spd reduced problem (Schur complement system) and with the special multigrid preconditioner. Section 5 contains some results of our numerical experiments on a parallel machine with distributed memory that show the high efficiency of our approach for a real-life application.

2 3D Magnetostatic Field Problems

The magnetostatic equations, in which we are interested throughout the paper, can be rewritten as

$$\mathrm{curl}(H) = J, \quad H = \nu B, \quad \mathrm{div}(B) = 0, \tag{1}$$

where H and B denote the magnetic field intensity and the magnetic flux density, respectively. The permeability μ ($\nu = 1/\mu$) and the impressed current density J are supposed to be given. If permanent magnets are present, J has to be replaced by $J + \mathrm{curl}(H_0)$, where H_0 is the initial (permanent) magnetic field intensity. Furthermore, we note that the current density J is physically divergence-free, i.e.,

$$\mathrm{div}(J) = 0. \tag{2}$$

Theoretically, the computational domain Ω coincides with the space $\mathbb{R}^3$ in any case. The behavior of the magnetic field at infinity is described by radiation conditions. In practice, one may often simplify the problem by considering a bounded, simply connected computational domain $\Omega \subset \mathbb{R}^3$ with Lipschitz boundary $\Gamma = \partial\Omega$ and by replacing the radiation condition by the boundary condition

$$B \cdot n = 0 \quad \text{on } \partial\Omega, \tag{3}$$

where n stands for the unit outward normal with respect to $\partial\Omega$. Of course, this modification causes some modeling error. However, if the sources are

completely surrounded by iron or, more generally, if Ω is chosen large enough, this error can be neglected. Introducing some vector potential u for the B-field $B = \mathrm{curl}(u)$, we can easily derive the canonical variational formulation: find $u \in H_0(\mathrm{curl}, \Omega)$ such that

$$\int_\Omega \nu \,\mathrm{curl}(u) \cdot \mathrm{curl}(v)\, dx = \int_\Omega J \cdot v\, dx \quad \forall\, v \in H_0(\mathrm{curl}, \Omega), \tag{4}$$

from the magnetostatic equations (1) in Ω and the boundary conditions (3) on Γ, where $H_0(\mathrm{curl}, \Omega) := \{v \in L^2(\Omega)^3 |\ \mathrm{curl}(v) \in L^2(\Omega)^3,\ v \times n = 0 \text{ on } \Gamma\}$. Condition (2) ensures the solvability of the variational problem (4). However, the solution is not unique. Indeed, if one adds to some solution u of (4) a gradient-field $\nabla\varphi \in L^2(\Omega)^3$ with $\nabla\varphi \times n = 0$, one obtains again a solution. From now on we allow the coefficient $\nu = 1/\mu$ to be piecewise constant with $\nu \geq \nu_{\min} > 0$. We mention that the normal component of the B-field and the tangential component of the H-field are continuous across the interfaces.

In order to obtain some variational formulation that has a unique solution, we have to take into account some gauging condition for the vector potential u. The Coulomb gauging

$$\mathrm{div}(u) = 0 \tag{5}$$

aims at divergence-free vector potentials. Treating the weak formulation of the Coulomb gauging (5) as an equality constraint, we arrive at the following mixed variational formulation that is fundamental for our approach to the numerical solution of the magnetostatic Maxwell equations (1): Find $(u, p) \in \mathbb{X} \times \mathbb{M} := H_0(\mathrm{curl}, \Omega) \times H_0^1(\Omega)$ such that

$$a(u, v) + b(v, p) = \langle f, v \rangle \quad \forall\, v \in H_0(\mathrm{curl}, \Omega), \tag{6}$$

$$b(u, q) = 0 \qquad \forall\, q \in H_0^1(\Omega), \tag{7}$$

where $a(u, v) := \int_\Omega \nu \,\mathrm{curl}(u) \cdot \mathrm{curl}(v)\, dx$, $b(v, p) := \int_\Omega v \cdot \nabla p\, dx$, and $\langle f, v \rangle := \int_\Omega J \cdot v\, dx$. Now it is not difficult to conclude from the Brezzi-Babuška theory that the mixed variational problem (6) - (7) has a unique solution. Moreover, choosing $v = \nabla p \in H_0(\mathrm{curl}, \Omega)$ in (6), we immediately observe from (2) and the Friedrich's inequality that

$$p = 0. \tag{8}$$

This simple observation is crucial for our approach. Indeed, adding an arbitrary symmetric and positive definite bilinear form $c(\cdot, \cdot) : \mathbb{M} \times \mathbb{M} \to \mathbb{R}^1$ to the second equation of our mixed variational problem (6) - (7), we arrive at the equivalent mixed variational problem: Find $(u, p) \in \mathbb{X} \times \mathbb{M}$ such that

$$a(u,v) + b(v,p) = \langle f, v\rangle \quad \forall\, v \in H_0\,(\mathrm{curl}, \Omega), \tag{9}$$
$$b(u,q) - c(p,q) = 0 \qquad \forall\, q \in H_0^1(\Omega). \tag{10}$$

Let now be $\mathbb{X}_h \subset \mathbb{X}$ and $\mathbb{M}_h \subset \mathbb{M}$ the lowest order edge element space (see [15]) and the space of piecewise linear nodal elements on a shape-regular tetrahedral triangularization with mesh width h of Ω, respectively [4]. Then the mixed finite element approximation to the regularized mixed variational problem (9) - (10) reads as follows: Find $(u_h, p_h) \in \mathbb{X}_h \times \mathbb{M}_h$ such that

$$a(u_h, v_h) + b(v_h, p_h) = \langle f, v_h\rangle \quad \forall\, v_h \in \mathbb{X}_h\,, \tag{11}$$
$$b(u_h, q_h) - c(p_h, q_h) = 0 \qquad \forall\, q_h \in \mathbb{M}_h\,. \tag{12}$$

Again, using the Brezzi-Babuška theory, we conclude that the mixed finite element scheme (11) - (12) has a unique solution. From the same theory, we can derive the standard a–priori error estimates (see, e.g., [3]). Similarly to the continuous case, we observe that $p_h = 0$ since $q_h \in \mathbb{M}_h$ implies $\nabla q_h \in \mathbb{X}_h$. Let us choose the standard finite element basis in the spaces $\mathbb{X}_h$ and $\mathbb{M}_h$. Then the mixed finite element scheme (11) - (12) is equivalent to the regular, symmetric, but indefinite system

$$\begin{pmatrix} A & B^T \\ B & -C \end{pmatrix} \begin{pmatrix} \underline{u}_h \\ \underline{p}_h \end{pmatrix} = \begin{pmatrix} \underline{f}_h \\ 0 \end{pmatrix} \tag{13}$$

of linear finite element equations, where the matrices A, B, C and the first component $\underline{f}_h$ of the right-hand side are derived from the bilinear forms $a(\cdot,\cdot)$, $b(\cdot,\cdot)$, $c(\cdot,\cdot)$, and the linear form $\langle f, \cdot\rangle$, respectively.

Eliminating $\underline{p}_h = -C^{-1}B\underline{u}_h$ from the second equation in (13) and inserting it into the first equation, we obtain the spd Schur complement system

$$G\underline{u}_h := (A + B^T C^{-1} B)\underline{u}_h = \underline{f}_h. \tag{14}$$

Let $\widetilde{C}$ be some spd matrix that is spectrally equivalent to C (briefly, $\widetilde{C} \approx C$), i.e. there are some positive constants c_1 and c_2, independent of h, such that

$$c_1(\widetilde{C}\underline{q}_h, \underline{q}_h) \le (C\underline{q}_h, \underline{q}_h) \le c_2(\widetilde{C}\underline{q}_h, \underline{q}_h) \quad \forall \underline{q}_h \in \mathbb{R}^{m_h}\,, \tag{15}$$

where $m_h := dim(\mathbb{M}_h)$. Then the original Schur complement system (14) is equivalent to the modified Schur complement system

$$\widetilde{G}\underline{u}_h := (A + B^T \widetilde{C}^{-1} B)\underline{u}_h = \underline{f}_h. \tag{16}$$

Instead of solving the symmetric, but indefinite system (13), we solve the spd modified Schur complement system (16).

Let us choose the spd bilinear form

$$c(p,q) := \frac{1}{\nu_{\min}} \int_{\Omega} \nabla p \, \nabla q \, dx, \tag{17}$$

corresponding to the Laplace operator scaled by $1/\nu_{\min}$, and let us consider a spd preconditioner C_H for the spd matrix $H := A + M$, i.e. there are some positive constants c_3 and c_4, independent of h, such that

$$c_3(C_H \underline{v}_h, \underline{v}_h) \leq (H \underline{v}_h, \underline{v}_h) \leq c_4(C_H \underline{v}_h, \underline{v}_h) \quad \forall \underline{v}_h \in \mathbb{R}^{n_h}, \tag{18}$$

where $n_h := dim(\mathbb{X}_h)$ and M is the scaled mass matrix in $\mathbb{X}_h$ defined by

$$(M \underline{u}_h, \underline{v}_h) := \nu_{\min} \int_{\Omega} u_h \, v_h \, dx. \tag{19}$$

The discrete LBB–condition and the spectral equivalence inequalities (18) imply that $C_H \approx G \approx \widetilde{G}$ (see [13] for the detailed proof). Once a good preconditioner C_H and an appropriate regularizator $\widetilde{C}$ is available, we can solve the modified Schur complement system (16) by the preconditioned conjugate gradient method. In practice, we choose the multigrid preconditioner

$$C_H := H(I - M_H)^{-1} \tag{20}$$

and the multigrid regularizator

$$\widetilde{C} := C_C := C(I - M_C)^{-1}, \tag{21}$$

where M_H and M_C are the corresponding multigrid iteration operators with respect to H and C. Choosing appropriate symmetric multigrid cycles, we can now conclude from the results of [1,10,11] that the pcg method is asymptotically optimal with respect to the operation count and to the memory demand [12]. The numerical results obtained from the serial implementation of this algorithm confirm this statement [14]. In this paper, we are interested in the parallel implementation of this algorithm. The parallelization of this algorithm is far from being straightforward because of the peculiarities connected with the multigrid regularizator C_C and with the special multigrid preconditioner C_H.

3 A Unified Data Distribution Concept

3.1 Notations

We use a non-overlapping domain decomposition for our approach, i.e., we decompose $\overline{\Omega}$ in P subdomains $\overline{\Omega}_s$ such that $\overline{\Omega} = \bigcup_{s=1}^{P} \overline{\Omega}_s$ with $\Omega_s \cap \Omega_q = \emptyset$,

$\forall q \neq s,\ s,q = \overline{1,P}$ holds. Each subdomain Ω_s is discretized by a mesh $\tau_{h,s}$, such that the whole triangulation $\tau_h = \bigcup_{s=1}^{P} \tau_{h,s}$ of Ω is conform. A global finite element space $\mathbb{V}_h$ is defined with respect to τ_h and the local spaces $\mathbb{V}_{h,s}$ are projections of $\mathbb{V}_h$ onto $\tau_{h,s}$. The index set of unknowns in Ω_s is denoted by ω_s. Note, that $\omega = \bigcup_{s=1}^{P} \omega_s$ but $N := |\omega| \leq \sum_{s=1}^{P} |\omega_s| =: \sum_{s=1}^{P} N_s$ holds, i.e. different subdomains may share unknowns whereas elements from different subdomains do not overlap.

The mapping of a vector $\underline{u} \in \mathbb{R}^N$ in global numbering onto a local vector $\underline{u}_s \in \mathbb{R}^{N_s}$ in subdomain $\overline{\Omega}_s$ ($s = \overline{1,P}$) is represented symbolically by subdomain connectivity matrices $\mathcal{A}_s$ of dimension $N_s \times N$ with entries

$$\mathcal{A}_s^{[i,j]} := \begin{cases} 1 & \text{if} \quad j = \text{global number of } i \\ 0 & \text{else} \end{cases} \qquad \forall i \in \omega_s\ ,\ \forall j \in \omega\ . \tag{22}$$

The transpose $\mathcal{A}_s^T$ of these binary matrices $\mathcal{A}_s : \mathbb{R}^N \mapsto \mathbb{R}^{N_s}$ maps a local vector back to the global one. The index set of all those subdomains, an unknown $u^{[j]}$, $j \in \omega$ belongs to, is denoted by

$$\sigma^{[j]} := \{s \,|\, u^{[j]} \in \overline{\Omega}_s\} = \{s \,|\, \exists i \in \omega_s : \mathcal{A}_s^{[i,j]} \neq 0\}\ . \tag{23}$$

3.2 Vector and matrix types

We store the data of a vector component $u^{[i]}$ in the subdomain Ω_s if $s \in \sigma^{[i]}$. There are (at least) two opportunities to store those components and finally that vector (see also Fig. 3.2).

- A vector $\underline{\mathfrak{u}}_s$ ($\underline{\mathfrak{u}}$) is called an accumulated vector if each vector component $\mathfrak{u}^{[i]}$ is stored in all subdomains Ω_s, $s \in \sigma^{[i]}$ with its full value. The local vectors $\underline{\mathfrak{u}}_s$ can be represented as

$$\underline{\mathfrak{u}}_s := \mathcal{A}_s \cdot \underline{\mathfrak{u}}\ . \tag{24}$$

- A vector $\underline{\mathsf{r}}$ is called a distributed vector if it is decomposed into local vectors $\underline{\mathsf{r}}_s$ such that

$$\underline{\mathsf{r}} = \sum_{s=1}^{P} \mathcal{A}_s^T \cdot \underline{\mathsf{r}}_s \tag{25}$$

  holds, i.e., all subdomains Ω_s, $s \in \sigma^{[i]}$ store only $\underline{\mathsf{r}}_s$ and possess a portion of the full vector value $\mathsf{r}^{[i]}$ which can be determined only by taking the sum in (25).

The conversion of a distributed vector $\underline{\mathfrak{v}}$ into an accumulated vector $\underline{\mathfrak{w}}$ can be done by evaluating the sum in (25) followed by the restriction (24), i.e.,

$$\underline{\mathfrak{w}} \leftarrow \underline{\mathfrak{v}} \qquad : \qquad \underline{\mathfrak{w}}_s := \mathcal{A}_s \cdot \underline{\mathfrak{w}} = \mathcal{A}_s \cdot \sum_{s=1}^{P} \mathcal{A}_s^T \cdot \underline{\mathfrak{v}}_s \ . \tag{26}$$

The conversion in the other direction is not unique - we prefer an equal weighted distribution of the accumulated vector. A matrix weighted distribution is also feasible in case of jumping coefficients. The weights are chosen such that the re-conversion (26) will result in the original vector :

$$\underline{\mathfrak{v}} \leftarrow \underline{\mathfrak{w}} \qquad : \qquad \underline{\mathfrak{v}}_s := (R_s)^{-1} \cdot \underline{\mathfrak{w}}_s \ , \tag{27}$$

with $R_s = \underset{i \in \omega_s}{\text{diag}} \{ |\sigma^{[i]}| \}$, i.e., $R_s^{[i,i]}$ stores the number of subdomains the component i belongs to.

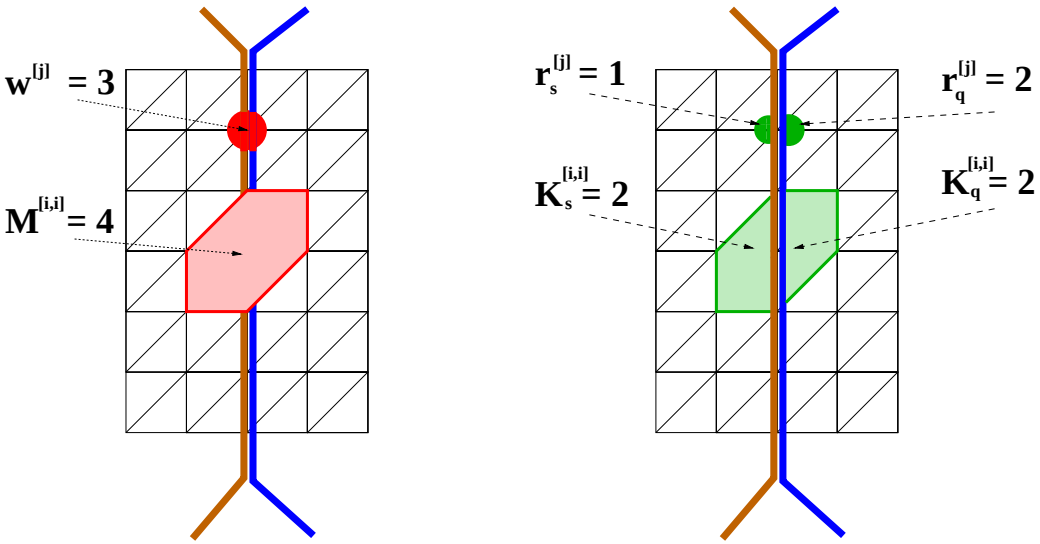

Fig. 1. Illustration for accumulated and distributed vectors and matrices.

The matrix defined by the bilinear form in (6) can also be stored in two ways. With respect to an element-wise domain decomposition, we can store the f.e. matrix accumulated or distributed.

- A matrix $\mathfrak{M}$ is called accumulated if its local restrictions $\mathfrak{M}_s$ possess the full entries of it, and we can write

$$\mathfrak{M}_s := \mathcal{A}_s \cdot \mathfrak{M} \cdot \mathcal{A}_s^T \ . \tag{28}$$

- We call a matrix K distributed if we have locally stored matrices K_s such that

$$\mathsf{K} := \sum_{s=1}^{P} \mathcal{A}_s^T \cdot \mathsf{K}_s \cdot \mathcal{A}_s \tag{29}$$

holds, i.e., each subdomain $\overline{\Omega}_s$ stores only a part of its full values.

We obtain distributed matrices K_s automatically after the local f.e. accumulation with respect to $\mathbb{V}_{h,s}$, due to our non-overlapping construction principle. The sum in (29) is equivalent to a global matrix accumulation which will not be carried out in our approach.

3.3 Basic operations

This section collects only some result from previous papers, for details see [5–7,9]. It is trivial that additive combinations of vectors from the same type do not require any communication. The inner product of different type vectors requires one global reduce operation of the local inner products:

$$\langle \underline{\mathfrak{w}}, \underline{\mathsf{r}} \rangle \;=\; \underline{\mathfrak{w}}^T \underline{\mathsf{r}} \;=\; \underline{\mathfrak{w}}^T \sum_{s=1}^{P} \mathcal{A}_s^T \underline{\mathsf{r}}_s \;=\; \sum_{s=1}^{P} (\mathcal{A}_s \underline{\mathfrak{w}})^T \underline{\mathsf{r}}_s \;=\; \sum_{s=1}^{P} \langle \underline{\mathfrak{w}}_s, \underline{\mathsf{r}}_s \rangle \quad . \tag{30}$$

Any other combination of vector types requires more effort. The multiplication of a distributed matrix with an accumulated vector

$$\mathsf{K} \cdot \underline{\mathfrak{w}} \;=\; \sum_{s=1}^{P} \mathcal{A}_s^T \mathsf{K}_s \mathcal{A}_s \cdot \underline{\mathfrak{w}} \;=\; \sum_{s=1}^{P} \mathcal{A}_s^T \left(\mathsf{K}_s \cdot \underline{\mathfrak{w}}_s \right) \;=\; \sum_{s=1}^{P} \mathcal{A}_s^T \underline{\mathsf{v}}_s \;=\; \underline{\mathsf{v}} \;, \tag{31}$$

results in a distributed vector. The realization requires no communication at all because we only have to compute locally $\underline{\mathsf{v}}_s = \mathsf{K}_s \cdot \underline{\mathfrak{w}}_s$.

The situation changes if we use an accumulated matrix $\mathfrak{M}$. Here, we have to ensure that the pattern of $\mathfrak{M}$ fulfills the condition

$$\forall i, j \in \omega \;: \qquad \sigma^{[i]} \not\subset \sigma^{[j]} \implies \mathfrak{M}^{[i,j]} = 0 \quad . \tag{32}$$

If the pattern condition (32) holds then the operations

$$\underline{\mathfrak{w}} \;=\; \mathfrak{M} \cdot \underline{\mathfrak{u}} \qquad \text{and} \qquad \underline{\mathsf{d}} \;=\; \mathfrak{M}^T \cdot \underline{\mathsf{r}} \tag{33}$$

can be performed locally without any communication, i.e. $\forall s = \overline{1, P}$

$$\underline{\mathfrak{w}}_s \;=\; \mathfrak{M}_s \cdot \underline{\mathfrak{u}}_s \qquad \text{and} \qquad \underline{\mathsf{d}}_s \;=\; \mathfrak{M}_s^T \cdot \underline{\mathsf{r}}_s \quad . \tag{34}$$

3.4 Basic algorithms

The operations (30) and (31) allow already to formulate a parallel pcg algorithm with preconditioning.

Algorithm 1 Parallel pcg $\textsc{pcg}(\mathsf{K}, \underline{\mathfrak{u}}, \underline{\mathfrak{f}}, C_K)$

repeat
$\underline{\mathfrak{v}} \leftarrow \mathsf{K} \cdot \underline{\mathfrak{s}}$
$\alpha \leftarrow \sigma / \langle \underline{\mathfrak{s}}, \underline{\mathfrak{v}} \rangle$
$\underline{\mathfrak{u}} \leftarrow \underline{\mathfrak{u}} + \alpha \cdot \underline{\mathfrak{s}}$
$\underline{\mathfrak{r}} \leftarrow \underline{\mathfrak{r}} - \alpha \cdot \underline{\mathfrak{v}}$
$\underline{\mathfrak{w}} \Leftarrow C_K^{-1} \cdot \underline{\mathfrak{r}}$
$\sigma \leftarrow \langle \underline{\mathfrak{w}}, \underline{\mathfrak{r}} \rangle$
$\beta \leftarrow \sigma / \sigma_{\text{old}} \quad , \quad \sigma_{\text{old}} \leftarrow \sigma$
$\underline{\mathfrak{s}} \leftarrow \underline{\mathfrak{w}} + \beta \cdot \underline{\mathfrak{s}}$
until termination

Besides the inner products, only the preconditioning step $\underline{\mathfrak{w}} \Leftarrow C_K^{-1} \cdot \underline{\mathfrak{r}}$ involves communication indicated by using $\Leftarrow$ instead of $\leftarrow$. In the case of $C_K = I$, i.e., no preconditioning, this step reduces to a type conversion (26) involving communication. We require that the communication costs for applying any other preconditioner C_K^{-1} are in the same range.

One possible choice for the preconditioner is $C_K^{-1} = (I - M_K)K^{-1}$, with M_K being the multigrid iteration operator for K. The parallel multigrid iteration presented in Alg. 2 needs a smoother SMOOTH with a good parallel performance, e.g., a block Jacobi smoother with Gauss-Seidel smoothing in blocks containing interior unknowns of the subdomains. Furthermore, the interpolation $\mathfrak{P}$ has to fulfill the pattern condition (32) and we take $\mathfrak{P}^T$ as restriction. Together with an appropriate coarse grid solver we can formulate the parallel multigrid algorithm.

Algorithm 2 Parallel multigrid $\textsc{pmg}(\mathsf{K}, \underline{\mathfrak{u}}, \underline{\mathfrak{f}}, \ell)$

if $\ell == 1$ **then**
$\underline{\mathfrak{u}} \Leftarrow \textsc{Solve}\left(\sum_{s=1}^{P} \mathcal{A}_s^T \mathsf{K}_s \mathcal{A}_s \cdot \underline{\mathfrak{u}} = \underline{\mathfrak{f}} \right)$
else
$\widetilde{\underline{\mathfrak{u}}} \Leftarrow \textsc{Smooth}(\mathsf{K}, \underline{\mathfrak{u}}, \underline{\mathfrak{f}})$
$\underline{\mathfrak{d}} \leftarrow \underline{\mathfrak{f}} - \mathsf{K} \cdot \widetilde{\underline{\mathfrak{u}}}$
$\underline{\mathfrak{d}}^H \leftarrow \mathfrak{P}^T \cdot \underline{\mathfrak{d}}$
$\underline{\mathfrak{w}}^H \leftarrow 0$
$\underline{\mathfrak{w}}^H \Leftarrow \textsc{pmg}(\mathsf{K}^H, \underline{\mathfrak{w}}^H, \underline{\mathfrak{d}}^H, \ell - 1)$
$\underline{\mathfrak{w}} \leftarrow \mathfrak{P} \cdot \underline{\mathfrak{w}}^H$
$\widehat{\underline{\mathfrak{u}}} \leftarrow \widetilde{\underline{\mathfrak{u}}} + \underline{\mathfrak{w}}$
$\underline{\mathfrak{u}} \Leftarrow \textsc{Smooth}^T(\mathsf{K}, \widehat{\underline{\mathfrak{u}}}, \underline{\mathfrak{f}})$
end if

Despite the coarse grid solver, only the smoothing sweep requires communication. The coarse grid can be solved directly (either in parallel or sequentially on one processor) or again by some iterative method similar to the PCG in Alg. 1.

4 Parallel Multigrid Maxwell Solver

We want to solve (16) by the pcg algorithm (Alg. 1) using a multigrid preconditioner (Alg. 2) for the realization of C_K^{-1}. In this section, we will discuss how these components have to be adapted to the case of our Maxwell solver presented in Section 2.

Our reduced primal formulation (16) has been derived from the block system (13). As discussed in the previous section, the matrices are generated locally, such that the local components A_s, B_s, C_s are available. Denoting the subdomain connectivity matrices with respect to the spaces $\mathbb{X}_h$ and $\mathbb{M}_h$ by $\mathcal{A}_{\mathbb{X},s}$ and $\mathcal{A}_{\mathbb{M},s}$, respectively, we have the following relations:

$$\mathsf{A} = \sum_{s=1}^{P} \mathcal{A}_{\mathbb{X},s} \mathsf{A}_s \mathcal{A}_{\mathbb{X},s}^T, \quad \mathsf{B} = \sum_{s=1}^{P} \mathcal{A}_{\mathbb{M},s} \mathsf{B}_s \mathcal{A}_{\mathbb{X},s}^T, \quad \mathsf{C} = \sum_{s=1}^{P} \mathcal{A}_{\mathbb{M},s} \mathsf{C}_s \mathcal{A}_{\mathbb{M},s}^T.$$

The system matrix in (16) is defined by $\widetilde{\mathsf{G}} := \mathsf{A} + \mathsf{B}^\mathsf{T} \widetilde{C}^{-1} \mathsf{B}$, where $\widetilde{C}$ is a preconditioner for C. In order to apply $\textsc{pcg}(\widetilde{\mathsf{G}}, \underline{\mathfrak{u}}, \underline{\mathfrak{f}}, C_{\widetilde{G}})$ we explain in Alg. 3 how the matrix-by-vector operation is defined for the distributed matrix $\widetilde{\mathsf{G}}$. Hereby, the required operation $\widetilde{C}^{-1}$ is being realized by one multigrid iteration

Algorithm 3 The operation $\underline{\mathfrak{v}} \Leftarrow \widetilde{\mathsf{G}} \cdot \underline{\mathfrak{s}}$.

$\underline{\mathfrak{q}} \leftarrow \mathsf{B} \cdot \underline{\mathfrak{s}}$
$\underline{\mathfrak{p}} \Leftarrow \textsc{pmg}(\mathsf{C}, \underline{\mathfrak{p}}, \underline{\mathfrak{q}}, \ell)$
$\underline{\mathfrak{v}} \leftarrow \mathsf{A} \cdot \underline{\mathfrak{s}} + \mathsf{B}^\mathsf{T} \cdot \underline{\mathfrak{p}}$

step $\textsc{pmg}(\mathsf{C}, \underline{\mathfrak{p}}, \underline{\mathfrak{q}}, \ell)$ in the space $\mathbb{M}_h$. Although the matrix $\widetilde{\mathsf{G}}$ is distributed, the corresponding matrix-by-vector operation requires as many communications as one multigrid iteration step for C in $\mathbb{M}_h$.

Furthermore, the operation C_K^{-1} in Alg. 1 is now realized by one iteration step of $\textsc{Hybridpmg}(\widetilde{\mathsf{H}}, \mathsf{C}, \underline{\mathfrak{u}}, \underline{\mathfrak{f}}, \ell)$ defined in Alg. 4. Comparing Alg. 4 and Alg. 2 we observe that only the smoother has to be adapted to our special application. In particular we use a hybrid smoother as proposed in [11] which is suitable for parallelization. As in [11], we introduce the lifting operator $\mathfrak{L} : \mathbb{X}_h \to \mathbb{M}_h$ where $\mathfrak{L}^{[i,j]} := -1$, $\mathfrak{L}^{[i,k]} := 1$ if the oriented edge with the unknown i in $\mathbb{X}_h$ connects the two unknowns j and k in $\mathbb{M}_h$, see Fig. 2. Otherwise we have $\mathfrak{L}^{[i,j]} := 0$. We observe that $\mathfrak{L}^T$ satisfies the pattern condition (32).

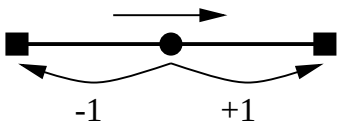

Fig. 2. The lifting operator $\mathfrak{L} : \mathbb{X}_h \to \mathbb{M}_h$.

Algorithm 4 Parallel hybrid multigrid HYBRIDPMG$(\mathsf{H}, \mathsf{C}, \underline{\mathfrak{u}}, \underline{\mathfrak{f}}, \ell)$

if $\ell == 1$ **then**
  $\underline{\mathfrak{u}} \Leftarrow$ SOLVE$\,(\sum_{s=1}^{P} \mathcal{A}_s^T \mathsf{H}_s \mathcal{A}_s \cdot \underline{\mathfrak{u}} \;=\; \underline{\mathfrak{f}}\,)$
else
  $\widetilde{\underline{\mathfrak{u}}} \Leftarrow$ HYBRIDSMOOTH$(\mathsf{H}, \mathsf{C}, \underline{\mathfrak{u}}, \underline{\mathfrak{f}})$
  $\underline{\mathfrak{d}} \leftarrow \underline{\mathfrak{f}} - \mathsf{H} \cdot \widetilde{\underline{\mathfrak{u}}}$
  $\underline{\mathfrak{d}}^H \leftarrow \mathfrak{P}^T \cdot \underline{\mathfrak{d}}$
  $\underline{\mathfrak{w}}^H \leftarrow 0$
  $\widetilde{\underline{\mathfrak{w}}}^H \Leftarrow$ HYBRIDPMG$(\mathsf{H}^H, \mathsf{C}^H, \underline{\mathfrak{w}}^H, \underline{\mathfrak{d}}^H, \ell - 1)$
  $\underline{\mathfrak{w}} \leftarrow \mathfrak{P} \cdot \widetilde{\underline{\mathfrak{w}}}^H$
  $\widehat{\underline{\mathfrak{u}}} \leftarrow \widetilde{\underline{\mathfrak{u}}} + \underline{\mathfrak{w}}$
  $\underline{\mathfrak{u}} \Leftarrow$ HYBRIDSMOOTH$^T(\mathsf{H}, \mathsf{C}, \widehat{\underline{\mathfrak{u}}}, \underline{\mathfrak{f}})$
end if

Now Alg. 5 is the correct definition of the parallel hybrid smoother HYBRIDSMOOTH$(\mathsf{H}, \mathsf{C}, \underline{\mathfrak{u}}, \underline{\mathfrak{f}})$. Note, using the matrix C derived from (17) for defining the smoother in $\mathbb{M}_h$ we have to use the correct scaling by $\nu_{\min}^2$ corresponding to the definition of M in (19). Now, the smoother SMOOTH can be

Algorithm 5 Parallel hybrid smoother HYBRIDSMOOTH$(\mathsf{H}, \mathsf{C}, \underline{\mathfrak{u}}, \underline{\mathfrak{f}})$

$\widetilde{\underline{\mathfrak{u}}} \Leftarrow$ SMOOTH$(\mathsf{H}, \underline{\mathfrak{u}}, \underline{\mathfrak{f}})$
$\underline{\mathfrak{q}} \leftarrow \mathfrak{L} \cdot (\underline{\mathfrak{f}} - \mathsf{H} \cdot \widetilde{\underline{\mathfrak{u}}})$
$\underline{\mathfrak{p}} \leftarrow 0$
$\widetilde{\underline{\mathfrak{p}}} \Leftarrow$ SMOOTH$(\nu_{\min}^2 \cdot \mathsf{C}, \underline{\mathfrak{p}}, \underline{\mathfrak{q}})$
$\underline{\mathfrak{u}} \leftarrow \widetilde{\underline{\mathfrak{u}}} + \mathfrak{L}^T \cdot \widetilde{\underline{\mathfrak{p}}}$

any standard smoother with a good parallel performance, e.g., a block Jacobi smoother with Gauss-Seidel smoothing in blocks containing interior unknowns of the subdomains. Since HYBRIDSMOOTH involves at least two smoothing steps, one in $\mathbb{X}_h$ and one in $\mathbb{M}_h$, at least two subsequent communications are required.

Note, the post–smoothing step HYBRIDSMOOTH$^T(\mathsf{H}, \mathsf{C}, \underline{\mathfrak{u}}, \underline{\mathfrak{f}})$. in Alg. 4 is obtained from Alg. 5 by executing step 1 after steps 2-3-4-5 instead of executing the given order 1-2-3-4-5.

5 Numerical Results

In this section, we present an example from magnetostatics and we apply the algorithms presented above.

The geometry of our model problem together with the corresponding coarse surface mesh is shown in Fig. 3 on the left. We consider the model of a trans-

former with a kernel, three coils and the air domain around these parts. The outer boundary is given by an iron casing of high conductivity that motivates the boundary condition (3). The magnetic field which is to be computed is generated by tangential currents within the three coils. The iron core has a permeability of 1000. The resulting magnetic flux density B is shown in Fig. 3 on the right.

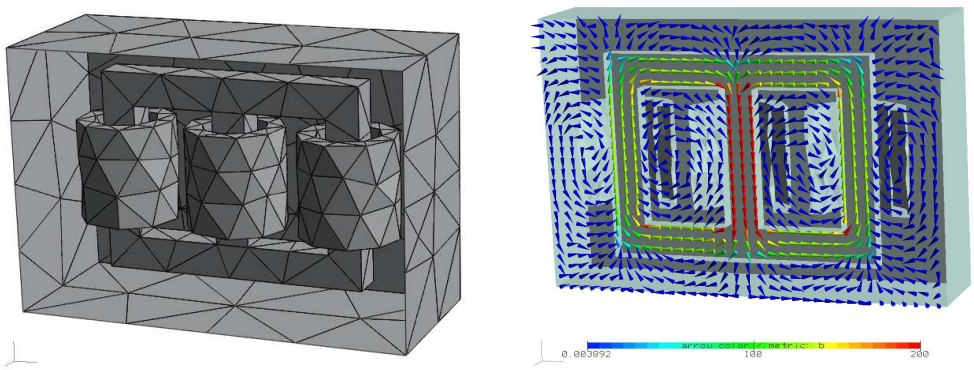

Fig. 3. Geometry, initial mesh (left), resulting magnetic flux density B (right).

The basis for our domain decomposition is a tetrahedral mesh generated fully automatically by NETGEN [17]. The surface mesh shown in Fig. 3 corresponds to a volume mesh of 2465 tetrahedra. We apply a modified recursive spectral bisection (rsb) algorithm which allows to use any number P of subdomains, see [8] an references therein. The use of the rsb ensures that the elements are distributed almost equally to the P processors being used. The partition of the mesh defines now the domain decomposition introduced in Section 3.1. The partitions into 4 and 32 subdomains are shown in Fig. 4. Finer meshes corresponding to the levels $\ell = 2, 3, 4$ are obtained from uniform

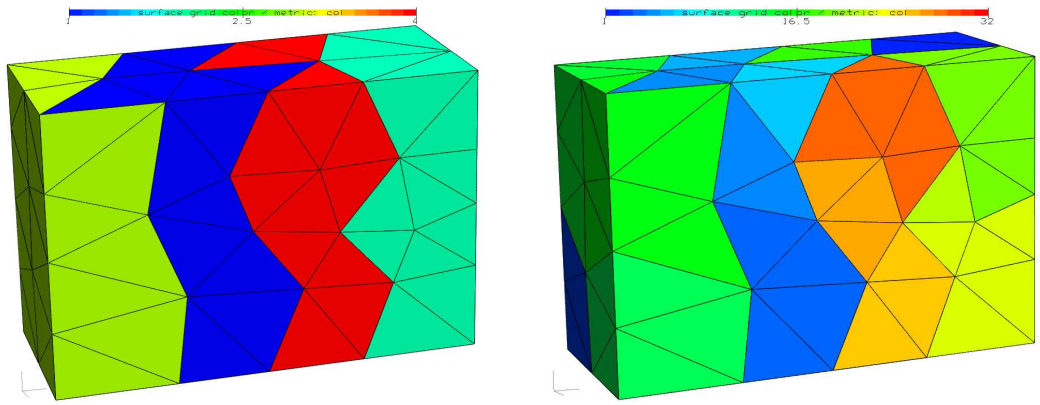

Fig. 4. Partitions into 4 and 32 subdomains.

		1		4	16	32	48	60	
ℓ	dof	It.	T[sec]	T[sec]	T[sec]	T[sec]	T[sec]	It.	T[sec]
1	3466	5	0.1	0.1	0.1	0.1	0.1	5	0.1
2	26907	8	7.1	2.9	1.5	1.4	1.4	11	1.7
3	212597	9	84	28	7.3	4.7	3.8	13	4.1
4	1691370	11	1398	496	81	42	28	14	24
T($\ell = 4$)		2593		806	160	86	61	54	
T($\Sigma\, \ell = 1..4$)		2852		886.4	188.4	108.5	83.8	79.6	

Table 1
Wall-clock time for the solver on each level (upper part), overall wall-clock time for $\ell = 4$ and accumulated for all levels ($\ell = 1..4$) in seconds.

refinement resulting in overall 1262080 tetrahedra. This refinement is purely local and can be realized without communication. However, newly created unknowns at interfaces between different subdomains have to be identified uniquely by all surrounding processors. For this purpose, one root process per interface receives data from all surrounding subdomains, it identifies all unknowns uniquely and it finally distributes this information again to all adjacent processors. This setup phase is required once after each refinement step.

The numerical experiments presented here are carried out on a SGI Origin 2000 machine with 64 CPU R12000, 300 MHz and overall 20 GB main memory. The numerical simulations are carried out using the object oriented C++ code *FEPP* [18,14]. The message passing is based on MPI from the *SGI Message Passing Toolkit 1.2*. The wall-clock time has been measured by *MPI_WTIME()*.

The system (16) has been solved using the PMG algorithm with the relative accuracy 10^{-4}. For the multigrid preconditioner C_H (20), a V–cycle with 1 pre– and 1 post–smoothing step HYBRIDSMOOTH has been used. The multigrid regularizator $\widetilde{C}$ (21) has been realized by a V–cycle with 2 pre– and 2 post–smoothing steps using a standard smoother with good parallel efficiency as described before. Table 1 shows the corresponding results including number of unknowns (dof), number of iterations (It.) and wall-clock time in seconds (T[sec]). We increase the number of unknowns from top to bottom, while the number of processors is increased from left to right. First, we observe that the number of iterations is almost independent of the number of unknowns. This shows the optimality of our algorithm. However, the number of iterations depends slightly on the number of processors. That is because our smoothers depend on the partition of the mesh. In particular, the blocks of unknowns at interfaces where Jacobi steps are performed grow with the number of subdomains.

P	1	4	16	32	48	60
$\Sigma\,\ell = 1..4$	1.0	3.2	15.1	26.2	34.1	35.6
$\ell = 4$	1.0	3.2	16.2	30.0	42.5	48.0
pcg ($\ell = 4$)	1.0	2.8	17.2	33.0	49.6	58.5
1 Iter. ($\ell = 4$)	1.0	3.3	21.9	42.1	63.2	74.4

Table 2
Speedup results for overall time, time for $\ell = 4$, pcg (solver) for $\ell = 4$ and one iteration of pcg for $\ell = 4$.

The wall-clock times for the pcg are given in the upper part of Table 1 for each level ℓ separately. Additionally we present the overall time for the finest grid ($\ell = 4$) and together for all grids ($\Sigma\ell = 1..4$) in the lower part of Table 1. This time includes the grid refinement together with the setup phase for the vector accumulation, the assembling of the matrices and the solution of the system by the pcg.

Table 2 and Fig. 5 show the corresponding speedup results. First the overall speedup for the accumulated time over all 4 levels is given. It performs well until $P = 16$ and is no longer optimal for $P = 60$. This loss of efficiency is due to the setup phase for interface unknowns which shows rather bad scalability in the current implementation. So it scales from 32 sec for $P = 16$ to 15 sec for $P = 60$ only. The speedup computed for $\ell = 4$ shows a slightly better behavior since coarse grid effects are neglected. However, the speedup computed for the solver and $\ell = 4$ only, shows much better results. Here the speedup is almost optimal for $P = 60$. For a more detailed analysis we consider the speedup with respect to one iteration for $\ell = 4$. Now we observe even super-speedups. However this is due to cache effects. Looking at Fig. 5 which shows the results

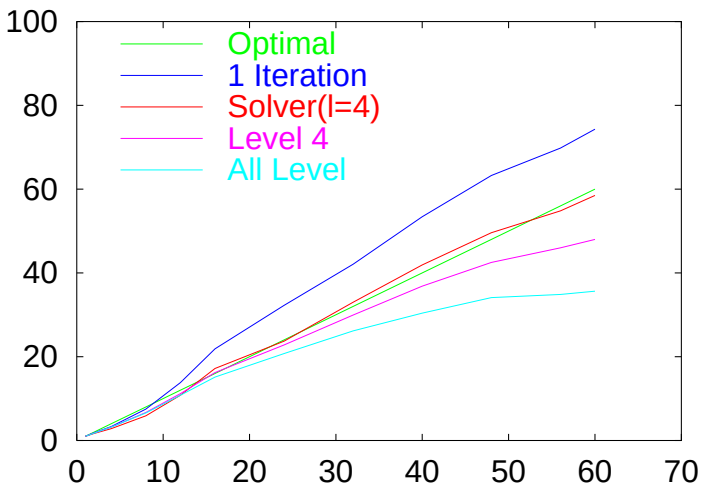

Fig. 5. Speedup results.

of Table 2 we see that for $P = 16$ the local problems are small enough to fit into local caches.

Summarizing, we observe an optimal scalability of our iterative solver. In the case of outer iterations, e.g. for nonlinear problems, where the linear solver is called repeatedly on a fixed mesh, the time required for the setup phase can be neglected and the solver which shows optimal efficiencies will dominate.

References

[1] D. Arnold, R. Falk, and R. Winther. Multigrid in H(div) and H(curl). *Numer. Math.*, 1999. (to appear).

[2] R. Beck, P. Deuflhard, R. Hiptmair, R. Hoppe, and B. Wohlmuth. Adaptive multilevel methods for edge element discretizations of Maxwell's equations. *Surv. Mat. Ind.*, 8:271–312, 1999.

[3] F. Brezzi and M. Fortin. *Mixed and Hybrid Finite Element Methods*, volume 15 of *Springer Series in Computational Mathematics*. Springer-Verlag, 1991.

[4] P. Ciarlet. *The Finite Element Method for Elliptic Problems*. North-Holland Publishing Company, Amsterdam-New York-Oxford, 1978.

[5] G. Haase. Parallel incomplete cholesky preconditioners based on the non-overlapping data distribution. *Parallel Computing*, 24(11):1685–1703, 1998.

[6] G. Haase. *Parallelization of Numerical Algorithms for PDEs*. Teubner, 1999. in German.

[7] G. Haase. A parallel amg for overlapping and non-overlapping domain decomposition. *Electronic Transactions on Numerical Analysis (ETNA)*, Feb. 2000. accepted for publication.

[8] G. Haase and M. Kuhn. Preprocessing for 2d FE-BE domain decomposition methods. *Comp. Vis. Sci.*, 2:25–35, 1999.

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

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

[11] R. Hiptmair. Multigrid methods for Maxwell's equations. *SIAM J. Numer. Anal.*, 36:204–225, 1999.

[12] 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.

[13] M. Kuhn. *Efficient Parallel Numerical Simulation of Magnetic Field Problem.* PhD thesis, Johannes Kepler University Linz, 1998.

[14] M. Kuhn, U. Langer, and J. Schöberl. Scientific computing tools for 3d magnetic field problems. In J. Whiteman, editor, *The Mathematics of Finite Elements and Applications*, pages 111–111. John Wiley & Sons, 2000.

[15] J. Nédélec. A new family of mixed finite elements in R^3. *Numer. Math.*, 50:57–81, 1986.

[16] M. Schinnerl, J. Schöberl, M. Kaltenbacher, U. Langer, and R. Lerch. Multigrid methods for the fast numerical simulation of coupled magnetomechanical systems. *ZAMM*, 1999. (to appear).

[17] J. Schöberl. NETGEN - An advancing front 2D/3D-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1:41–52, 1997.

[18] J. Schöberl. Object-oriented Finite Element Code *FEPP*, 1997. See also: http://www.sfb013.uni-linz.ac.at.