The multigrid method for an elliptic problem on a rectangular domain with an internal conducting structure and an inner empty space

K. S. Kang
High Level Support Team (HLST)
Max-Planck-Institut für Plasmaphysik
Boltzmannstraße 2
D-85748 Garching bei München
Germany
kskang@ipp.mpg.de

September 14, 2011

Abstract

This article is a technical report on the HLST project "2D Kinetic model of the Scrape-Off Layer" (KinSOL2D). The multigrid method is a well-known, fast and efficient algorithm to solve many classes of problems, but its implementation is quite complex. So this report gives a basic guideline for the implementation of the multigrid algorithm as a solver and as a preconditioner for the Krylov subspace iterative method. We report the implementation of the multigrid method for an elliptic problem on a rectangular domain with an internal conducting structure and an inner empty space. Issues which arose from performance tests on the HPC-FF computer located at the Jülich Supercomputing Center (JSC) are addressed.

Contents

1 Introduction

The Particle-in-Cell (PIC) code BIT1 [24] is restricted so far to 1D3V (1-dim real space + 3-dim velocity space) plasma and 2D3V neutral particle modeling with a reasonable scaling up to 1000 and more cores. Ongoing work is focused on enhancement of the code in real space to 2D3V plasma simulations of the Scrape-Off-Layer (SOL). The increase of the dimensionality of the code to 2D or even 3D seems to be straight forward. However, the Poisson solver in 2D has been identified as a bottleneck for the scaling properties. It is mandatory that this part of the code also scales to very high core numbers to maintain the good scaling property of the whole code. So the work plan is to develop a good scaling Poisson solver in 2D.

2 Assessment of potential solvers

Possible candidates for solvers are a multigrid solver or, depending on the type of the matrix, a preconditioned Conjugated Gradient Method (CGM) or Generalized Minimal Residual Method (GMRES) [23]. A combination of both is also thinkable where the multigrid method is used as a preconditioner for either the CGM or the GMRES method. In contrast, a parallel direct solver doesn't seem to be a good choice as it usually doesn't scale as efficiently to large numbers of cores and its memory consumption is larger compared to iterative methods.

The multigrid method is a well-known, fast and efficient algorithm to solve many classes of problems including the linear and nonlinear elliptic, parabolic, hyperbolic, Navier-Stokes equation, and Magnetohydrodynamics (MHD) [5, 13, 18, 20, 25]. Although the multigrid method is complex to implement, researchers in many areas think of it as an essential algorithm and apply it to their codes because the number of operations of the multigrid method depends on the degrees of freedom times the number of levels (log of the number of degrees of freedom).

To implement and analyze the multigrid method, we have to consider two main parts of the multigrid algorithm, the intergrid transfer operators and the smoothing operator, separately. The intergrid transfer operators depend on the discretization scheme and are highly related with the discretization of the matrix. The smoothing operator will be implemented according to the matrix-vector multiplication. So we have to determine the appropriate discretization method which includes the generation of the matrix and an efficient implementation of the matrix-vector multiplication.

If a multigrid solver converges, it usually converges very fast which is the case on most problems. However, the multigrid method as a solver does not guarantee convergence. In contrast, iterative Krylov subspace methods, which include the CGM and GMRES, guarantee convergence and can be further improved by preconditioners to speed up the convergence rate.

The multigrid method is also well-known to act as a very efficient preconditioner. The preconditioned CGM can be used only for symmetric and positive definite problems and has to use the A-norm instead of the L^2-norm. For non-symmetric or non-positive definite systems, which can arise e.g. through the boundary condition treatment the GMRES method has to be used. In general, the preconditioned system of a symmetric system is not symmetric for the same inner product. However under certain conditions such a system can be symmetric in a different inner product (A-inner product or energy inner product) and the less costly CGM method can be

used. The preconditioned GMRES works for non-symmetric or non-positive definite problems, but needs more working memory. To reduce the working memory in GMRES, we can use Restart GMRES which does not strictly guarantee the convergence, but converges for most problems.

In section 3, we explain the basics of the multigrid method and related issues as e.g. the implementation of the program in serial and parallel versions. We describe the model problem, its discretization and the intergrid transfer operators in section 4. Then, we explain the parallelization of the program in section 5. An overview of the numerical experiments of the multigrid method is given in section 6. Finally, the scaling properties of the multigrid method on HPC-FF at the Jülich Supercomputing Center (JSC) are shown in section 7.

3 The multigrid method as a solver and preconditioner

In this section, we consider and discuss the issues about the implementation of the multigrid method as a solver and as a preconditioner for the iterative Krylov subspace method.

3.1 Basics of the multigrid method

We start to explain the multigrid method in an abstract form as in [3]. We consider the finite functional spaces $\{V_k\}$ for $k = 1, \ldots, J$ and operators $A_k : V_k \to V_k$ and the intergrid transfer operators as e.g. the prolongation operator $I_k : V_{k-1} \to V_k$ and the restriction operator $P_{k-1}^0 : V_k \to V_{k-1}$ defined by

$$(I_k v, w)_k = (v, P_{k-1}^0 w)_{k-1}, \quad \forall v \in V_{k-1}, \forall w \in V_k, \tag{3.1}$$

for the inner products $(\cdot, \cdot)_k$ of V_k.

Let $R_k : V_k \to V_k$ for $k = 1, \ldots, J$ be the linear smoothing operators, R_k^T denotes the adjoint of R_k with respect to the inner product $(\cdot, \cdot)_k$, and defined by

$$R_k^{(l)} = \begin{cases} R_k, & l \text{ is odd}, \\ R_k^T, & l \text{ is even}. \end{cases} \tag{3.2}$$

The multigrid operator $B_k : V_k \to V_k$ is defined recursively as follows.

Multigrid Algorithm (MG): Let $1 \leq k \leq J$ and p be a positive integer. Set $B_1 = A_1^{-1}$ (lowest level solver). Assume that B_{k-1} has been defined then $B_k g$ for $g \in V_k$ will be defined by

(1) Set $x^0 = 0$ and $q^0 = 0$. (initialization)

(2) Define x^l for $l = 1, \ldots, m(k)$ by

$$x^l = x^{l-1} + R_k^l(g - A_k x^{l-1}). \quad \text{(pre-smoothing)}$$

(3) Define $y^{m(k)} = x^{m(k)} + I_k q^p$, where q^i for $i = 1, \ldots, p$ is defined by

$$q^i = q^{i-1} + B_{k-1}[P_{k-1}^0(g - A_k x^{m(k)}) - A_{k-1} q^{i-1}]. \quad \text{(correction)}$$

(4) Define x^l for $l = m(k) + 1, \ldots, 2m(k)$ by

$$x^l = x^{l-1} + R_k^l(g - A_k x^{l-1}). \quad \text{(post-smoothing)}$$

(5) Set $B_k g = y^{2m(k)}$.

In the MG algorithm, $m(k)$ gives the number of pre- and post-smoothing iterations and can be varied as a function of k.

The different multigrid algorithms are classified by the terms: Full, V-cycle, variable V-cycle and W-cycle multigrid algorithm according to the number of smoothing iterations $m(k)$ and p. The full multigrid algorithm starts with the coarsest level to get the initial solution on the finest level, i.e., $m(k) = 0$ of the first cycle. When $p = 1$, we call the MG algorithm a V-cycle multigrid algorithm. The V-cycle multigrid algorithm has a fixed $m(k) = m$ for all levels k and the variable V-cycle has an increasing number of smoothing iterations $m(k)$ as the level k is decreased. When $p = 2$, we call the MG algorithm a W-cycle multigrid algorithm which has a fixed $m(k) = m$. We illustrate the basic idea of the V-cycle multigrid method and its structure of levels in Figs. 1 and 2. Also, we plot the cycle of the Full, V-cycle and W-cycle multigrid method in Fig. 3.

From the beginning of 1980's, many results on convergence of multigrid algorithms were published [1, 2, 4, 10, 17]. Many researchers showed that the W-cycle multigrid algorithm has a fixed bounded error reduction factor when a sufficient number of smoothing iterations is used. Instead the variable V-cycle has just a uniform boundary of the condition number of the preconditioned system. Even then, the V-cycle multigrid method converges typically well in numerical experiments. The theoretical results are limited to some specific problems, and the required number of iterations increases as the number of levels increases. As the number of levels increases with the logarithm of the unknowns n, the number of iterations is of $O(\log(n))$.

The W-cycle and variable V-cycle multigrid methods need more work on the coarser levels on which computations have poor scaling properties. So we only consider the V-cycle multigrid method, being optimal for calculations on the massively parallel machines such as the HPC-FF computer.

A typical implementation of the multigrid algorithm can be seen in Fig. 4. First of all, the matrix A_J is generated on the finest level that has to be solved

$$A_J u_J = f_J.$$

This matrix and/or system will be fixed by the discretization scheme which is highly dependent on the partial differential equation given by the physicist. Depending on the particular multigrid algorithm being chosen one can either generate the matrices A_k on each level k in the same way as A_J was generated or instead in a more abstract manner by multiplying the matrices A_{k+1} with the intergrid transfer operators from both sides.

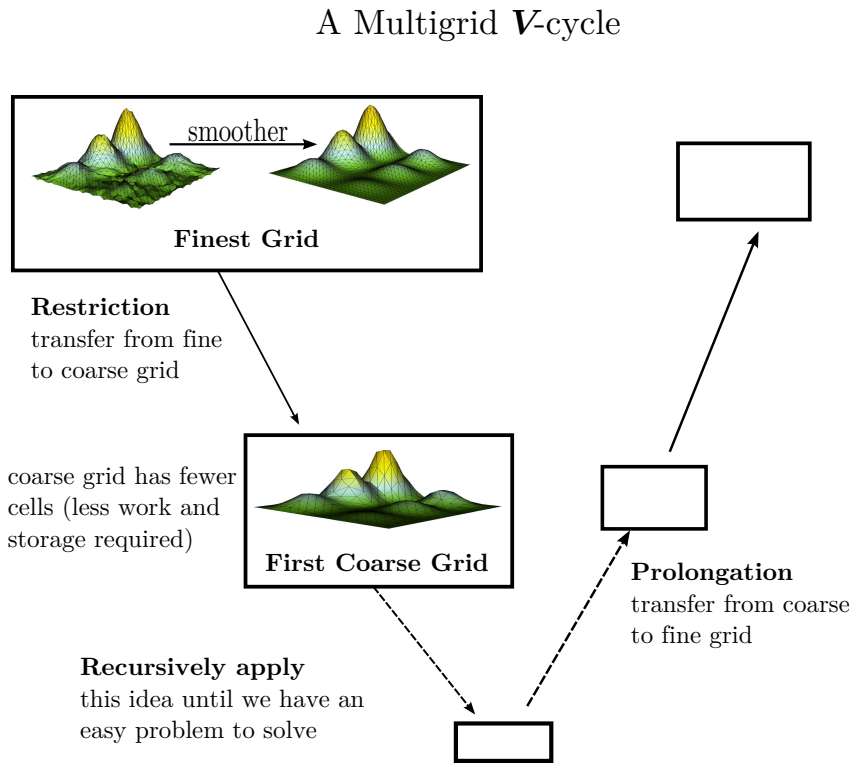


Figure 1: Basic idea of the V-cycle multigrid method.

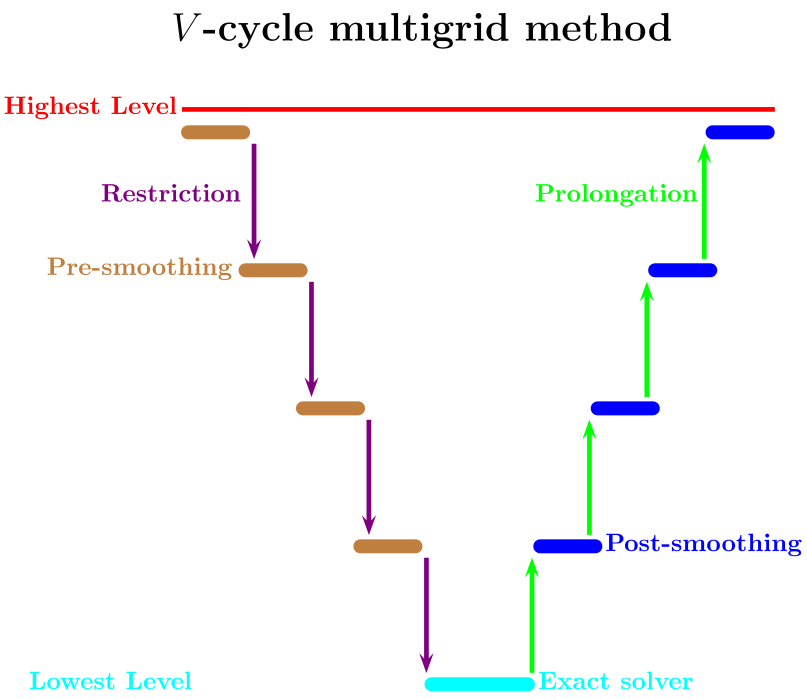
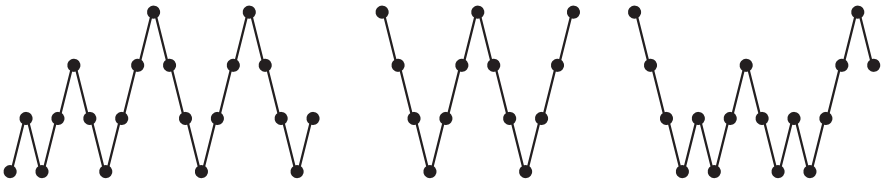


Figure 2: The different levels of the V-cycle multigrid method.



(a) Full multigrid (b) V-cycle MG (c) W-cycle MG

Figure 3: Full, V-cycle and W-cycle multigrid method with four levels.

```
MGCycle(k, A_k, u_k, f_k)
  If k = 0, the u_k = A_k^{-1} f_k
  else GaussSeidelIter(A_k, u_k, f_k)
       r_k = f_k − A_k u_k
       q_{k−1} = P^0_{k−1} r_k
       x_{k−1} = 0
       MGCycle(k − 1, A_{k−1}, x_{k−1}, q_{k−1})
       if p = 2 (W-cycle), then
            r_{k−1} = q_{k−1} − A_{k−1} x_{k−1}
            y_{k−1} = 0
            MGCycle(k − 1, A_{k−1}, y_{k−1}, r_{k−1})
            x_{k−1} = x_{k−1} + y_{k−1}
       u_k = u_k + I_k x_{k−1}
       GaussSeidelIter(A_k, u_k, f_k)
```

Figure 4: An implementation of the multigrid method.

Next, we need an implementation of the smoothing operators which can be used as an iterative solver. The smoothing operators have to be implemented on each level except on the lowest one.

Then, we need to generate the intergrid transfer operators, in detail the restriction and prolongation operators which have the adjoint relation with each other on the same level.

Finally, we have to implement the lowest level solver which provides an exact solution on the lowest level. It can be either a direct method or an iterative method.

We will consider the issues of the implementation of the matrices and operators in the following sections.

3.2 Discretization scheme

To solve a given partial differential equation, we have to determine the discretization scheme acting on the selected meshes. There are many different discretization schemes and they have quite different properties. So at the beginning, a decision has to be made according to the problem under consideration. In this section, we summarize the properties of the well-known discretization schemes as, e.g. the finite difference method (FDM), the finite

element method (FEM), and the finite volume method (FVM). However, we will avoid the spectral method as it leads to dense matrices which are most efficiently solved by direct methods and not iterative ones, like the multigrid method.

The numerical solution to the PDE is an approximation to the exact solution and is obtained by using a discrete representation to the PDE at the points x_j. Let us denote this numerical solution as U such that

$$U_j \approx u(x_j).$$

Depending on the location of the points x_j, we can classify the discrete scheme as cell-centered or vertex-centered. Thus, the numerical solution is a collection of finite values

$$U = [U_1, U_2, \ldots, U_m].$$

The finite different method (FDM): The FDM starts from the definition of the derivative in calculus

$$\frac{\partial u}{\partial x}(x_j) = \lim_{h \to 0} \frac{u(x_j + h) - u(x_j)}{h}.$$

We use this formula with a small finite value of $h = \Delta x$, i.e., we approximate

$$\frac{\partial u}{\partial x}(x_j) \approx \frac{u(x_j + h) - u(x_j)}{h} \quad \text{(forward difference)}.$$

To analyze the error of the FDM, we consider the local truncation error (LTE) which is the error that results by substituting the exact solution into the finite difference formula. Errors in the approximations to the derivative are calculated using Taylor approximations around a grid point x_j, i.e.,

$$u(x_j + \Delta x) = u(x_j) + \frac{\partial u}{\partial x}(x_j)\Delta x + \frac{\partial^2 u}{\partial x^2}(x_j)\frac{(\Delta x)^2}{2} + O((\Delta x)^3).$$

Thus,

$$\frac{\partial u}{\partial x}(x_j) \approx \frac{u(x_j + \Delta x) - u(x_j)}{\Delta x} + \frac{\partial^2 u}{\partial x^2}(x_j)\frac{\Delta x}{2} + O((\Delta x)^2).$$

The forward difference is a first order accurate approximation to the partial derivative $\frac{\partial u}{\partial x}$ at x_j and the LTE is of $O(\Delta x)$.

As shown above by invoking Taylor's theorem, we can prove that the LTE of FDM goes to zero as Δx approach zero, i.e., fulfills consistency.

To show the convergence of the discretization problem, one usually applies the Lax-Equivalence Theorem, i.e., proof its stability.

Lax-Equivalence Theorem: A consistent approximation to a well-posed problem is convergent if and only if it is stable.

The stability implies that the numerical solution remains bounded at any given point in time t for the time dependent problem. In general, the stability of the FDM scheme is harder to prove than consistency. The stability of the FDM scheme can be proved using either

- Eigenvalue analysis of the matrix representation of the FDM
- Fourier analysis on the grid (von Neumann analysis)
- In the case of hyperbolic PDEs, computing the domain of dependence of the numerical scheme resulting, e.g. in the CFL condition

The FDM can be easily understood without higher mathematical knowledge and its consistency can be usually shown without difficulty. But, the FDM scheme needs special treatment on general domains in two- or three-dimensions.

The finite element method (FEM): For a given PDE, we have a weak formulation in the functional space V by multiplying both sides by an arbitrary test function $v \in V$ and integrating over volume. After the discretization of the domain (T_h), we define a finite-dimensional functional space V_h which can be either a subspace of V (conforming) or not (non-conforming). Usually, we need the property that functions in V_h can get arbitrarily close to functions in V as h decreases to zero.

We construct the basis functions $\{\phi_j\}_{j=1}^M$ of the finite dimensional space V_h which have a local support. Then we approximate the solution u and the test function v with the basis functions, i.e.,

$$u(x) \approx u_h(x) = \sum_{j=1}^{M} u_j \phi_j(x), \quad v(x) = \sum_{j=1}^{M} v_j \phi_j(x).$$

Because v_j can be an arbitrary vector, we can choose the unit vector $e_j = (0, \ldots, 0, 1, 0, \ldots, 0)$ and receive M equations for M unknowns u_j. Thus, the discretization of the derivatives results in a linear system.

The FEM has a high degree of flexibility and can be applied to complicated geometries. Also, we can easily choose higher-order approximations. There is always a strong mathematical foundation to understand and apply the FEM to a PDE.

The finite volume method (FVM): Many PDEs of interest are derived from physical models with underlying conservation laws, i.e., the rate of change of $u(x,t)$ within a volume Ω is equal to the flux past the boundary

$$\frac{\partial}{\partial t}\int_{\Omega} u(x,t)\,\mathrm{d}x + \int_{\partial\Omega} \overrightarrow{f}(u)\cdot\overrightarrow{n}\,\mathrm{d}s = 0$$

where f is a flux function. Instead of the pointwise approximations on a grid, the FVM approximates the average integral value on a reference volume which means that the solution of the FVM satisfies the conservation law within the numerical errors.

The FVM can handle discontinuities in solutions as well as the natural choice of heterogeneous materials as one can assign each grid cell different material parameters. Even though the analysis for convergence, accuracy and stability of the FVM can not be proven without a strong mathematical foundation in contrast to the FEM, many authors [6, 7, 8, 9, 11, 12, 17, 21] have developed corresponding methods to show the convergence, accuracy and stability of the FVM.

3.3 Intergrid transfer operators

The intergrid transfer operators are key operators for the multigrid method. They consist of the fine-to-coarse (restriction) and the coarse-to-fine (prolongation) transfer operators. These two operators are adjoint to each other on a certain inner product of the vector space.

According to how the intergrid transfer operators are constructed, we can classify them into the geometric and algebraic multigrid method.

The **geometric multigrid method (GMM)** was studied at first and had a very good performance on many interesting problems including the Poisson problem and the Navier Stokes problem. The intergrid transfer operators in the GMM can be defined according to the available geometric information and the chosen discretization scheme. In the conforming cases, there exists the so-called injection operators from the coarse level function space to the fine level function space. With the help of these injection operators the intergrid transfer operators can be defined. However, such injection

operators do not exist for the nonconforming cases, so one may define the intergrid transfer operators as in [17]. There is a minimum approximation order of the intergrid transfer operators according to the approximation order of the discretization scheme [25]. The order of the intergrid transfer operators clearly affects the convergence rate per iteration of the multigrid method as shown in [19]. Usually, the higher order intergrid transfer operators have better convergence rate per iteration but usually need more CPU operations and/or are more complicated to implement. So we need to compromise the order of the intergrid transfer operators in the multigrid method. For the implementation of the GMM, one needs to know about the domain, i.e., how to handle the geometric information and discretization. Due to these requirements of the GMM, the implementation of a problem with complex geometry can be very difficult. Hence, for some problems the performance of the GMM can be too poor to be acceptable.

The finest meshes in the GMM usually are constructed by refinement from the coarsest mesh. Such an approach is called a "down-top approach". It is preferred by mathematicians as it is easy to understand and well embedded in the mathematical theory. However, this approach has some restrictions when it is applied to problems with complex domains including e.g. small holes inside the domain.

Instead one can construct the coarser meshes including the coarsest mesh itself from the finest mesh. We call such an approach a "top-down approach". The top-down approach can be applied to any complex domain, but causes other problems which do not occur in the down-top approach. Such problems which are hardly considered by mathematicians have to be investigated and to be resolved. For the special problem of a rectangular domain with an inner empty space and an internal conducting structure, we will use the Neumann boundary condition on the inner empty space and the Dirichlet boundary condition on the internal conducting area. We consider two kinds of treatment for the Neumann boundary condition. In addition, we will face the relative positioning effect of the internal conducting area in relation to the uniformly structured mesh of the rectangular domain in Sec. 6.4

The **algebraic multigrid method (AMM)** is a black-box method which can be applied to any linear system. In the AMM, the intergrid transfer operators are defined according to the matrix of the system and the matrix equations on the coarser levels are defined by the intergrid transfer operators. The AMM only considers the finest meshes to solve the PDE, so many researchers are developing the AMM as program libraries to iteratively

solve matrix equations.

The convergence rate of the AMM per iteration is usually slower than the GMM, but improvements are being made with the use of sophisticated algorithms to generate the intergrid transfer operators by considering the matrices with methods from graph theory. Good results with fast convergence rates per iteration on the AMM are now starting to be achieved. Nevertheless, current approaches still need a lot of work to generate efficient intergrid transfer operators.

3.4 Smoothing operators

The smoothing operator is another essential part of the multigrid method. The starting point of the multigrid method is that many iterative methods tend to reduce the high-frequency components of the error rapidly but reduce the low-frequency components of the error much more slowly. So good smoothing operators reduce the high-frequency components of the error rapidly. Another desirable property for a smoothing operator is its simple and easy implementation. Traditionally, the damped Jacobi and the Gauss-Seidel method are preferred by many researchers because these two methods have the above mentioned properties. But, many other iterative methods can be used as smoothing operators. Next, we will summarize these iterative methods with their advantages and disadvantages.

The simplest type of iterative method for solving

$$Ax = b \tag{3.3}$$

with the $N \times N$ matrix A has the form

$$x^{(k+1)} = Gx^{(k)} + c \tag{3.4}$$

where the matrix G and vector c are chosen so that a fixed point of Eq. (3.4) is a solution to Eq. (3.3). This method is said to be stationary if G and c are constant over all iterations. One way to obtain a suitable matrix G is by the following splitting ansatz,

$$A = M - N$$

with the non-singular matrix M. We can then take $G = M^{-1}N$ and $c = M^{-1}b$, so that the iterative scheme becomes

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b, \text{ i.e., } Mx^{(k+1)} = Nx^{(k)} + b.$$

Thus, the iteration scheme is convergent [14, 22] if

$$\rho(G) = \rho(M^{-1}N) < 1.$$

To describe the following iterative methods, we split

$$A = D + L + U \tag{3.5}$$

where D is a diagonal matrix and L and U are the strict lower and upper triangular parts of A, respectively.

The **Richardson** iterative method is the simplest iterative method with

$$x^{(k+1)} = x^{(k)} + \frac{1}{\lambda_N}\left(b - Ax^{(k)}\right), \text{ i.e., } G = I - \frac{1}{\lambda_N}A, \text{ and } c = \frac{1}{\lambda_N}b \tag{3.6}$$

where λ_N is the largest eigenvalue of A or an appropriate approximation. The smoothing property of the Richardson method can be proved easily. So this method was used for the analysis of the multigrid method at the end of 70's and the beginning of 80's.

The **Jacobi** method is a simple iterative method with $M = D$ and $N = -(L + U)$, i.e.,

$$x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)}) = -D^{-1}(L + U)x^{(k)} + D^{-1}b,$$

and is guaranteed to converge under conditions that are often satisfied in practice. But, the convergence rate of the Jacobi method is usually unacceptably slow. Also, a simple analysis on the convergence rate of the Jacobi method shows that the Jacobi method reduces the high- and low-frequency components of the error relatively slowly. Instead, it reduces the middle range frequency components of the error rapidly [16].

The **damped Jacobi** method is defined by

$$x^{(k+1)} = (1 - \omega)x^{(k)} + \omega D^{-1}(b - (L + U)x^{(k)}) = (I - \omega D^{-1}A)x^{(k)} + wD^{-1}b.$$

The damped Jacobi method has been developed to further improve the smoothing property of the Jacobi method at the cost of a damped convergence rate. For optimal smoothing property the tuning parameter is within the range of $\omega = 0.6 \sim 0.8$. However, the (damped) Jacobi method requires a doubled storage space for the solution because the old values are needed throughout each sweep, and therefore the updated values of the solution

cannot overwrite the old ones until the sweep has been completed. On the other hand, due to this property, the Jacobi method is well suited on parallel machines because it doesn't introduce any additional communication overhead. Hence, it can be used to produce results which are identical for both the serial and parallel version of the code. This is especially helpful when debugging the parallel implementation of the multigrid algorithm.

The **Gauss-Seidel** method uses each updated component of the solution as soon as it has been computed, i.e., $M = D + L$ and $N = -U$. The Gauss-Seidel method can be written as

$$x^{(k+1)} = D^{-1}(b - Lx^{(k+1)} - Ux^{(k)}) = (D + L)^{-1}(b - Ux^{(k)}).$$

In addition to faster convergence, another benefit of the Gauss-Seidel method is that a doubled storage is not needed for the solution, since the newly computed components can overwrite the old ones immediately. However, this feature makes the implementation of the algorithm typically harder on parallel machines. An exception is the special case of the red-black Gauss-Seidel method on structured grids.

The **successive over-relaxation (SOR)** uses the step of the next Gauss-Seidel iteration as a search direction, but with a fixed search parameter denoted by ω $(0 < \omega < 2)$, i.e, $x^{(k+1)} = (1 - \omega)x^{(k)} + \omega x_{GS}^{(k+1)}$. Hence, the SOR method can be written as

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \omega[D^{-1}(b - L^{(k+1)} - Ux^{(k)}) - x^{(k)}] \\ &= (D + \omega L)^{-1}[(1 - \omega)D - \omega U]x^{(k)} + \omega(D + \omega L)^{-1}b. \end{aligned}$$

The SOR can accelerate the convergence rate of the Gauss-Seidel method. Like the Gauss-Seidel method, the SOR method makes repeated forward sweeps through the unknowns, updating them successively. The **symmetric SOR (SSOR)** alternates forward and backward sweeps through the unknowns. SSOR is not necessarily faster than SOR, but it has the advantage of being symmetric which makes SSOR useful as a preconditioner.

The **Kancmarz** method is introduced in [15] and defined as an iteration

$$x^{(k+1)} = x^{(k)} + \frac{b_i - a_i^T x^{(k)}}{a_i^T a_i} a_i,$$

where a_i be the i-th column vector. Kaczmarz's approach is a projection method that is also referred to as algebraic reconstruction technique and is

used for solving linear systems from image reconstruction problems. The Kaczmarz method is defined by applying the Gauss-Seidel method on AA^T which is always symmetric and positive, so it can be applied on singular problems. This method can also be used as a smoother in the multigrid method as shown in [16].

The **alternative direction iteration (ADI)** method, as introduced by Peaceman and Rachford, is an iterative scheme for $Ax = b$ ($A = A_1 + A_2$ with A_1 and A_2 symmetric and positive definite) which consists of solving the following systems $\forall k \geq 0$

$$\begin{aligned}(I + \alpha_1 A_1)x^{(k+1/2)} &= (I - \alpha_1 A_2)x^{(k)} + \alpha_1 b,\\ (I + \alpha_2 A_2)x^{(k+1)} &= (I - \alpha_2 A_1)x^{(k+1/2)} + \alpha_2 b,\end{aligned}$$

where α_1 and α_2 are two real parameters. For the 2-dim Poisson problem, A_1 and A_2 are the second order derivative operators on the x- and y-directions. So the ADI method solves two one-dim problems which can be easily analyzed and solved. This method is effective when the domain has a relatively long width in comparison with the height or if the coefficients ϵ_{11} and ϵ_{22} of the elliptic PDE

$$-\epsilon_{11}\frac{\partial^2 u}{\partial x^2} - \epsilon_{22}\frac{\partial^2 u}{\partial y^2} = f$$

are largely different in absolute value such as e.g. $\epsilon_{11} = 1$ and $\epsilon_{22} = 10000$.

The **incomplete LU (ILU)** factorization method is a process that computes $P = L_{\text{in}}U_{\text{in}}$ where L_{in} is a lower triangular matrix and U_{in} is an upper triangular matrix. These matrices are approximations of the exact matrices L and U of the LU factorization, but have the same sparse pattern as the lower- and upper-part of the original matrix. The ILU method has a very good performance as a smoother for the multigrid method, but it is complicated to implement and very hard to analyze.

3.5 The Krylov subspace method and preconditioners

We introduce the **Krylov subspace** of order m

$$K_m(A; v) = \text{span}\{v, Av, \ldots, A^{m-1}v\}. \tag{3.7}$$

The Krylov subspace is a subspace of the set spanned by all vectors $u \in R^N$ that can be written as $u = p_{m-1}(A)v$, where p_{m-1} is a polynomial in A of

degree $\leq m-1$. Clearly, we have $K_1(A;v) \subseteq K_2(A,v) \subseteq K_3(A,v), \ldots$, and the dimension increases at most by one for each step.

Theorem 3.1 *Let $A \in R^{N\times N}$ and $v \in R^N$. The Krylov subspace $K_m(A;v)$ has a dimension equal to m if and only if the degree of v with respect to A, denoted by $deg_A(v)$, is not less than m, where the degree of v is defined as the minimum degree of a monic non-null polynomial p in A, for which $p(A)v = 0$.*

We can classify the Krylov methods by how they compute the best approximation to the solution in $K_m(A;v)$.

- The residual $r_m = b - Ax_m$ is orthogonal to $K_m(A;v)$: Conjugate Gradient
- The residual r_m has a minimum norm for x_m in $K_m(A;v)$: GMRES and the minimal residual method (MINRES)
- The residual r_m is orthogonal to a different space $K_m(A^T;v)$: BiConjugate Gradients
- The error d_m has a minimum norm: the symmetric LQ method (SYMMLQ).

The above mentioned conjugate gradient method (CGM) works only if A is symmetric and positive definite to the inner product of the vector space. For the CGM, we have the following theorems about its termination and convergence rate.

Theorem 3.2 *Let A be a symmetric and positive matrix. The conjugate gradient method to solve Eq. (3.3) terminates after at most N steps. Moreover, the error $d^{(m)}$ at the m-th iteration (with $m < N$) is*

$$\|d^{(m)}\|_A \leq 2\rho^k \|d^{(0)}\|_A, \text{ with } \rho = \frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1} \tag{3.8}$$

where $\kappa(A)$ is the condition number of A, i.e., $\kappa(A) = \frac{\lambda_N}{\lambda_1}$ with the largest eigenvalue λ_N and the lowest eigenvalue λ_1 of A.

Theorem 3.2 can be applied for any Krylov method and shows that the convergence rate ρ is very good when $\kappa(A)$ is small and the convergence rate ρ is converging to 1 as $\kappa(A)$ is increasing. For large real-world problems, the condition number $\kappa(A)$ is very large and the Krylov space method converges very slowly.

In practice, the Krylov space solvers are combined with **preconditioning**: $Ax = b$ is replaced by

$$\underbrace{MA}_{\hat{A}}x = \underbrace{Mb}_{\hat{b}}, \text{ or } \underbrace{AM}_{\hat{A}}\underbrace{M^{-1}x}_{\hat{x}} = b, \text{ or } \underbrace{M_{\mathrm{L}}AM_{\mathrm{R}}}_{\hat{A}}\underbrace{M_{\mathrm{R}}^{-1}x}_{\hat{x}} = \underbrace{M_{\mathrm{L}}b}_{\hat{b}}.$$

The first two cases are referred to as left and right preconditioning, respectively, while for the last case we apply a split preconditioner $M_{\mathrm{L}}M_{\mathrm{R}}$. To be a good preconditioner, $\hat{A}$ needs a small condition number, i.e., M and $M_{\mathrm{L}}M_{\mathrm{R}}$ are approximate inverses of A ($M^{-1} \approx A$). The preconditioner M also has the properties that the system $M^{-1}y = z$ can be easily solved for any z. Applying a preconditioned Krylov subspace method just means to apply the method to the preconditioned problem $\hat{A}\hat{x} = \hat{b}$ instead of $Ax = b$.

To use preconditioned CGM, we have to check the symmetry of the preconditioner to the inner product. We summarize the implementation of the CGM and preconditioned CGM (PCGM) in Fig. 5. In Fig. 5 (b), compared to the CGM we need only one step more to compute $z(k+1) = Mr^{(k+1)}$.

For the non-symmetric problem, we consider the GMRES only. The GMRES can be considered as a generalization of the MINRES algorithm and is theoretically equivalent to the Generalized Conjugate Residual (GCR) method and to the ORTHODIR [23]. The GMRES terminates at most after N iterations, yielding the exact solution. We describe the GMRES algorithm in Fig. 6 (a). For the GMRES method, the number of vectors for which storage is required increases as k and the number of multiplications as $\frac{1}{2}k^2N$ when the iteration k is increased. To remedy this difficulty, we use the restarted GMRES method (GMRES(m)) which restarts the algorithm every m steps where m is some fixed integer parameter as in Fig. 6 (b).

For the GMRES and GMRES(m) algorithms in Fig. 6, we have to implement a routine to get y_k which minimizes

$$\|\beta_1 - \bar{H}_m y_k\|$$

in step 3 as in [23].

We obtain the preconditioned GMRES (PGMRES) when we apply the GMRES to the preconditioned problem $\hat{A}\hat{x} = \hat{b}$.

The multigrid method is also a well-known preconditioner. So we use PCGM and PGMRES with a multigrid preconditioner because the convergence of PCGM and PGMRES is always guaranteed.

Given $x^{(0)} \in R^N$,
set $r^{(0)} = b - Ax^{(0)}$
and $p^{(0)} = r^{(0)}$
then, for $k = 0, 1, \ldots,$
until convergence, compute

1. $\alpha_k = \dfrac{{r^{(k)}}^T r^{(k)}}{{p^{(k)}}^T A p^{(k)}},$

2. $x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)},$

3. $r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)},$

4. $\beta_k = \dfrac{{r^{(k+1)}}^T r^{(k+1)}}{{r^{(k)}}^T r^{(k)}},$

5. $p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}.$

(a) CGM

Given $x^{(0)} \in R^N$,
set $r^{(0)} = b - Ax^{(0)}$,
$z^{(0)} = Mr^{(0)}$ and $p^{(0)} = z^{(0)}$
then, for $k = 0, 1, \ldots,$
until convergence, compute

1. $\alpha_k = \dfrac{\left(z^{(k)}\right)^T r^{(k)}}{\left(p^{(k)}\right)^T A p^{(k)}},$

2. $x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)},$

3. $r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)},$

4. $z^{(k+1)} = M r^{(k+1)},$

5. $\beta_k = \dfrac{\left(z^{(k+1)}\right)^T r^{(k+1)}}{\left(z^{(k)}\right)^T r^{(k)}},$

6. $p^{(k+1)} = z^{(k+1)} + \beta_k p^{(k)}.$

(b) PCGM

Figure 5: The implementation of the CGM and PCGM.

1. Start: Choose $x^{(0)} \in R^N$,
   set $r^{(0)} = b - Ax^{(0)}$
   and $v^{(1)} = \dfrac{r^{(0)}}{\|r^{(0)}\|}$.
2. Iterate: For $k = 0, 1, \ldots,$
   until satisfied do:
   $h_{i,j} = (Av_j, v_i)$, $i = 1, 2, \ldots, j$,
   $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$,
   $h_{j+1,i} = \|\hat{v}_{j+1}\|$, and
   $v_{j+1} = \dfrac{\hat{v}_{j+1}}{h_{j+1,j}}$,
3. From the approximate solution:
   $x^{(k)} = x^{(0)} + V_k y_k$,
   where y_k minimizes
   $\|\beta e_1 - \bar{H}_m y_k\|$, $y_k \in R^m$.

(a) GMRES

1. Start: Choose $x^{(0)} \in R^N$,
   set $r^{(0)} = b - Ax^{(0)}$
   and $v^{(1)} = \dfrac{r^{(0)}}{\|r^{(0)}\|}$.
2. Iterate: For $k = 0, 1, \ldots, m-1$
   do:
   $h_{i,j} = (Av_j, v_i)$, $i = 1, 2, \ldots, j$,
   $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$,
   $h_{j+1,i} = \|\hat{v}_{j+1}\|$, and
   $v_{j+1} = \dfrac{\hat{v}_{j+1}}{h_{j+1,j}}$,
3. From the approximate solution:
   $x^{(k)} = x^{(0)} + V_k y_k$,
   where y_k minimizes
   $\|\beta e_1 - \bar{H}_m y_k\|$, $y_k \in R^m$.
4. Restart: Compute
   $r^{(m)} = b - Ax^{(m)}$:
   if satisfied the stop. else compute
   $x^{(0)} = x^{(m)}$, $v^{(1)} = \dfrac{r^{(m)}}{\|r^{(m)}\|}$
   and go to 2.

(b) GMRES(m)

Figure 6: The implementation of the GMRES and restarted GMRES.

4 The model problem and its discretization

From now on, we focus on the PDE defined on the rectangular domain with an internal conducting structure and an inner empty space. Such a domain represents the simulation domain for the outer plasma of confined fusion experiments such as ASDEX, JET, or ITER.

We consider the following second order partial differential equation:

$$-\left[\frac{\partial}{\partial x}\epsilon(x,y)\frac{\partial}{\partial x}+\frac{\partial}{\partial y}\epsilon(x,y)\frac{\partial}{\partial y}\right]\phi(x,y,t)=\rho(x,y,t) \tag{4.1}$$

with the boundary condition for the outer wall

$$\phi_w(x,y,t)=0, \qquad \oint \overrightarrow{E}_w \cdot \mathrm{d}\overrightarrow{S}=\frac{1}{\epsilon_0}\sigma_w, \tag{4.2}$$

where integration goes over the wall surface and E_w and σ_w denote the electric field and total surface charge at the outer wall, the boundary condition at the inner empty surface

$$E_n(x,y,t)=\frac{\partial\phi(x,y,t)}{\partial n}=0, \tag{4.3}$$

where E_n is the electic field normal to the surface (n is the normal vector on the boundary), and the boundary condition for the internal conductor

$$\phi_c(x,y,t)=\phi_c(t) \tag{4.4}$$

as in Fig. 7.

It is well known that on the corners of the inner empty space the solution ϕ of Eq. (4.1) for a general function ρ has corner singularities. Such corner singularities affect the performance of the multigrid algorithm because the theory predicts that the convergence factor of the multigrid algorithm is a function of the regularity of the general solution. The same problem occurs for the internal conducting area. However, as the internal conducting area is very small compared to the whole simulation domain we suppose that at least the impact of the corner singularities triggered by the internal conducting area should be of minor importance. Another more dominant problem seems to be the relative orientation, i.e., the matching of the inner empty space and the internal conducting area with the nodal points of the grids of the coarser levels.

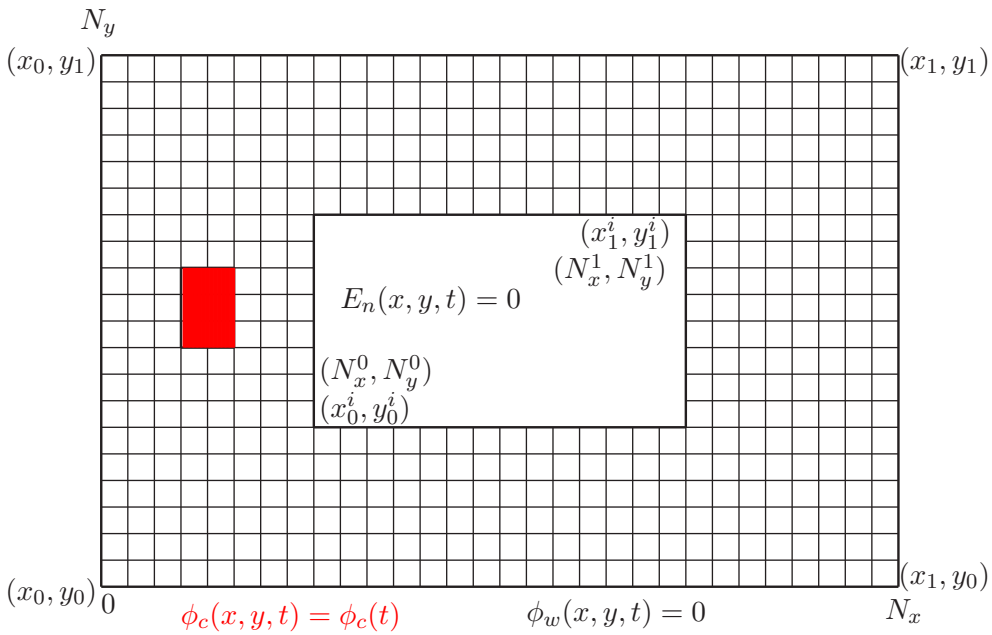


Figure 7: Rectangular domain with internal conducting structure [■: internal structure (conductor)]

4.1 The finite difference method

For compatibility reason regarding to KinSOL2D project, we consider the finite difference scheme on uniform meshes which is easy to understand and the code developer use it. By using the finite difference scheme, we get the following finite difference equation for Eq. (4.1)

$$\begin{aligned} &-\frac{(\phi_{i+1,j}-\phi_{i,j})\epsilon_{i+\frac{1}{2},j}+(\phi_{i-1,j}-\phi_{i,j})\epsilon_{i-\frac{1}{2},j}}{\Delta x^2} \\ &-\frac{(\phi_{i,j+1}-\phi_{i,j})\epsilon_{i,j+\frac{1}{2}}+(\phi_{i,j-1}-\phi_{i,j})\epsilon_{i,j-\frac{1}{2}}}{\Delta y^2}=\rho_{i,j}, \end{aligned} \tag{4.5}$$

where

$$\epsilon_{i,j\pm\frac{1}{2}}=\frac{1}{2}(\epsilon_{i+\frac{1}{2},j\pm\frac{1}{2}}+\epsilon_{i-\frac{1}{2},j\pm\frac{1}{2}}),\ \epsilon_{i\pm\frac{1}{2},j}=\frac{1}{2}(\epsilon_{i\pm\frac{1}{2},j+\frac{1}{2}}+\epsilon_{i\pm\frac{1}{2},j-\frac{1}{2}})$$

and $\epsilon_{i\pm\frac{1}{2},j\pm\frac{1}{2}}$ are dielectric constants defined at the cell center as in Fig. 8.

The boundary condition on the outer wall (Dirichlet boundary condition)

is

$$\phi_{0,j} = \phi_{N_x,j} = \phi_{i,0} = \phi_{i,N_y} = 0, \quad \text{for } i = 0, \ldots, N_x \text{ and } j = 0, \ldots, N_y. \tag{4.6}$$

As an example, Eq. (4.5) at $(1, j)$ will be

$$\begin{aligned} &- \frac{(\phi_{2,j} - \phi_{1,j})\epsilon_{\frac{3}{2},j} - \phi_{1,j}\epsilon_{\frac{1}{2},j}}{\Delta x^2} \\ &- \frac{(\phi_{1,j+1} - \phi_{1,j})\epsilon_{1,j+\frac{1}{2}} + (\phi_{1,j-1} - \phi_{1,j})\epsilon_{1,j-\frac{1}{2}}}{\Delta y^2} = \rho_{1,j}. \end{aligned}$$

The boundary condition for the internal conductor area (Dirichlet boundary condition) is

$$\phi_{i,j} = \phi_c \tag{4.7}$$

for (i, j) on the conductor surface S. As an example, Eq. (4.5) at (i, j) where $c = (i - 1, j)$ is part of the internal conductor will be

$$\begin{aligned} &- \frac{(\phi_{i+1,j} - \phi_{i,j})\epsilon_{i+\frac{1}{2},j} - \phi_{i,j}\epsilon_{i-\frac{1}{2},j}}{\Delta x^2} \\ &- \frac{(\phi_{i,j+1} - \phi_{i,j})\epsilon_{i,j+\frac{1}{2}} + (\phi_{i,j-1} - \phi_{i,j})\epsilon_{i,j-\frac{1}{2}}}{\Delta y^2} = \rho_{i,j} + \frac{\phi_c \epsilon_{i-\frac{1}{2},j}}{\Delta x^2}. \end{aligned}$$

The boundary condition on the inner empty surface (Neumann boundary condition) is

$$\begin{aligned} \frac{\partial \phi}{\partial x} &= 0, \qquad \text{on } x_0^i \times (y_0^i, y_1^i) \\ \frac{\partial \phi}{\partial y} &= 0, \qquad \text{on } (x_0^i, x_1^i) \times y_0^i \\ -\frac{\partial \phi}{\partial x} &= 0, \qquad \text{on } x_1^i \times (y_0^i, y_1^i) \\ -\frac{\partial \phi}{\partial y} &= 0, \qquad \text{on } (x_0^i, x_1^i) \times y_1^i, \end{aligned}$$

and, by using the central difference scheme on the boundary point (as in

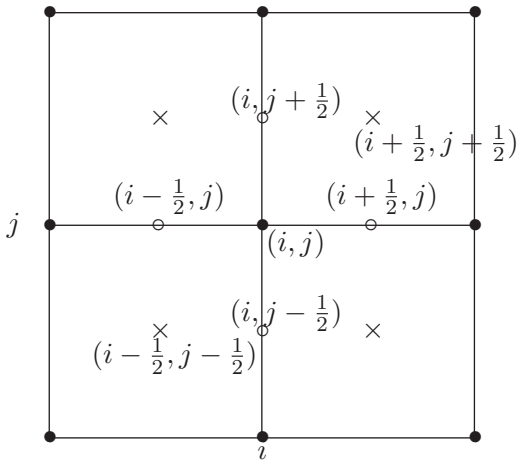


Figure 8: Notation for the finite difference equations (•: node points, ×: cell center for ϵ, ∘: edge points for ϵ).

Fig. 9 (a), we have

$$\begin{aligned}
\frac{\phi_{N_x^0+1,j} - \phi_{N_x^0-1,j}}{2\Delta x} &= 0, \quad \text{for } j = N_y^0+1, \ldots, N_y^1-1, \\
\frac{\phi_{i,N_y^0+1} - \phi_{i,N_y^0-1}}{2\Delta y} &= 0, \quad \text{for } i = N_x^0+1, \ldots, N_x^1-1, \\
-\frac{\phi_{N_x^1+1,j} - \phi_{N_x^1-1,j}}{2\Delta x} &= 0, \quad \text{for } j = N_y^0+1, \ldots, N_y^1-1, \\
-\frac{\phi_{i,N_y^1+1} - \phi_{i,N_y^1-1}}{2\Delta y} &= 0, \quad \text{for } i = N_x^0+1, \ldots, N_x^1-1,
\end{aligned} \tag{4.8}$$

i.e.,

$$\begin{aligned}
\phi_{N_x^0+1,j} &= \phi_{N_x^0-1,j}, \quad \phi_{N_x^1-1,j} = \phi_{N_x^1+1,j}, \\
\phi_{i,N_y^0+1} &= \phi_{i,N_y^0-1}, \quad \phi_{i,N_y^1-1} = \phi_{i,N_y^1+1},
\end{aligned} \tag{4.9}$$

for $j = N_y^0+1, \ldots, N_y^1-1$ and $i = N_x^0+1, \ldots, N_x^1-1$. As an example, Eq. (4.5) at (N_x, j), for $j = N_y^0+1, \ldots, N_y^1-1$, will be

$$\begin{aligned}
&- \frac{(\phi_{i-1,j} - \phi_{i,j})(\epsilon_{i+\frac{1}{2},j} + \epsilon_{i-\frac{1}{2},j})}{\Delta x^2} \\
&- \frac{(\phi_{i,j+1} - \phi_{i,j})\epsilon_{i,j+\frac{1}{2}} + (\phi_{i,j-1} - \phi_{i,j})\epsilon_{i,j-\frac{1}{2}}}{\Delta y^2} = \rho_{i,j},
\end{aligned}$$

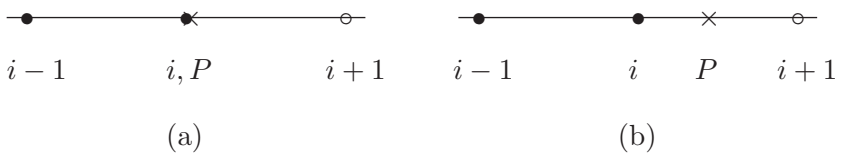


Figure 9: Notation for the finite difference scheme on the Neumann boundary condition (•: node points, ×: real boundary points, ∘: node points in the inner empty space).

Now, we consider the handling of the Neumann boundary condition when the node (usually on coarser meshes) does not lay on the real boundary, i.e., the real boundary lies between two nodes as in Fig. 9 (b).

We assign the Neumann boundary condition on the boundary point to the solution ϕ by interpolating the value of the solution. As an example, we look at the point on $x_0^i \times (y_0^i, y_1^i)$ (P as in Fig. 9 (b)).

First, we consider as the simplest function, a linear function on (x_i, x_{i+1}) which is generated by the values of the nodes x_i and x_{i+1}, i.e.,

$$\phi(x) = \frac{x_{i+1} - x}{x_{i+1} - x_i}\phi(x_i) + \frac{x - x_i}{x_{i+1} - x_i}\phi(x_{i+1}).$$

If we apply the Neumann boundary condition to the function on the point P, we have

$$\frac{\partial \phi}{\partial x} = \frac{\phi_{i+1} - \phi_i}{x_{i+1} - x_i} = 0,$$

i.e.,

$$\phi_{i+1} = \phi_i. \tag{4.10}$$

Second, we consider a quadratic function on (x_{i-1}, x_{i+1}) which is generated by the values of the nodes x_{i-1}, x_i, and x_{i+1}, i.e.,

$$\begin{aligned}\phi(x) = &\frac{1}{2h^2}(\phi_{i+1} - 2\phi_i + \phi_{i-1})x^2 \\ &- \frac{1}{2h^2}((x_i + x_{i-1})\phi_{i+1} - 2(x_{i-1} + x_{i+1})\phi_i + (x_i + x_{i+1})\phi_{i-1})x \\ &+ \frac{1}{2h^2}(x_i x_{i-1}\phi_{i+1} - 2x_{i-1}x_{i+1}\phi_i + x_i x_{i+1}\phi_{i-1}).\end{aligned}$$

If we imply the Neumann boundary condition to the function on the point P, we have,

$$\frac{\partial \phi}{\partial x} = \frac{2(x - x_i) + h}{2h^2}\phi_{i+1} - \frac{2(x - x_i)}{h^2}\phi_i + \frac{2(x - x_i) + h}{2h^2}\phi_{i-1} = 0,$$

i.e.,

$$\phi_{i+1} = \frac{4(x - x_i)}{2(x - x_i) + h}\phi_i - \frac{2(x - x_i) - h}{2(x - x_i) + h}\phi_{i-1}. \tag{4.11}$$

Here, we give an example of ϕ_{i+1} in as in Eq. (4.11).

If $x = x_i$ [same as in Fig. 9 (a)], then $\phi_{i+1} = \phi_{i-1}$.

If $x = (x_i + x_{i+1})/2 = x + h/2$ (P is the mid point of the two node points), then $\phi_{i+1} = \phi_i$ is the same as in the zeroth order case.

If $x = x_{i+1} = x + h$, then $\phi_{i+1} = \frac{4}{3}\phi_i - \frac{1}{3}\phi_{i-1}$.

If $x = x_i + \frac{h}{4}$, then $\phi_{i+1} = \frac{2}{3}\phi_i + \frac{1}{3}\phi_{i-1}$.

If $x = x_i + \frac{3}{4}h$, then $\phi_{i+1} = \frac{6}{5}\phi_i - \frac{1}{5}\phi_{i-1}$.

Remark. We could use higher order functions to implement the Neumann boundary condition. But, the order of the boundary scheme is related to the order of the approximating scheme of the partial differential equation. Usually, the order of the difference scheme for the Neumann boundary condition is less than or equal to the order of the scheme of the partial differential equation, so the second order interpolation for the Neumann boundary condition of this problem is sufficient.

4.2 The intergrid transfer operators

In this section, we describe the intergrid transfer operators in detail: the restriction and prolongation operators. The definition of the intergrid transfer operators on uniform grids is very straight forward, but the definition of the operator on boundaries with Neumann boundary condition has to be handled with care.

The linear restriction operator is defined by

$$\begin{aligned} P_{k-1}u(P_{2i,2j}) = {} & \frac{1}{4}u(P_{2i,2j}) \\ & + \frac{1}{8}\left\{u(P_{2i-1,2j}) + u(P_{2i+1,2j}) + u(P_{2i,2j-1}) + u(P_{2i,2j+1})\right\} \\ & + \frac{1}{16}\left\{u(P_{2i-1,2j-1}) + u(P_{2i+1,2j+1}) + u(P_{2i+1,2j-1}) + u(P_{2i-1,2j+1})\right\} \end{aligned}$$

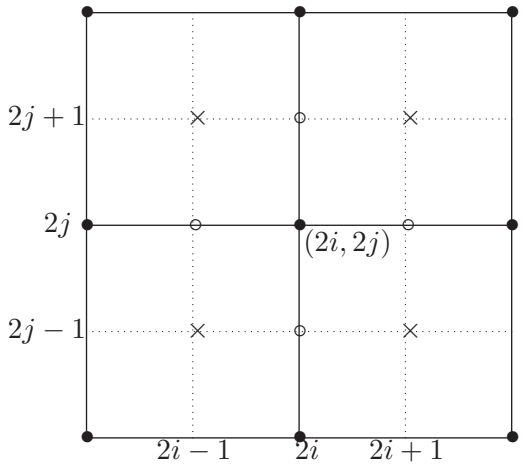


Figure 10: Notation for the coarse and fine meshes.

and the linear prolongation operator is defined by

$$\begin{aligned}
I_k u(P_{2i,2j}) &= u(P_{2i,2j}) \\
I_k u(P_{2i-1,2j}) &= \frac{1}{2}(u(P_{2i-2,2j}) + u(P_{2i,2j})) \\
I_k u(P_{2i,2j-1}) &= \frac{1}{2}(u(P_{2i,2j-2}) + u(P_{2i,2j})) \\
I_k u(P_{2i-1,2j-1}) &= \frac{1}{4}(u(P_{2i-2,2j-2}) + u(P_{2i-2,2j}) + u(P_{2i,2j-2}) + u(P_{2i,2j})).
\end{aligned}$$

These linear restriction and interpolation operators can be written by the following stencil notation

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}.$$

These operators will be defined on the boundary in accordance with the boundary condition. The values of the function on the Dirichlet boundary are all zero and the values of the function on the Neumann boundary are defined by the values of the function on the related interior points.

We need to know the function values at the nodes which are within the inner empty space and near the boundary. We consider two cases, one is the boundary at a straight piece of the inner empty space and the other is the boundary at one of the corners of the inner empty space.

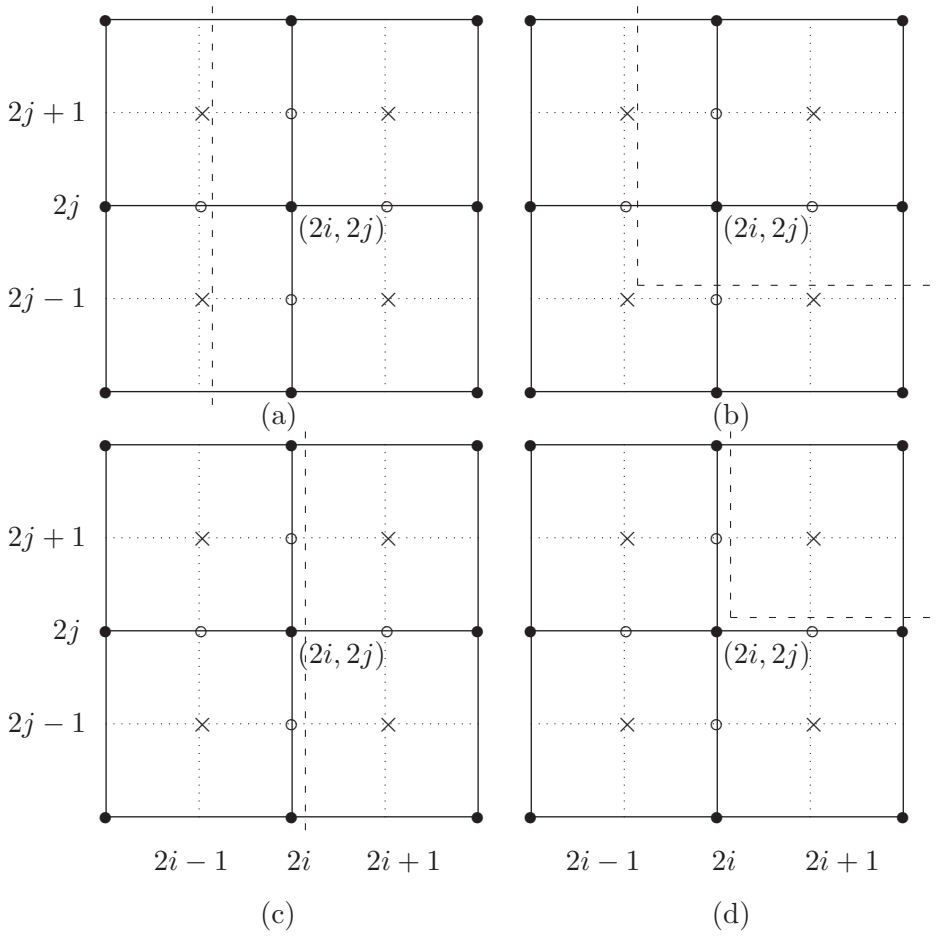


Figure 11: Notation of the Neumann boundary condition for the coarse and fine meshes (dashed line: inner boundary).

We consider the case that the boundary is located between x_{2i-1} and x_{2i} as in Fig. 11 (a). Then we need the coarse function value $u(P_{2i,2j})$ to compute the values $I_k u(P_{2i-1,2j})$, $I_k u(P_{2i-1,2j-1})$, and $I_k u(P_{2i-1,2j+1})$. In this case, we get the value $u(P_{2i,2j})$ by Eqs. (4.10) or (4.11). In a similar way, we compute $u(P_{2i+1,2j})$, $u(P_{2i+1,2j-1})$, and $u(P_{2i+1,2j+1})$ by Eqs. (4.10) or (4.11) to compute $P_{k-1}u(P_{2i,2j})$ in Fig. 11 (c).

For the case of the Neumann boundary on a corner as in Fig. 11 (b), we have two different values for $u(P_{2i,2j})$ by Eqs. (4.10) or (4.11), i.e., one $u(P_{2i,2j})^x$ is from $u(P_{2i-2,2j})$ and $u(P_{2i-4,2j})$ and the other $u(P_{2i,2j})^y$ is from $u(P_{2i,2j-2})$ and $u(P_{2i,2j-4})$. In this case, we use $u(P_{2i,2j})^x$ for $I_k u(P_{2i-1,2j})$, $u(P_{2i,2j})^y$ for $I_k u(P_{2i,2j-1})$, and $(u(P_{2i,2j})^x + u(P_{2i,2j})^y)/2$ for $I_k u(P_{2i-1,2j-1})$. In the same way, we compute the function value $u(P_{2i+1,2j+1})$ in Fig. 11 (d).

5 The parallelization concept

In this section, we consider the distribution of the domain, the data handling and the implementation on a parallel machine.

First, the discretized domain is divided into $n_x \times n_y$ subdomains as shown in Fig. 12. Each subdomain will be handled by one core. For the inner empty space, we do not assign these parts to any core if the subdomain is totally within the inner empty space. Hence, we can request less cores from the batch system which is quantized in multiples of 8 for the HPC-FF computer. As a result no more than seven cores will idle. In the future this number could be further reduced but at the moment it seems to be justified by the fact that it is only a small fraction of the total number of cores. In the case of the internal conducting structure no exception is made so that all subdomains are assigned to a core each independently of whether they include parts of the internal conducting structure or not.

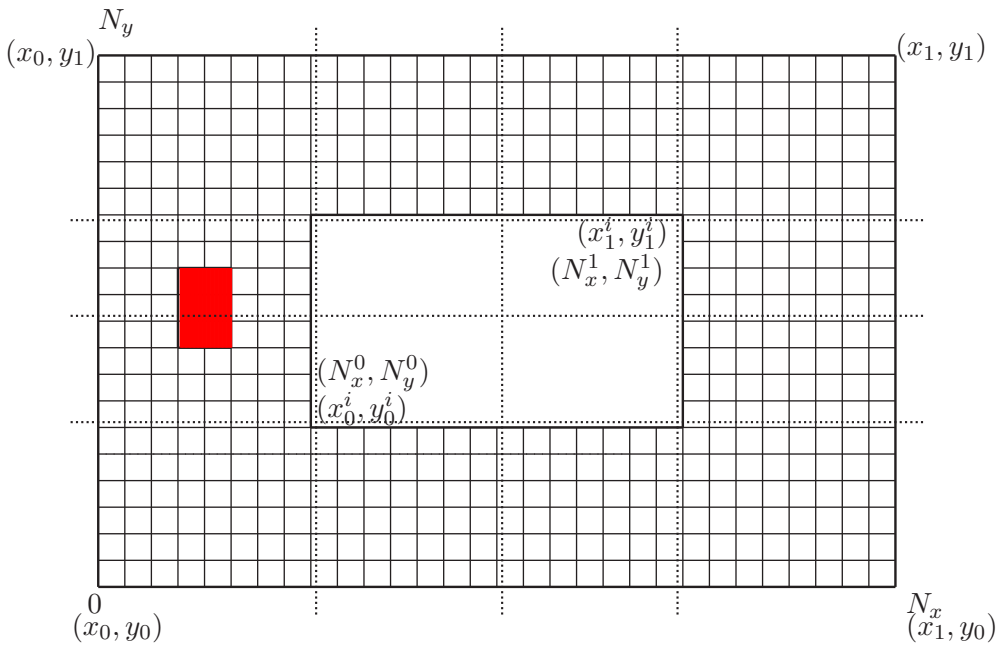


Figure 12: $n_x \times n_y$ subdomains of the rectangular domain with the internal conducting structure with $n_x = 4$ and $n_y = 4$ (■: internal structure (conductor)).

For each core, we classify real and ghost nodes which are positioned on the core as shown in Fig. 13. The real nodes are part of the subdomain which is assigned to the core. They are handled and updated by the core. Instead, the ghost nodes are originally part of exterior subdomains being located on other cores but nevertheless their values are needed for calculations done on the core. Hence, the values of the ghost nodes are first updated by the core to which they belong to as real nodes and then their values are communicated to update the ghost node values. In the same way cells are also classified as real and ghost cells.

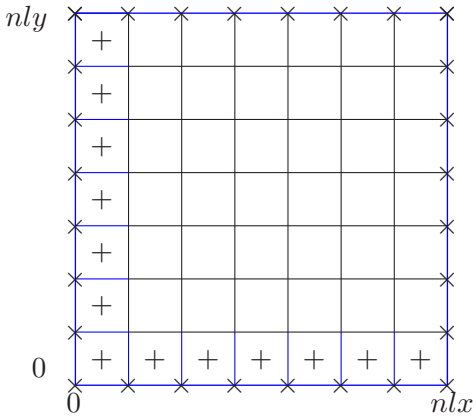


Figure 13: Typical local domain on each core (×: ghost node, +: ghost cell).

In this situation, we have to distribute the coefficient ϵ to each real cell and the right hand-side function ρ to each real node. For the geometric multigrid algorithm, we need additional routines to generate the coarser level discretizations and to compute the ϵ values on the coarser levels. Finally, this has to be tied together by data transfer routines from one core to another. The according pseudo-code takes the following form:

1. Read in the domain and discretization information.

2. Divide it into `npx`×`npy` sub-domains and assign each domain to a core. Each core has the information of one sub-domain and the rank of the neighbor cores.

   The list of sub-domain information is:

- `nlx`, `nly` : The number of intervals in each direction of the sub-blocks
- `lsx, lsy, lex, ley` : The number of ghost nodes in each direction and their beginning and ending, i.e., 0 or 1.
- `ngx[nlx+1], ngy[nly+1]` : The global number of the nodes in each direction. Needed for coarsening.
- `node[(nlx+1)*(nly+1)]` : The properties of the nodes, e.g. region within the plasma, part of the inner empty surface, part of the inner- or outer-boundary region of the plasma or part of the internal conducting structure.
- `hx, hy, px[nlx+1], py[nly+1]` : The length of the interval in each direction and the position of the node in each direction.
- `eps[nlx*nly]` : The value of ϵ for each cell.

3. Compute ϵ for each real cell and ρ for each real node on the finest level.
4. Exchange of ϵ and ρ with neighboring cores on the finest level.
5. From the sub-domain information of the finest grid, we obtain the sub-domain information of the coarser grid.
6. Compute the value of the matrix element on each grid level and impose the boundary condition according to the properties of the sub-domain.
7. Generate the intergrid transfer operators on each grid level.
8. Solve the linear problem.

To solve the discretized system, we have to implement the matrix-vector multiplication routine and a relaxation routine such as the Jacobi iteration and/or the Gauss-Seidel iteration. Finally, the data transfer routines for the function value on the ghost nodes are provided.

6 Numerical Experiments

In this section, we report the numerical results of the multigrid method as a solver and as a preconditioner of the Preconditioned GMRES (PGMRES). We consider test cases with an inner empty space but without an internal conducting structure and vice versa. Finally, we will also perform test cases for the full problem, including both the inner empty space and the internal conducting structure.

As a measure for speed of convergence, we compute the averaged residual error reduction factor per V-cycle iteration which is directly related to the required number of iterations needed to reach a given residual error. As a termination criterion for the multigrid algorithm we define a reduction of the initial residual error on the finest level by a factor of 10^{-10} (the stop criterion). The smaller the reduction factor is, the faster the convergence of the multigrid algorithm is and the less iterations we need to reach the given precision of the solution. In the following we will make an assessment of four schemes: a multigrid solver with a Jacobi smoother (MGJA), a multigrid solver with a local Gauss-Seidel smoother (MGGS), the PGMRES method with a multigrid preconditioner using the Jacobi smoother (GMJA), and the PGMRES method with a multigrid preconditioner using the local Gauss-Seidel smoother (GMGS). For all cases, we run two pre-smoothing and two post-smoothing iterations.

6.1 Validation of the discretization

We test the correct discretization of the elliptic problem on a rectangular domain with an internal conducting structure and an inner empty space by comparing the converged numerical solution with the exact solution. To construct an exact solution for a zero Neumann boundary condition on the boundary of the inner empty space, we choose the following sine function $\phi(x, y)$:

$$\phi(x, y) = \frac{1}{2\pi^2} \sin^2 \pi x \sin^2 \pi y.$$

The Dirichlet boundary condition of the internal conductor is given by the values $\phi(x, y)$ at the boundary of the conductor. To validate the system on the finest level, we compute the L^2-error of the solution with the exact solution on the domain $(0, 4) \times (0, 4)$ with an inner empty space of $(1, 3) \times (1, 3)$ and an internal conductor of $(0.46875, 0.53125) \times (1,875, 2.21500)$. The values of the L^2-errors for different grid resolution are listed in Table 1. The table contains the level l indicating a uniform discretization of the domain

with $2^l \times 2^l$ cells. Thus, the mesh size h of each direction is $4.0/2^l$. It can be clearly seen that the error converges by second order with respect to h, i.e., $O(h^2)$ when the grid resolution is increased by a factor of two.

level	L^2 error	h	error ratio
7	0.00005786	0.031250000	–
8	0.00001440	0.015625000	4.0182
9	0.00000359	0.007812500	4.0082
10	0.00000090	0.003906250	4.0039
11	0.00000022	0.001953125	3.9996
12	0.00000006	0.000976562	3.9984

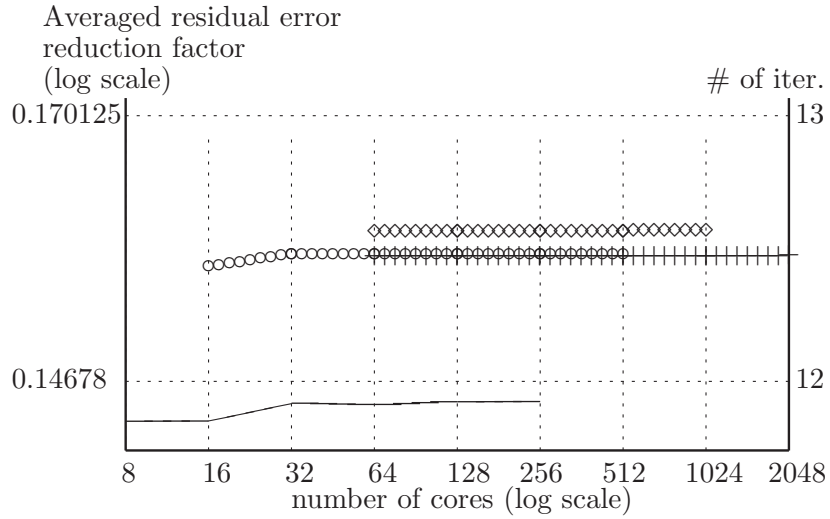
Table 1: Discretization error in the L^2-norm and the measured error ratio between succeeding refinements.

6.2 The local Gauss-Seidel iterative method

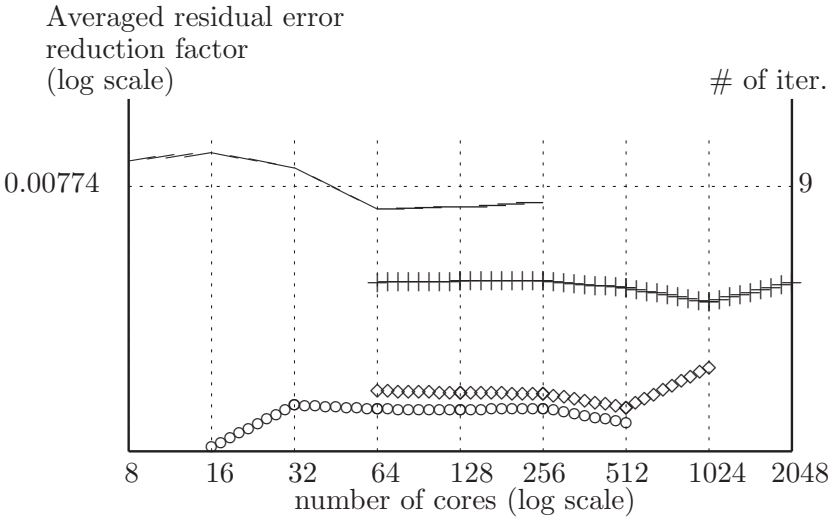
The Gauss-Seidel iteration is a preferred smoothing operator because it is simple in its implementation and has a good performance. But, the Gauss-Seidel iteration is an algorithm which involves communication when distributed over subdomains, so it becomes quite inefficient on parallel machines. A possible resolution of this problem is to perform the Gauss-Seidel iteration exclusively on each core, i.e., in one Gauss-Seidel cycle no data communication between cores takes place. Such a reduced algorithm is called the local Gauss-Seidel iteration as it applies the Gauss-Seidel iteration only locally on each core. The performance of the local Gauss-Seidel iteration typically degrades with an increasing number of cores and can reach, in the worst case, the performance of the Jacobi iteration.

In the following we investigate the convergence rate of the local Gauss-Seidel iteration both as a multigrid solver and as a multigrid preconditioner for the PGMRES method. In this context it is of special interest to see the influence of the number of cores on the convergence rate which will be measured by the averaged residual error reduction factor. We consider two different domains, one is the full square domain with an internal conducting area of $(0.484375, 0.515625) \times (1.9375, 2.0625)$ (FSQc8) and the other is the square domain with an inner empty space of $(1.0, 3.0) \times (1.0, 3.0)$ with an internal conducting area of $(0.484375, 0.515625) \times (1.9375, 2.0625)$ (Halc8).

For each domain, we consider different finest levels, e.g. the levels 12, 13, and 14 in combination with a total number of levels, as e.g. 8 and 9

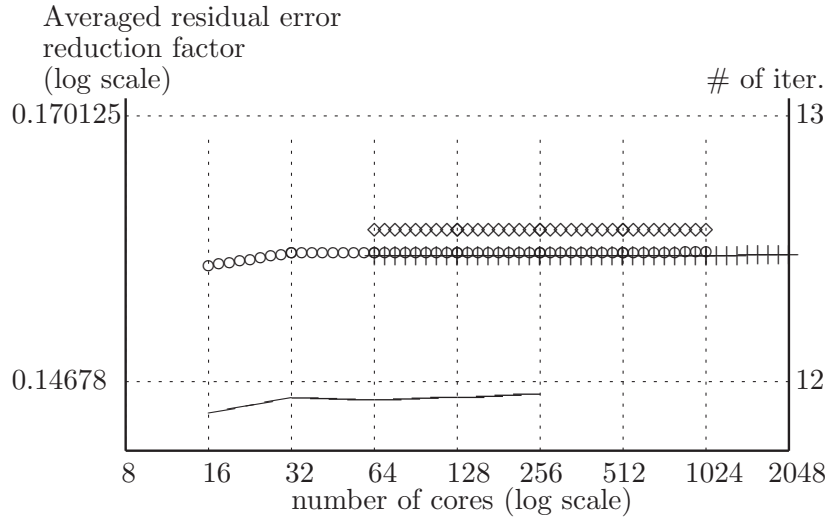


(a) The multigrid method using a local Gauss-Seidel iteration as a smoother.

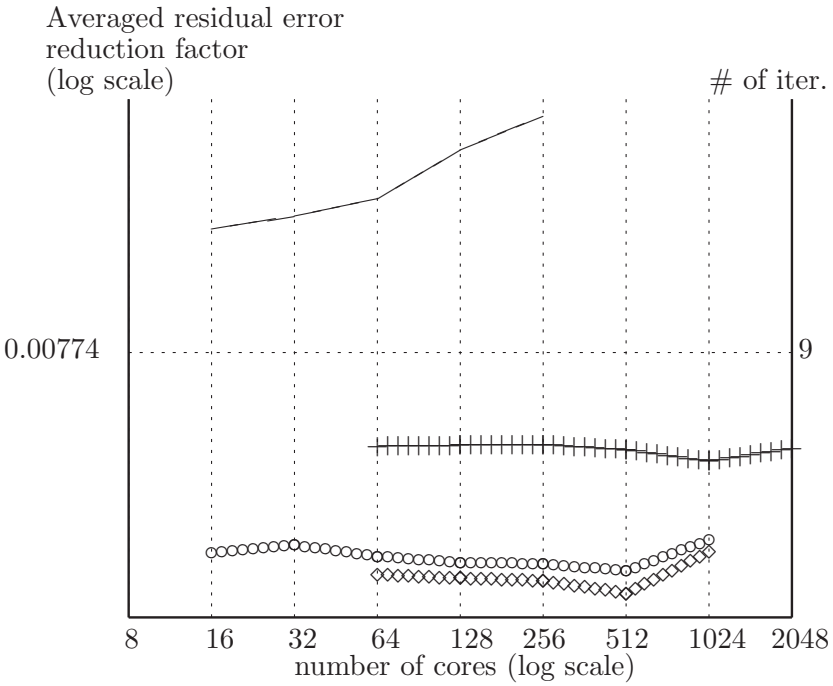


(b) The PGMRES with a multigrid preconditioner using a local Gauss-Seidel iteration as a smoother.

Figure 14: The averaged residual error reduction factor as a function of the number of cores used in the parallel run for the domain FSQc8, for symbols please see text.



(a) The multigrid method using a local Gauss-Seidel iteration as a smoother.



(b) The PGMRES with a multigrid preconditioner using a local Gauss-Seidel iteration as a smoother.

Figure 15: The averaged residual error reduction factor as a function of the number of cores used in the parallel run for the domain Halc8, for symbols please see text.

levels. The combination of finest level with the number of levels is denoted by {finest level}–{number of levels}, i.e., $12-8$ means that the finest level is 12 and the number of levels is 8. The results for the domains FSQc8 and Halc8 are shown in Fig. 14 and Fig. 15 where we distinct between the cases: $12-8$ (solid), $13-8$ ($\circ$) and $14-8$ ($+$) and $14-9$ ($\diamond$).

The graphs show that the averaged residual error reduction factor does not change much as the number of cores increases. Hence, we conclude that the efficiency of the local Gauss-Seidel iteration as a smoother in the multigrid method is not significantly reduced by the parallel distribution on many cores.

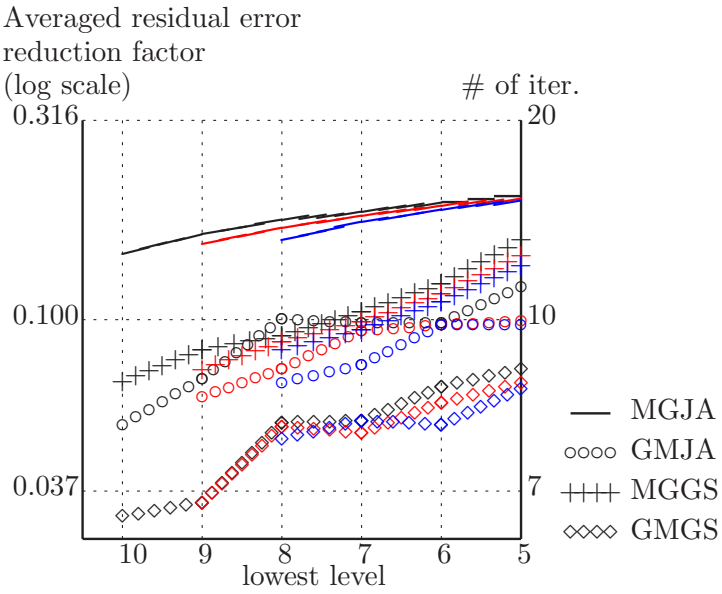
6.3 The handling of the Neumann boundary condition of the inner empty space

In the following we discretize the Neumann boundary condition for different inner empty spaces. As standard test case we define the domain Hall1, a square domain $(0.0, 4.0) \times (0.0, 4.0)$ with an inner empty space of $(1.0, 3.0) \times (1.0, 3.0)$ which is aligned with all coarser grids of the multigrid V-cycle. In addition, we define on the same square domain $(0.0, 4.0) \times (0.0, 4.0)$ different inner empty spaces which are slightly larger or smaller than the standard case;

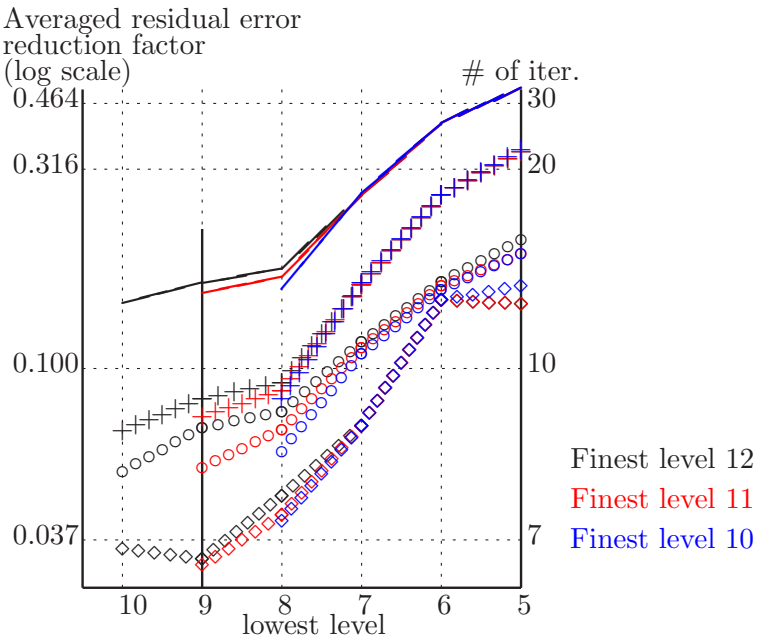
- DomainI9: $(0.9921875, 3.0078125) \times (0.9921875, 3.0078125)$
- DomainI8: $(0.984375, 3.015625) \times (0.984375, 3.015625)$
- DomainO9: $(1.0078125, 2.9921875) \times (1.0078125, 2.9921875)$
- DomainO8: $(1.015625, 2.984375) \times (1.015625, 2.984375)$

As a result the inner boundary lines of DomainI9 and DomainO9 are aligned with level 9 or higher and the inner boundary lines of DomainI8 and DomainO8 with level 8 or higher. For even smaller levels the alignment gets lost.

In Fig. 16 we plot the results for the domains Hall1 (a) and DomainI9 (b) for different finest levels 10 (blue), 11 (red), and 12 (black) and for the following schemes: MGJA (solid), MGGS (circles), GMJA (crosses), and GMGS (diamonds). For case DomainI9 the last level with alignment of the inner empty space is marked by a vertical line at level 9 in Fig. 16 (b). The results are shown as a function of the lowest level on which the problem is solved (to high precision) with the GMRES method.



(a) Case Hall1



(b) Case DomainI9

Figure 16: The averaged residual error reduction factor and the corresponding number of iterations of the multigrid method as a solver and as a preconditioner for the PGMRES method with Jacobi and Gauss-Seidel smoothers with a first order approximation for the Neumann boundary condition of the inner empty space.

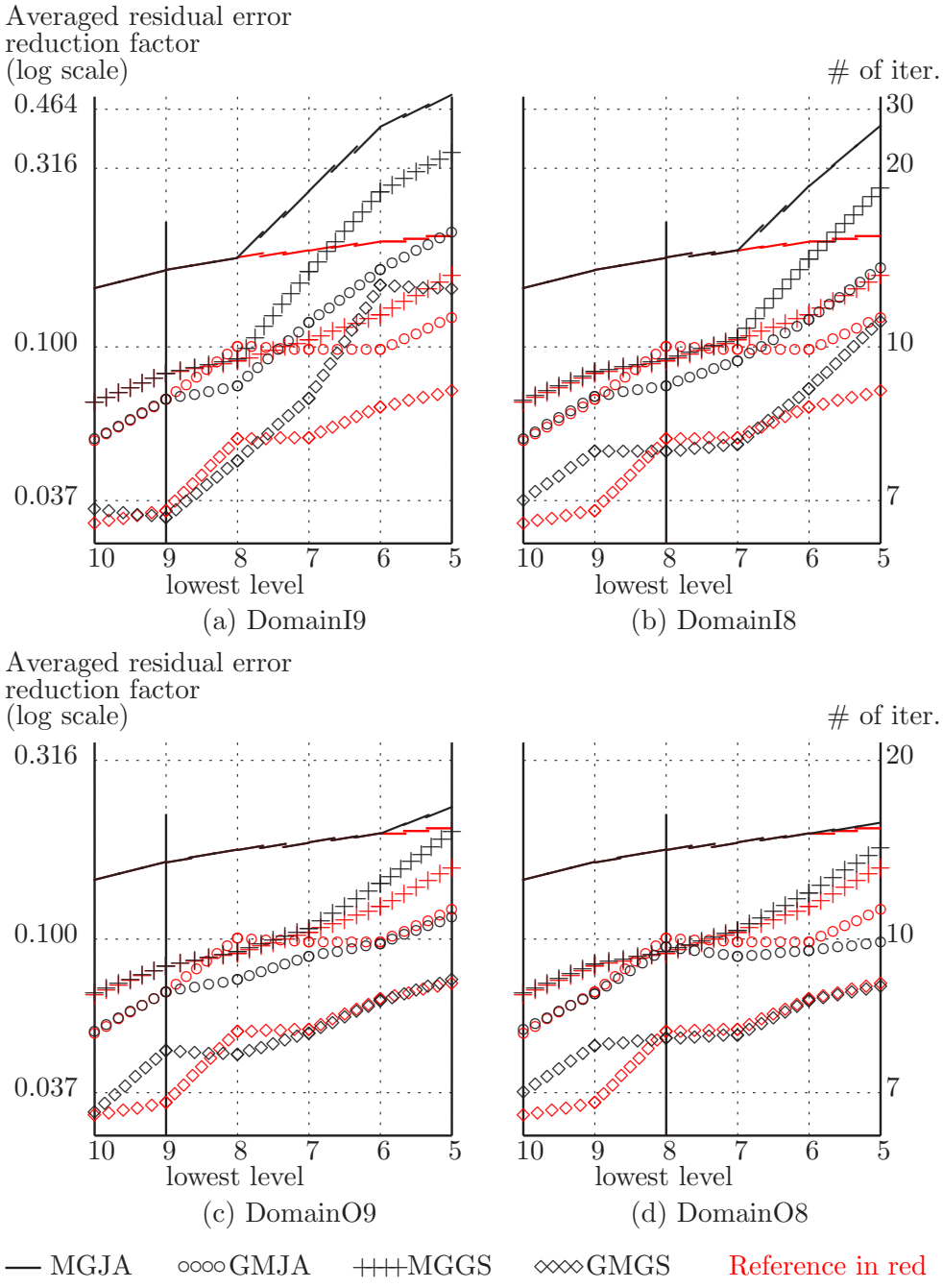


Figure 17: The averaged residual error reduction factor and the corresponding number of iterations of the multigrid method as a solver and as a preconditioner for the PGMRES method with Jacobi and Gauss-Seidel smoothers with a first order approximation for the Neumann boundary condition of the inner empty space.

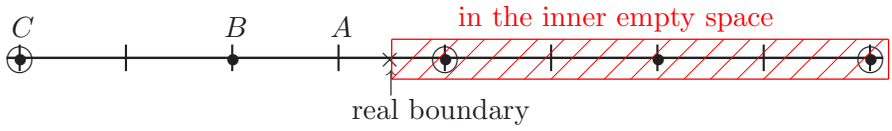
As expected the performance of the Jacobi smoother is not as good as the performance of the Gauss-Seidel smoother. Also it is clearly proven that the averaged residual error reduction factor of the PGMRES is smaller than for the multigrid solver. We also see that the averaged residual error reduction factor gets larger the larger the total number of levels in the V-cycle becomes. This is due to the fact that the problem is solved on the lowest level. So the smaller the number of levels in the multigrid V-cycle becomes the faster the multigrid method converges for the price of a more costly low level solver.

In addition, Fig. 16 shows that for both cases, Hall1 (a) and DomainI9 (b), the results only slightly depend on the highest level on which the original problem has been discretized. Hence, for the following numerical tests we will usually restrict ourselves to the case with a finest level of 12.

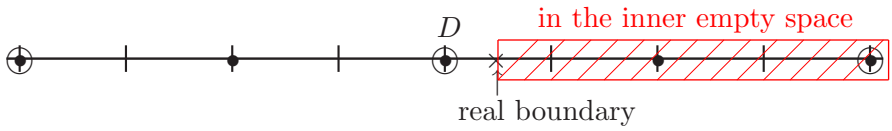
Next, we show in Fig. 17 the results of the test cases with a non-aligned inner empty space plotted in black and compare them with the results of the aligned reference case Hall1 plotted in red. The graphs in Fig. 17 show for the DomainI9 (a) and DomainI8 (b) that the averaged residual error reduction factor significantly increases compared to the aligned case when the lowest level becomes smaller by one or more levels than the last aligned level of the inner empty space. However, such an effect can not be found for the results of DomainO9 (c) and DomainO8 (d). Instead the efficiency of the averaged residual error reduction factor only slightly degrades.

To understand the different behavior of the DomainI and DomainO test cases we study the cases DomainI9 and DomainO9 in more detail. In Fig. 18 we show the relative position of the real boundary in one dimension in relation to the adjacent grid points of the eighth (vertical bar), seventh (bullet) and sixth level (open circle) for DomainI9 (a) and DomainO9 (b). For both domains the real boundary is still aligned with the grid on level 9. However, for DomainI9 the relative distance between the position of the real boundary and the nearest grid point becomes larger with each decreasing grid level (points A, B, and C). Instead, for DomainO9 the distance keeps the same (point D). It follows that the approximation of the inner boundary on a certain lower grid level is better the closer it is to the next grid point.

In the hope to further improve the efficiency of the averaged residual error reduction factor for the non-aligned cases, DomainI9 and DomainI8, we implement a second order scheme for the Neumann boundary condition as explained in Sec. 4.1. We compare the averaged residual error reduction factor of this second order scheme with the first order scheme for the case DomainI9 in Fig. 19. Again we show the results together with the results for the reference case Hall1 and take results for all four schemes un-



(a) Case DomainI9



(b) Case DomainO9

Figure 18: Location of the non-aligned inner boundary on the lower level (×: real boundary points on level 9, |: level 8 nodes, •: level 7 nodes, ◯: level 6 nodes).

der consideration. Especially for the multigrid schemes MGJA and MGGS a clear improvement can be noted. The averaged residual error reduction factor starts to degrade now when the lowest level becomes smaller than 7. In addition, the degradation for the non-aligned cases seems to be not as pronounced compared to the aligned case for decreasing lowest levels.

6.4 The effect of the internal conducting structure

In this section we consider the effect of the position of the internal conducting structure. For our test case the internal conducting structure is assumed as being small compared to the full domain and being located somewhere inside the domain. The different meshes of the V-cycle are generated from the finest mesh on which we want to solve the problem and the properties of the nodes on coarser meshes are inherited from the finest mesh. Depending on the position and the size of the internal conducting structure, the structure might not match exactly the nodes or might even disappear within the resolution of certain coarser meshes. In Fig. 20, we illustrate the matching and non-matching of the internal conducting area on a coarser grid. The visibility of the conducting area to the multigrid algorithm is given by its projection on the nodes of the grids. In Fig. 21 we mark the nodes which are within the conducting area for different grids of the V-cycle. We distinguish

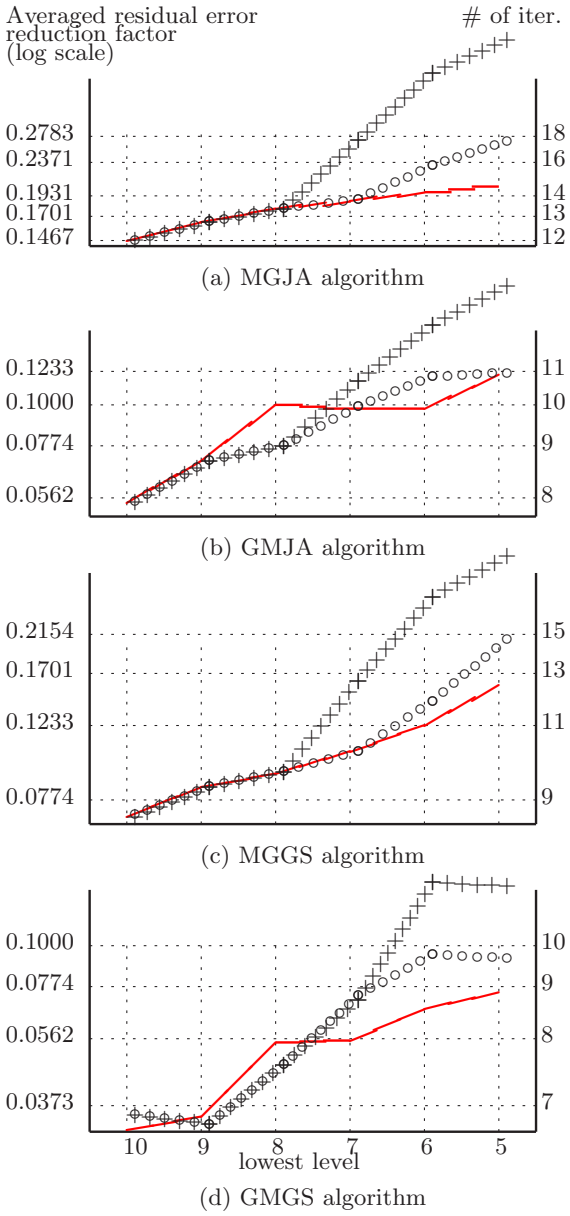
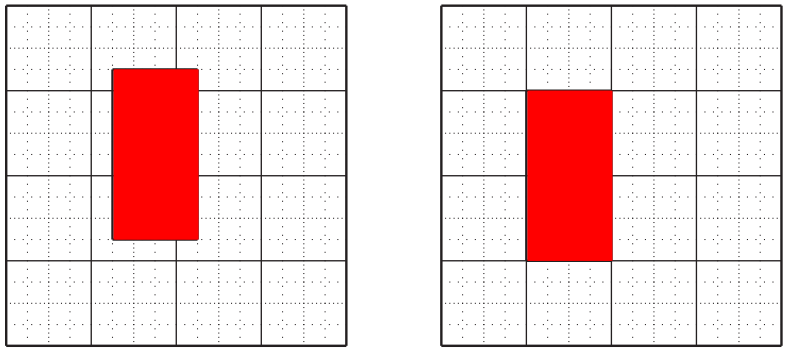


Figure 19: The averaged residual error reduction factor and the corresponding number of iterations for different iterative solvers with a first and second order approximation for the Neumann boundary condition of the inner empty space (solid line: domain Hall1, +: the first order scheme on DomainI9, ∘: the second order scheme on DomainI9).



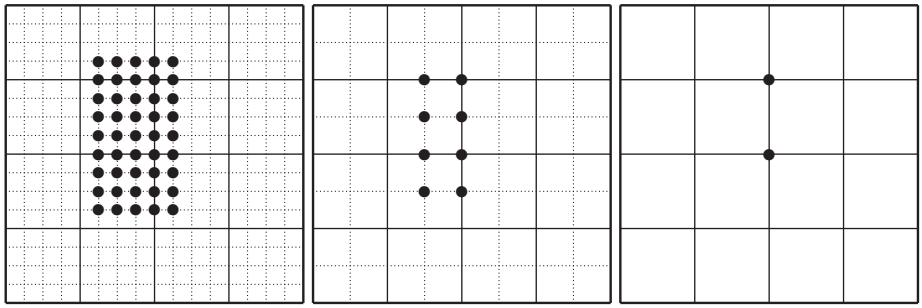
(a) Unmatched area (b) Matched area

Figure 20: The positioning of the internal conducting area (■: Internal structure (conductor), line: coarse mesh, dotted line: fine mesh).

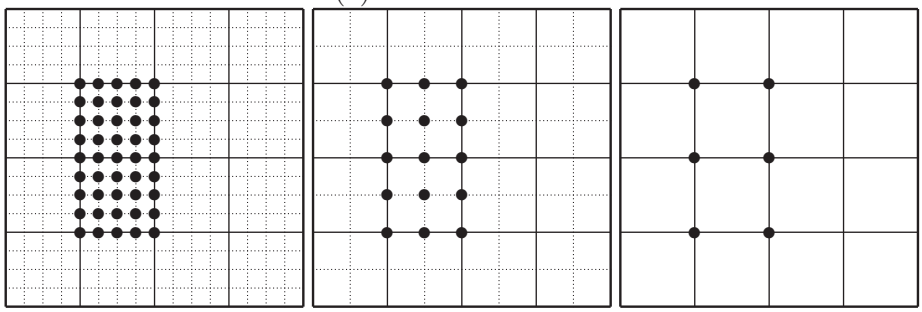
between the cases of a matching (a) and non-matching area (b).

Similarly to the tests with different inner empty spaces in Sec. 6.3 we define now different test cases to measure the performance of the multigrid algorithm for an aligned and non-aligned internal conducting area. We consider a square domain $(0.0, 4.0) \times (0.0, 4.0)$ with no inner empty space which includes the internal conducting area. The size of the internal conducting area keeps constant but its position changes to test the effect of the alignment. Our reference test case has an inner conducting area of $(0.468745, 0.531255) \times (1.874995, 2.125005)$ denoted by C0. Case C0 is centered around the coarsest node point and still aligned on level 5 in the y-direction. As the conducting area is four times more elongated in the y-direction than in the x-direction the alignment is lost already below level 7 in the x-direction. For additional test cases, we move the internal conducting structure only along the x-direction by the following values:

- C1: 0.03125, aligned on level 6
- C2: 0.0156250, aligned on level 8
- C3: 0.0078125, aligned on level 9
- C4: 0.00390625, aligned on level 10
- C5: 0.001953125, aligned on level 11



(a) Unmatched area



(b) Matched area

Figure 21: Effect of the internal conducting area on different multigrid levels (•: nodes covered by the internal conducting area).

In Fig. 22 we plot the results for the case C0 (a) and case C3 (b) for different finest levels 10 (blue), 11 (red), and 12 (black) and for the following schemes: MGJA (solid), MGGS (circles), GMJA (crosses), and GMGS (diamonds). The results are shown as a function of the lowest level on which the problem is solved (to high precision) with the GMRES method. We only show results which converge within 30 iterations to our required precision. This effects in particular the MGJA solver for case C3 in Fig. 22 (b) which has difficulties to converge within 30 iterations.

In agreement with Sec. 6.3 the results only slightly depend on the highest level on which the original problem has been discretized. Also the internal conducting area has hardly any effect on the convergence rate of the solvers as long as it is aligned to the grids of the multigrid levels. This can be clearly seen in Fig. 22 (b) where the averaged residual error reduction factor makes a jump to larger values when decreasing the lowest level from 9 to 8 and hence loosing the alignment in the x-direction. For the multigrid solvers MGJA and MGGS it further degrades significantly when extending the V-cycle to even smaller lowest levels. However, this effect is much less pronounced for the GMJA and GMGS algorithms where the multigrid method is used as a preconditioner.

To see the effect of the positioning of the internal conducting area in more detail, we scan now through the results of our test cases C0 to C5 for a full square domain without an inner empty space and a finest level of 12. Again, the averaged residual error reduction factor is plotted as a function of the lowest level. In Fig. 23 the results for a Jacobi smoother and in Fig. 24 the results for a Gauss-Seidel smoother are collected. A distinction between the results for the multigrid method as a solver (a) and as a preconditioner for the PGMRES (b) is made. For each graph, we plot a reference case denoted by NoC which lacks the internal conducting area.

From the Figs. 23 and 24, we can deduce that the multigrid method as a solver and the PGMRES method with a multigrid preconditioner have best convergence properties when the inner conducting area is absent. As already mentioned in Sec. 4 this is caused by the so-called corner singularities of the conducting area which affect the performance of the multigrid method. Next best results are achieved when the conducting area is aligned with as many grids of the different multigrid levels as possible. The deeper such an alignment is achieved within the V-cycle the better are the results. Otherwise the results degrade (significantly) when there are lower levels with no such an alignment. This becomes plausible when one looks again at Fig. 21. If there is no alignment for all levels of the V-cycle from a certain coarser level on, the internal conducting area is mapped only onto one node point, i.e.,

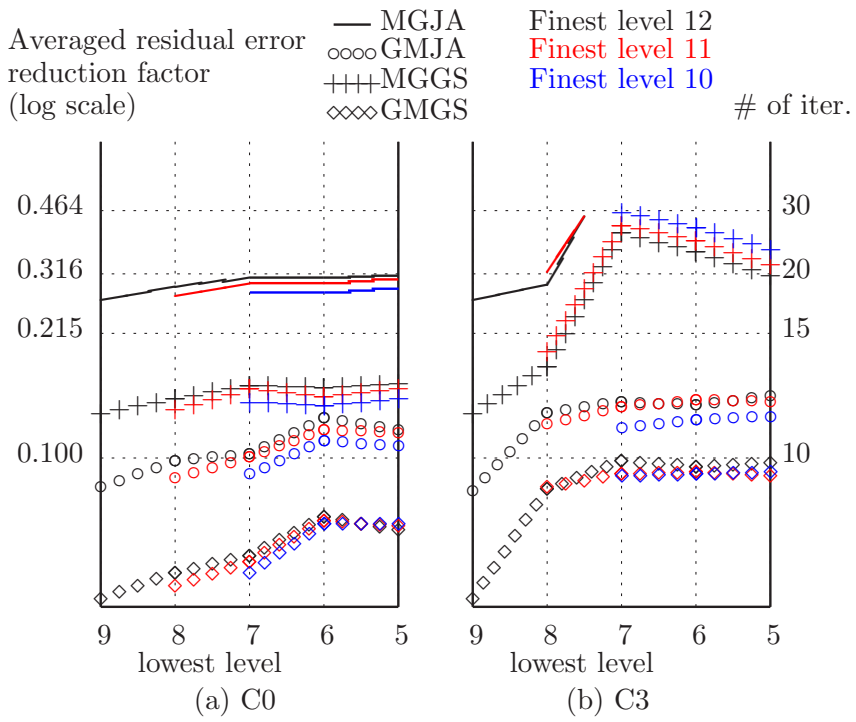
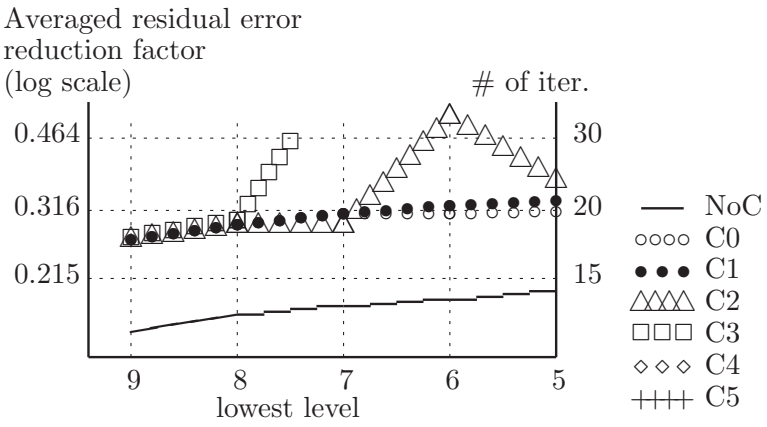
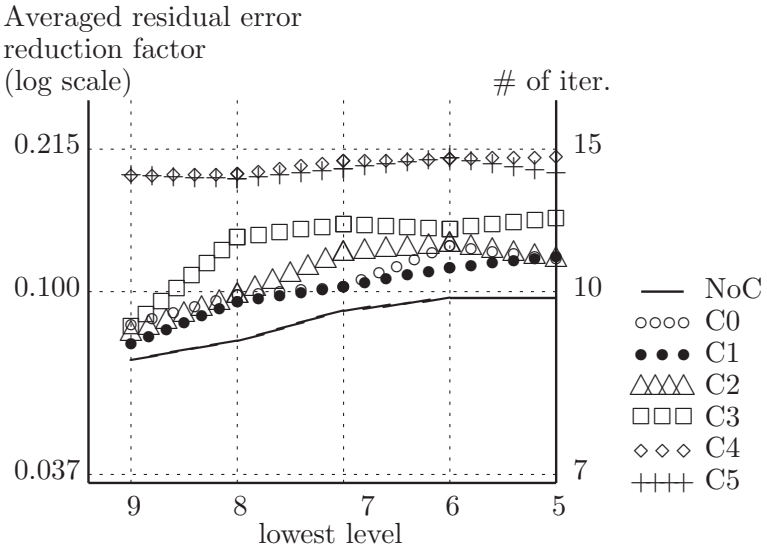


Figure 22: The averaged residual error reduction factor of four different solvers for the full square domain with the inner conducting areas C0 and C3.

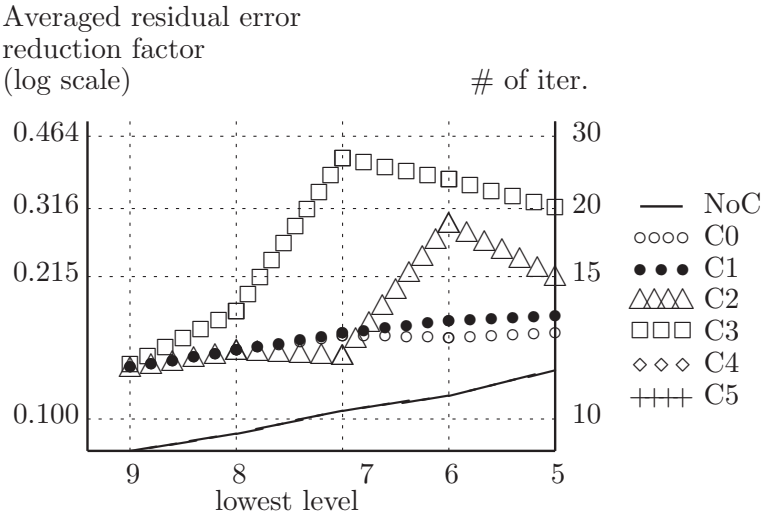


(a) Multigrid method as a solver

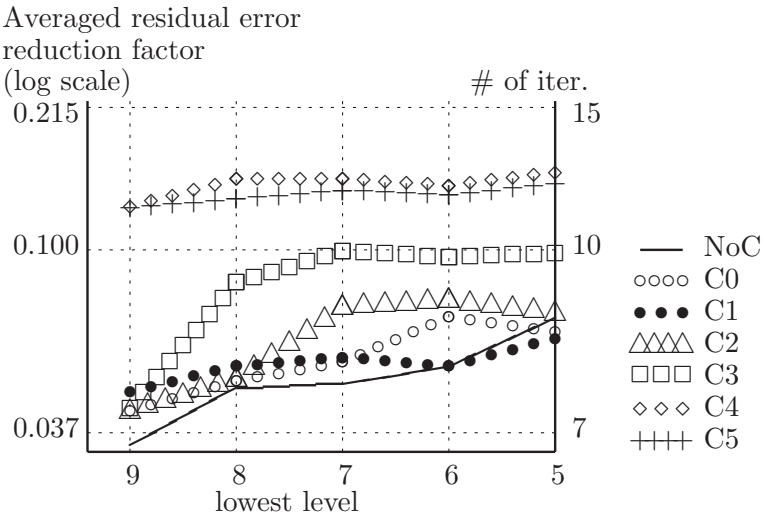


(b) PGMRES with multigrid preconditioner

Figure 23: The averaged residual error reduction factor and the corresponding number of iterations as a function of the lowest level for an internal conducting area having the slightly shifted positions of the cases C0–C5. A reference case without an internal conducting area is marked as NoC. A Jacobi smoother has been used for a multigrid method with a finest level of 12.



(a) The multigrid method as a solver



(b) PGMRES with multigrid preconditioner

Figure 24: The averaged residual error reduction factor and the corresponding number of iterations as a function of the lowest level for an internal conducting area having the slightly shifted positions of the cases C0–C5. A reference case without an internal conducting area is marked as NoC. A Gauss-Seidel smoother has been used for a multigrid method with a finest level of 12.

its structure can not be resolved any longer by the grid. This seems to be no problem as long as the structure is symmetrically positioned around such a node point. However, if the structure is not aligned with the grid the formation of its misalignment gets lost and it will not be correctly treated on the grid of the coarser levels. In the worst case this can lead to a non converging multigrid solver result. However, this effect is much more pronounced for the multigrid method as a solver. Hence, in such cases the multigrid method should only be used as a preconditioner to speed up the convergence rate of the PGMRES method.

6.5 The effect of the size of the inner empty space and the inner conducting structure

Now, we consider the full problem with an inner empty space and with an inner conducting structure.

First, we investigate the effect of the size of the inner empty space on the performance of the solvers. To do this, we select two domains, one is Ha1 which has the inner empty space $(1.0, 3.0) \times (1.0, 3.0)$ and the other is HaH which has a larger inner empty space $(0.5, 3.5) \times (0.5, 3.5)$. For the inner conducting structure, we choose the same size as in the previous section but with a different positioning for the cases Ha1 and HaH, i.e., $(0.468745, 0.531255) \times (1.874995, 2.125005)$ for the Ha1 domain and $(0.218745, 0.281255) \times (1.874995, 2.125005)$ for the HaH domain. We call this inner conducting structure case c7 because it is aligned on the seventh level in the x-direction. In Fig. 25 we compare the results of the cases Ha1 and HaH with the FSQ case where the inner empty space is absent. It can be clearly seen that the averaged residual error reduction factor is hardly influenced by the presence and size of the inner empty space as long as it is aligned with the grid.

Next, we investigate the effect of the size of the inner conducting structure which is reduced by a factor of two between the cases c7, c8 and c9. In addition to case c7 we define case c8 with $(0.4843745, 0.5156255) \times (1.937495, 2.062505)$ being aligned on the eighth level and case c9 with $(0.49218745, 0.50781255) \times (1.968745, 2.0312505)$ being aligned on the ninth level.

The results for all three inner conducting structures combined with all three domains (FSQ, Ha1, and HaH) do not show much difference. As an example we just show the averaged residual reduction factor on the Ha1 domain depending on the cases c7, c8 and c9 in Fig. 26. The difference one sees between results is a matter of two effects: the varying size of the

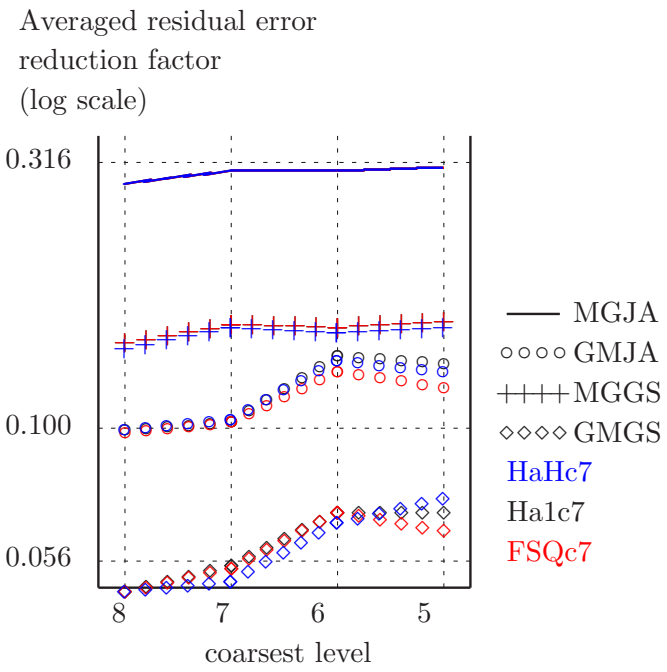


Figure 25: The averaged residual error reduction factor for different iterative solvers as a function of the lowest level for three different domains (case FSQ without and cases Ha1 and HaH with an inner empty space) with an internal conducting structure aligned at the seventh level (for details please see text). The finest grid level is 12.

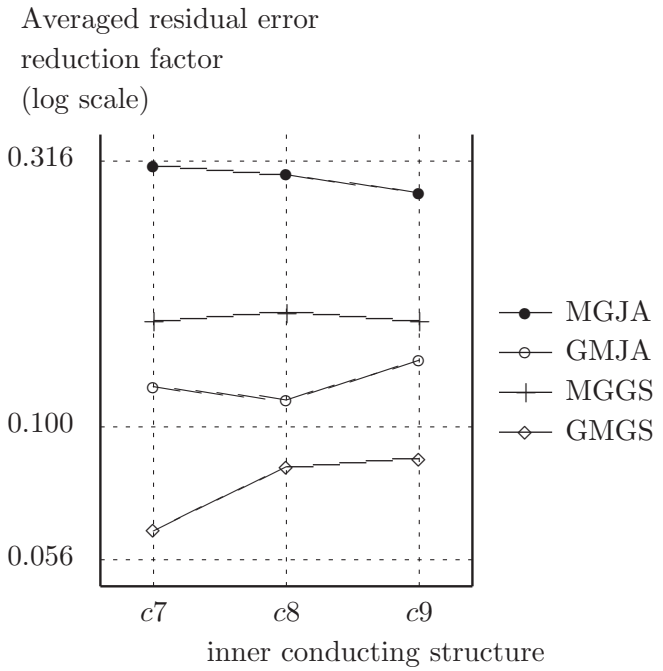


Figure 26: The averaged residual error reduction factor for different iterative solvers depending on three internal conducting structures of different size (reduced by a factor of two between the cases c7, c8 and c9) with a Ha1 domain (for details please see text). The finest grid level is level 12 and the lowest level 5.

inner conducting structure and the alignment on different levels. The results differ in a range which can be already expected from the alignment on different levels as presented in Sec. 6.4. Hence, the actual size of the inner conducting structures seems to have a very small influence on the results. This is in agreement with the results for a varying size of the inner empty space discussed before.

7 The scaling property on HPC-FF

In this section, we consider the scaling properties of our multigrid algorithm for solving our elliptic problem on a rectangular domain with an internal conducting structure and an inner empty space.

If the algorithm is executed on a parallel machine, the scaling behavior of the algorithm according to the number of cores being used is of key interest. Here we distinct between the strong and weak scaling properties. To measure the strong scaling property a fixed problem size is executed on an increasing number of cores. Instead the weak scaling property investigates the behavior of a program when the work dedicated to each core, e.g. the number of operations, is kept the same. Hence, the overall problem size scales with the number of cores.

However, for the multigrid algorithm it is nearly impossible to fix the number of operations per core while increasing the total problem size. If we add, e.g., for a 2-dim problem an additional level in the multigrid method, then the number of degrees of freedom (DoF) of the finest level becomes four times that of the previous one. In such a case, the total number of levels increases by one as long as the lowest level is kept constant. This means that the total number of operations will increase by more than a factor of four. Also incrementing the lowest level by one does not help as the new lowest level would have a four times larger number of DoF compared to the previous one. As a consequence, the number of operations needed for the solver on the lowest level increases also by more than a factor of four. In both cases, the total number of operations would be more than four times larger than for the original one. Thus, the number of operations per core would always (slightly) increase and not be fixed.

Instead we will define a semi-weak scaling by fixing the number of DoF of the finest level on each core. As a side-effect, the number of operations per core and with it the execution time will (slightly) increase with the number of cores even if there is no communication overhead.

7.1 Algorithmic scaling

To classify the efficiency of a given algorithm, as e.g. an iterative solver, it is of interest to know the number of operations needed to calculate a result with a given precision (error). Especially the behavior of the solver in relation to the number of unknowns, the so-called algorithmic scaling, is of interest. For many iterative solvers including CGM, GMRES, the Jacobi iteration, and Gauss-Seidel iteration the number of operations in each iteration is

almost proportional to the number of DoF. So the efficiency of an iterative solver can be estimated by the required number of iterations. We summarize the required storage and number of operations (flops) for some well-known solvers in Table 2.

Method	Storage	Flops	# of iterations
Gauss-Elimination (banded)	$n^{5/3}$	$n^{7/3}$	Not applicable
Gauss-Seidel Iteration	n	$n^{5/3}\log n$	$n^{2/3}\log n$
Optimal SOR	n	$n^{4/3}\log n$	$n^{1/3}\log n$
Conjugate Gradient Method	n	$n^{3.5/3}\log n$	$n^{1/6}\log n$
Full Multigrid Method	n	$n\log n$	$\log n$

Table 2: Order of required storage and flops for linear solvers.

It can be seen that the multigrid method has a very good algorithmic scaling property because the required number of iterations to achieve a given residual error increases according to the number of levels (log of the number of DoF) as already mentioned in section 3.1

To prove the predicted multigrid scaling property for our model problem, we consider the Ha1 domain with an inner conducting structure defined as in case c9 (see previous section). We choose level 5 with a mesh size of $h = 0.125$ as the lowest level. Depending on the finest level of the test problem, which varies from 10 ($h = 0.00390625$) to 13 ($h = 0.00048828125$), the number of DoF is listed in Table 3.

In Fig. 27 one can see the averaged residual error reduction factor and the corresponding number of iterations depending on the finest level of the different test cases. For all solvers under consideration the number of iterations increases only slightly as the number of levels increases. Thus, the good algorithmic scaling property of the multigrid algorithm has been confirmed for our problem.

Problem	Level 10	Level 11	Level 12	Level 13
Ha1c9	785,323	3,143,383	12,577,711	50,319,199

Table 3: The number of DoF of the different test problems.

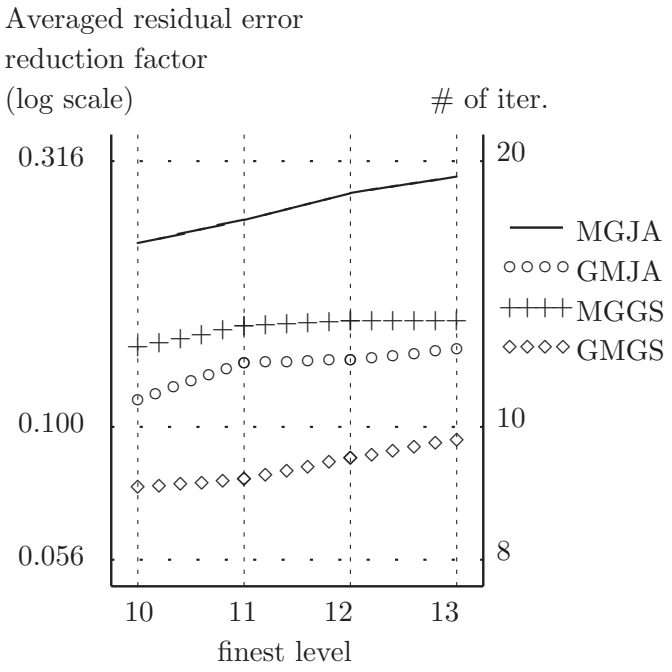


Figure 27: The averaged residual error reduction factor and the corresponding number of iterations for four different solvers depending on the finest level for case Ha1c9 with an inner empty space and an internal conducting structure. The lowest level 5 is kept constant.

7.2 Strong scaling

In this section we discuss the strong scaling property of the multigrid method on the HPC-FF computer. As test problems, we choose the following three cases, FSQ without and cases Ha1 and HaH with an inner empty space. In addition, we choose for all cases an inner conducting structure c8 which is aligned on the eighth level in the x-direction. The measured execution times are done under the assumption that the initial residual error has diminished by a factor of 10^{-9} until the iterative process is stopped. To improve statistics the same problem is solved 200 times to derive an averaged solution time. Especially, for the multigrid solvers the required number of iterations is determined just once at the first run. In subsequent runs the calculation of the residual error is omitted.

For each domain, we consider different finest levels, e.g. the levels 11,

12, and 13 in combination with a total number of levels, as e.g. 7, 8, and 9 levels. The combination of finest level with the number of levels is denoted by {finest level}–{number of levels}, i.e., $12-8$ means that the finest level is 12 and the number of levels is 8.

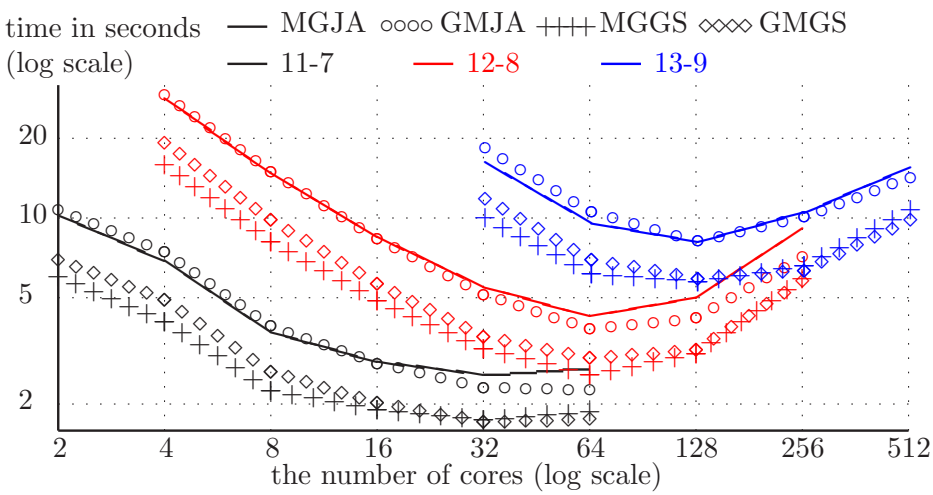
Our program is affected by two main limitations. One is related with the memory and the size of the problem, i.e., a certain number of cores is necessary to provide the needed amount of memory. Hence, the larger the highest level is, the higher will be the minimal required number of cores. Another limitation is related to the lowest level, i.e., the given number of cores has to be less than or equal to the number of DoF of the lowest level. The latter limitation could be removed with a special treatment as proposed in [19]. However, we will not follow this idea because it would result in certain performance penalties.

For the problem FSQc8, we choose the cases $11-7$, $12-8$, and $13-9$. Accordingly, all these cases have the same lowest level and therefore they all would scale to the same maximum number of cores. However, we stop displaying results when the solution time starts increasing with increasing number of cores. In addition to the solution times as a function of cores we also display the corresponding speed-up curves normalized to the solution time of the minimum number of cores in the lower part of the Figs. 28, 29, and 30. For a better comparison we plot the speed-up curves together with a solid line for the ideal strong scaling.

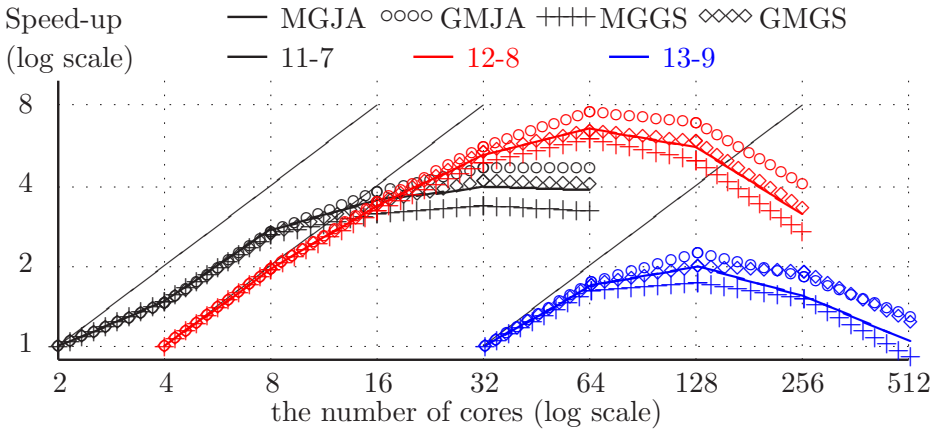
Each domain is divided into $n = n_x * n_y$ sub-domains, i.e. n_x in x-direction and n_y in y-direction. We increase the number of sub-domains n by alternate doubling of the number of cores in each direction, i.e., start with $n_x * n_y$ then $(2n_x) * n_y$, $(2n_x) * (2n_y)$, etc.

The results for the domain FSQc8 are shown in Fig. 28. As usual we distinct between the following solvers: a multigrid solver with a Jacobi smoother (MGJA) (solid line), a multigrid solver with a Gauss-Seidel smoother (MGGS) (crosses), the PGMRES method with a multigrid preconditioner using the Jacobi smoother (GMJA) (circles), or the PGMRES method with a multigrid preconditioner using the Gauss-Seidel smoother (GMGS) (diamonds). The larger the number of DoF is, the higher the multigrid solvers scale. The fastest execution times are reached at 32 cores for case $11-7$, 64 cores for case $12-8$, and 128 cores for case $13-9$. Thereby the particular solver is only of minor importance. It is obvious again that the multigrid is especially suited for large problem sizes.

The cases Ha1c8 and HaHc8 differ from case FSQc8 in the sense that they have an inner empty space. If some cores are mapped onto the inner empty space they are forced to idle. This means e.g. for case Ha1c8 that we



(a) The solution times



(b) The speed-up

Figure 28: The solution times in seconds of the multigrid method as a solver and as a preconditioner for the PGMRES method with Jacobi and Gauss-Seidel smoothers as a function of the number of cores for domain FSQc8. Results for the highest levels 11, 12, and 13 are shown for a lowest level of 5. In the lower part (b) ideal scaling is depicted by a solid line.

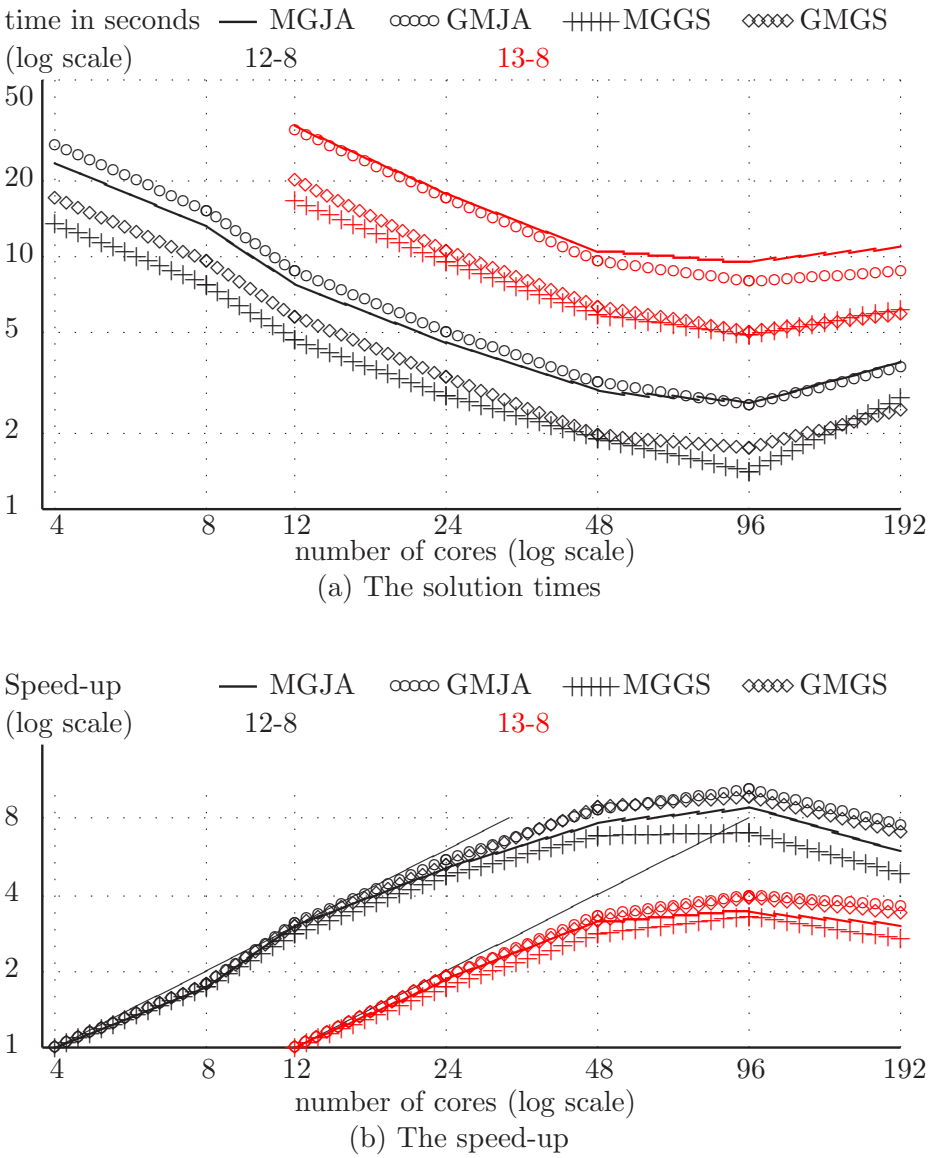


Figure 29: The solution times in seconds of the multigrid method as a solver and as a preconditioner for the PGMRES method with Jacobi and Gauss-Seidel smoothers as a function of the number of cores for domain Ha1c8. Results for the highest levels 12 and 13 are shown for a lowest level of 5. In the lower part (b) ideal scaling is depicted by a solid line.

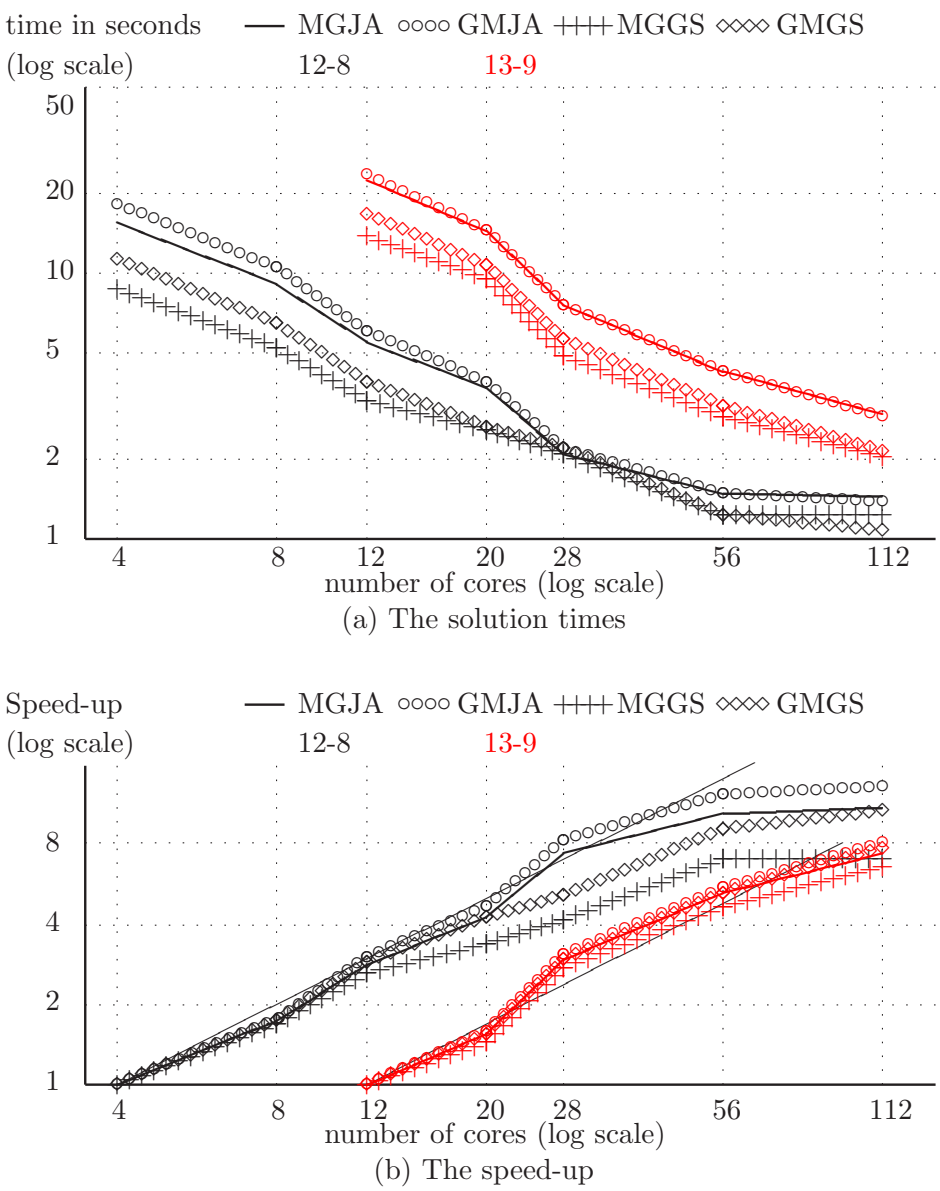


Figure 30: The solution times in seconds of the multigrid method as a solver and as a preconditioner for the PGMRES method with Jacobi and Gauss-Seidel smoothers as a function of the number of cores for domain HaHc8. Results for the highest levels 12 and 13 are shown for a lowest level of 5. In the lower part (b) ideal scaling is depicted by a solid line.

start with 4 cores then double to 8 cores but instead of doubling again to 16 cores we just use 12 cores. This is feasible as 4 cores are mapped onto the inner empty space and hence are not needed. From 12 cores on we double again the number of cores by refining the mesh either in x- or y-direction but still excluding the cores being mapped onto the inner empty space. For case HaHc8, which has a larger inner empty space relative to the size of the domain, the sequence of feasible cores is 4, 8, 12, 20, 28 and then doubling of the number of cores. Due to the limitation on the lowest level the maximum usable number of cores is 512 for case Ha1c8 and 112 for case HaHc8.

The results for the domain Ha1c8 are shown in Fig. 29. The fastest execution times are reached at 96 cores for the cases $12-8$ and $13-9$. Correspondingly, the results for the domain HaHc8 are shown in Fig. 30. Here, the fastest execution times are reached at 112 cores for the cases $12-8$ and $13-9$.

The scaling results for the domains Ha1c8 and HaHc8 are quite similar. Of course, the execution time is longer for higher number of DoF. This is correlated with an improvement of the scaling property for larger problem sizes. Hence, the 13-9 cases have the better scaling property. If we compare the scaling of the different solvers, we can see again that they scale quite similar. However, the multigrid solver with a Gauss-Seidel smoother (MGGS) has usually the fastest execution times. The existence and size of the inner empty space has an impact on the scaling of the multigrid solver. Efficient scaling to highest core numbers is achieved for the domain FSQc8 without an inner empty space followed by the domains HaHc8 and Ha1c8.

7.3 Semi-weak scaling

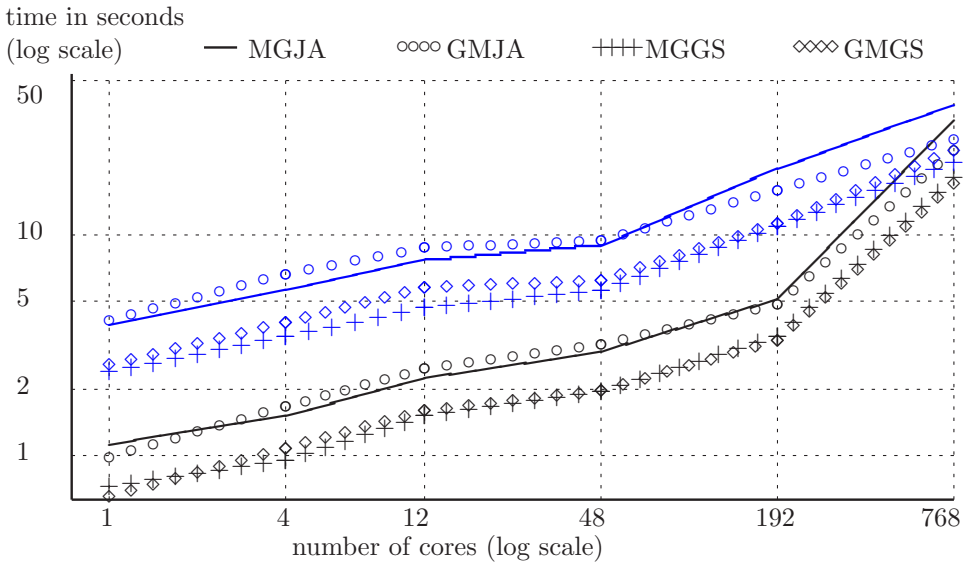
For testing the semi-weak scaling property we consider the case that the number of DoF on the finest level is fixed per core while the number of cores is increased. As mentioned before, the semi-weak scaling property is the combination of the algorithmic scaling and the weak scaling. We consider the domains Ha1c8 and HaHc8 including an inner empty space and an internal conducting structure already used for the strong scaling in the previous section.

For each problem we choose four different problem sizes per core which consist of 262 144 DoF, 523 264 DoF, 1 048 576 DoF, and 2 095 104 DoF on the finest level. To be precise, the number of DoF per core is not exactly the same as some cores share parts of the inner conducting structure and thus have to handle less number of DoF. As the inner conducting structure is relatively small this affects only a small number of cores. Hence, the above

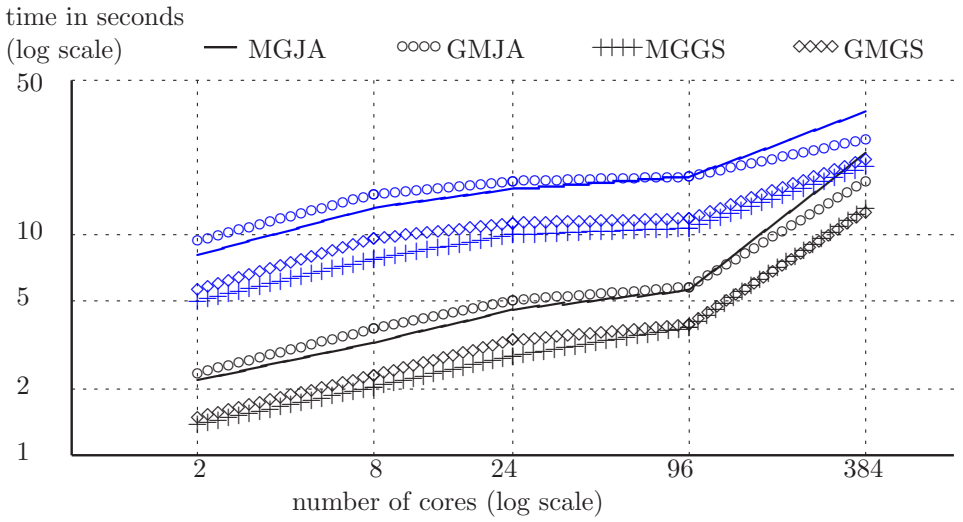
given number of DoF reflect the dominating case where the number of DoF for each core is not influenced by the inner conducting structure.

To keep the work per core as constant as possible, we invoke an additional level in the multigrid V-cycle when quadrupling the number of cores. As a result, the mesh of the finest level will increase by a factor of two in each direction so that the DoF of the coarsest mesh on each core stays constant. We report the execution times as a function of the number of cores for the domains Ha1c8 and HaHc8 in Figs. 31 and 32.

As a matter of the semi-weak scaling, the execution time is always increasing with the number of cores, in contrast to a perfect weak scaling that would imply a constant execution time. Nevertheless for very large core numbers we can see that the execution time for the solving increases significantly. Then, the communication overhead becomes obvious, which limits the scalability of the multigrid algorithm to very large numbers of cores. For Figs. 31 and 32 it is convenient to compare the cases which have been sampled at the same number of cores and which are plotted within the same graph. It shows that the "by a factor of four larger cases" always have a better semi-weak scaling property. This is plausible as as the communication costs undergo a relative decrease. For the absolute semi-weak scaling property good results are achieved up to ≈ 100 cores.

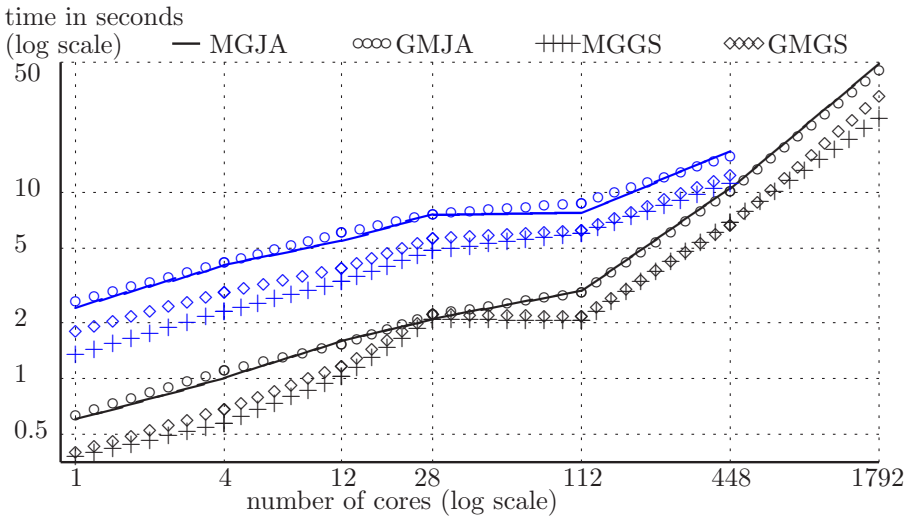


(a) 262 144 DoF (black) and 1 048 576 DoF (blue)

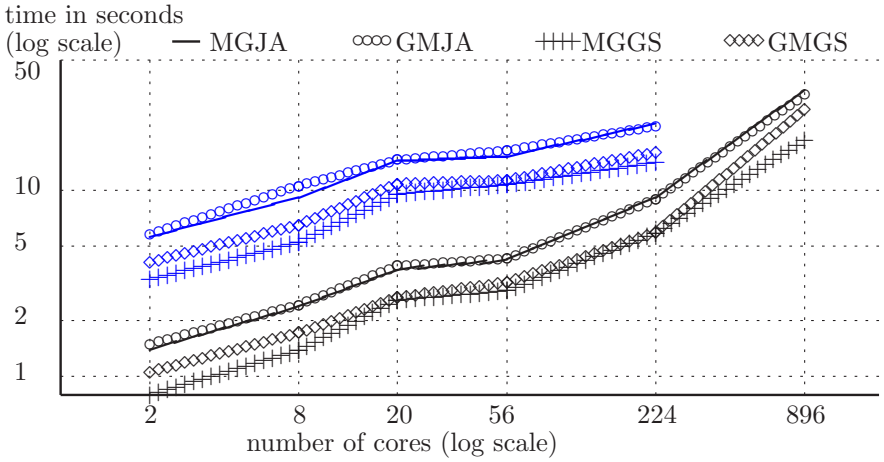


(b) 523 264 DoF (black) and 2 095 104 DoF (blue)

Figure 31: The solution times in seconds of the multigrid method as a solver and as a preconditioner for the PGMRES method with Jacobi and Gauss-Seidel smoothers as a function of the number of cores for domain Ha1c8. Four different cases with different fixed number of DoF per core (semi-weak scaling) are depicted.



(a) 262 144 DoF (black) and 1 048 576 DoF (blue)



(b) 523 264 DoF (black) and 2 095 104 DoF (blue)

Figure 32: The solution times in seconds of the multigrid method as a solver and as a preconditioner for the PGMRES method with Jacobi and Gauss-Seidel smoothers as a function of the number of cores for domain HaHc8. Four different cases with different fixed number of DoF per core (semi-weak scaling) are depicted.

8 Conclusions

For the considered model problem of an elliptic PDE on a rectangular domain we implemented and tested a multigrid solver with a Jacobi smoother, a multigrid solver with a Gauss-Seidel smoother, the Preconditioned Generalized Minimal Residual Method (PGMRES) with a multigrid preconditioner using the Jacobi smoother, and the PGMRES method with a multigrid preconditioner using the Gauss-Seidel smoother. In contrast to standard problems, the domain contains both an internal conducting structure with a Dirichlet boundary condition and an inner empty space with a Neumann boundary condition. As soon as the inner empty space becomes unaligned with the grids of the multigrid algorithm, the second order implementation of the Neumann boundary gives much better results than a first order counterpart.

As long as these internal structures are aligned with the grids of the different levels of the multigrid V-cycle there seems to be just a small negative effect of the iterative multigrid method and the property of the multigrid method as an efficient preconditioner on the convergence rate. However, if the internal structures are not aligned, the convergence rate can be significantly reduced. In the worst case the multigrid solver will not converge at all. But even then, the multigrid method can still be used as an efficient preconditioner for the GMRES iterative solver which will always converge. Instead, there is only little influence of the size of the inner empty space and inner conducting structure.

It is known that the effect of parallelization is only small on the Gauss-Seidel smoother so that the performance of the local Gauss-Seidel smoother is always superior to the Jacobi smoother. Hence, for the solvers tested here either the multigrid solver with local Gauss-Seidel smoother (MGGS) or the GMRES method with a multigrid preconditioner and a local Gauss-Seidel smoother (GMGS) give the best results.

For the multigrid method one benefits from both the fast execution time, which can be orders of magnitude faster than ordinary iterative solvers and the good hard and semi-weak scaling properties which go up to more than a hundred cores for our test cases. It was shown that the strong and semi-weak scaling properties become better, the larger the test cases are. This is plausible as the communication costs undergo a relative decrease. In addition, the iterative multigrid method has a very low memory consumption compared to direct solvers. Thus, the multigrid method is suitable for large problems on massively parallel machines like HPC-FF.

Acknowledgments

I would like to thank R. Hatzky and D. Tskhakaya for helpful discussions.

References

[1] R. Bank and T. Dupont, *An optimal order process for solving finite element equations*, Math. Comp., **36** (1981), pp. 35–51.

[2] D. Braess and R. Verfürth, *Multigrid methods for nonconforming finite elements methods*, SIAM J. Numer. Anal., **27** (1990), pp. 979–986.

[3] J. Bramble, Multigrid Methods, Pitman, London, 1993.

[4] J. Bramble, J. Pasciak, and J. Xu, *The analysis of multigrid algorithms with non-nested spaces or non-inherited quadratic forms*, Math. Comp., **56** (1991), pp. 1–34.

[5] A. Brandt, *Multigrid techniques with applications to fluid dynamics: 1984 guide*, in VKI Lecture Series, Mar. 1984, 176 pp.

[6] Z. Cai, *On the finite volume element method*, Numerische Mathematik, **58** (1991), pp. 713–735.

[7] Z. Cai, J. Mandel, and S. McCormick, *The finite volume element method for diffusion equations on general triangulations*, SIAM J. Numer. Anal., **28** (1991), pp. 392–402.

[8] P. Chatzipantelidis, *Finite Volume Methods for Elliptic PDE's: A New Approach*, Mathematical Modelling and Numerical Analysis, **36** (2002), pp. 307–324.

[9] S. H. Chou, *Analysis and convergence of a covolume method for the generalized Stokes Problem*, Math. Comp. **66** (1989), pp. 85–104.

[10] S. H. Chou and D. Y. Kwak, *Multigrid algorithms for a vertex-centered covolume method for elliptic problems*, Numer. Math. **90** (2002), pp. 441–458.

[11] S. H. Chou and X. Ye, *Unified analysis of finite volume methods for second order elliptic problems*, SIAM J. Numer. Anal., **45** (2007), pp. 1639–1653.

[12] R. E. Ewing, T. Lin, and Y. Lin, *On the accuracy of the finite volume element method based on piecewise linear polynomials*, SIAM J. Numer. Anal., **39** (2002), pp. 1865–1888.

[13] W. Hackbush, *Multigrid Methods and Applications*, Springer-Verlag, Berlin, Germany, 1985.

[14] M. T. Heath, *Scientific Computing: An Introductory Survey*, The McGraw-Hill Companies, INC., 1996.

[15] S. Kaczmarz, *Angenäherte Auflösung von Systemen linear Gleichungen*, Bulletine de l'Academie Polonaise des Sciences Letters, **35** (1937), pp. 355–357.

[16] K. S. Kang, *Convergence estimates for multigrid algorithms with general smoothing*, Ph. D. Thesis, Korea Advanced Institute of Science and Technology, 1999.

[17] K. S. Kang, *Covolume-based intergrid transfer operator in P_1 nonconforming multigrid method*, Applied Numerical Mathematics, **51** (2004), pp. 47–67.

[18] K. S. Kang, *P_1 nonconforming finite element method for radiation transport*, SIAM J. Sci. Comput., **25** (2003), pp. 369–384.

[19] K. S. Kang, Parallelization of the Multigrid Method on High Performance Computers, IPP-Report 5/123, 2010.

[20] K. S. Kang and D. E. Keyes, *Implicit symmetrized streamfunction formulations of magnetohydrodynamics*, Int. J. Numer. Meth. Fluids, **58** (2008), pp. 1201–1222.

[21] K. S. Kang and D. Y. Kwak, *Error Estimate in L^2 of a covolume method for the generalized Stokes problem*, Numer. Methods Partial Differential Eq., **22** (2006), pp. 165–179.

[22] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical Mathematics*, Texts in Applied Mathematics **37**, Springer, 2007.

[23] Y. Saad and M. H. Schultz, *GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems*, SIAM J. Sci. Stat. Comput., **7** (1986), pp. 856–869.

[24] D. Tskhakaya and R. Schneider, *Optimization of PIC codes by improved memory management*, J. Comp. Phys., **225** (2007), pp. 829–839.

[25] P. Wesseling, *An Introduction to Multigrid Methods*, Wiley, Chichester, 2003.