The parallel multigrid and domain decomposition methods for an elliptic problem on a hexagonal domain with structured triangular meshes

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

May 6, 2014

Abstract

This article is a technical report on the HLST project "Parallel Multigrid on Triangles" (MGTRI) submitted by B. Scott for developing a fast parallel solver for the gyrokinetic simulation on a massively parallel computer. We consider a second order elliptic partial differential equation on a hexagonal domain with structured triangular meshes. We implemented a parallel matrix-vector multiplication algorithm tailored to the structured triangular meshes on the hexagonal domain. The algorithm achieved very good strong and weak scaling properties. In addition, we investigated and implemented a parallel multigrid method and two non-overlapping domain decomposition methods, namely the FETI-DP and BDDC.

Contents

1 Introduction

The underlying physics code project GEMT by B. Scott is intended to generate a new code by combining two existing codes: GEMZ and GKMHD. GEMZ [18], is a simulation code for turbulent tokamak dynamics using gyrofluid equations [24], based on a conformal, logically Cartesian grid [20]. GKMHD is an MHD equilibrium solver which is intended to evolve the Grad-Shafranov MHD equilibrium with a reduced MHD description which is derived self consistently from the gyrokinetic theory.

GKMHD already uses a triangular grid, which is logically a spiral form with the first point on the magnetic axis and the last point on the X-point of the flux surface geometry. The solving method is to relax this coordinate grid onto the actual contours describing the flux surface of the equilibrium solution. Such a structure is logically the same as in a generic tokamak turbulence code. Presently GKMHD is not parallelized. Hence, a major building block of making the code parallel is to implement a matrix-vector multiplication and a parallelized solver on a triangular grid which is the topic of this report.

The matrix-vector multiplication is the key component of iterative methods such as the conjugated gradient method (CGM), the generalized minimal residual method (GMRES) [21], and the multigrid method. For the structured grid on a rectangular domain with triangular or rectangular elements, the matrix-vector multiplication is well developed and tested. In addition, there are some popular libraries available to solve partial differential equations with a finite element method on unstructured triangular meshes. In this project, we consider a new approach to handle a structured grid of a regular hexagonal domain with regular triangle elements. We show that the matrix-vector multiplication of this approach has almost perfect strong and weak scaling properties.

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 equations, and magnetohydrodynamics (MHD) [1, 2, 9, 10, 17, 19, 27]. Although the multigrid method is complex to implement, researchers in many areas think of it as an essential algorithm. Its main strength is its complexity of $N \log N$, i.e., the number of operations of the multigrid method depends on the number of degrees of freedom (DoF) times the number of levels (log of the DoF).

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 operators, separately. The intergrid transfer operators

depend on the discretization scheme and are highly related with the discretization of the matrix. The smoothing operators are implemented according to the matrix-vector multiplication. The multigrid method on a triangular mesh was studied by many researchers and reached a mature state. However, its implementation focused mostly on structured rectangular domains or unstructured grids with very complicated data structures. Instead, the problem under consideration here is based on a structured grid on a regular hexagonal domain. Hence, the data structure has to be adapted to our problem to get an optimal parallel performance. We implemented the smoothing operators and the intergrid transfer operators according to the distribution of the triangular grid information on each core which is used to implement the parallel matrix-vector multiplication.

Due to the ratio between computational and communication costs, the larger the number of DoF per core is, the better are the scaling properties of an iterative method like the multigrid method. For sufficiently large numbers of DoF per core, the scaling properties are very good [12]. Unfortunately, for very large numbers of cores, the number of DoF per core typically falls below a critical threshold and the communication cost on the coarser levels becomes more dominant than the computational cost. Especially, the exact solver on the the coarsest level has to be improved because the number of DoF per core is very small $(1 - 4)$ and the required number of iterations of a method such as CGM and GMRES increases according to the square root of the total number of DoF which is similar to the number of cores.

Thus, we have to find a special handling on the coarser level and/or look for other methods which are more efficient for small numbers of DoF per core than the multigrid method. Regarding the former, we consider to reduce the number of cores which are used to solve the problem on the coarser levels. Considering the complexity of the data communication, we can conclude that the case which uses only one core below a certain coarse level is the best choice. So, for a certain coarse level, the executing core gathers data from all cores, executes the coarse level iteration, and sends the results back to all cores. This algorithm also avoids the coarsest level limitation where there must be at least one DoF per core. Instead of having only one core solving the coarser level problem while the others are idling, we chose to replicate the same computation on each core; then we use these results for computations on the finer level without the need to send the results to all cores. Numerical experiments show that this implementation performs better, especially, on the large numbers of cores. However, it still needs more improvement to run efficiently on a massively parallel computer.

A potential candidate for the coarsest level solution is the domain decom-

position method (DDM) which is intrinsically parallel and scalable on massively parallel computers for solving partial differential equations (PDEs). The DDM can be further classified into the so-called overlapping and non-overlapping methods. On the one hand the overlapping method can be used as a preconditioner of the Krylov iterative method. On the other hand, the non-overlapping method is suited for problems which have discontinuous coefficients in the PDEs or have many distinct parts in the domain. The convergence rate of the DDM for reaching a certain error threshold shows the following behavior: for the one-level DDM it depends on the number of cores and for the two-level DDM it depends on the ratio between its coarsest and finest mesh sizes.

We implemented, analysed and developed the following two-level DDMs: the two-level Schwarz (overlapping) method [3, 25, 26], the balanced domain decomposition method with constraints (BDDC) [16], and the dual-primal finite element tearing and interconnection (FETI-DP) [7] method. While implementating the two-level Schwarz method, we were confronted with the same problem on the coarses levels as in the case of the multigrid method. Therefore, no performance improvements compared to the multigrid method were seen. Consequently we investigate only the performance of the BDDC and FETI-DP.

2 The model problem and its discretization

In this section, we explain the model problem we are interested in and how we discretize it. We consider a structured triangulation with regular triangles of a regular hexagonal domain. This triangulation is fitted for a regular hexagonal domain and may be extend to a general domain by a conformal mapping. As discretization scheme, we consider both a finite element method (FEM) and a finite volume method (FVM).

The FEM and FVM have been studied and used in many areas and have different aspects. The FEM has been well analysed on mathematical grounds with highly developed mathematical theories, but hardly keeps conservation properties which are important for some application problems. The FVM is originated to conserve quantities locally, but an error analysis of the FVM is more difficult and can be obtained by the comparison with an analysis of the FEM.

We consider a Poisson type second order elliptic partial differential equation (PDE) on a regular hexagonal domain (as shown in Fig. 1) with a Dirichlet boundary condition

$$\begin{cases} a(x,y)u - \nabla \cdot b(x,y)\nabla u &= f, \quad \text{in } \Omega, \\ u &= 0, \quad \text{on } \partial\Omega, \end{cases} \tag{2.1}$$

where $f \in L^2(\Omega)$, $b(x,y)$ is a positive function and $a(x,y)$ is a non-negative function. It is well known that (2.1) has a unique solution.

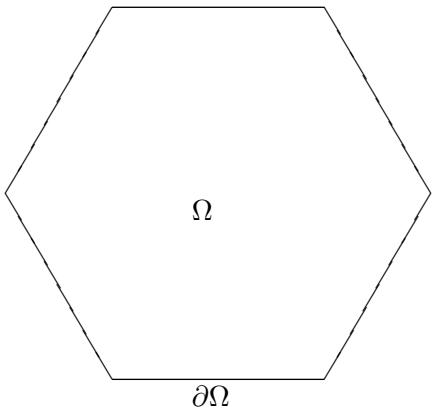


Figure 1: A regular hexagonal domain of the model problem.

2.1 Finite element method (FEM)

First, we consider the finite element formulation for (2.1).

Multiplying each side with the test function v and integrating over Ω, we obtain the following equation

$$\int_\Omega a(x,y)uv\,\mathrm{dx} - \int_\Omega (\nabla \cdot b(x,y)\nabla u)v\,\mathrm{dx} = \int_\Omega fv\,\mathrm{dx}$$

and, by using the Gauss divergence theorem with a Dirichlet boundary condition, we get the following equation

$$\int_\Omega a(x,y)uv\,\mathrm{dx} + \int_\Omega b(x,y)\nabla u \cdot \nabla v\,\mathrm{dx} = \int_\Omega fv\,\mathrm{dx}. \tag{2.2}$$

By denoting

$$a_E(u,v) = \int_\Omega a(x,y)uv\,\mathrm{dx} + \int_\Omega b(x,y)\nabla u \cdot \nabla v\,\mathrm{dx},$$

the second-order elliptic problem (2.1) can be written in the follwoing form: Find $u \in H_0^1(\Omega)$ such that

$$a_E(u,v) = \int_\Omega fv\,\mathrm{dx} \tag{2.3}$$

for any test function $v \in H_0^1(\Omega)$ where $H_0^1(\Omega)$ is the space of the first differentiable functions in Ω with zero values on the boundary $\partial\Omega$.

By choosing a finite dimensional space of $H_0^1(\Omega)$, we can classify different discretization methods including as e.g. the spectral method, the discontinuous Galerkin method, the conforming and the non-conforming FEM.

Here we consider only piecewise linear finite element spaces defined on the triangulation with regular triangles as in Fig. 2 (solid lines). Let h_1 and $\mathcal{T}_{h_1} \equiv \mathcal{T}_1$ be given, where $\mathcal{T}_1$ is a partition of Ω into regular triangles and h_1 is the maximum diameter of the elements of $\mathcal{T}_1$. For each integer $1 < k \leq J$, let $h_k = 2^{-(k-1)}h_1$ and the sequence of triangulations $\mathcal{T}_{h_k} \equiv \mathcal{T}_k$ be constructed by a nested-mesh subdivision method, i.e., let $\mathcal{T}_k$ be constructed by connecting midpoints of edges of the triangles in $\mathcal{T}_{k-1}$. Let $\mathcal{T}_h \equiv \mathcal{T}_J$ be the finest grid.

Define the piecewise linear finite element spaces

$$V_k = \{v \in C^0(\Omega) : v|_K \text{ is linear for all } K \in \mathcal{T}_k\}.$$

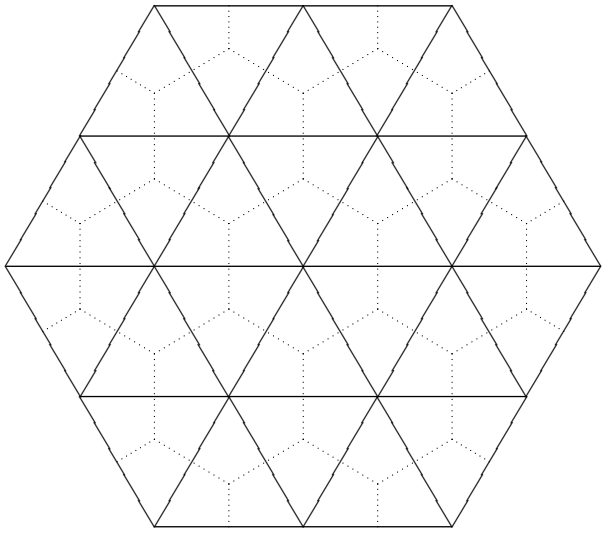


Figure 2: The primal (solid lines) and dual (dotted lines) triangulations.

Then, the finite element discretization problem can be written as follows: Find $u_J \in V_J$ such that

$$a_E(u_J, v) = \int_\Omega f v \, \mathrm{dx} \tag{2.4}$$

for any test function $v \in V_J$.

We have the following well-known error analysis result about the FEM.

Theorem 2.1 *If $u \in H^2(\Omega)$ and u_J the solution of (2.1) and (2.4). Then there exists a constant C, independent of h_J, such that*

$$\|u - u_J\|_0 + h_J \|u - u_J\|_{1,h_J} \leq C h_J^2 \|u\|_2. \tag{2.5}$$

The equation (2.4) can be written as the following linear system;

$$A_J u_J = f_J \tag{2.6}$$

where A_J is a matrix of $\dim V_J \times \dim V_J$ and u_J and f_J are vectors with dimension $\dim V_J$.

2.2 Finite volume method (FVM)

Next, we consider a finite volume formulation for (2.1). The FVM originates from a different starting point, i.e., from control volumes. According to control volumes, the FVM can be classified by overlapping [22] and non-overlapping FVM. In this report, we consider the non-overlapping control volume which is corresponding to the previous triangulations in Fig. 2 (dotted lines).

Let $\mathcal{T}_k^*$ be a finite set of control volumes as shown in Fig. 2 with dotted lines. By the integral (2.1) over each control volume K_P^*, we have the following equation

$$\int_{K_P^*} a(x,y)u \,\mathrm{dx} - \int_{K_P^*} (\nabla \cdot b(x,y)\nabla u) \,\mathrm{dx} = \int_{K_P^*} f \,\mathrm{dx}$$

and, by using the Gauss divergence theorem, we derive

$$\int_{K_P^*} a(x,y)u \,\mathrm{dx} - \int_{\partial K_P^* \setminus \partial K_P^* \cap \partial\Omega} b(x,y)\frac{\partial u}{\partial n} \,\mathrm{d}\sigma = \int_{K_P^*} f \,\mathrm{dx}. \tag{2.7}$$

By restricting the solution space to V_J, we have the following problem: Find $u_J^* \in V_J$ such that, for all $K_P^* \in \mathcal{T}_J^*$,

$$\int_{K_P^*} a(x,y)u_J \,\mathrm{dx} - \int_{\partial K_P^* \setminus \partial K_P^* \cap \partial\Omega} b(x,y)\frac{\partial u_J}{\partial n} \,\mathrm{d}\sigma = \int_{K_P^*} f \,\mathrm{dx}. \tag{2.8}$$

Define associated test function spaces Y_k, $k = 1, \ldots, J$, as the space of piecewise constant functions:

$$Y_k = \{z \in L^2(\Omega) : z|_{K_P^*} \text{ is a constant function}\}.$$

Also, define a bilinear form $a_k^*(\cdot,\cdot) : V_k \times Y_k \to \mathbb{R}$ by

$$a_k^*(u_k, v) = \int_{K_P^*} a(x,y)u_k v \,\mathrm{dx} - \int_{\partial K_P^* \setminus \partial K_P^* \cap \partial\Omega} b(x,y)\frac{\partial u_J}{\partial n} v \,\mathrm{d}\sigma$$

for $u_k \in V_k$ and $v \in Y_k$.

Then, Eq. (2.8) can be written by

$$a_J^*(u_J, v) = \int_{K_P^*} fv \,\mathrm{dx}. \tag{2.9}$$

for all $v \in Y_J$.

From results on convergence analysis in [4] and [5], we got the following error estimations for the non-overlapping FVM.

Theorem 2.2 *If $u \in H^2(\Omega)$ and $u_J^* \in V_J$ as the solution of (2.1) and (2.9). Then there exists a constant C, independent of h_J, such that*

$$\|u - u_J^*\|_{1,h_J} \leq Ch_J\|u\|_2. \tag{2.10}$$

Furthermore, assume that $f \in H^1(\Omega)$ and $a(x,y), b(x,y) \in W^{2,\infty}$. Then there exists a constant C, independent of h_J, such that

$$\|u - u_J^*\| \leq Ch_J^2(\|u\|_2 + \|f\|_1). \tag{2.11}$$

The equation (2.9) can be written as the following linear system;

$$A_J^* u_J = f_J \tag{2.12}$$

where A_J^* is a matrix of $\dim V_J \times \dim Y_J$, u_J is a vector with dimension $\dim V_J$, and f_J is a vector with dimension $\dim Y_J$. But, in most of the cases, we choose the same cardinality of V_J and Y_J, i.e., $\dim V_J = \dim Y_J$.

In the case of $a(x,y)$ and $b(x,y)$ being piecewise constant functions, the discretized system from the FEM and the FVM are only different in the source term (right-hand side of the system).

From the above, we know that A_J and A_J^* has the same dimension, so we consider the multigrid method to solve the system

$$A_J u_J = f_J, \tag{2.13}$$

where $A_J : V_J \to V_J$ is a symmetric positive system.

3 Parallelization of the structured grid

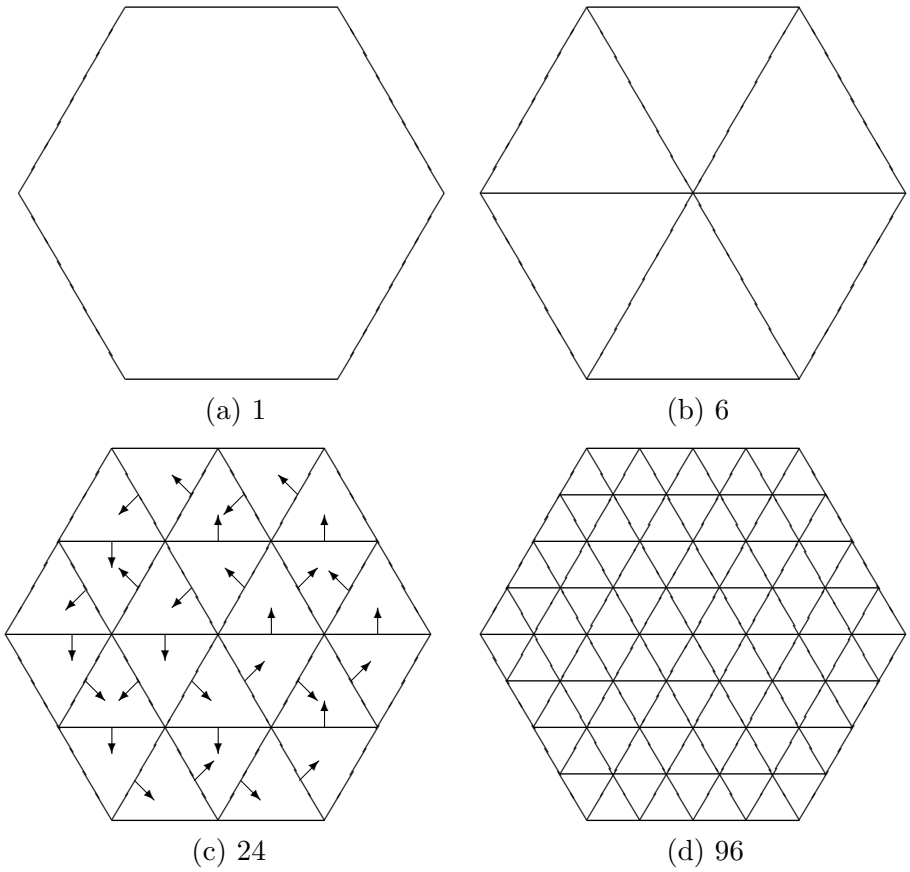
The issue of the parallelization is how to distribute the work (load balancing) and how to reduce the data communication costs to a minimum. To get good load balancing, we distribute the data in such a way that each core handles a similar amount of cells or nodes. In contrast to an unstructured grid, the information of a structured grid can be employed by the programmer. It gives the opportunity to minimize the storage requirement for the triangular grid information and to optimize the data communication on a distributed memory computer.

In our implementation, we use real and ghost nodes on each core. The values on the real nodes are handled and updated locally. The ghost nodes are the part of the distributed sub-domains located on other cores whose values are needed for the local calculations. Hence, the values of the ghost nodes are first updated by the cores to which they belong to as real nodes and then transferred to the cores that need them. To reduce the data communication during matrix elements computation, the computation of the matrix elements within the overlapping regions which belong to more than one core as real cells can be calculated concurrently by those cores.

3.1 Load balancing

We consider of how to divide the hexagonal domain into sub-domains with the same number of cores. Except for the single core case, we divide the hexagonal domain in regular triangular sub-domains and each core handles one sub-domain. Hence, the feasible numbers of cores are limited to the numbers 6×4^n for $n = 0, 1, 2, \ldots$ as e.g. can be seen in Fig. 3. For each core we have to define what are real and ghost nodes on the common boundary regions of the sub-domains. We declare the nodes on the common boundaries between two sub-domains to be real nodes of the sub-domain which is either radially outward or located in the counter clockwise direction. As shown in Fig. 3 (c). For our problem with a Dirichlet boundary condition on the outer boundary, we can handle these boundary nodes as ghost nodes. The value of there boundary ghost nodes are determined by the boundary condition and thus do not have to be transferred from other cores.

We number the sub-domains beginning at the center and going outwards following the counter clockwise direction as shown in Fig. 4. Each sub-domain can be further divided into triangles; this process is called triangulation. In this process each line segment of the sub-domain is divided into 2^n parts. It can be shown that, independently of the total number of



(a) 1 (b) 6

(c) 24 (d) 96

Figure 3: The sub-domains according to the number of cores.

sub-domains and triangulation chosen, there are just three domain types shown in Fig. 5. These give detailed information of the real and ghost nodes being connected to other sub-domains and cells which are needed to compute the matrix elements for the real nodes. We summarize the number of real nodes and ghost nodes and the total number of nodes in each sub-domain in Table 1. To see how good the load balancing is, we measure the ratio of the largest number of real nodes to the smallest number of real nodes as "ratio" which is $[2^n(2^n+3)]/[2^n(2^n+1)]$. It tends to "one" as n is increased.

The local numbering of the nodes of the three sub-domain types (a), (b) and (c) in Fig. 5 are necessary to describe the inter sub-domain communication pattern. Usually, we number real nodes first and then ghost nodes. We

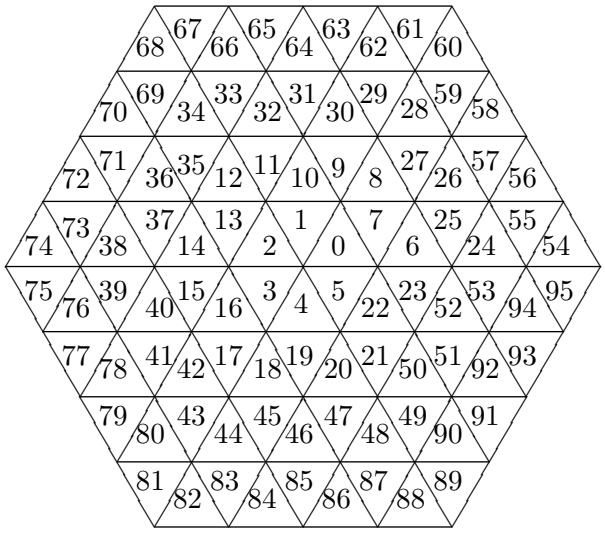
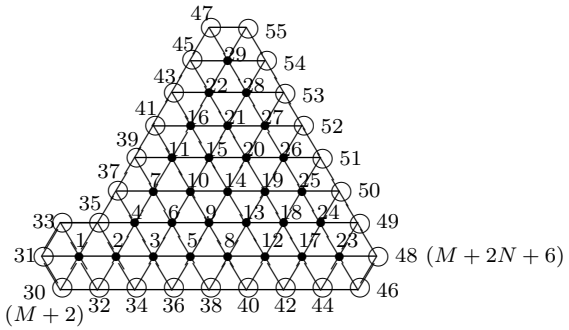


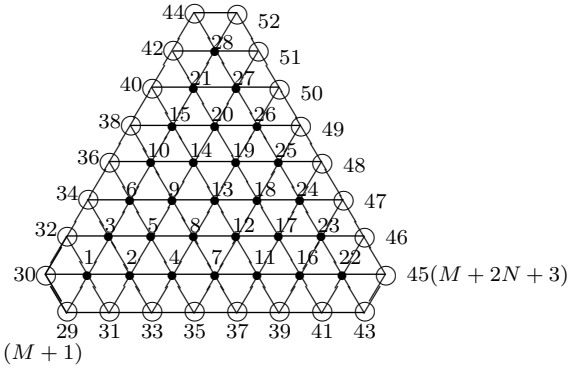
Figure 4: The numbering of the sub-domains.

level	Type I		Type II		Type III		Ratio
	Real	Ghost	Real	Ghost	Real	Ghost	
1(1)	2	8	1	6	2	8	2.0000
2(3)	7	14	6	12	9	14	1.5000
3(7)	29	26	28	24	35	26	1.2500
4(15)	121	50	120	48	134	50	1.1167
5(31)	497	98	496	96	526	98	1.0604
6(63)	2017	194	2016	192	2078	194	1.0308
7(127)	8129	386	8128	384	8254	386	1.0155
$n(N)$	$M+1$	$3N+5$	M	$3N+3$	$M+N$	$3N+5$	

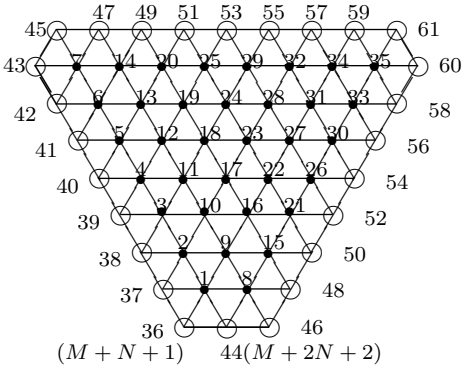
Table 1: The number of real and ghost nodes ($N = 2^n - 1$ and $M = \frac{N(N+1)}{2} = 2^{n-1}[2^n - 1]$.)



(a) Sub-domain type I: 0, 6, 9, 12, 15, 18, 21, 24, ...



(b) Sub-domain type II: 1, 2, 3, 4, 5, 11, 14, 17, 20, 23, 28, ...



(c) Sub-domain type III: 7, 10, 13, 16, 19, 22, 25, 27, ...

Figure 5: The local numbering of the node on the three sub-domain types (•: real nodes, ○: ghost nodes, $N = 2^n - 1$ and $M = \frac{N(N+1)}{2}$).

depict the local numbering and properties of the nodes for $n = 3$ in Fig. 5.

3.2 Data communication step

To get the value on the ghost nodes from the other cores for all sub-domains, we implement certain communication steps. The communication steps are the dominating part of the parallelization process and thus a key issue for the performance of the parallel code. The easiest way to implement the data communication would be that every ghost node value is received from the core which handles it as a real node value. However, such implementation would need several steps and their number would vary among different cores. This approach could be well suited for unstructured grids, but it would be too slow in our case. We solve the problem by using a sophisticated data communication routine which needs a fixed number (five) of steps for each core.

Our dedicated data communication steps are as follows:

Step1: Radial direction (Fig. 6. a)

Step2: Counter clockwise rotational direction (Fig. 6. b)

Step3: Clockwise rotational direction (Fig. 6. c)

Step4: Radial direction (same as in (1), Fig. 6. a)

Step5: Mixed communications (Fig. 6. d)

In these communication steps, the number of nodes which needs to be updated is one to four only after Step3. We illustrate the processe in Fig. 6 and show the updated values on nodes after the first three steps in Fig. 7 depending on their relative positions.

3.3 Krylov subspace methods

In this subsection, we review the Krylov subspace method which is used as a popular iterative solver whose performance can be improved by a preconditioner.

Define the **Krylov subspace** of order m by

$$K_m(A; v) = \text{span}\{v, Av, \ldots, A^{m-1}v\}. \tag{3.1}$$

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

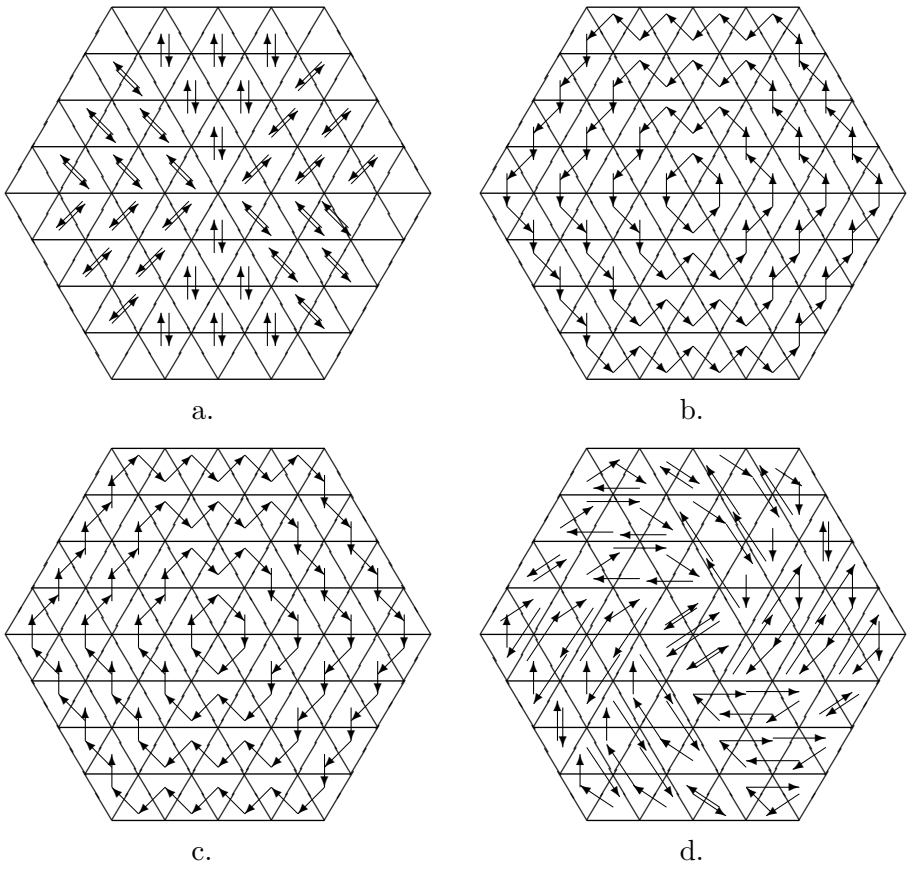
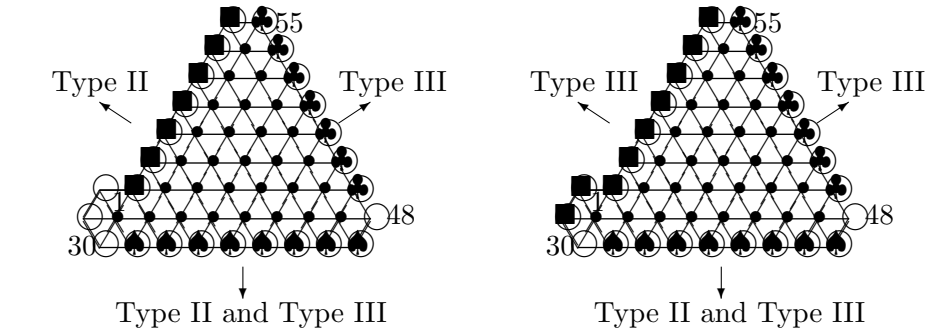


Figure 6: The data communication steps.

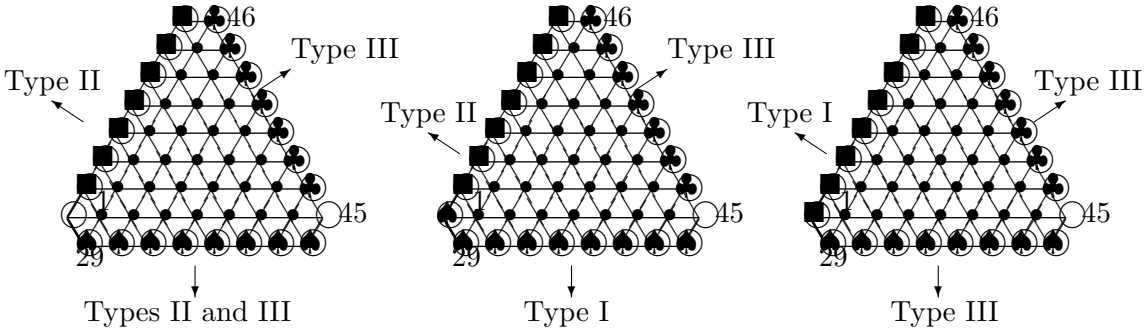
degree $\leq m-1$. Clearly, we have $K_1(A;v) \subseteq K_2(A,v) \subseteq K_3(A,v), \ldots$, and the dimension of the subspace increases at most by one for each step.

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 $p(A)v = 0$.*

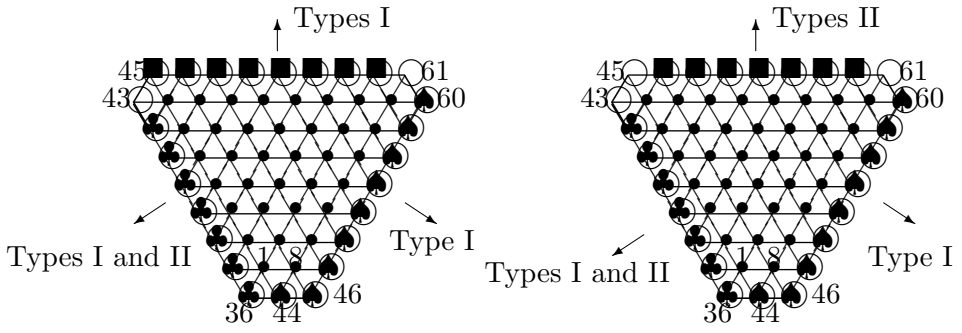
The CGM and the GMRES are well-known and well-analyzed Krylov subspace methods. The CGM works only if A is symmetric and positive definite to the inner product of the vector space. For the CGM, we have the



(a) Type I



(b) Type II



(c) Type III

Figure 7: After third communication step (Step3) (♣: first step, ♠: second step, ■: third step).

following theorem about its termination and convergence rate.

Theorem 3.2 *Let A be a $N \times N$ symmetric and positive definite matrix. The CGM to solve Eq. (2.13) terminates after at most N steps. Moreover, the error $d^{(m)}$ at the m-th iteration (with $m < N$) is*

$$\|d^{(m)}\|_A \leq 2\rho^m \|d^{(0)}\|_A, \text{ with } \rho = \frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \tag{3.2}$$

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 smallest eigenvalue λ_1 of A.

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

The performance of the Krylov subspace method can be improved by replacing the original problem with a preconditioned problem, i.e., the original problem $Ax = b$ is replaced by a preconditioned problem $\hat{A}\hat{x} = \hat{b}$ where

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

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 an efficient preconditioner, $\hat{A}$ needs to have a small condition number, i.e., M or $M_{\mathrm{L}}M_{\mathrm{R}}$ is a approximate inverses of A ($M^{-1} \approx A$). Another required property of the preconditioner M is 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 Krylov subspace method to the preconditioned problem $\hat{A}\hat{x} = \hat{b}$ instead of the original problem $Ax = b$. To use the preconditioned CGM (PCGM), we have to check the symmetricity of the preconditioned system to the inner product.

4 The multigrid method

The motivation of the multigrid method is the fact that simple iterative methods, such as Jacobi and Gauss-Seidel methods, reduce well the high-frequency error but have difficulties to reduce the low-frequency error, which can be well approximated after projection on a coarser level problem. We illustrate the basic idea of the V-cycle multigrid method in Fig. 8.

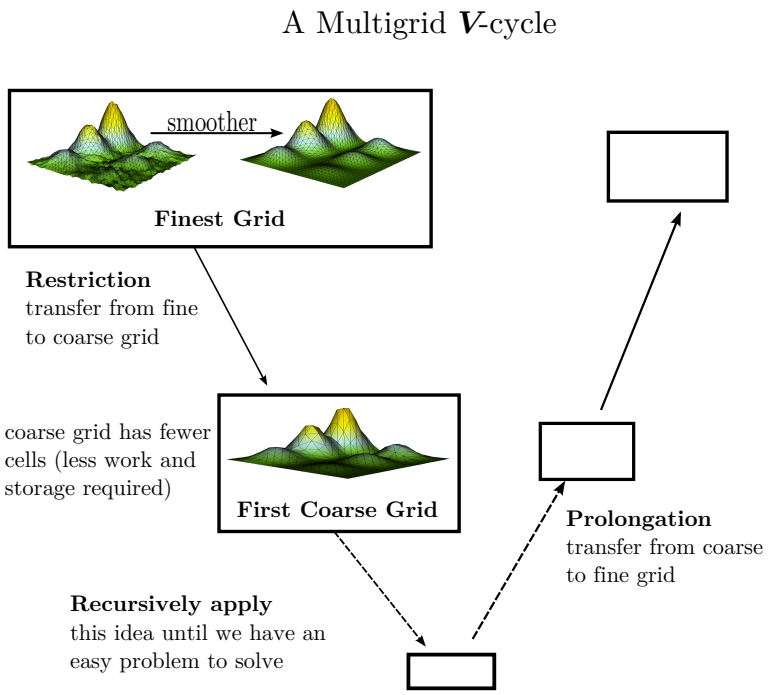


Figure 8: Basic idea of the multigrid method.

As shown in Fig. 8, the multigrid method consists of two main steps, one is the smoothing operator and the other is the intergrid transfer operator. As the smoothing operator, we can use any type of iterative method including, Richardson type (simplest one), Jacobi iteration, Gauss-Seidel iteration, Kaczmarz iteration, and incomplete LU decomposition (probably the most complex one) method. But, the smoothing operator has to be easily implemented and be able to reduce effectively the high frequency error. So the typically preferred smoothing operator is the Gauss-Seidel method.

The other important ingredient are the intergrid transfer operators, which consists of the prolongation and the restriction operator. These op-

erators depend on the geometry and functional spaces being used (e.g. the conforming or the nonconforming method), discretization schemes (e.g. the FEM, the finite difference method, the FVM, or the covolume method), and the cell-centered or the vertex-centered scheme, etc. There are natural intergrid transfer operators for the conforming and inherited functional spaces. For the nonconforming method or non-inherited functional spaces, we could define the intergrid transfer operators in several different ways, e.g., geometrical based for the FEM [11] and control volume based for the FVM.

Even if we do not have any geometric information, we can define the intergrid transfer operators according to the properties of a linear operator, i.e., the algebraic multigrid method. This method is a black-box method and requires more effort to develop efficient intergrid transfer operators.

For now, we review the analysis of the multigrid method for the linear finite element method and consider its efficient implementation on a massively parallel computer.

4.1 Basic analysis

The multigrid method for the linear finite element method with different smoothing operators is well-analysed in the literature. So, we briefly review it and mention the latest analysis theorem [1].

First, we consider the abstract framework for the trial function spaces $\{V_k\}_{k=1}^J$. Let $I_k : V_{k-1} \to V_k$ be the natural injection operator. Then we have

$$a_k(I_k w, I_k w) = a_{k-1}(w, w), \quad \forall w \in V_{k-1}. \tag{4.1}$$

Let A_k $(k = 1, \ldots, J)$ be the matrix representations of the form $a(\cdot,\cdot)_k$ on $V_k \times V_k$ with respect to a certain discrete inner product $(\cdot,\cdot)_k$. Define $P_{k-1} : V_k \to V_{k-1}$ by

$$(A_{k-1}P_{k-1}w, v)_{k-1} = (A_k w, I_k v)_k, \quad \forall v, \in V_{k-1}, w \in V_k. \tag{4.2}$$

The restriction operators $P_{k-1}^0 : U_k \to U_{k-1}$ are defined by

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

Now the discretized equation can be rewritten as

$$A_J u_J = f_J, \tag{4.3}$$

where f_J is the vector representation of f.

It is easy to see that $I_k P_{k-1}$ is a symmetric operator with respect to the $a(\cdot,\cdot)_k$ form.

Finally, let $S_k : V_k \to V_k$ for $k = 1, \ldots, J$ be the linear smoothing operators, let S_k^T denote the adjoint of R_k with respect to the $(\cdot,\cdot)_k$ inner product, and define

$$S_k^{(l)} = \begin{cases} S_k, & l \text{ odd}, \\ S_k^T, & l \text{ even}. \end{cases}$$

Following reference [1], the multigrid operator $B_k : V_k \to V_k$ is defined recursively as follows.

Multigrid Algorithm 3.1. Let $1 \le k \le J$ and p be a positive integer. Set $B_0 = A_0^{-1}$. Assume that B_{k-1} has been defined and define $B_k g$ for $g \in V_k$ by

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

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

$$x^l = x^{l-1} + S_k^{(l+m(k))}(g - A_k x^{l-1}).$$

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

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

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

$$y^l = y^{l-1} + S_k^{(l+m(k))}(g - A_k y^{l-1}).$$

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

In the multigrid algorithm 3.1, $m(k)$ gives the number of pre- and post-smoothing iterations and can vary as a function of k. If $p = 1$, we have a V-cycle multigrid algorithm. If $p = 2$, we have a W-cycle multigrid algorithm. Other versions of multigrid algorithms without pre- or post-smoothing can be analyzed similarly. For variable V-cycle multigrid algorithm the number of smoothing $m(k)$ increases exponentially as k decreases (i.e., $p = 1$ and $m(k) = 2^{J-k}$).

Theorem 4.1 *Let $\{V_k\}$, $k = 1, \ldots, J$ be the set fo the usual conforming finite element space and let S_k be the Jacobi or Gauss-Seidel method. Then there exists a $\delta < 1$ such that the following estimate holds.*

$$|a_J((I - B_J A_k)v, (I - B_J A_J)v)| \le \delta^2 a_J(v, v), \quad \forall v \in V_J. \tag{4.4}$$

Remark about the W-cycle and the variable V-cycle algorithms For simplicity, we presented the proof for the V-cycle method with one smoothing since the theoretical analysis does not give an improved rate of convergence for either $p > 1$ or $m(k) > 1$.

4.2 Reducing the number of cores on lower levels

The main issue of the parallelization of the multigrid method is the computation time spent on the coarser levels. In general, the ratio of communication to computation on a coarse level grid is larger than on a fine level grid. Because the multigrid method works on both the coarse and the fine grid levels, to get good scaling performance, we should avoid operating on the coarser levels if possible. Usually, the W-cycle and the variable V-cycle multigrid method require more work on the coarser level problems, so we consider only the parallelization of the V-cycle multigrid method.

In addition to the execution time on the coarser levels, we have to consider the solving time on the coarsest level. As a coarsest level solver, we can use either a Krylov subspace method or a direct method. The solving time of both methods increases with the problem size. So in considering the solution time of the coarsest level we need to find the optimal coarsening level, as well as the ratio of the communication to computation on each level.

As we studied in [12], we can use a small number of cores to perform computations for the coarser levels from a certain level on. We can use different numbers of cores on the coarser levels according to the problem size on that level. But such an implementation would be very complex and would have only a small benefit as there would be significant overhead due to the gathering of the data on the executing cores. So, among all possible algorithms, we consider the one which executes only on one core after having gathered all data. Such a variation of the multigrid algorithm can solve the coarsest level problem on one core only, independent on the total number of cores. We illustrate this enhanced multigrid algorithm in Fig. 9.

In a real parallel implementation, we could use `MPI_Reduce` to gather all the data to one core and then `MPI_Bcast` to send the results of the coarser level multigrid method to all cores. Instead of having only one core solving the coarser level problems whilst the other cores are idling, we choose to replicate the computation of the coarser levels on each core; subsequently we use these results for computations on the finer levels. In the current implementation, we use `MPI_Allreduce` instead of combinations of `MPI_Reduce` and `MPI_Bcast` and as it usually yields a better performance, depending on the `MPI` implementation on a given machine.

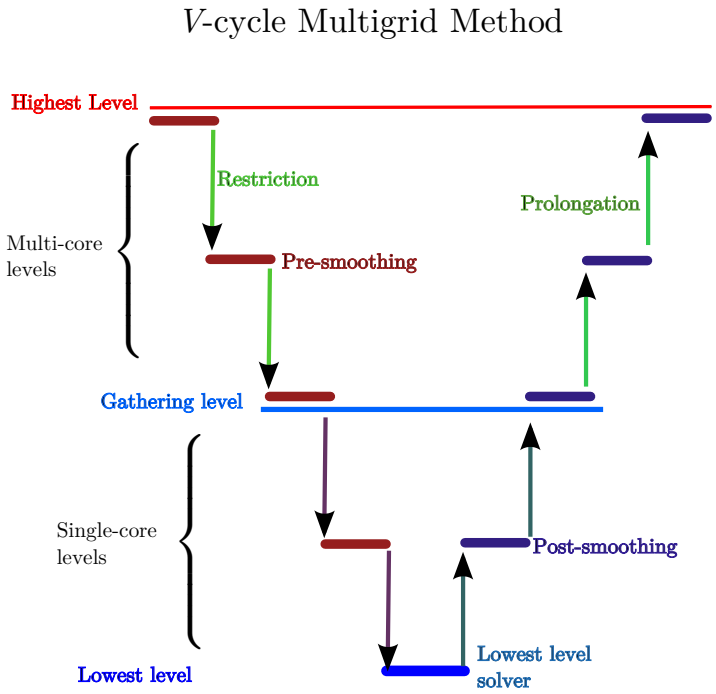


Figure 9: The V-cycle multigrid algorithm including single core levels.

In general, the balance between computation and communication costs is highly dependent on the machine architecture and problem sizes. Hence, we need to determine the level at which we need to stop coarsening according to the number of cores on each machine and problem size.

5 Overlapping domain decomposition methods

The domain decomposition methods (DDMs) have been developed to solve large problems by decomposing them into many smaller ones. As a result, these methods lead to intrinsically parallel algorithms. The required number of iterations of the two-level DDM does not depend on the number of sub-domains, therefore many computational scientists are researching on this topic.

The earliest known DDM was introduced by Schwarz in 1870 [23]. Though not originally intended as a numerical method, the classical alternating Schwarz method might be used to solve elliptic boundary value problems on a domain which is the union of two sub-domains by alternatively solving the same elliptic boundary problem restricted to the individual sub-domain. In this section we review the Schwarz methods and its convergence analysis and consider the issues of a real implementation on a massively parallel computer.

5.1 Schwarz method

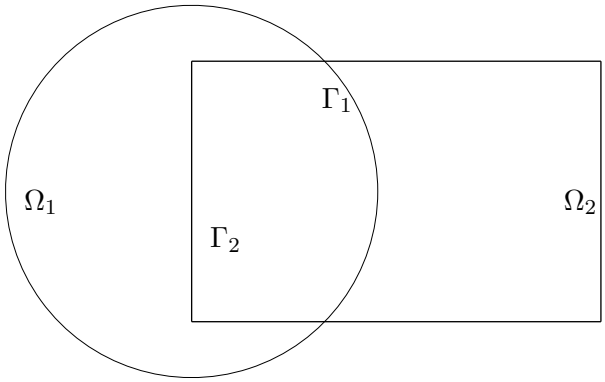


Figure 10: A typical example domain of an overlapping DDM with two sub-domains.

Suppose that Ω has the covering set $\{\Omega_i, 1 \le i \le N\}$ with

$$\overline{\Omega} = \bigcup_i \overline{\Omega}_i,$$

as shown in Figs. 10 and 11.

To solve Eq. (2.1), we consider the following local problems; Solve u_{Ω_i} with

$$\begin{cases} a(x,y)u_{\Omega_i} - \nabla \cdot b(x,y)\nabla u_{\Omega_i} &= f, \quad \text{in } \Omega_i, \\ u_{\Omega_i} &= 0, \quad \text{on } \partial\Omega_i \cap \partial\Omega, \\ u_{\Omega_i} &= u_0, \quad \text{on } \partial\Omega_i \setminus \partial\Omega, \end{cases} \tag{5.1}$$

for some u_0.

By using the same arguments and discretization schemes as in Section 2, we can write (5.1) by

$$A_{\Omega_i} u_{\Omega_i} = f|_{\Omega_i} \tag{5.2}$$

for $A_{\Omega_i} : V_{\Omega_i} \to V_{\Omega_i}$ and V_{Ω_i} is a linear finite element space defined on Ω_i. From the linear functional property, we know that there are projection (interpolation) operators

$$R_{\Omega_i} : V_J \to V_{\Omega_i}.$$

First, we consider an alternating (multiplicative) Schwarz method which solves (5.1). For a given initial solution u^0, we get $u^{n+1}_{\Omega_i}$ by solving the following problems

$$\begin{cases} a(x,y)u^{n+1}_{\Omega_i} - \nabla \cdot b(x,y)\nabla u^{n+1}_{\Omega_i} &= f, \quad \text{in } \Omega_i, \\ u^{n+1}_{\Omega_i} &= 0, \quad \text{on } \partial\Omega_i \cap \partial\Omega, \\ u^{n+1}_{\Omega_i} &= \begin{cases} u^{n+1}_{\Omega_j}, & \text{on } \partial\Omega_i \cap \Omega_j \text{ and } j < i, \\ u^{n}_{\Omega_j}, & \text{on } \partial\Omega_i \cap \Omega_j \text{ and } j > i, \end{cases} \end{cases} \tag{5.3}$$

for $i = 1, \ldots, N$.

We can write the alternating Schwarz method in the following way;

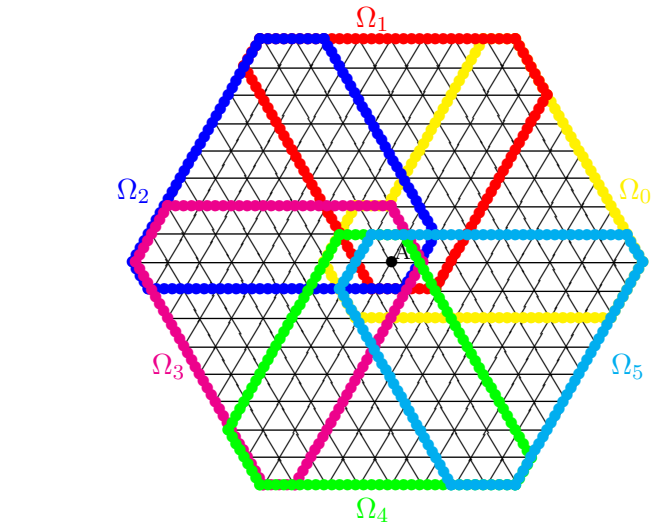
$$u^{n+1} - u^0 = (I - P_{\Omega_N}) \cdots (I - P_{\Omega_1})(u^n - u^0)$$

where $P_{\Omega_i} = R_{\Omega_i} A^{-1}_{\Omega_i} R^T_{\Omega_i} A_J$ for $i = 1, \ldots, N$.

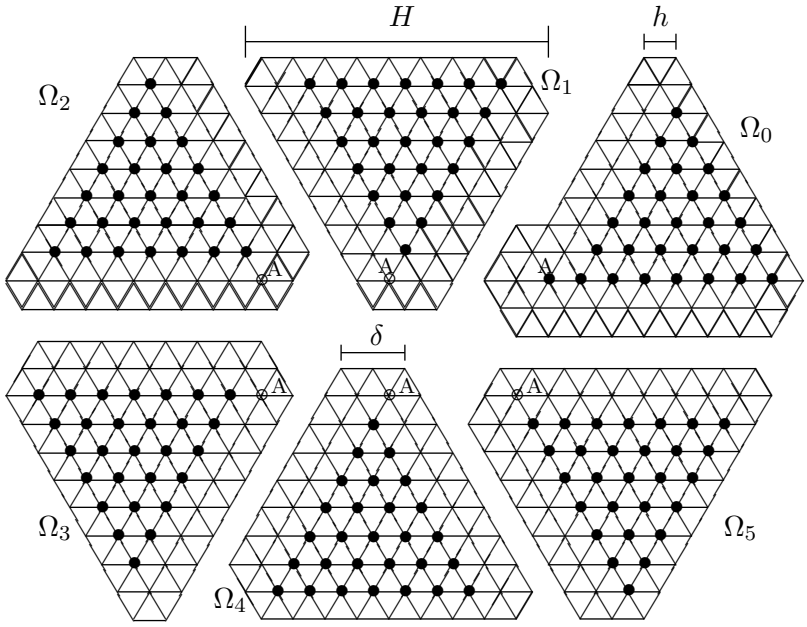
Next, we consider an additive Schwarz method.

For a given initial solution u^0, we get $u^{n+1}_{\Omega_i}$ by solving the following problems

$$\begin{cases} a(x,y)u^{n+1}_{\Omega_i} - \nabla \cdot b(x,y)\nabla u^{n+1}_{\Omega_i} &= f, \quad \text{in } \Omega_i, \\ u^{n+1}_{\Omega_i} &= 0, \quad \text{on } \partial\Omega_i \cap \partial\Omega, \\ u^{n+1}_{\Omega_i} &= u^n_{\Omega_j}, \quad \text{on } \partial\Omega_i \cap \Omega_j, \end{cases} \tag{5.4}$$



(a) On a real domain



(b) Separated by sub-domains

Figure 11: Overlapping sub-domains with $2h$-width (•: real nodes, A: origin).

for $i = 1, \ldots, N$. Using the above notation, we can write the additive Schwarz method as

$$u^{n+1} - u^0 = \sum_{i=1}^{N} \left(R_{\Omega_i}^T A_{\Omega_i}^{-1} R_{\Omega_i} A_J \right) (u^n - u^0). \tag{5.5}$$

The overlapping Schwarz method, when used in conjunction with the Krylov subspace method, has the following four properties of convergence behavior:

- the number of iterations grows as $1/H_i$,
- if δ is kept proportional to H_i, the number of iterations is bounded independently of h and H_i/h,
- the number of iterations for the multiplicative Schwarz method is roughly half of that needed for the additive Schwarz method, and
- convergence is poor for $\delta = 0$ but improves rapidly as δ is increased,

where h is the fine mesh size, δ is the overlapping size of sub-domains, and H_i is the domain size of the sub-domains.

5.2 Two-level Schwarz method

We consider the two-level overlapping method. This method is given in terms of overlapping partitions of Ω, so-called sub-domains Ω_i, $i = 1, \ldots, N$, which themselves are unions of finite elements. They have diameters of order H_i and all attended by a coarse shape-regular mesh with mesh size H as shown in Fig. 12. The mesh size H of the coarse mesh need not be related to the diameters H_i nor the mesh size h.

We define a coarser level finite element space V_0, a bilinear operator $A_{\Omega_0} : V_0 \to V_0$, and a projection $R_{\Omega_0} : V_J \to V_0$. Then we have a two-level additive Schwarz method, as in (5.5) by

$$u^{n+1} - u^0 = P_{\text{ad}}(u^n - u^0) \tag{5.6}$$

where $P_{\text{ad}} = \sum_{i=0}^{N} \left(R_{\Omega_i}^T A_{\Omega_i}^{-1} R_{\Omega_i} A_J \right)$.

From previous analysis results on the two level overlapping method [3, 25, 26], we have the following theorem.

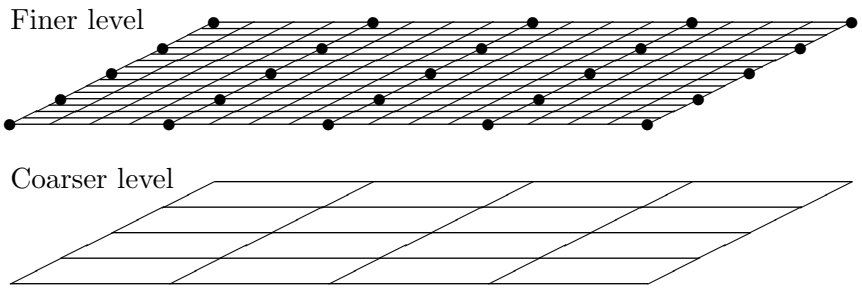


Figure 12: Two level mesh.

Theorem 5.1 *The condition number of the additive Schwarz operator satisfies*

$$k(P_{\text{ad}}) \leq C \left(1 + \frac{H}{\delta}\right),$$

where C is independent of h, H, and δ.

Theorem 5.1 shows that the condition number of P_{ad} does not depend on the number of sub-domains which is a required property for scalable programs on a massively parallel computer.

5.3 Implementation issues

In this subsection we consider the implementation on a massively parallel computer.

The effectiveness of the multiplicative Schwarz method has been shown theoritically and practically. But, this method needs sophisticated implementation on a parallel computer because it needs values which have to be updated on other sub-domains as it is also the case for the Gauss-Seidel method. So we discard the multiplicative Schwarz method.

For the additive method the condition number of P_{ad} depends on the overlapping width δ as shown in the previous subsections. To get a good condition number, we need a larger δ which means more values have to be transferred from other cores, i.e., higher costs for the data communication

step. If we use the Schwarz additive method with minimal overlapping δ to minimize the data communication cost, the resulting method is identical with the block Jacobi iteration which is well known. However, it is not an efficient preconditioner for the Krylov subspace method.

The condition number of the two-level Schwarz DDM does not depend on the number of sub-domains which is an appreciated property when it comes to the application on massively parallel computers. But this method needs a coarser level solver and requires a projection operator from fine level to coarse level. Therefore, this method has the same performance issue as the multigrid method which is related to the coarser level computations and the coarsest level solver.

From the above mentioned reasons there is no benefit to consider the one- and two-level Schwarz methods as an alternative coarsest level solver for the multigrid method. So we discarded these methods.

6 Non-overlapping DDMs

A non-overlapping DDM is the natural method for problems which have discontinuous coefficients and should be implemented on a distributed computer. The method can be classified by how the values on inner-boundary conditions, which are defined on the common boundaries of two and more sub-domains, are handled. The condition number of the non-overlapping DDM has the same property as the overlapping DDM, i.e., the one-level method depends on the number of sub-domains and the two-level method does not. Among the many non-overlapping DDMs, BDDC and FETI-DP are well developed two-level DDMs which have good performance on a massively parallel computer for many 2D and 3D problems. So we considered and implemented them.

6.1 Basic non-overlapping domain decomposition methods

In this subsection we consider the basic approach of the non-overlapping DDM and define some notations which are required to explain these methods.

Consider Eq. (2.1) on a region Ω with zero Dirichlet boundary condition on $\partial\Omega$. Suppose that Ω is partitioned into N non-overlapping sub-domains $\{\Omega_i, 1 \leq i \leq N\}$ with

$$\overline{\Omega} = \bigcup_i \overline{\Omega}_i \ ; \quad \Omega_i \cap \Omega_j = \emptyset, \quad i \neq j,$$

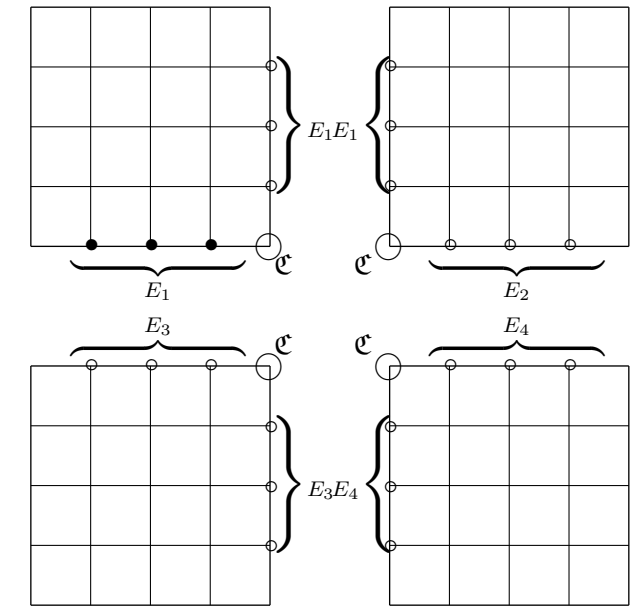
as shown in Fig. 13. If $\Gamma_i = \partial\Omega_i \setminus \partial\Omega$, then the interface Γ is defined as

$$\Gamma = \bigcup_i \Gamma_i.$$

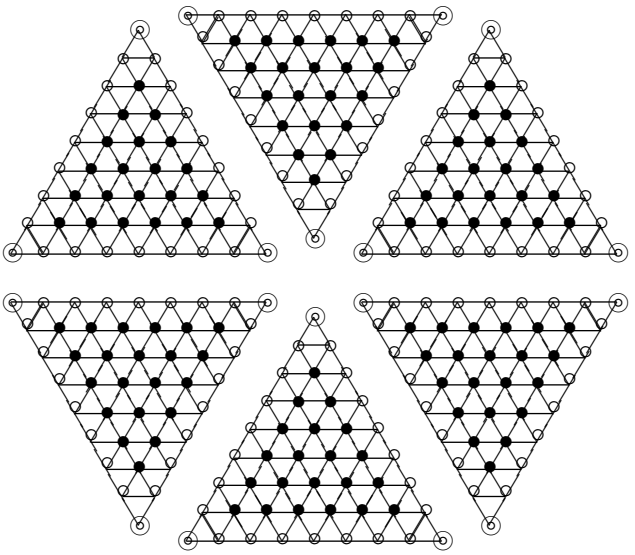
Under suitable regularity assumptions on f and boundaries of the sub-domains, typically f is square-summable and the boundaries satisfy the Lipschitz condition, (2.1) is equivalent to the following problem [26]:

$$\begin{aligned} a(x,y)u_{\Omega_i} - \nabla \cdot b(x,y)\nabla u_{\Omega_i} &= f && \text{in } \Omega_i, \\ u_{\Omega_i} &= 0 && \text{on } \partial\Omega_i \setminus \Gamma, \\ u_{\Omega_i} &= u_{\Omega_j} && \text{on } \partial\Omega_i \cap \partial\Omega_j \subset \Gamma, \\ \frac{\partial u_{\Omega_i}}{\partial \mathbf{n}_{\Omega_i}} &= -\frac{\partial u_{\Omega_j}}{\partial \mathbf{n}_{\Omega_j}} && \text{on } \partial\Omega_i \cap \partial\Omega_j \subset \Gamma, \end{aligned} \tag{6.1}$$

where u_{Ω_i} is the restriction of u to Ω_i and $\mathbf{n}_{\Omega_i}$ the outward normal to Ω_i. To simplify the notation we denote u_{Ω_i} by u_i and $\mathbf{n}_{\Omega_i}$ by $\mathbf{n}_i$.



(a) For a case with a rectangular domain and with rectangular sub-domains.



(b) For a case with hexagonal domain and with regular triangular sub-domains.

Figure 13: Sub-domains of the non-overlapping DDM.

The imposed conditions on the interface Γ are called **transmission conditions** and they are equivalent to the equality of any two independent linear combinations of the traces of the functions and their normal derivatives (flux).

We consider a triangulation of the domain Ω and a finite element approximation of the problem (6.1). We assume that the sub-domains consist of unions of elements or, equivalently, that the sub-domain boundaries do not cut through any elements. Such an approximation leads to a linear system

$$Au = f \tag{6.2}$$

with a symmetric, positive definite matrix A.

Also, we classify the nodes according to their location on the sub-domain Ω_i: interior nodes which are not shared with any other substructure are denoted by I, edge nodes which are shared by two or more substructures are denoted by Γ. Also, we may distinguish corner nodes from edge nodes which are used for a coarser problem and are denoted by $\mathfrak{C}$. For convenience we denote the set of edge nodes except the corner ones by $E = \Gamma \setminus \mathfrak{C}$ and $\Lambda = I \cup E$. We define a restriction operator R_{α_i} to the set α_i ($\alpha = I, \Gamma, \mathfrak{C}, E, \Lambda$ and $i = 1, \ldots, N$).

The linear system (6.2) can be written as

$$\begin{pmatrix} A_{II} & A_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} u_I \\ u_\Gamma \end{pmatrix} = \begin{pmatrix} f_I \\ f_\Gamma \end{pmatrix}, \tag{6.3}$$

where we have partitioned the degre of freedom (DoF) into those interior to the sub-domains and those laying on Γ, i.e.,

$$A_{II} = \begin{pmatrix} A_{II}^1 & 0 & 0 & \cdots & 0 \\ 0 & A_{II}^2 & 0 & \cdots & 0 \\ 0 & 0 & A_{II}^3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A_{II}^N \end{pmatrix}, \; A_{\Gamma I}^T = A_{I\Gamma} = \begin{pmatrix} A_{I_1\Gamma} \\ A_{I_2\Gamma} \\ A_{I_3\Gamma} \\ \vdots \\ A_{I_N\Gamma} \end{pmatrix}$$

and $A_{II}^i (= A_{I_i I_i})$ are symmetric, positive definite matrices of the DoF of the interior nodes of Ω_i with reordering.

The Schur complement system: In the first step of many iterative domain decomposition methods, the unknowns in the interior of the subdomains u_I are eliminated. This corresponds to a block factorization of the matrix of (6.3):

$$A = LR = \begin{pmatrix} I_{\mathrm{d}} & 0 \\ A_{\Gamma I} A_{II}^{-1} & I_{\mathrm{d}} \end{pmatrix} \begin{pmatrix} A_{II} & A_{I\Gamma} \\ 0 & A_{\Gamma\Gamma} - A_{\Gamma I} A_{II}^{-1} A_{I\Gamma} \end{pmatrix} \tag{6.4}$$

and the resulting linear system is

$$\begin{pmatrix} A_{II} & A_{I\Gamma} \\ 0 & S_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} u_I \\ u_\Gamma \end{pmatrix} = \begin{pmatrix} f_I \\ g_\Gamma \end{pmatrix} \tag{6.5}$$

where $I_{\rm d}$ is the identity matrix, $S_{\Gamma\Gamma} = A_{\Gamma\Gamma} - A_{\Gamma I}A_{II}^{-1}A_{I\Gamma}$, and $g_\Gamma = f_\Gamma - A_{\Gamma I}A_{II}^{-1}f_I$. Here $S_{\Gamma\Gamma}$ is called the Schur complement matrix relative to the unknowns on Γ.

We need to introduce another notation to explain the non-overlapping DD algorithm. We consider local Neumann boundary problems

$$\begin{aligned} a(x,y)u_i - \nabla \cdot b(x,y)\nabla u_i &= f \quad \text{in } \Omega_i, \\ u_i &= 0 \quad \text{on } \partial\Omega_i \cap \partial\Omega, \\ \frac{\partial u_i}{\partial \mathbf{n}_{\Omega_i}} &= g_i \quad \text{on } \partial\Omega_i \setminus \partial\Omega. \end{aligned} \tag{6.6}$$

If we add the following additional conditions on the common inner boundaries

$$u_i = u_j, \quad \text{and } g_i = -g_j, \quad \text{on } \partial\Omega_i \cap \partial\Omega_j, \tag{6.7}$$

then the problem (6.6) is equivalent to (6.1).

By using the FEM or FVM of Section 2, we have the following local linear systems

$$A^i u_i = f_i, \tag{6.8}$$

where

$$A^i = \begin{pmatrix} A^i_{II} & A_{I_i\Gamma_i} \\ A_{\Gamma_i I_i} & A_{\Gamma_i\Gamma_i} \end{pmatrix}, \quad u_i = \begin{pmatrix} u_{I_i} \\ u_{\Gamma_i} \end{pmatrix}, \quad f_i = \begin{pmatrix} f_{I_i} \\ f_{\Gamma_i} + g_{\Gamma_i} \end{pmatrix}.$$

In general, $A_{I_i\Gamma_i} = A_{I\Gamma}|_{\Omega_i}$, but, for the nodes in $\partial\Omega_i \cap \partial\Omega_j$, $u_{\Gamma_i} \neq u_{\Gamma_j}$ and $A_{\Gamma_i\Gamma_i} \neq A_{\Gamma_j\Gamma_j}$. Hence, we have $A_{\Gamma\Gamma} = \sum_{i=1}^N A_{\Gamma_i\Gamma_i}$.

From (6.8), we have

$$A = \sum_{i=1}^N A^i \tag{6.9}$$

and we can write $A^i = R_{\Omega_i}^T A R_{\Omega_i}$ for the projection operators R_{Ω_i}.

According to the handling of the transmission conditions, we classify the non-overlapping DDMs as Dirichlet-Neumann algorithm, Neumann-Neumann algorithm, and Dirichlet-Dirichlet algorithm. From these basic domain decomposition methods, in an effort to overcome the difficulties in their analysis and implementation, the BDDC and the FETI-DP methods have been developed. In the next subsections, we consider these well-developed methods and their implementation issues on a massively computer.

6.2 The Neumann-Neumann algorithm and BDDC

The basic Neumann-Neumann (NN) algorithm can be described as follows. We start from an initial guess u_Γ^0. For a given u_Γ^n, we compute u_Γ^{n+1} by:

NN1. Solve the Dirichlet problems on each $\Omega_i (i = 1, \ldots, N)$ with data u_Γ^n on Γ

$$\begin{aligned} a(x,y)u_i^{n+1/2} - \nabla \cdot b(x,y)\nabla u_i^{n+1/2} &= f && \text{in } \Omega_i, \\ u_i^{n+1/2} &= 0 && \text{on } \partial\Omega_i \setminus \Gamma, \\ u_i^{n+1/2} &= u_{\Gamma_i}^n && \text{on } \Gamma. \end{aligned} \tag{6.10}$$

NN2. Solve the Neumann problems on each $\Omega_i (i = 1, \ldots, N)$ with the Neumann data on Γ chosen as the difference of the normal derivatives of the solutions of the Dirichlet problems

$$\begin{aligned} a(x,y)\psi_i^{n+1} - \nabla \cdot b(x,y)\nabla \psi_i^{n+1} &= 0 && \text{in } \Omega_i, \\ \psi_i^{n+1} &= 0 && \text{on } \partial\Omega_i \setminus \Gamma, \\ \frac{\partial \psi_i^{n+1}}{\partial \mathbf{n}_i} &= \sum_{j=1}^{N} \frac{\partial u_j^{n+1/2}}{\partial \mathbf{n}_j} && \text{on } \Gamma. \end{aligned} \tag{6.11}$$

NN3. Update u_Γ^{n+1} with the solutions of the Neumann problems

$$u_{\Gamma_i}^{n+1} = u_{\Gamma_i}^n - \theta \left(\sum_{j=1}^{N} \psi_j^{n+1}|_{\Gamma_i} \right) \tag{6.12}$$

with a suitable $\theta \in (0, \theta_{\max})$.

In this algorithm, we impose a continuity condition on the inner boundary nodes Γ, so we have $u_{\Gamma_i}^n = u_{\Gamma_j}^n = u_\Gamma^n$ on the common boundary nodes $\partial\Omega_i \cap \partial\Omega_j$ for all n.

By using matrix notation, we have, for $i = 1, \ldots, N$,

$$A_{II}^i u_{I_i}^{n+1/2} = f_{I_i} - A_{I_i\Gamma_i} u_\Gamma^n, \tag{6.13}$$

$$A^i \begin{pmatrix} \psi_{I_i}^{n+1} \\ \psi_{\Gamma_i}^{n+1} \end{pmatrix} = \begin{pmatrix} 0 \\ r_\Gamma \end{pmatrix}, \tag{6.14}$$

where the residual r_Γ is defined by

$$r_\Gamma = \sum_{j=1}^{N} \left(A_{\Gamma_j I_j} u_{I_j}^{n+1/2} + A_{\Gamma_j \Gamma_j} u_\Gamma^n - f_{\Gamma_j} \right)$$

By eliminating $u_{I_i}^{n+1/2}$ and $\psi_{I_i}^{n+1}$ from (6.13) and (6.14) and denoting $g_{\Gamma_i} = f_{\Gamma_i} - A_{\Gamma_j I_j} A_{I_j I_j}^{-1} f_{I_j}$, $S_i = A_{\Gamma_j \Gamma_j} - A_{\Gamma_j I_j} A_{I_j I_j}^{-1} A_{I_j \Gamma_j}$, $g_\Gamma = \sum_{i=1}^{N} g_{\Gamma_i}$ and $S = \sum_{i=1}^{N} S_i$, the problem NN1 gives

$$\begin{aligned} r_\Gamma &= -\left(\sum_{j=1}^{N} (f_{\Gamma_j} - A_{\Gamma_j I_j} A_{I_j I_j}^{-1} f_{I_j}) - (A_{\Gamma_j \Gamma_j} - A_{\Gamma_j I_j} A_{I_j I_j}^{-1} A_{I_j \Gamma_j}) u_\Gamma^n \right) \\ &= -(g - S u_\Gamma^n) \end{aligned}$$

which shows that the difference r_Γ of the local fluxes is equal to minus the residual of the Schur complement system. Using a block factorization of the local matrix A^i, the problem NN2 gives

$$\psi_{\Gamma_i}^{n+1} = S_i^{-1} r_\Gamma = -S_i^{-1} (g_\Gamma - S u_\Gamma^n).$$

Therefore, we have

$$u_\Gamma^{n+1} - u_\Gamma^n = \theta \left(\sum_{j=1}^{N} S_i^{-1} \right) (g_\Gamma - S u_\Gamma^n)$$

which shows that the Neumann-Neumann algorithm is also a preconditioned Richardson iteration for the Schur complement system with the preconditioner $\sum_{j=1}^{N} S_i^{-1}$.

The system A^i in (6.14) may be a symmetric and positive semi-definite matrix. The Balancing Domain Decomposition (BDD) preconditioner was introduced by Mandel et al. [15] and solves the difficulty of the singularity of the generated matrix A^i of (6.14) by using the equilibrium conditions which lead to a simple and natural construction of a coarse problem.

The Balancing Domain decomposition by constraints (BDDC) algorithm is an algorithm for substructuring based on a constrained energy minimization concept [6, 16, 14].

By using the notation in (6.9), the model problem (2.1) can be written as

$$Au = \left(R^T \overline{A} R \right) u = f, \tag{6.15}$$

where

$$R = \begin{pmatrix} R_{\Omega_1} \\ \vdots \\ R_{\Omega_N} \end{pmatrix}, \quad \overline{A} = \begin{pmatrix} A^1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A^N \end{pmatrix}.$$

The BDDC preconditioner for A is specified by a symmetric and positive definite weight matrix D and a constraint matrix C_u. The matrix D has the same dimensions and block diagonal structure as A and forms a decomposition of unity in the sense that

$$R^T D R = I_{\mathrm{d}}.$$

The coarse vector u_c is given by

$$u_c = C_u u$$

where C_u specifies the constraints that are enforced between substructures in the BDDC preconditioner. These constraints have the effect of loosely coupling the substructures and naturally introducing a coarse problem. For example, C_u can be used to enforce equality of the substructure DoF's averages across faces, edges, or at individual DoF on substructure boundaries usually called corners.

Let R_{ci} select the rows of $C_u R_i^T$ with at least one non-zero entry and define

$$C_i = R_{ci} C_u R_i^T, \quad R_c = \begin{pmatrix} R_{c1} \\ \vdots \\ R_{cN} \end{pmatrix}, \quad C = \begin{pmatrix} C_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_N \end{pmatrix}.$$

The energy minimizing coarse basis functions Ψ are obtained from the solution of the saddle-point problem

$$\begin{pmatrix} \overline{A} & C^T \\ C & 0 \end{pmatrix} \begin{pmatrix} \Psi \\ \Pi \end{pmatrix} = \begin{pmatrix} 0 \\ R_c \end{pmatrix} \tag{6.16}$$

and the coarse stiffness matrix is defined as

$$S_c = \Psi^T \overline{A} \Psi.$$

Let

$$P_1 = R_I^T A_{II}^{-1} R_I, \quad P_2 = R^T D \Psi S_c^{-1} \Psi^T D R, \quad P_3 = R^T D Q D R, \tag{6.17}$$

where Q is defined such that

$$\begin{pmatrix} \overline{A} & C^T \\ C & 0 \end{pmatrix} \begin{pmatrix} Qg \\ \mu \end{pmatrix} = \begin{pmatrix} g \\ 0 \end{pmatrix}, \qquad \forall g$$

for some μ.

The BDDC preconditioner on a residual vector r can be defined as follows:

B1. Initial static condensation correction and residual update

$$u_0 = P_1 r, \quad r_1 = r - Au_0.$$

B2. Coarse grid and substructure corrections and residual update

$$u_1 = P_2 r_1, \quad u_2 = P_3 r_1, \quad r_2 = r_1 - A(u_1 + u_2).$$

B3. Final static condensation correction

$$u_3 = P_1 r_2.$$

B4. Preconditioned residual

$$Pr = u_0 + u_1 + u_2 + u_3.$$

Simplest real case: One of the simplest constrainted cases in 2D is the enforced equality at individual DoF on corners which share three or more substructures ($\mathfrak{C}$ in Fig. 13 (a) and $\bullet$ in Fig 14) and average values on edges (E_i in Fig. 13 (a) and $\circ$ in Fig. 14).

Because the operator P_1 is able to get the solutions of the local Dirichlet boundary problem, this step is well-defined and easly to understand. So we consider how to construct the operator S_c. To do this, we have to solve the saddle-point problem (6.16), i.e., solve the following saddle-point problems, for $i = 1, \ldots, N$,

$$\begin{pmatrix} A^i_{\Lambda\Lambda} & A^i_{\mathfrak{C}\Lambda} & {C^i_\Lambda}^T & 0 \\ A^i_{\Lambda\mathfrak{C}} & A^i_{\mathfrak{C}\mathfrak{C}} & 0 & I_{\mathrm{d}} \\ C^i_\Lambda & 0 & 0 & 0 \\ 0 & I_{\mathrm{d}} & 0 & 0 \end{pmatrix} \begin{pmatrix} \Psi^e_{\Lambda_i} & \Psi^c_{\Lambda_i} \\ \Psi^e_{\mathfrak{C}_i} & \Psi^c_{\mathfrak{C}_i} \\ \Upsilon^e_{\Lambda_i} & \Upsilon^c_{\Lambda_i} \\ \Upsilon^e_{\mathfrak{C}_i} & \Upsilon^c_{\mathfrak{C}_i} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ I_{\mathrm{d}} & 0 \\ 0 & I_{\mathrm{d}} \end{pmatrix}. \tag{6.18}$$

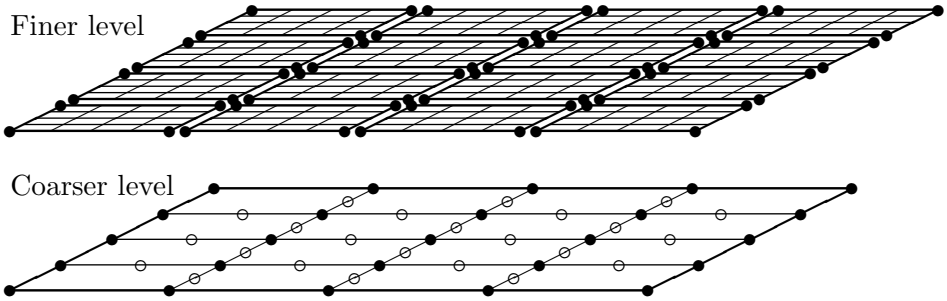


Figure 14: Two level mesh of the non-overlapping DDM.

From (6.18), we have

$$\begin{aligned}
\Psi^e_{\mathfrak{C}_i} &= 0,\\
\Psi^c_{\mathfrak{C}_i} &= I_{\mathrm{d}},\\
\Psi^e_{\Lambda_i} &= -(A^i_{\Lambda\Lambda})^{-1} {C^i_\Lambda}^T \Upsilon^e_{\Lambda_i},\\
\Psi^c_{\Lambda_i} &= -(A^i_{\Lambda\Lambda})^{-1}(A^i_{\mathfrak{C}\Lambda} + {C^i_\Lambda}^T \Upsilon^{\mathrm{c}}_{\Lambda_i}),\\
\Upsilon^e_{\mathfrak{C}_i} &= -A^i_{\Lambda\mathfrak{C}} \Psi^e_{\Lambda_i},\\
\Upsilon^c_{\mathfrak{C}_i} &= -A^i_{\Lambda\mathfrak{C}} \Psi^c_{\Lambda_i} - A^i_{\mathfrak{C}\mathfrak{C}},\\
C^i_\Lambda \Psi^e_{\Lambda_i} &= I_{\mathrm{d}},\\
C^i_\Lambda \Psi^c_{\Lambda_i} &= 0,
\end{aligned}$$

and

$$\begin{aligned}
\Upsilon^e_{\Lambda_i} &= -(C^i_\Lambda (A^i_{\Lambda\Lambda})^{-1} {C^i_\Lambda}^T)^{-1} = -S^i,\\
\Upsilon^c_{\Lambda_i} &= -(C^i_\Lambda (A^i_{\Lambda\Lambda})^{-1} {C^i_\Lambda}^T)^{-1} C^i_\Lambda (A^i_{\Lambda\Lambda})^{-1} A^i_{\mathfrak{C}\Lambda} = -S^i C^i_\Lambda (A^i_{\Lambda\Lambda})^{-1} A^i_{\mathfrak{C}\Lambda},
\end{aligned}$$

i.e.,

$$\begin{aligned}
\Psi^e_{\Lambda_i} &= (A^i_{\Lambda\Lambda})^{-1} {C^i_\Lambda}^T S^i,\\
\Psi^c_{\Lambda_i} &= -(A^i_{\Lambda\Lambda})^{-1} \left(I - {C^i_\Lambda}^T S^i C^i_\Lambda (A^i_{\Lambda\Lambda})^{-1} \right) A^i_{\mathfrak{C}\Lambda},
\end{aligned}$$

where $S^i = (C^i_\Lambda (A^i_{\Lambda\Lambda})^{-1} {C^i_\Lambda}^T)^{-1}$.

Then we define the global coarse matrix

$$S_c = \sum_{i=1}^{N} \begin{pmatrix} {\Psi^e_{\Lambda_i}}^T & 0 \\ {\Psi^c_{\Lambda_i}}^T & I_{\rm d} \end{pmatrix} \begin{pmatrix} A^i_{\Lambda\Lambda} & A^i_{\mathfrak{C}\Lambda} \\ A^i_{\Lambda\mathfrak{C}} & A^i_{\mathfrak{C}\mathfrak{C}} \end{pmatrix} \begin{pmatrix} \Psi^e_{\Lambda_i} & \Psi^c_{\Lambda_i} \\ 0 & I_{\rm d} \end{pmatrix} = \begin{pmatrix} S^i_{ee} & S^i_{ec} \\ {S^i_{ec}}^T & S^i_{cc} \end{pmatrix},$$

where

$$\begin{aligned} S^i_{ee} &= S^i, \\ S^i_{ec} &= S^i C^i_\Lambda (A^i_{\Lambda\Lambda})^{-1} A^i_{\mathfrak{C}\Lambda}, \\ S^i_{cc} &= A^i_{\mathfrak{C}\mathfrak{C}} - {A^i_{\mathfrak{C}\Lambda}}^T \left((A^i_{\Lambda\Lambda})^{-1} - (A^i_{\Lambda\Lambda})^{-1} {C^i_\Lambda}^T S^i C^i_\Lambda (A^i_{\Lambda\Lambda})^{-1} \right) A^i_{\mathfrak{C}\Lambda}. \end{aligned}$$

In a similar way, we have to solve (6.18) for each i and $Qg = z$, i.e.,

$$\begin{pmatrix} A^i_{\Lambda\Lambda} & A^i_{\mathfrak{C}\Lambda} & {C^i_\Lambda}^T & 0 \\ A^i_{\Lambda\mathfrak{C}} & A^i_{\mathfrak{C}\mathfrak{C}} & 0 & I_{\rm d} \\ C^i_\Lambda & 0 & 0 & 0 \\ 0 & I_{\rm d} & 0 & 0 \end{pmatrix} \begin{pmatrix} z_{\Lambda_i} \\ z_{\mathfrak{C}_i} \\ \mu_i \\ \phi_i \end{pmatrix} = \begin{pmatrix} g_{\Lambda_i} \\ g_{\mathfrak{C}_i} \\ 0 \\ 0 \end{pmatrix}$$

and have $z_{\mathfrak{C}_i} = 0$, $\mu_i = S^i {C^i_\Lambda}^T (A^i_{\Lambda\Lambda})^{-1} g_{\Lambda_i}$, and

$$z_{\Lambda_i} = \left(I - (A^i_{\Lambda\Lambda})^{-1} {C^i_\Lambda}^T S^i C^i_\Lambda \right) (A^i_{\Lambda\Lambda})^{-1} g_{\Lambda_i}.$$

The above matrices $A^i_{\Lambda\Lambda} (i = 1, \ldots, N)$ are symmetric and positive definite matrices, so we can solve the local problems by either a direct method or an iterative method.

6.3 The Dirichlet-Dirichlet algorithm and FETI-DP

We consider the Dirichlet-Dirichlet algorithm. We start from an initial guess $\lambda^0_\Gamma = \delta \lambda_{\Gamma_i}$ (δ is $+1$ or -1) of the flux on Γ. For given λ^n_Γ, we compute the following steps:

DD1. Solve Neumann problems on each $\Omega_i (i = 1, \ldots, N)$ with normal derivatives λ^n_Γ on Γ

$$\begin{aligned} a(x,y) u_i^{n+1/2} - \nabla \cdot b(x,y) \nabla u_i^{n+1/2} &= f_i \qquad \text{in } \Omega_i, \\ u_i^{n+1/2} &= 0 \qquad \text{on } \partial\Omega_i \setminus \Gamma, \\ \frac{\partial u_i^{n+1/2}}{\partial \mathbf{n}_i} &= \lambda^n_\Gamma \qquad \text{on } \Gamma. \end{aligned} \tag{6.19}$$

DD2. Solve Dirichlet problems on each $\Omega_i (i = 1, \ldots, N)$ with the Dirichlet boundary data on Γ chosen as the difference of the trace of the solutions of DD1

$$\begin{aligned} a(x,y)\psi_i^{n+1} - \nabla \cdot b(x,y)\nabla\psi_i^{n+1} &= 0 && \text{in } \Omega_i, \\ \psi_i^{n+1} &= 0 && \text{on } \partial\Omega_i \setminus \Gamma, \\ \psi_i^{n+1} = u_i^{n+1/2} - \sum_{j=1}^{N} u_j^{n+1/2}/N_\Gamma & && \text{on } \Gamma \end{aligned} \tag{6.20}$$

where N_Γ is the number of sub-domains which share the node of Γ.

DD3. Update flux λ_Γ^{n+1} with the values on Γ of the normal derivatives of the solutions of DD2

$$\lambda_\Gamma^{n+1} = \lambda_\Gamma^n - \theta \left(\frac{\partial \psi_i^{n+1}}{\partial \mathbf{n}_i} + \frac{\partial \psi_j^{n+1}}{\partial \mathbf{n}_j} \right) \quad \text{on } \Gamma \tag{6.21}$$

with a suitable $\theta \in (0, \theta_{\max})$.

By using matrix notation, we have, for $i = 1, \ldots, N$,

$$\begin{pmatrix} A_{II}^i & A_{I\Gamma}^i \\ A_{\Gamma I}^i & A_{\Gamma\Gamma}^i \end{pmatrix} \begin{pmatrix} u_{I_i}^{n+1/2} \\ u_{\Gamma_i}^{n+1/2} \end{pmatrix} = \begin{pmatrix} f_I^i \\ f_\Gamma^i + \lambda_i^n \end{pmatrix}, \tag{6.22}$$

$$A_{II}^i \psi_{I_i}^{n+1} = -A_{I\Gamma}^i \left(\sum_{j=1}^{N} \delta_j u_{\Gamma_j}^{n+1/2} \right), \tag{6.23}$$

$$\lambda_\Gamma^{n+1} = \lambda\Gamma^n - \theta \left[\sum_{j=1}^{N} A_{\Gamma I}^i u_i^{n+1} + A_{\Gamma\Gamma}^i \left(\sum_{j=1}^{N} \delta_j u_{\Gamma_j}^{n+1/2} \right) \right] \tag{6.24}$$

where δ_j is $+1$ or -1.

The Finite Element Tearing and Interconnecting (FETI) method is one of the Dirichlet-Dirichlet type algorithms and uses Lagrange multipliers λ to impose the continuity on the inner boundary nodes, which have the same roles as the flux λ_Γ [8].

We have the following saddle point problem:

$$\begin{pmatrix} A & B^T \\ B^T & 0 \end{pmatrix} \begin{pmatrix} u \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{6.25}$$

where

$$A = \begin{pmatrix} A_1 & 0 & \dots & 0 \\ 0 & A_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & A_N \end{pmatrix}, \quad A_i = \begin{pmatrix} A_{I_iI_i} & A_{\Gamma_iI_i} \\ A_{I_i\Gamma_i} & A_{\Gamma_i\Gamma_i} \end{pmatrix},$$

$$B = \begin{pmatrix} B_1 & B_2 & \dots & B_N \end{pmatrix}, \quad B_i = \begin{pmatrix} 0 & B_{\Gamma_i} \end{pmatrix}, \quad B_{\Gamma_i} = \begin{pmatrix} \delta & 0 & \dots & 0 \\ 0 & \delta & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \delta \end{pmatrix}$$

and δ is 1 or -1.

By block Gauss elimination, we have

$$\begin{pmatrix} A & B^T \\ 0 & -BA^{-1}B^T \end{pmatrix} \begin{pmatrix} u \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ BA^{-1}f \end{pmatrix}. \tag{6.26}$$

The FETI method solves the following problem to get λ

$$BA^{-1}B^T\lambda = -BA^{-1}f. \tag{6.27}$$

To solve (6.27), we may use the preconditioned CG method. After getting λ we can get the solution u by

$$u = A^{-1}(f - B^T\lambda).$$

But, the matrix A_i which comes from the Neumann boundary problem is usually a symmetric and positive semi-definite matrix. To avoid the singularity of the matrix, the FETI-DP method [7, 16, 13] imposes the continuity on the corner nodes which belong to more than two sub-domains (as shown in Fig. 14 •). Then we have

$$\begin{pmatrix} A_{\Lambda\Lambda} & A_{\mathfrak{C}\Lambda}^T & B^T \\ A_{\mathfrak{C}\Lambda} & A_{\mathfrak{C}\mathfrak{C}} & 0 \\ B & 0 & 0 \end{pmatrix} \begin{pmatrix} u_\Lambda \\ u_\mathfrak{C} \\ \lambda \end{pmatrix} = \begin{pmatrix} f_\Lambda \\ f_\mathfrak{C} \\ 0 \end{pmatrix} \tag{6.28}$$

Eliminating u_Λ by one step of the block Gaussian elimination, we obtain the reduced system

$$\begin{pmatrix} S_{\mathfrak{C}\mathfrak{C}} & -A_{\mathfrak{C}\Lambda}A_{\Lambda\Lambda}^{-1}B^T \\ -BA_{\Lambda\Lambda}^{-1}A_{\mathfrak{C}\Lambda}^T & -BA_{\Lambda\Lambda}^{-1}B^T \end{pmatrix} \begin{pmatrix} u_\mathfrak{C} \\ \lambda \end{pmatrix} = \begin{pmatrix} f_\mathfrak{C} - A_{\mathfrak{C}\Lambda}A_{\Lambda\Lambda}^{-1}f_\Lambda \\ -BA_{\Lambda\Lambda}^{-1}f_\Lambda \end{pmatrix} \tag{6.29}$$

where $S_{\mathfrak{C}\mathfrak{C}} = A_{\mathfrak{C}\mathfrak{C}} - A_{\mathfrak{C}\Lambda}A_{\Lambda\Lambda}^{-1}A_{\mathfrak{C}\Lambda}^T$.

By also eliminating the $u_{\mathfrak{C}}$ we obtain the reduced system

$$F\lambda = d \tag{6.30}$$

where

$$F = BA_{\Lambda\Lambda}^{-1}B^T + BA_{\Lambda\Lambda}^{-1}A_{\mathfrak{C}\Lambda}^T S_{\mathfrak{C}\mathfrak{C}}^{-1} A_{\mathfrak{C}\Lambda}A_{\Lambda\Lambda}^{-1}B^T = B\tilde{A}^{-1}B^T$$
$$d = BA_{\Lambda\Lambda}^{-1}f_\Lambda - BA_{\Lambda\Lambda}^{-1}A_{\mathfrak{C}\Lambda}^T S_{\mathfrak{C}\mathfrak{C}}^{-1}(f_{\mathfrak{C}} - A_{\mathfrak{C}\Lambda}A_{\Lambda\Lambda}^{-1}f_\Lambda) = B\tilde{A}^{-1}\tilde{f}.$$

In (6.27) and (6.30), $BA^{-1}B^T$ and F have less DoF than A in (2.13) because the number of DoF is given by the number of nodes on the inner-boundary. But, these matrices are more dense than A and globally defined, symmetric, positive definite matrices. From the construction of these matrices we can perform matrix-vector multiplication locally. So we can solve (6.27) and (6.30) by using iterative methods such as PCGM with a locally defined preconditioner.

Preconditioner: We define a Dirichlet preconditioner for (6.30) as

$$F_D^{-1} = \sum_{s=1}^{N} B \begin{pmatrix} 0 & 0 \\ 0 & S_{EE} \end{pmatrix} B^T \tag{6.31}$$

where

$$S_{EE} = K_{EE} - K_{IE}^T K_{II}^{-1} K_{IE}$$

and a lumped preconditioner as

$$F_L^{-1} = \sum_{s=1}^{N} B \begin{pmatrix} 0 & 0 \\ 0 & K_{EE} \end{pmatrix} B^T \tag{6.32}$$

as defined in [7].

The lumped preconditioner is obtained by simplifying the primal Schur complement of the Dirichlet preconditioner to its leading term K_{EE}. This simplification reduces the arithmetic complexity of the preconditioner step. The Dirichlet preconditioner is mathematically optimal. It is computationally more expensive than the lumped preconditioner, but it is computationally more efficient for plate and shell problems. On the other hand, the lumped preconditioner is not mathematically optimal, but it is computationally more efficient than the Dirichlet preconditioner for second-order problems. The required number of iterations for the PCGM with the Dirichlet preconditioner is smaller than with the lumped preconditioner as expected. The solution times of the PCGM have the same property, so we use the PCGM with the Dirichlet preconditioner for FETI-DP.

After solving (6.30), we can get finally the solution u by the following computations

$$u_{\mathfrak{C}} = S_{\mathfrak{C}\mathfrak{C}}^{-1}(f_{\mathfrak{C}} - A_{\mathfrak{C}\Lambda}A_{\Lambda\Lambda}^{-1}f_{\Lambda} + A_{\mathfrak{C}\Lambda}A_{\Lambda\Lambda}^{-1}B^T\lambda) \tag{6.33}$$

$$u_{\Lambda} = A_{\Lambda\Lambda}^{-1}(f_{\Lambda} - A_{\mathfrak{C}\Lambda}^{T}u_{\mathfrak{C}} - B^T\lambda). \tag{6.34}$$

7 Numerical experiments

In this section, we consider the numerical experimental results of the scaling properties of the matrix-vector multiplication, the CGM, the parallel multigrid method, the BDDC, and the FETI-DP.

As a model problem, we choose the simplest problem with $a(x, y) = 0$ and $b(x, y) = 1.0$ in (2.1), i.e., the Poisson problem. To test the numerical performance, we use the finite element discretization formula which is identical with the finite volume discretization for the test problem. As a termination criterion for the solvers, we define a reduction of the initial residual error on the finest level by a factor 10^{-8}.

We execute the implemented algorithm on the HELIOS machine. The machine is dedicated to the Fusion community in Europe and Japan. It is located in the International Fusion Energy Research Centre (IFERC) at Aomori, Japan. IFERC is funded for EU(F4E)-Japan by the broader approach collaboration. It is made of 4410 Bullx B510 blades nodes of two Intel Sandy-Bridge EP 2.7 GHz processors with 64 GB memory each and connected by Infiniband QDR. So it has 70 560cores in total and a Linpack performance of 1.23 Petaflops.

7.1 Matrix-Vector Multiplication and CGM

We consider the scaling properties of the matrix-vector multiplication which is the basic operation of iterative methods such as CGM, GMRES, the multigrid methods, etc. First, we investigate the strong scaling property which measures the solution time of a fixed problem while the number of cores is increased. We tested four different problems with 780k, 3.1M, 12.6M, and 50M of DoF and report the results in Fig. 15. We ran 40 000 times the matrix-vector multiplication and took the average to get the execution time for one matrix-vector multiplication. The results in Fig. 15 show that the matrix-vector multiplication has a very good strong scaling property up to a certain number of cores which depends on the number of DoF.

Next, we consider the weak scaling property which measures the execution time of a fixed number of operations per core while the number of cores and hence also the problem size are increased. We depict the time of a matrix-vector multiplication for 2.2k, 8.5k, 33k, 132k, 527k, and 2M DoF per core in Fig. 16. The results show that the matrix-vector multiplication has an almost perfect weak scaling property for all cases.

Now, we consider the scaling properties of the PCGM with a Jacobi preconditioner. It is well-known that the required number of iterations of

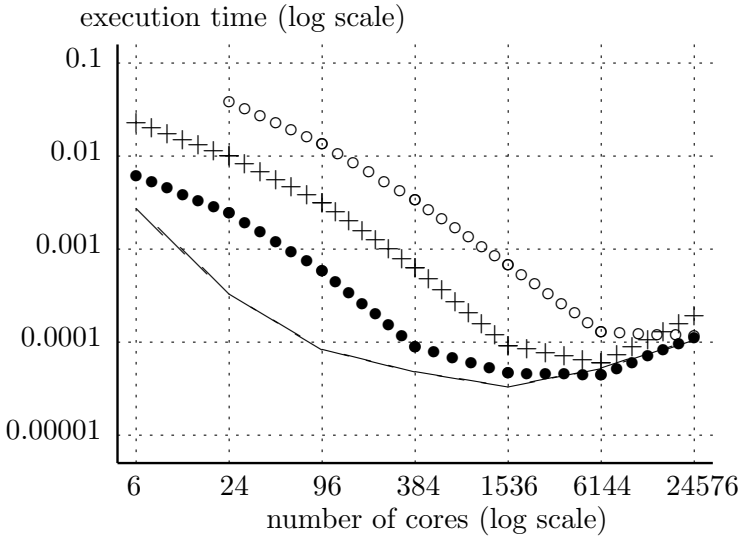


Figure 15: The execution time of the matrix-vector multiplication in seconds as a function of the number of cores for domains with 780k DoF (solid line), 3.1M DoF (•), 12.6M DoF (+), and 50M DoF (○).

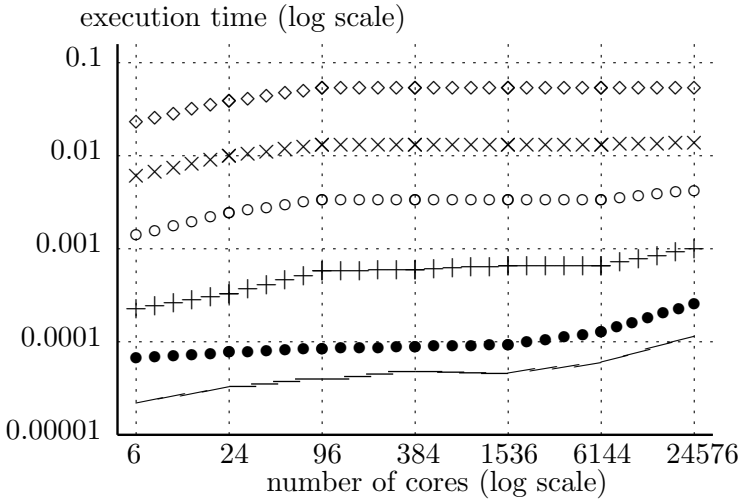


Figure 16: The execution time of the matrix-vector multiplication in seconds as a function of the number of cores for domains with 2.2k DoF (solid line), 8.5k DoF (•), 33k DoF (+), 132k DoF (○), 527k DoF (×), and 2M DoF(◇) per core.

the CGM increases with the number of DoF. This fact is true for the PCGM with a Jacobi preconditioner. The Jacobi preconditioner does not depend on the number of cores, so that the required number of iterations does not change according to the number of cores. We report the required number of iterations in Table 2 according to the size of the problem by denoting the levels. We limit the maximum number of iteration to 15 000. From level 13 on, we need more than 15 000 iterations and report up to level 12. The results in Table 2 show that the required number of iterations increases with the square root of the number of DoF.

Levels	DoF	iterations	iteration/$\sqrt{\text{DoF}}$
6	12 097	213	1.937
7	48 769	411	1.861
8	195 841	809	1.812
9	784 897	1 591	1.796
10	3 142 657	3 056	1.724
11	12 576 769	5 614	1.583
12	50 319 361	10 965	1.546

Table 2: The required number of iterations for the PCGM with a Jacobi preconditioner according to the levels.

We consider the strong and weak scaling properties of the PCGM with a Jacobi preconditioner. For the strong scaling property, we tested four different problems with 780k, 3.1M, 12.6M, and 50M DoF and depicted the results in Fig. 17. The solution times decrease as the number of cores increase up to a certain number of cores, which is 96 cores for 780k DoF, 384 cores for 3.1M DoF, and 1536 cores for 12.6M DoF. A further increase beyond this optimal number of cores gives a degradation of the solution times.

For the weak scaling property, we consider the solution time and the execution time of one iteration of the PCGM. The required number of iterations of the PCGM increases with the problem size as shown in Table 2, so we can expect that the solution time will increase with the problem size. From a certain level on we cannot solve the problem within the maximum number of iterations. So, we consider the execution time of one iteration of the PCGM which can be measured even if we cannot solve the problem within our iteration constraints. We tested four cases: 2.2k DoF, 8.5k DoF, 33k DoF, and 132k DoF per core and depicted the results in Fig. 18.

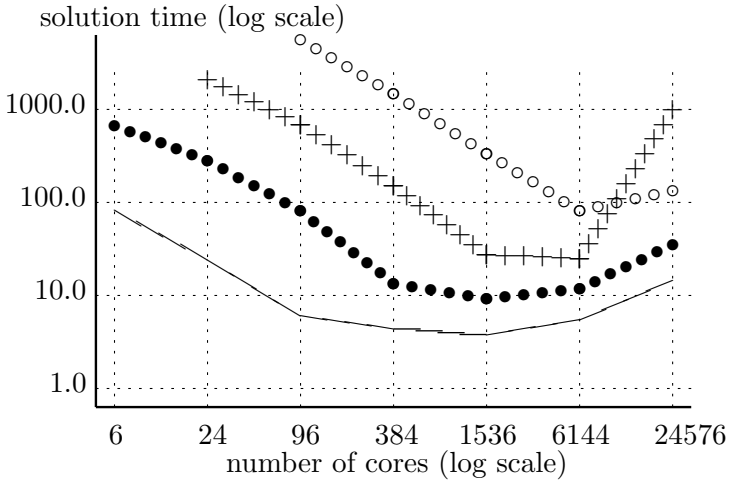


Figure 17: The solution times in seconds of the PCGM with a Jacobi preconditioner as a function of the number of cores for domains with 780k DoF (solid line), 3.1M DoF (•), 12.6M DoF (+), and 50M DoF (○).

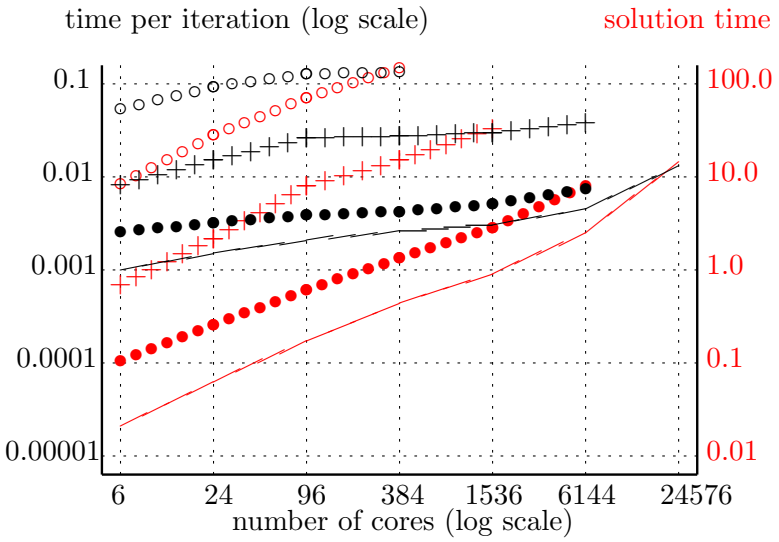


Figure 18: The execution time per PCGM iteration in seconds (in black) and the solution times of the PCGM with a Jacobi preconditioner in seconds (in red) as a function of the number of cores for domains with 2.2k DoF (solid line), 8.5k DoF (•), 33k DoF (+), and 132k DoF (○) per core.

The solution time increases with the number of cores, i.e., the problem size, as shown in Fig. 18 in red. The execution times of one iteration have a relatively good weak scaling property for all cases. Again we find a better weak scaling property for larger numbers of DoF per core similar to the matrix-vector mulplication results.

7.2 Scaling properties of the multigrid method

We consider the V-cycle multigrid method as a solver and as a preconditioner for the PCGM. For the multigrid method one should test both the Gauss-Seidel and Jacobi smoothers. However, the multigrid method with a Gauss-Seidel smoother is better than with a Jacobi smoother. So we consider the former only. Usually, the Jacobi smoother is used for debugging purposes in a parallel implementation because its results does not depend on the number of cores. In the multigrid method, we use the PCGM with symmetric Gauss-Seidel preconditioner as a lowest level solver and run two pre-smoothing and two post-smoothing iterations for all cases. The symmetric Gauss-Seidel iteration runs forward and backward Gauss-Seidel iterations alternatingly. Therefore, we swap forward and backward Gauss-Seidel iterations in both the pre- and post-smoothing iterations.

We tested different data gathering levels on a fixed number of cores. Without gathering the data, the feasible coarsest level of the multigrid algorithm is the level that has at least one DoF per core, i.e., one cannot go to the lower levels on the parallel algorithm. For the multigrid method without gathering the data, we have to use the feasible coarsest level as lowest level and solve exactly the lowest level problem by using the PCGM. For the multigrid method with gathering the data, this level is the lowest gathering level. So the lowest gathering level will increase with the number of cores. We tested four different cases, 2.2k, 8.5k, 33k, and 132k DoF per core with the lowest gathering level as gathering level and depicted the results in Fig. 19. They show that the gathering of the data is needed for large numbers of cores. The solution time of the solver with the gathering data shows a significant improvement in these cases, especially for small numbers of DoF per core.

One question which arises in this context is on which level we gather the data for each core. To answer this question, we tested different gathering levels on three fixed sets of cores: 384, 1536, and 6144 cores. The lowest gathering level for 384 cores is five, for 1563 cores six, and for 6144 cores seven. After gathering the data, we can introduce lower levels as the lowest level of the multigrid algorithm. So the lowest level of the multigrid method

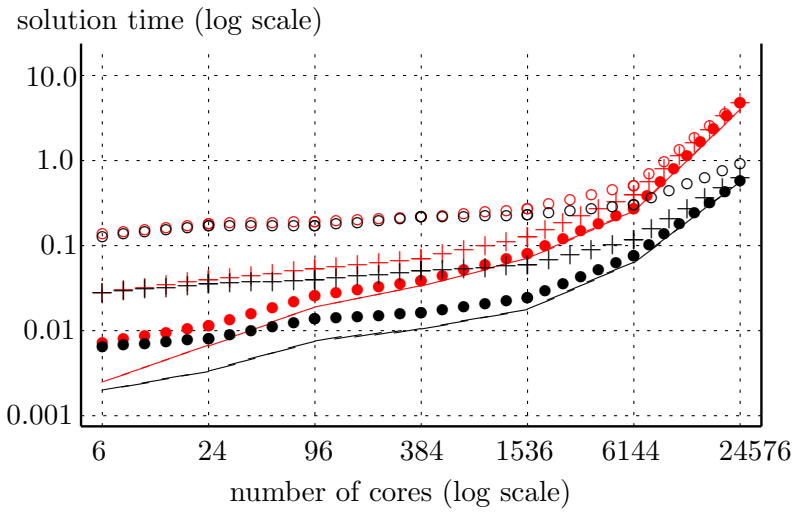


Figure 19: The solution times in seconds of the multigrid method as a preconditioner for the PCGM with the Gauss-Seidel smoother with (in black) and without (in red) gathering data as a function of the number of cores for domains with 2.2k DoF (solid line), 8.5k DoF (•), 33.4k DoF(+), and 132k DoF(∘) per core

can be any level from one to the gathering level.

We tested three different gathering levels for each set of cores, namely, level 5, 6, 7 for 384 cores, level 6, 7, 8 for 1536 cores, and level 7, 8, 9 for 6144 cores. The results are depicted in Fig. 20. We chose problems which have the same number of DoF per core because we can easily compare the differences of their solution times.

The results in Fig. 20 show that the lowest solving level of the multigrid method does not affect the solution time up to level five. Also, they show that the lowest possible gathering level has always the best performance and that the difference is more significant as the number of cores is increased. Therefore, we can conclude that we have to use the lowest possible gathering level as gathering level and may use any level less than six as the coarsest solving level in the multigrid algorithm.

From now on, we choose the lowest possible gathering level and level one as the coarsest solving level for the multigrid algorithm. In Fig. 21 we report the solution times of the solvers for a fixed problem size (strong scaling property). The coressponding speed-up is depicted in Fig. 22. The numerical results in Figs. 21 and 22 show an almost perfect strong scaling

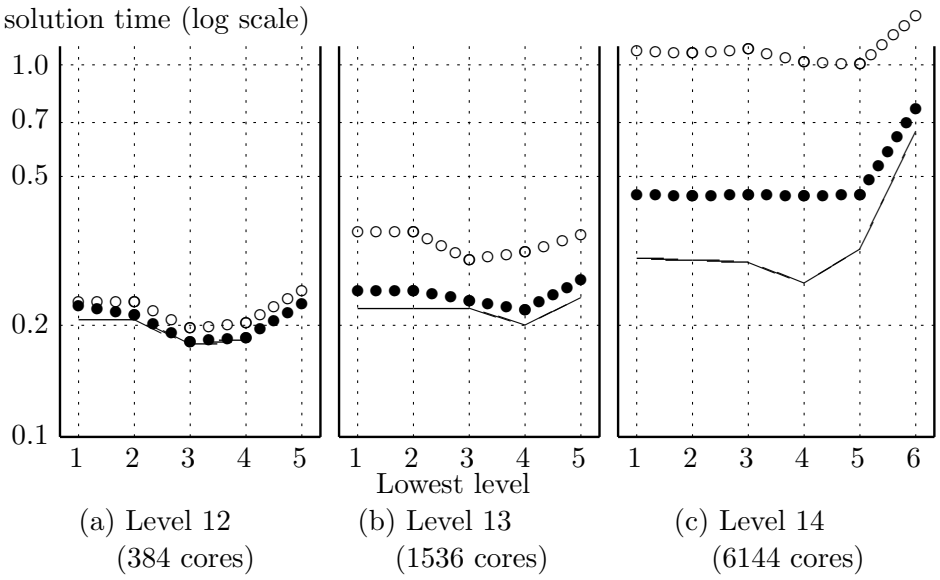


Figure 20: The solution times in seconds of the multigrid method with gathering the data as a preconditioner for the PCGM method as a function of the lowest level on which the direct solution is performed. Results for different gathering levels: lowest possible gathering level (solid line), second lowest level (•), and third lowest level (∘).

property up to 1 536 cores. In general, problems with a larger number of DoF have a better strong scaling property.

For the multigrid algorithm it is nearly impossible to fix the number of operations per core while increasing the total problem size, so we consider a semi-weak scaling by fixing the number of DoF of the finest level on each core. We tested six different numbers of DoF per core on the finest level, from 2.2k DoF to 2.1M DoF and depicted the results in Fig. 23. The results show that the multigrid method as a solver and as a preconditioner has really good semi-weak scaling properties when the number of DoF per core is large (see 527k DoF and 2.1M DoF per core cases). This is the typical behavior of the multigrid algorithm, i.e., the weak scaling property becomes better as the number of DoF per core is increased as shown in Fig. 23. In comparison with the execution time of the matrix-vector multiplication in Fig 16 and the execution time of one iteration of the PCGM in Fig. 18, the (semi)-weak scaling property of the multigrid algorithm is the worst case. Especially, the cores with a small number of DoF are affected. Nevertheless, the (semi)-weak scaling property could be improved with a more scalable lowest level

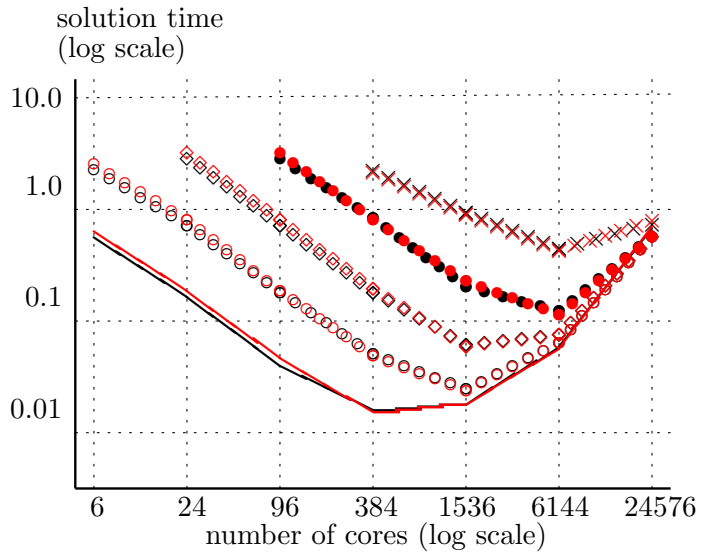


Figure 21: The solution times in seconds of the multigrid method as a solver (in red) and as a preconditioner for the PCGM (in black) as a function of the number of cores for domains with 3.1M DoF (solid line), 12.5M DoF (○), 50M DoF (◇), 201M DoF (●), and 805M DoF (×).

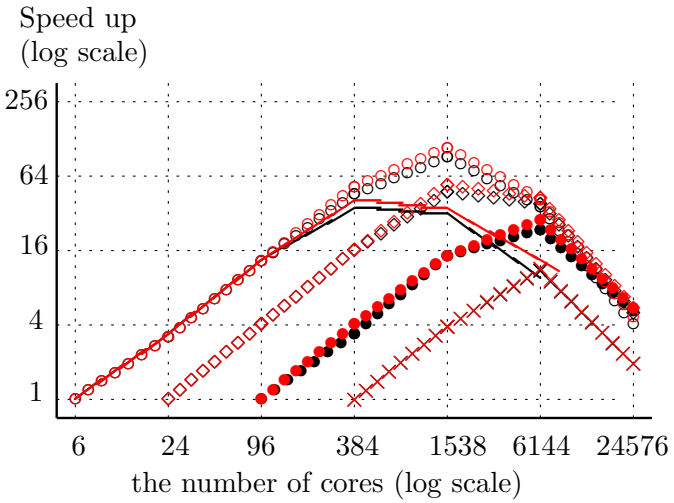


Figure 22: The speed up of the multigrid method as a solver (in red) and as a preconditioner for the PCGM (in black) as a function of the number of cores for domains with 3.1M DoF (solid line), 12.5M DoF (○), 50M DoF (◇), 201M DoF (●), and 805M DoF (×).

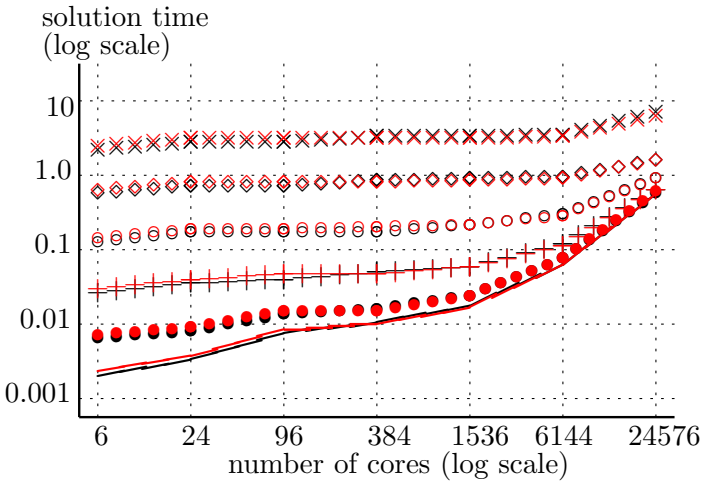


Figure 23: The solution times in seconds of the multigrid method as a solver (in red) and as a preconditioner for the PCGM (in black) as a function of the number of cores. Six different cases with fixed numbers of DoF [2.2k DoF (solid line), 8.5k DoF (•), 33.4k DoF (+), 132k DoF (○), 527k DoF (◇), and 2.1M DoF (×)] per core on the finest level.

solver for the multigrid algorithm. This could be probably achieved by either using an OpenMP/MPI hybridization, or processor accelerators such as GPUs or Intel MIC cards.

7.3 Scaling properties of the BDDC and FETI-DP

In this section we consider the two-level non-overlapping DDMs, the FETI-DP and BDDC. In similarity with the two-level Schwarz method, the required number of iterations of these two-level non-overlapping methods does not depend on the number of sub-domains, but instead depends on the ratio of the mesh size of the triangulation (fine level, h) and the size of the sub-domains (coarse level, H). We list the required number of iterations of the FETI-DP and BDDC in Table 3. It shows that the required number of iterations depends on the ratio of the mesh size of the fine level to the coarse level as it is the case for the Schwarz method, except that it does not increase as rapidly.

To implement the FETI-DP and BDDC methods, we have to solve local problems with Dirichlet and/or Neumann boundary conditions on each

h/H	1/8		1/16		1/32		1/64		1/128	
# cores	FD	BD	FD	BD	FD	BD	FD	BD	FD	BD
24	12	7	14	8	16	9	18	10	20	12
96	15	8	17	9	20	11	23	13	26	14
384	16	8	19	10	22	11	24	13	28	14
1536	16	8	20	10	23	11	26	13	29	14
6144	16	8	19	10	23	11	26	13	30	14
24576	16	8	19	9	23	11	26	13	29	14

Table 3: The required number of iterations of FETI-DP and BDDC

sub-domain and one globally defined coarse level problem (S_c for the BDDC and $S_{\mathfrak{cc}}$ for the FETI-DP). Furthermore, we need to communicate data with neighboring sub-domains and data on the coarse level. Solving the local problems and communicating data with neighboring sub-domains are performed in parallel. So these local steps do alter the performance by changing the number of cores. Otherwise, the dimension of the global coarse level problem would grow as the number of cores increases. The dimensionality of the coarse level problem used for BDDC (S_c) is the same as of the coarsest gathering level used for the multigrid method. Instead, the dimensionality of the coarse level problem of the FETI-DP ($S_{\mathfrak{cc}}$) is one level lower.

We use the same gathering algorithm of the multigrid method to solve the global coarse level problem. In both FETI-DP and BBDC, every sub-domain has some contribution to the matrices and vectors on the coarse level and uses the solution of the coarse level problem. So we gather these contributions on each core using `MPI_Allreduce` and use the solution after solving the coarse problem without any data communication.

To solve the local and global problems, we used two direct methods, the LAPACK (Intel MKL) library with dense matrix format and the IBM WSMP library with sparse matrix format, and the multigrid method as an iterative method. As we expected, the LAPACK solver is the fastest for small number of DoF problems, whereas the multigrid method has to be used for large number of DoF problems due to the increasing memory consumption. For comparison to our previous results, we chose the solver which performs best. We tested five different cases with a fixed number of DoF per core, from 32 DoF to 8k DoF. The results in Fig. 24 show that the FETI-DP is faster than the BDDC even though the latter requires a smaller number of iterations, as shown in Table 3. These results also show

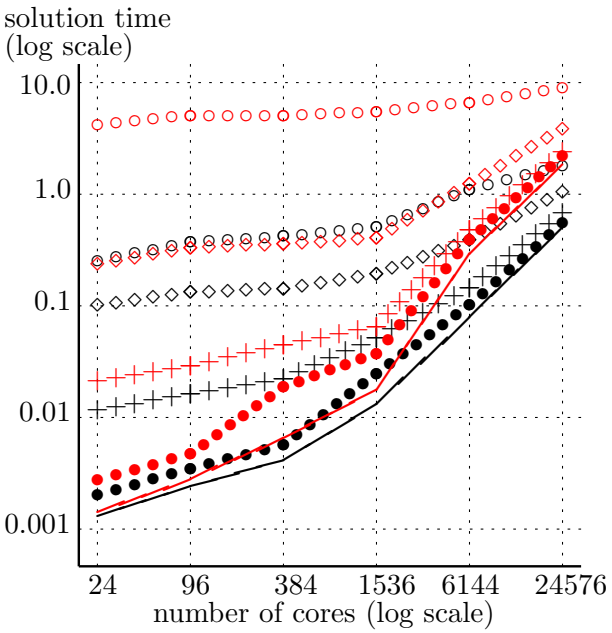


Figure 24: The solution times in seconds of the BDDC (in red) and the FETI-DP (in black) as a function of the number of cores. Five different cases with fixed number of DoF per core are depicted [32 DoF (solid line), 128 DoF (•), 500 DoF (+), 2k DoF (◇), and 8k DoF (○) per core].

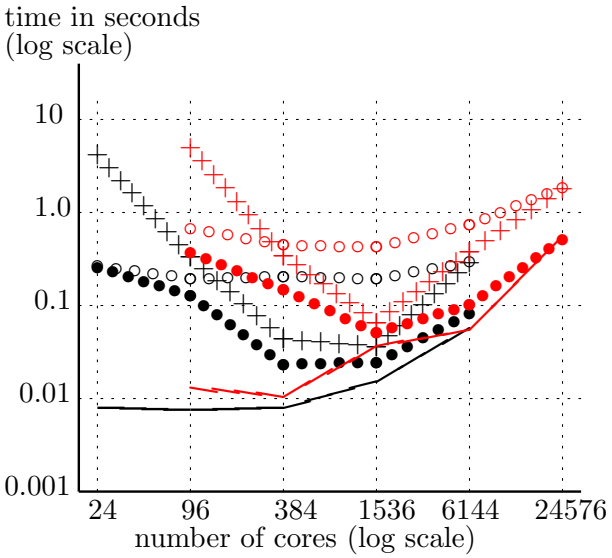


Figure 25: The solution times in seconds of the multigrid method as a preconditioner for the PCGM (solid line), the FETI-DP (•), the BDDC (+), and the CGM (∘) as a function of the number of cores. Two different cases with a different number of DoF [780k DoF (in black), and 3.1M DoF (in red)].

that the weak scaling property is improved as the number of DoF per core is increased.

7.4 The lowest level solver

In this section we consider the lowest level solver of the parallel multigrid method. In accordance with most structured discretization problems, we can use any coarser level as the lowest level in our problem. But most of the practical problems suffer from the lowest level limitation because the solution on the lower level problem does not well approximate the solution of the continuous problem. To solve such problems, we may use a direct method as the lowest level solver when the size of the lowest level problem is small enough. However, the size of the lowest level problem is usually too large to use the direct method due to memory constrains. For such problems we may use the CGM or DDM.

From now on, we compare the solution times of the CGM and DDM

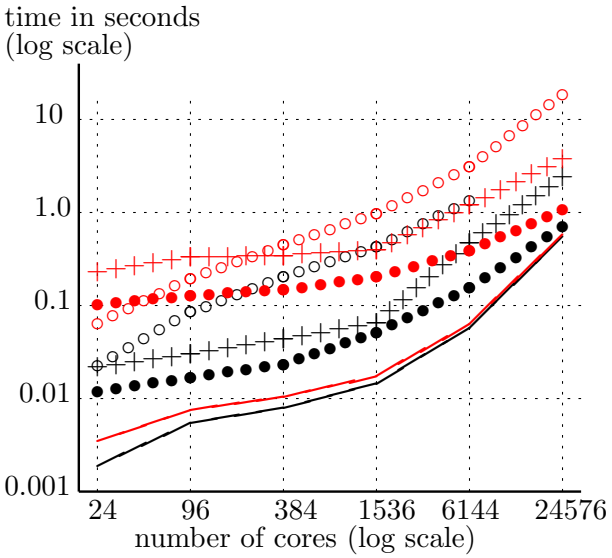


Figure 26: The solution times in seconds of the multigrid method as a preconditioner for the PCGM (solid line), the FETI-DP (•), the BDDC (+), and theCGM (∘) as a function of the number of cores. Two different cases with different fixed number of DoF per core [590 DoF (in black), and 2200 DoF (in red) per core.]

(BDDC and FETI-DP). First, we compare the solution times when the size of the problems is fixed (strong scaling). We test two cases, one has 780k DoF and the after 3.1M DoF and depict the results in Fig. 25 with the solution time of the multigrid method used as a preconditioner as a reference. The multigrid method achieves the fastest result. The FETI-DP is always faster than the CGM and the BDDC is faster than the CGM except on the smaller number of cores.

Next, we compare the solution times when the number of DoF per core is fixed (weak scaling). We also test two cases, one has 590 DoF and the after 2 200 DoF per core and depict the results in Fig. 26. The multigrid method with gathering the data is the fastest.

The results in Figs. 25 and 26 show that the FETI-DP can be used as the lowest level solver in the multigrid method for problems which have a large number of degree of freedom on the lowest level.

8 Conclusions

We have implemented a sophisticated communication pattern between cores to exchange the information necessary for the ghost nodes on a structured triangular grid on a regular hexagonal domain. Such a communication pattern has a very good parallel performance for a matrix-vector multiplication and the CGM.

We have implemented the multigrid method as a solver and as a preconditioner of the preconditioned conjugate gradient method with a Gauss-Seidel smoother. To improve the performance on massively parallel computers, we consider the gathering of the data for each core on a certain coarser level. The numerical experimental results show that the performance improvement of the gathering data algorithm is significant and the optimal data gathering level is the coarsest one for a parallel multigrid method. They show also that the multigrid algorithm has a very good semi-weak scaling property up to 24 576 cores for the larger problem sizes.

Furthermore, we implemented three different domain decomposition methods: the two-level Schwarz, the FETI-DP, and the BDDC methods. The two-level Schwarz method appeared to be inefficient for our test problem. In contrast, the FETI-DP and BDDC methods proved to be feasible. The FETI-DP method showed a better perfromance than the BDDC method and almost the same as the multigrid method for smaller problem sizes.

The multigrid method is the fastest solver for our model problem for most of the test cases, but it needs to be improved for cases with smaller number of degree of freedom (DoF) per core. For such problems the FETI-DP method is not a suitable candidate as lowest level solver of the multigrid method.

Acknowledgments

This work was carried out using the HELIOS supercomputer system at the Computational Simulation Centre of the International Fusion Energy Research Centre (IFERC-CSC), Aomori, Japan, under the Broader Approach collaboration between Euratom and Japan, implemented by Fusion for Energy and JAEA.

I would like to thank R. Hatzky and other HLST team members for helpful discussions.

I would like to thank B. Scott for submitting the project MGTRI as project coordinator and for helpful discussions.

I would like to thank D. Tskhakaya for helpful discussions.

This project has received funding from the Euratom research and training programme 2014–2018.

References

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

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

[3] X. C. Cai, *An optimal two-level overlapping domain decomposition method for elliptic problems in two and three dimensions*, SIAM J. Sci. Comp., **14** (1993), pp. 1–8

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

[5] 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.

[6] C. R. Dohrmann, *A preconditioner for substructuring based on constrained energy minimization*, SIAM J. Sci. Comput., **25** (2003), pp. 246–258.

[7] C. Farhat, M. Lesoinne, P. le Tallec, K. Pierson, and D. Rixen, *FETI-DP: A dual-primal unified FETI method – part I: A faster alternative to the two-level FETI method*, Internat. J. Numer. Methods Engrg, **42** (2001), pp. 1523–1544.

[8] C. Farhat and F. X. Roux, *A Method of Finite Element Tearing and Interconnecting and its Parallel Solution Algorithm*, Internat. J. Numer. Methods Engrg, **32** (1991), pp. 1205–1227.

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

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

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

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

[13] A. Klawonn and O. Rheinbach, *Inexact FETI-DP methods*, Technical Report SM-E-609, Univeristy of Duisburg-Essen, July 2005.

[14] J. Li and O. B. Widlund, *On the use of inexact subdomain solvers for BDDC algorithms*, Technical Reports TR2005-871, Department of Computer Science, Courant Institute. July 2006.

[15] J. Mandel, *Balancing domain decomposition*, Communications in Numerical methods in Engineering, **9** (1993), pp. 233–241.

[16] J. Mandel and C. R. Dohrmann, *Convergence of a balancing domain decomposition by constraints and energy minimization*, Numerical Linear Algebra with Applications, **10** (2003), pp. 639–659.

[17] S. McCormick, *Multilevel Adaptive Methods for Partial Differential Equations*, SIAM, Philadelphia, 1989.

[18] E. Poli, A. Bottino, W. A. Hornsby, A. G. Peeters, T. Ribeiro, B. D. Scott, and M. Siccino, *Gyrokinetic and gyrofluid investigation of magnetic islands in tokamaks*, Plasma Physics and Controlled Fusion, **52** (2010), Art. No. 124021.

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

[20] T. Ribeiro and B. D. Scott, *Conformal Tokamak Geometry for Turbulence Computations*, IEEE Trans. Plasma. Sci., **38** (2010), pp. 2159–2168.

[21] 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.

[22] R. Sadourny, A. Arakawa, and Y. Mintz, *Integration of the nondivergent barotropic vorticity equation with an Icosahedral-Hexagonal grid for the sphere*, Monthly Weather Review, **96** (1968), pp. 351–356.

[23] H. A. Schwarz, *Gesammelte Mathematische Abhandlungen*, Volume 2, pp. 133–143. Springer, Berlin, 1890. First published in Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich, volume 15, 1870, pp. 272–186.

[24] B. D. Scott, *Free Energy Conservation in Local Gyrofluid Models*, Physics of Plasmas, **12** (2005), Art. No. 102307.

[25] B. Smith, P. Bjørstad, and W. Gropp, *Domain Decomposition Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambrifge University Press, 1996.

[26] A. Toselli and O. Widlund, *Domain Decomposition Methods — Algorithms and Theory*, Springer Series in Computational Mathematics, **24**, Springer-Verlag, Berlin Heidelberg, 2010.

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