Czech Technical University in Prague
Faculty of Mechanical Engineering

Numerical solution of the incompressible flow using a domain decomposition method

Ph.D. thesis

Ing. Martin Hanek

Program
Mechanical Engineering

Study field
Mathematical and Physical Engineering

Supervisor
Prof. RNDr. Pavel Burda, CSc.

Co-supervisor
Ing. Jakub Šístek, Ph.D.

Prague 2022

Acknowledgements

Here, I would like to thank several people, who influenced my work. First, I would like to thank my advisor Prof. Pavel Burda. He has advised me since my bachelor thesis, introduced me to numerical mathematics, and has been helpful during my whole studies.

I greatly thank Dr. Jakub Šístek. He has been very supportive during my research and taught me parallel computing and domain decomposition methods. He has provided me with an opportunity to use and modify in-house software which he has been partly developing. Also, most of my research was joint work with him and he motivated me during my studies.

I also thank Dr. Eduard Stach for providing me the challenging problem of oil flow in hydrostatic bearings and helping me with the related experimental mesurements.

Finally, I would like to thank my family for their support and patience and for creating beautiful background which allows me to fully focus on my research and Ph.D. thesis.

My research during the Ph.D. studies was supported by the Student Grants SGS16/206/OHK2/3T/12 and SGS19/154/OHK2/3T/12, by the Czech Academy of Sciences under grant 18-09628S, and by the IT4Innovations under projects OPEN-4-10, OPEN-6-15, OPEN-10-43, OPEN-14-36, and OPEN-17-40.

Abstrakt

V této disertační práci se zabývám numerickým řešením Navierových-Stokesových (N-S) rovnic pro stacionární nestlačitelné proudění pomocí metody Balancing Domain Decomposition by Constraints (BDDC). Pro diskretizaci N-S rovnic je použita metoda konečných prvků (MKP).

V práci je představena víceúrovňová metoda BDDC, která je vhodná pro řešení velkých soustav lineárních algebraických rovnic. Tato metoda a její teorie je dobře známa pro Poissonovu úlohu, úlohy lineární pružnosti a proudění popsané Stokesovými rovnicemi. Práce je věnována rozšíření této víceúrovňové metody na nesymetrickou úlohu vznikající diskretizací N-S rovnic. Pro tuto formu BDDC jsou také diskutovány různé vážící operátory, včetně vlastního vychá- zejícího z "upwind" schématu, a vhodného pro úlohy proudění. Dále je představen vlastní geometrický dělič sítě, který je vhodný pro tento typ úloh, speciálně pro úlohy s dimenzionálně komplexní geometrií.

Všechny nové přístupy prezentované v této práci jsou podrobeny numerickým experimentům na různých úlohách. Konkrétně jsou tu prezentovány výsledky slabé škálovatelnosti víceúrovňové metody BDDC pro proudění v 3D kavitě, výsledky pro test vlivu rozhraní mezi podoblastmi ve 2D a 3D na úloze zužujícího se kanálku a výsledky využívající získané poznatky pro simulaci proudění oleje uvnitř hydrostatického ložiska.

Abstract

The Ph.D. thesis is devoted to numerical simulations of the Navier-Stokes (N-S) equations for stationary incompressible flow using Balancing Domain Decomposition by Constraints (BDDC) method. The Finite Element Method (FEM) is used for the discretization of the N-S equations.

In the thesis, the multilevel BDDC method is introduced which is suitable for solving the large system of linear algebraic equations. This method and its theory are well known for the Poisson problem, for linear elasticity, and flow problems described by Stokes equations. The thesis is devoted to an extension of the multilevel BDDC method to the nonsymmetric problem arising from the discretization of the N-S equations. The weights operators for this form of the BDDC method are discussed including my own based on an upwind scheme suitable for flow problems. Next, my geometric partitioner is presented which is suitable for these problems, especially for problems with dimensionally complex geometry.

All new approaches presented in this thesis are tested numerically for different problems. Namely, weak scalability of multilevel BDDC method is tested on the flow in the 3D cavity, the test of the influence of the interface between subdomains is tested on the problem of narrowing channel, and all these pieces of knowledge are employed for simulations of oil inside the hydrostatic bearing.

Contents

Chapter 1

Introduction

1.1 Motivation

The rapid rise of the progress of computers regarding their memory and computational speed is providing a useful tool in many areas of science and technology. Especially in numerical mathematics, which is related to computational mechanics, physics, and mathematics. One of the most common methods in numerical mathematics is the finite element method (FEM), which has been developed and used since the first computers for simulating many physical processes described by partial differential equations (PDEs).

The finite element method is used and developed for many problems of mechanical engineering like structural analysis and flow of fluids. The application of FEM for problems of structural analysis (for example, linear elasticity) was there from the beginning of the development of the FEM method, and therefore, these applications are very advanced. The application of FEM for flow problems came slightly later. This was caused by the complexity of physics and the mathematical models behind these problems. Therefore, another widely spread method was developed, namely, the finite volume method (FVM).

With the growing performance of computers, the possibility of large-scale simulations with high resolution are becoming possible. As the size of the problem grows, the computational time and memory requirements grow as well. Due to the limitation of memory of a single computer as well as growing demands on the computational time, the idea of dividing the computation into several processors, started a new field of interest. This approach ignited the development of algorithms in numerical mathematics suitable for parallel computing. In general, these methods suitable for solving PDEs in parallel are called domain decomposition methods, and their development went hand in hand with the fast development of supercomputers.

Application of the FEM in computational fluid dynamics (CFD) comes with a lot of challenges. For incompressible flow, it is not an easy task to simulate flows in detail or cases with high Reynolds numbers. Computation with fine meshes and therefore with a large number of unknowns are a new way for CFD. For such problems, domain decomposition and parallelization of the algorithms are a must.

There are many publications concerning FEM for CFD. One of the first monograph is from Girault and Raviart [33]. The theory of mixed methods was also presented by Brezzi and Fortin in [15]. Here, the finite elements satisfying the Babuška-Brezzi stability condition are presented. FEM for incompressible fluids is discussed by Gresho and Sani in [36]. Elman et al. present discretization of incompressible flows by FEM together with fast iterative solvers in [26].

The main interest of this thesis is the development of the Balancing Domain Decomposi-

tion by Constraints (BDDC) method for incompressible Navier-Stokes equations, therefore, to nonsymmetric systems arising from the discretization of the equations by the finite element method. The idea of domain decomposition (DD) methods is the division of the solution domain into smaller parts called subdomains or substructures and distributing the whole computation to several problems, defined and solved on these subdomains. In the case of finite elements, this decomposition of the whole solution domain means the division into several submeshes. In this way, the methods allow massively parallel implementations.

1.2 State of the art

The topic of domain decomposition for elliptic partial differential equations is investigated by Smith, Bjorstad, and Gropp in [56]. A publication by Toselli and Widlund [58] is devoted to algorithms and theory for DD methods. Application of DD to problems of fluid mechanics is presented by Quarteroni and Valli in [52].

In domain decomposition, there are two basic types of division of the original mesh, namely, with overlapping and nonoverlapping subdomains. For overlapping subdomains, the common part of the original mesh that belongs to at least two subdomains is called the overlap which is formed by elements that belong to several subdomains. For nonoverllaping subdomains, the common part is called the interface, and it is formed only by the nodes on the boundaries of subdomains, where two or more subdomains touch each other. Both these methods allow distributing the computations to more processors and computing some parts of the simulation independently in parallel.

The idea of nonoverlapping DD methods is to form an interface problem using the elimination of interior unknowns of each subdomain. The Schur complement matrix and the right-hand side with respect to the interface are created. This matrix and right-hand side form a global interface problem which are much smaller than the original one. It can be so small that it can be solved by a direct method. Once the interface problem is solved, the interior unknowns can be found by solving a Dirichlet problem on each subdomain with a prescribed boundary value on the interface.

With larger problems, also the size of the interface problem grows and we can reach a memory limit for direct method, and the iterative methods are necessary. Thus, the condition number of the system of equations starts to be important. In [14], Brenner and Scott have shown that while the condition number of the global problem grows as $O\left(h^{-2}\right)$ with the characteristic size of an element h going to zero, the condition number of the Schur complement matrix for the interface problem grows only as $O\left(H^{-1}h^{-1}\right)$, where the characteristic size of subdomain H is much larger than h (see Brenner [12]). The next question that arises for Schur complement systems is the choice of a suitable preconditioner. These methods are referred to as iterative substructuring.

For solving Schur complement problems, a Krylov subspace method like the preconditioned conjugate gradient (PCG) method introduced by Concus, Golub, and O'Leary in [17] and biconjugate gradient stabilized (BiCGstab) method introduced by van der Vorst in [62] are suitable, because only the multiplication of a vector by Schur complement matrix is needed. This product can be computed without an explicit construction of the Schur complement, and only the Dirichlet problem on each subdomain in each iteration is solved.

There exist several scalable preconditioners for Schur complement problems with the condition number bounded independently of the number of subdomains. Two major methods introduced at end of the last century are the Finite Element Tearing and Interconnecting (FETI) method introduced by Farhat and Roux in [29], and the Balancing Domain De-

composition (BDD) method published by Mandel in [44]. In BDD, the Schur complement problem on the interface is preconditioned and the continuity of the interface unknowns is enforced strongly. Continuity at the interface in FETI method is enforced only weakly using Lagrange multipliers and the Schur complement system at the interface is converted to a dual problem of Lagrange multipliers of the same size. Both of these methods have their limitations due to the singularity of the subdomain systems and therefore new modifications of these methods have been introduced.

Modifications of these methods were introduced as Dual-Primal Finite Element Tearing and Interconnecting (FETI-DP) by Farhat et al. in [27] and as Balancing Domain Decomposition based on Constraints (BDDC) by Dohrmann in [21]. In FETI-DP, the continuity is enforced in the corners of subdomains by common values and the continuity at the rest of the interface between subdomains is enforced by Lagrange multipliers. Condition number of 2D FETI-DP method is bounded as $O\left(\log^2\left(1+H/h\right)\right)$ as shown by Mandel and Tezaur in [49]. This does not hold for the 3D problems and additional continuity constraints need to be added for fast convergence. Namely, the continuity of averages over edges and faces of subdomains. This was first shown experimentally by Farhat, Lesoinne, and Pierson in [28] and theoretically by Klawonn, Widlund, and Dryja in [39]. The bound here is $O\left(\log^2\left(1+H/h\right)\right)$.

The BDDC method was introduced by Dohrmann in [21] for the Poisson problem and linear elasticity. The underlying theory for the bound of $O\left(\log^2\left(1+H/h\right)\right)$ was presented by Mandel and Dohrmann in [46], and Mandel et al. in [47] has shown that the BDDC method is spectrally equivalent to the FETI-DP method [28]. This preconditioner was then used without an explicit coarse problem for systems with symmetric positive definite matrices and the change of basis by Li and Widlund in [43]. The multilevel extension of the BDDC method was presented first for three levels by Tu in [59], and later for an arbitrary number of levels by Mandel et al. in [48]. The multilevel BDDC method was combined with the adaptive selection of coarse unknowns and implemented into an open-source parallel solver *BDDCML* by Sousedík et al. in [57]. A recent overview of adaptive BDDC was provided by Pechstein and Dohrmann in [50]. The potential of the multilevel method to scale up to 499 thousand cores was demonstrated by Badia et al. in [7]. More on domain decomposition methods, in general, can be found in the monograph [58].

In [42], the BDDC method was first applied to a saddle-point system with symmetric indefinite matrices arising from the discretization of the Stokes problem. This approach uses finite elements with a discontinuous approximation of pressure, which leaves only unknowns for the velocity components at the interface. An approach for the Stokes problem using elements with continuous pressure was investigated by Šístek et al. in [55], and a different approach was later introduced by Li and Tu in [41].

By discretizing and linearizing the Navier-Stokes equations, we get saddle-point systems with nonsymmetric matrices. An application of the BDDC method to nonsymmetric matrices arising from advection-diffusion problems was presented by Tu and Li in [60] and [61], where the method was formulated without an explicit coarse problem. An explicit coarse problem of BDDC was presented by Yano for nonsymmetric problems arising from the Euler equations in [64].

Earlier domain decomposition methods for problems with nonsymmetric matrices include the Robin-Robin preconditioner introduced by Achdou et al. in [5, 6] for advection-diffusion problems.

An important building block of the BDDC method is the choice of weights used for averaging a discontinuous solution at the interface between subdomains. There are some standard types of weights like arithmetic average (also known as cardinality scaling), or

weighted average based on diagonal entries of subdomain matrices. A recent advanced type of scaling is the deluxe scaling presented by Dohrmann et al. in [23, 20], and by Beirão da Veiga et al. in [8].

To this point, I described only the nonoverlapping DD method, as they are the main interest of this thesis, therefore no other references to the overlapping DD method will be mentioned. For some details of overlapping DD methods, see, for example, [56] and [58].

1.3 Aims of the work

The aims of my research are two-fold:

- The first aim is to develop a numerical method for computational fluid dynamics employing the extension of the multilevel BDDC method towards the nonsymmetric systems arising from the discretization of the Navier-Stokes equations.
- The second aim is to perform missing detailed 3D simulations of the industrial problem of a flow of oil inside the whole moving hydrostatic bearings.

The problem of the bearing has been provided by Eduard Stach from the Research Center of Manufacturing Technology at the Faculty of Mechanical Engineering of the Czech Technical University in Prague.

By combining the approaches from [55] and [64], I apply the 2-level BDDC method to the Navier-Stokes equations for the lid-driven cavity benchmark problem. Next, I present an extension of the multilevel BDDC preconditioner to nonsymmetric problems and its application to linear systems obtained by Picard linearization of the Navier-Stokes equations. My computations employ the *BDDCML* solver and a parallel finite element package written in C++ described in [54], which I have extended towards the flow problems considered in my thesis. For a benchmark problem of flow in a 3-D lid-driven cavity, I compare the convergence behavior of the 2-, 3- and 4-level methods and perform weak scaling tests. In addition, I apply the 2-level variant of the BDDC preconditioner to several industrial problems of oil flow in hydrostatic bearings.

The main goal of this thesis is to develop and present the algorithm for the multilevel BDDC method for nonsymmetric systems arising from the discretization of the Navier-Stokes equations by FEM. This novel algorithm is tested on a 3D lid-driven cavity problem. Next, I test the 2-level BDDC method for this problem for different types of weight operators including my own based on the upwind scheme. Also, an experimental test of convergence for the 2- and 3-level method for cavity problem is presented. Next, I present the strategy of mesh partitioning, including my partitioner preserving 2D interfaces between each subdomain which is shown to be suitable for dimensionally complex geometries, and a mesh builder for building solution domain by individual subdomains for large meshes. Finally, I present the chronology of the simulation for the industrial problem of oil flow inside the hydrostatic bearing. Here, I gradually use all pieces of knowledge from previous numerical experiments to perform simulation for a realistic geometry of the hydrostatic bearing and also validate the calculation with an experiment.

1.4 Structure of the thesis

The thesis is organized as follows. In Chapter 2 the incompressible Navier-Stokes equations for the steady case are described. In addition, a weak formulation of these equations

and discretization by the finite element method are presented. Chapter 3 is devoted to domain decomposition methods. Basic principles and methods for nonoverlapping DD are introduced. This chapter covers the idea of discrete harmonic functions and Schur complement systems arising from iterative substructuring. Presented methods are the balancing Neumann-Neumann, two-level, and multilevel BDDC methods for symmetric positive definite systems. Condition number bounds for these methods are also presented. In Chapter 4, I present the novel approach of using the BDDC method and its multilevel variant for nonsymmetric systems arising from the Navier-Stokes equations. The rest of the chapter summarizes the used interface scaling including our new weight type, and partitioners used for mesh division. My partitioner based on geometry together with mesh builder designing the computational mesh by individual subdomains for large meshes are introduced. Chapter 5 presents all numerical results for the approaches contained in Chapter 4. This includes a comparison of weak scalability for 2-, 3-, and 4-level BDDC methods for the lid-driven cavity, the influence of the interface between subdomains and the convergence of the two-level method, and simulations of the industrial problem of oil flow inside the hydrostatic bearings. In Chapter 6 the results of the presented work and possible topics for further research are summarized.

Chapter 2

The Navier-Stokes equations and the finite element method

In this section, we introduce the mathematical model and its numerical approximation. We consider a stationary incompressible flow in a bounded three-dimensional domain Ω with Lipschitz boundary governed by the Navier-Stokes equations with zero body forces (see e.g. [26]),

$$(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \nu\Delta\boldsymbol{u} + \nabla p = \boldsymbol{f} \quad \text{in } \Omega, \tag{2.1}$$

$$\nabla \cdot \boldsymbol{u} = 0 \quad \text{in } \Omega, \tag{2.2}$$

where

- $\boldsymbol{u} = (u_1, u_2, u_3)^T [\mathrm{ms}^{-1}]$ denotes the unknown velocity vector as a function of space variable $\mathbf{x}$
- p [Pa] is an unknown pressure normalized by (constant) density as a function of $\mathbf{x}$
- ν [$\mathrm{m}^2\mathrm{s}^{-1}$] is a given kinematic viscosity
- $\boldsymbol{f}$ is a given volume force function
- Ω is a solution domain

but for complex unstructured meshes could be a problematic a definition of the overlap. In addition, we consider the following boundary conditions,

$$\boldsymbol{u} = \boldsymbol{g} \quad \text{on } \Gamma_D, \tag{2.3}$$

$$-\nu(\nabla\boldsymbol{u})\mathbf{n} + p\mathbf{n} = 0 \quad \text{on } \Gamma_N, \tag{2.4}$$

where Γ_D and Γ_N are parts of the boundary $\partial\Omega$, $\overline{\Gamma_D} \cup \overline{\Gamma_N} = \partial\Omega$, $\Gamma_D \cap \Gamma_N = \emptyset$, $\mathbf{n}$ is the outer unit normal vector of the boundary, and $\boldsymbol{g}$ is a given function.

This completed system has for low Reynolds numbers a unique solution (see [26]), except for case with prescribed velocity on the whole boundary, i.e. $\Gamma_D = \Gamma$. Then the solution of pressure is unique up to a constant.

A solution of equations (2.1)–(2.4) is called a classical solution. This puts certain demands on the differentiability of the pressure and velocity functions. In particular, a function of a velocity component has to have continuous second partial derivatives in Ω (i.e. $\boldsymbol{u} \in [C^2(\Omega)]^3$) and has to be continuous up to the boundary ($\boldsymbol{u} \in [C^0(\overline{\Omega})]^3$). Pressure has to have continuous

first partial derivatives in Ω ($p \in C^1(\Omega)$) and also has to be continuous up to the boundary ($p \in C^0(\overline{\Omega})$). In the case of non-smooth domain or non-continuity of function $(\boldsymbol{f}, \boldsymbol{g})$, which corresonds with most of the technical applications, data $\boldsymbol{f}$ and $\boldsymbol{g}$ do not have to be smooth or continuous enough to be considered as a classical solution. In these examples, an alternative description of the boundary value problem with weaker demands on solution is necessary. This approach is using the so-called *weak formulation.*

2.1 Weak formulation

To apply the numerical approximation by the finite element method, we first derive the mixed weak formulation. We use the mixed method, where we use different function spaces for test functions of velocity and pressure. We multiply equations (2.1) and (2.2) by these test functions from suitably chosen function spaces and integrate over the solution domain Ω. We get

$$\begin{aligned} \int_\Omega (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} \cdot \boldsymbol{v} \, \mathrm{d}\Omega - \nu \int_\Omega \Delta \boldsymbol{u} \cdot \boldsymbol{v} \, \mathrm{d}\Omega + \int_\Omega \nabla p \cdot \boldsymbol{v} \, \mathrm{d}\Omega &= \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \, \mathrm{d}\Omega, \qquad (2.5) \\ \int_\Omega q \nabla \cdot \boldsymbol{u} \, \mathrm{d}\Omega &= 0, \qquad (2.6) \end{aligned}$$

where

- $\boldsymbol{v}$ are velocity test functions
- q are pressure test functions

If $\boldsymbol{v}$ is smooth enough, requirement for smoothness of solution $(\boldsymbol{u}, p)$ in equation (2.5) can be lowered using the divergence theorem. We get

$$\begin{aligned} \int_\Omega (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} \cdot \boldsymbol{v} \, \mathrm{d}\Omega - \nu \int_\Gamma (\nabla \boldsymbol{u})\mathbf{n} \cdot \boldsymbol{v} \, \mathrm{d}\Gamma + \nu \int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} \, \mathrm{d}\Omega &+ \\ + \int_\Gamma p\boldsymbol{v} \cdot \mathbf{n} \, \mathrm{d}\Gamma - \int_\Omega p \nabla \cdot \boldsymbol{v} \, \mathrm{d}\Omega &= \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \, \mathrm{d}\Omega. \end{aligned}$$

The term $\nabla \boldsymbol{u} : \nabla$ is defined as $\nabla \boldsymbol{u} : \nabla := \nabla u_x \cdot \nabla v_x + \nabla u_y \cdot \nabla v_y + \nabla u_z \cdot \nabla v_z$.

It can be rewritten as

$$\begin{aligned} \int_\Omega (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} \cdot \boldsymbol{v} \, \mathrm{d}\Omega + \nu \int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} \, \mathrm{d}\Omega - \int_\Omega p \nabla \cdot \boldsymbol{v} \mathrm{d}\Omega &- \qquad (2.7) \\ - \int_\Gamma \left(\nu \frac{\partial \boldsymbol{u}}{\partial \, \mathbf{n}} - p\mathbf{n} \right) \cdot \boldsymbol{v} \, \mathrm{d}\Gamma &= \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \, \mathrm{d}\Omega. \end{aligned}$$

In this equation, we can see that the integral over the boundary corresponds with the boundary condition on the part of the boundary with Neumann boundary condition and therefore is equal to zero.

Equations (2.7) and (2.6) with inserted boundary condition (2.3) and (2.4) and assuming $\boldsymbol{f} = \mathbf{0}$ gives

$$\begin{aligned}\int_\Omega (\mathbf{u}\cdot\nabla)\mathbf{u}\cdot\mathbf{v}\ \mathrm{d}\Omega + \nu\int_\Omega \nabla\mathbf{u}:\nabla\mathbf{v}\ \mathrm{d}\Omega - \int_\Omega p\nabla\cdot\mathbf{v}\ \mathrm{d}\Omega \ &- \\ -\int_{\Gamma_D}\left(\nu\frac{\partial\mathbf{u}}{\partial\mathbf{n}} - p\mathbf{n}\right)\cdot\mathbf{v}\ \mathrm{d}\Gamma \ &= \ \mathbf{0}, \end{aligned} \tag{2.8}$$

$$\int_\Omega q\nabla\cdot\mathbf{u}\ \mathrm{d}\Omega = 0, \tag{2.9}$$

for all suitable functions $\mathbf{v}, q$.

Now we look on the suitable function spaces for the solution $(\boldsymbol{u}, p)$ and for the test functions $\boldsymbol{v}$, q, therefore for spaces, where integrals in equations (2.8) and (2.15) are finite.

First we look at the solution and test functions for velocity. In particular, at the integral

$$\int_\Omega \nabla\boldsymbol{u}:\nabla\boldsymbol{v}\ \mathrm{d}\Omega. \tag{2.10}$$

Here we use the function space of Lebesque measurable functions

$$L^2(\Omega) := \left\{\boldsymbol{u}:\Omega\to\mathbb{R}^3\ \middle|\ \int_\Omega \boldsymbol{u}^2 < \infty\right\}$$

with the norm

$$\|\boldsymbol{u}\| := \left(\int_\Omega \boldsymbol{u}^2\right)^{1/2}.$$

Integral (2.10) is finite for the first derivatives of all velocity components from $L^2(\Omega)$. In particular for a 3-D problem, $\frac{\partial u_x}{\partial x}, \frac{\partial u_x}{\partial y}, \frac{\partial u_x}{\partial z}, \frac{\partial u_y}{\partial x}, \frac{\partial u_y}{\partial y}, \frac{\partial u_y}{\partial z}, \frac{\partial u_z}{\partial x}, \frac{\partial u_z}{\partial y}, \frac{\partial u_z}{\partial z} \in L^2(\Omega)$ and $\frac{\partial v_x}{\partial x}, \frac{\partial v_x}{\partial y}, \frac{\partial v_x}{\partial z}, \frac{\partial v_y}{\partial x}, \frac{\partial v_y}{\partial y}, \frac{\partial v_y}{\partial z}, \frac{\partial v_z}{\partial x}, \frac{\partial v_z}{\partial y}, \frac{\partial v_z}{\partial z} \in L^2(\Omega)$. By using the Cauchy-Schwarz inequality, we get

$$\begin{aligned}\int_\Omega \nabla\boldsymbol{u}:\nabla\boldsymbol{v}\ \mathrm{d}\Omega \ &= \ \int_\Omega (\nabla u_x\cdot\nabla v_x + \nabla u_y\cdot\nabla v_y + \nabla u_z\cdot\nabla v_z)\,\mathrm{d}\Omega = \\ &= \ \int_\Omega\left(\frac{\partial u_x}{\partial x}\frac{\partial v_x}{\partial x} + \frac{\partial u_x}{\partial y}\frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial z}\frac{\partial v_x}{\partial z} + \frac{\partial u_y}{\partial x}\frac{\partial v_y}{\partial x} + \right. \\ &+ \ \left.\frac{\partial u_y}{\partial y}\frac{\partial v_y}{\partial y} + \frac{\partial u_y}{\partial z}\frac{\partial v_y}{\partial z} + \frac{\partial u_z}{\partial x}\frac{\partial v_z}{\partial x} + \frac{\partial u_z}{\partial y}\frac{\partial v_z}{\partial y} + \frac{\partial u_z}{\partial z}\frac{\partial v_z}{\partial z}\right)\mathrm{d}\Omega \le \\ &\le \ \left\|\frac{\partial u_x}{\partial x}\right\|\left\|\frac{\partial v_x}{\partial x}\right\| + \left\|\frac{\partial u_x}{\partial y}\right\|\left\|\frac{\partial v_x}{\partial y}\right\| + \left\|\frac{\partial u_x}{\partial z}\right\|\left\|\frac{\partial v_x}{\partial z}\right\| + \left\|\frac{\partial u_y}{\partial x}\right\|\left\|\frac{\partial v_y}{\partial x}\right\| + \\ &+ \ \left\|\frac{\partial u_y}{\partial y}\right\|\left\|\frac{\partial v_y}{\partial y}\right\| + \left\|\frac{\partial u_y}{\partial z}\right\|\left\|\frac{\partial v_y}{\partial z}\right\| + \left\|\frac{\partial u_z}{\partial x}\right\|\left\|\frac{\partial v_z}{\partial x}\right\| + \left\|\frac{\partial u_z}{\partial y}\right\|\left\|\frac{\partial v_z}{\partial y}\right\| + \\ &+ \ \left\|\frac{\partial u_z}{\partial z}\right\|\left\|\frac{\partial v_z}{\partial z}\right\| < \infty. \end{aligned}$$

Therefore for $\Omega\subset\mathbb{R}^3$, the Sobolev space $[H^1(\Omega)]^3$, defined as

$$[H^1(\Omega)]^3 := \left\{\mathbf{u}:\Omega\to\mathbb{R}^3\ \middle|\ u_i, \frac{\partial u_i}{\partial x_j}\in L^2(\Omega);\ i,j=1,2,3\right\},$$

is a suitable space for $\boldsymbol{u}$ and $\boldsymbol{v}$.

Analogously, it can be shown that the integral

$$\int_\Omega (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} \cdot \boldsymbol{v} \, \mathrm{d}\Omega \tag{2.11}$$

is finite for $\boldsymbol{u}, \boldsymbol{v} \in [H^1(\Omega)]^3$ (see [26]).

In the case of the integral on the boundary with Dirichlet boundary condition Γ_D, we look at the behaviour of the functions $\boldsymbol{u}$ and $\boldsymbol{v}$. Meanwhile for function $\boldsymbol{u}$ we have $\boldsymbol{u} = \boldsymbol{g}$ on Γ_D, for the test function we choose $\boldsymbol{v} = \boldsymbol{0}$ on Γ_D. It result in omiting boundary formula in equation (2.8) and we get the following function spaces

$$V_g := \left\{ \boldsymbol{u} \in [H^1(\Omega)]^3 \,|\boldsymbol{u} = \boldsymbol{g} \text{ on } \Gamma_D \right\}, \tag{2.12}$$

$$V := \left\{ \boldsymbol{v} \in [H^1(\Omega)]^3 \,|\boldsymbol{v} = \boldsymbol{0} \text{ on } \Gamma_D \right\}. \tag{2.13}$$

Here, the Dirichlet boundary condition is fulfilled in the sense of traces (more for example in [13]).

Now we look at the solution and test functions for the pressure. Because on the right-hand side of equation (2.8) there is no derivative of the pressure, the space $L^2(\Omega)$ is suitable for pressure p and choice of test functions q from $L^2(\Omega)$ secures that the integral on the left-hand side of equation (2.15) is finite.

If we put all of this together, we get the final formulation:

Find $\boldsymbol{u} \in V_g$ *and* $p \in L^2(\Omega)$ *satisfying*

$$\int_\Omega (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} \cdot \boldsymbol{v} \, \mathrm{d}\Omega + \nu \int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} \, \mathrm{d}\Omega - \int_\Omega p \nabla \cdot \boldsymbol{v} \, \mathrm{d}\Omega = 0 \quad \forall \boldsymbol{v} \in V, \tag{2.14}$$

$$\int_\Omega q \nabla \cdot \boldsymbol{u} \, \mathrm{d}\Omega = 0 \quad \forall q \in L^2(\Omega), \tag{2.15}$$

where the function spaces V_g and V are spaces from (2.12) and (2.13), respectively.

2.2 Finite element method

The first step for using the finite element method is creating a mesh. We divide the solution domain $\Omega \subset \mathbb{R}^3$ into N hexahedrons H_K, such that

$$\begin{aligned} \bigcup_{K=1}^{N} \overline{H}_K &= \overline{\Omega}, \\ \mu_{\mathbb{R}^3}(H_K \cap H_L) &= 0, \; K \neq L, \end{aligned}$$

so the neighboring elements have always a common face, edge or vertex.

Now we assume that on each element we approximate solutions of the velocity and the pressure and the corresponding test functions by polynomials of a certain degree. For solving the Navier-Stokes equations, it is suitable to choose polynomials of different degree for velocity and pressure. For the same order of approximation, pressure shows instability. I. Babuška and F. Brezzi have shown the condition (more details can be found for example in

[15]) which limits suitable pairs of polynomial degrees for the approximation of the velocity and the pressure.

The desired combinations of polynomial degrees are assured by suitable choice of finite elements. There are several types of finite elements that satisfy the Babuška-Brezzi (also known as *inf-sup*) condition. For my computations, I choose the hexahedral Taylor-Hood Q_2-Q_1 finite elements (see e.g. [26]). These elements locally approximate pressure functions by polynomials of the first degree and velocity components by polynomials of the second degree, while both fields are continuous on the inter-element boundaries.

The convergence theory of the $Q_2 - Q_1$ finite elements is well-established for the Stokes problem, see e.g. [10, 26]. The three-dimensional version of the elements satisfies the Babuška-Brezzi stability condition, one of the requirements for developing the optimal error estimate. Given a sufficient regularity of the solution $(\boldsymbol{u}, p)$ and a bounded aspect ratio of elements, the $Q_2 - Q_1$ elements satisfy the apriori error estimate [26, Theorem 5.6]

$$\|\nabla(\boldsymbol{u} - \boldsymbol{u}_h)\| + \|p - p_h\|_{0,\Omega} \leq Ch^2(\|D^3\boldsymbol{u}\| + \|D^2 p\|), \tag{2.16}$$

where $\|D^3\boldsymbol{u}\|$ and $\|D^2 p\|$ measure the regularity of the solution, h is the length of the longest edge in the mesh, $\|\cdot\|$ is the L^2-norm, and $\|\cdot\|_{0,\Omega}$ is the quotient space norm which removes the mean value of the pressure.

2.2.1 Finite element approximation

Due to their use in this thesis, we look closer at the Taylor-Hood finite elements. On each element we approximate values of velocity in the verticies, in the middles of edges, in the middles of faces and in middles of each element. Pressure degrees of freedom are only in vertices (see Figure 2.1). It correspond with the following approximation on each element H_K:

$$\begin{aligned} u_i &\in Q_2(H_K),\ i = 1, 2, 3, \text{ the so called tri-quadratic polynomial,} \\ p &\in Q_1(H_K), \text{ the so called tri-linear polynomial.} \end{aligned}$$

For an approximation by the finite element method, we assign to each node a shape function of the required degree which is equal to one in that node and equal to zero in all others. Linear combination of these shape functions approximate the solution on the element. These shape functions in local coordinates system $[\xi; \eta; \zeta]$ follows these equations:

Shape functions for approximation of each velocity component are

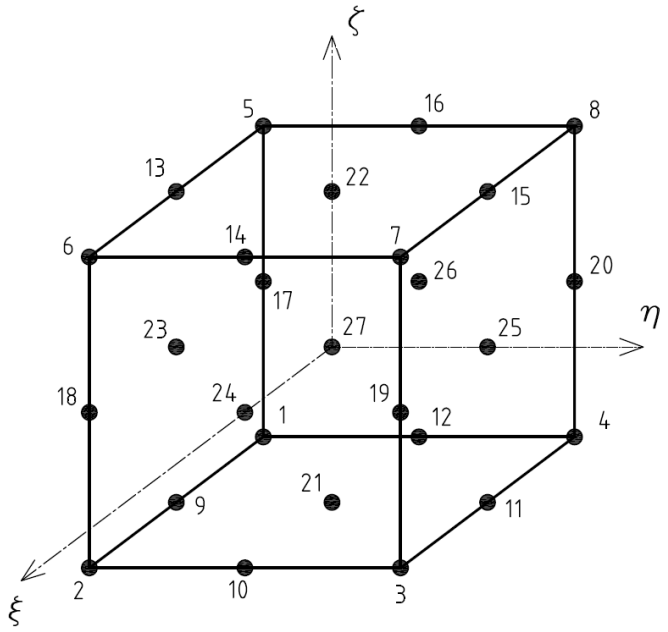

Figure 2.1: Taylor-Hood reference element

$$
\begin{aligned}
N_1 &= -1/8 \cdot \xi(1-\xi)\eta(1-\eta)\zeta(1-\zeta) \\
N_2 &= 1/8 \cdot \xi(1+\xi)\eta(1-\eta)\zeta(1-\zeta) \\
N_3 &= -1/8 \cdot \xi(1+\xi)\eta(1+\eta)\zeta(1-\zeta) \\
N_4 &= 1/8 \cdot \xi(1-\xi)\eta(1+\eta)\zeta(1-\zeta) \\
N_5 &= 1/8 \cdot \xi(1-\xi)\eta(1-\eta)\zeta(1+\zeta) \\
N_6 &= -1/8 \cdot \xi(1+\xi)\eta(1-\eta)\zeta(1+\zeta) \\
N_7 &= 1/8 \cdot \xi(1+\xi)\eta(1+\eta)\zeta(1+\zeta) \\
N_8 &= -1/8 \cdot \xi(1-\xi)\eta(1+\eta)\zeta(1+\zeta) \\
N_9 &= 1/4 \cdot (1-\xi)(1+\xi)\eta(1-\eta)\zeta(1-\zeta) \\
N_{10} &= -1/4 \cdot \xi(1+\xi)(1-\eta)(1+\eta)\zeta(1-\zeta) \\
N_{11} &= -1/4 \cdot (1-\xi)(1+\xi)\eta(1+\eta)\zeta(1-\zeta) \\
N_{12} &= 1/4 \cdot \xi(1+\xi)(1-\eta)(1+\eta)\zeta(1-\zeta) \\
N_{13} &= -1/4 \cdot (1-\xi)(1+\xi)\eta(1-\eta)\zeta(1+\zeta) \\
N_{14} &= 1/4 \cdot \xi(1+\xi)(1-\eta)(1+\eta)\zeta(1+\zeta) \\
N_{15} &= 1/4 \cdot (1-\xi)(1+\xi)\eta(1+\eta)\zeta(1+\zeta) \\
N_{16} &= -1/4 \cdot \xi(1+\xi)(1-\eta)(1+\eta)\zeta(1+\zeta) \\
N_{17} &= 1/4 \cdot \xi(1-\xi)\eta(1-\eta)(1-\zeta)(1+\zeta) \\
N_{18} &= -1/4 \cdot \xi(1+\xi)\eta(1-\eta)(1-\zeta)(1+\zeta) \\
N_{19} &= 1/4 \cdot \xi(1+\xi)\eta(1+\eta)(1-\zeta)(1+\zeta) \\
N_{20} &= -1/4 \cdot \xi(1-\xi)\eta(1+\eta)(1-\zeta)(1+\zeta) \\
N_{21} &= -1/2 \cdot (1-\xi)(1+\xi)(1-\eta)(1+\eta)\zeta(1-\zeta) \\
N_{22} &= 1/2 \cdot (1-\xi)(1+\xi)(1-\eta)(1+\eta)\zeta(1+\zeta) \\
N_{23} &= -1/2 \cdot (1-\xi)(1+\xi)\eta(1-\eta)(1-\zeta)(1+\zeta)
\end{aligned}
$$

$$
\begin{aligned}
N_{24} &= 1/2 \cdot \xi(1+\xi)(1-\eta)(1+\eta)(1-\zeta)(1+\zeta) \\
N_{25} &= 1/2 \cdot (1-\xi)(1+\xi)\eta(1+\eta)(1-\zeta)(1+\zeta) \\
N_{26} &= -1/2 \cdot \xi(1-\xi)(1-\eta)(1+\eta)(1-\zeta)(1+\zeta) \\
N_{27} &= (1-\xi)(1+\xi)(1-\eta)(1+\eta)(1-\zeta)(1+\zeta)
\end{aligned}
$$

Shape functions for approximation of pressure are

$$
\begin{aligned}
M_1 &= 1/8 \cdot (1-\xi)(1-\eta)(1-\zeta) \\
M_2 &= 1/8 \cdot (1+\xi)(1-\eta)(1-\zeta) \\
M_3 &= 1/8 \cdot (1+\xi)(1+\eta)(1-\zeta) \\
M_4 &= 1/8 \cdot (1-\xi)(1+\eta)(1-\zeta) \\
M_5 &= 1/8 \cdot (1-\xi)(1-\eta)(1+\zeta) \\
M_6 &= 1/8 \cdot (1+\xi)(1-\eta)(1+\zeta) \\
M_7 &= 1/8 \cdot (1+\xi)(1+\eta)(1+\zeta) \\
M_8 &= 1/8 \cdot (1-\xi)(1+\eta)(1+\zeta)
\end{aligned}
$$

On each element, the values of velocities and pressure are given as

$$
\begin{aligned}
u_x &= \sum_{i=1}^{N_u} u_{xi} N_i, \\
u_y &= \sum_{i=1}^{N_u} u_{yi} N_i, \\
u_z &= \sum_{i=1}^{N_u} u_{zi} N_i, \\
p &= \sum_{i=1}^{N_p} p_i M_i,
\end{aligned}
$$

where

- u_x denotes velocity component in x direction
- u_y denotes velocity component in y direction
- u_z denotes velocity component in z direction
- p denotes pressure
- u_{xi}, u_{yi}, u_{zi} are values of velocity u_x, u_y, u_z in node i
- p_i is the value of pressure in node i
- N_u is the number of nodes with velocity value on a given element (for Taylor-Hood element $N_u = 27$)
- N_p is the number of nodes with pressure value on a given element (for Taylor-Hood element $N_p = 8$)

2.2.2 Discretization of the weak formulation

To be able to seek the aproximate solution, we need also to discretize the weak formulation. During discretization of the weak formulation of the Navier-Stokes equations (2.14)–(2.15) we assume that we approximate $\boldsymbol{u}, p, \boldsymbol{v}, q$ using the finite dimensional subspaces of solutions and test function spaces. Especially $V_h \subset V$, $V_{gh} \subset V_g$ and $M_h \subset L^2(\Omega)$. Meanwhile, $\dim V_h < \infty$, $V_{gh} < \infty$, and $M_h < \infty$. First we look at the solution and test functions of velocity. Assume that $V_h \subset V$ a $V_{gh} \subset V_g$ are finite $3n$-dimensional spaces of functions for which $\{\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_{3n}\}$ are suitable vector basis functions. Next due to respecting the Dirichlet boundary condition in the definition of space V_g, we extend this set of basis functions by defining other functions $\boldsymbol{\phi}_{3n+1}, \boldsymbol{\phi}_{3n+2}, \ldots, \boldsymbol{\phi}_{3(n+\partial n)}$ so that the function $\sum_{j=3n+1}^{3n+\partial n} u_{xj}\phi_{xj}$ interpolates g_x on Γ_D, $\sum_{j=3n+\partial n+1}^{3n+2\partial n} u_{yj}\phi_{yj}$ interpolates g_y on Γ_D, and $\sum_{j=3n+2\partial n+1}^{3n+3\partial n} u_{zj}\phi_{zj}$ interpolates g_z on Γ_D. Finite element approximation $\boldsymbol{u}_h \in V_{gh}$ is therefore in each component uniquely connected with vectors of real coefficients, so that

$$\begin{aligned} u_{xh} &= \sum_{j=1}^{n} u_{xj}\phi_{xj} + \sum_{j=3n+1}^{3n+\partial n} u_{xj}N_{xj}, \\ u_{yh} &= \sum_{j=n+1}^{2n} u_{yj}\phi_{yj} + \sum_{j=3n+\partial n+1}^{3n+2\partial n} u_{yj}N_{yj}, \\ u_{zh} &= \sum_{j=2n+1}^{3n} u_{zj}\phi_{zj} + \sum_{j=3n+2\partial n+1}^{3n+3\partial n} u_{zj}N_{zj}, \end{aligned}$$

where n is equal to the number of nodes corresponding to velocity and $\phi_{xj}, \phi_{yj}, \phi_{zj}$ are basis functions for corresponding velocity components which we get by summing the all shape functions of Taylor-Hood elements for velocity corresponding to a given node in the mesh. To simplify the notation and unify vectors of velocity components, we put together all velocity components to one vector consecutively so that

$$\boldsymbol{u}_h = \boldsymbol{u}_{h1} + \boldsymbol{u}_{h2} = \sum_{j=1}^{3n} u_j\boldsymbol{\phi}_j + \sum_{j=3n+1}^{3(n+\partial n)} u_j\boldsymbol{\phi}_j, \tag{2.17}$$

therefore the first third of the vector corresponds to u_x values with vector basis functions $\boldsymbol{\phi}_j = (\phi_j, 0, 0)^T$, the second corresponds to u_y values with vector basis functions $\boldsymbol{\phi}_j = (0, \phi_j, 0)^T$, and the third corresponds to u_z values with vector basis functions $\boldsymbol{\phi}_j = (0, 0, \phi_j)^T$. During the discretization of solution and test functions of pressure, we assume that $M_h \subset L^2(\Omega)$ has scalar basis functions $\psi_1, \psi_2, \ldots, \psi_m$ and similarly as in the velocity case is the finite element approximation of pressure $p_h \in M_h$ is uniquely connected with vector of real coefficients $\mathbf{p} = (p_1, p_2, \ldots, p_m)^T$ by

$$p_h = \sum_{k=1}^{m} p_k\psi_k, \tag{2.18}$$

where m is equal to the number of nodes corresponding to the pressure, and ψ_k are basis functions of pressure which arise similary as basis functions of velocity, therefore by summing

shape functions of pressure of Taylor-Hood finite elements for a given node in the mesh. With regard to assuming the different basis functions for velocity and pressure, we call this approach the *mixed approximation* [26].

The result of the mixed approximation method is the discretized weak formulation, which we get by substituting (2.17) and (2.18) into the weak formulation (2.14)–(2.15). Therefore, we seek $\mathbf{u}_h \in V_{gh}$ and $p_h \in M_h$ satisfying

$$\int_\Omega (\boldsymbol{u}_h \cdot \nabla)\boldsymbol{u}_h \cdot \boldsymbol{v}_h \, \mathrm{d}\Omega + \nu \int_\Omega \nabla \boldsymbol{u}_h : \nabla \boldsymbol{v}_h \, \mathrm{d}\Omega - \int_\Omega p_h \nabla \cdot \boldsymbol{v}_h \, \mathrm{d}\Omega = \mathbf{0} \quad \forall \mathbf{v}_h \in V_h, \tag{2.19}$$

$$\int_\Omega q_h \nabla \cdot \boldsymbol{u}_h \, \mathrm{d}\Omega = 0 \quad \forall q_h \in M_h. \tag{2.20}$$

If equation (2.19) should be satisfied for all $\mathbf{v}_h \in V_h$ and equation (2.20) for all $q_h \in M_h$, it has to be satisfied even if we choose for test functions of velocity and pressure directly the basis from the given spaces. From (2.17) and (2.18) then it follows that equations (2.19) and (2.20) are equivalent to seeking u_j, p_j such that

$$\begin{aligned}
\int_\Omega (\boldsymbol{u}_{h1} \cdot \nabla) \left(\sum_{j=1}^{3n} u_j \boldsymbol{\phi}_j \right) \cdot \boldsymbol{\phi}_i \, \mathrm{d}\Omega &+ \nu \int_\Omega \nabla \left(\sum_{j=1}^{3n} u_j \boldsymbol{\phi}_j \right) : \nabla \boldsymbol{\phi}_i \, \mathrm{d}\Omega - \int_\Omega \left(\sum_{k=1}^{m} p_k \psi_k \right) \nabla \cdot \boldsymbol{\phi}_i \, \mathrm{d}\Omega \\
&= -\int_\Omega (\boldsymbol{u}_{h2} \cdot \nabla) \left(\sum_{j=3n+1}^{3(n+\partial n)} u_j \boldsymbol{\phi}_j \right) \cdot \boldsymbol{\phi}_i \, \mathrm{d}\Omega - \nu \int_\Omega \nabla \left(\sum_{j=3n+1}^{3(n+\partial n)} u_j \boldsymbol{\phi}_j \right) : \nabla \boldsymbol{\phi}_i \, \mathrm{d}\Omega \\
\int_\Omega \psi_l \nabla \cdot \left(\sum_{j=1}^{3n} u_j \boldsymbol{\phi}_j \right) \mathrm{d}\Omega &= -\int_\Omega \psi_l \nabla \cdot \left(\sum_{j=3n+1}^{3(n+\partial n)} u_j \boldsymbol{\phi}_j \right) \mathrm{d}\Omega,
\end{aligned}$$

which holds for all ϕ_i where $i = 1, \ldots, 3n$ and ψ_l where $l = 1, \ldots, m$. Since u_j and p_j are constant, we can put the sum in front of integral and get

$$\begin{aligned}
\nu \sum_{j=1}^{3n} u_j \int_\Omega \nabla \boldsymbol{\phi}_j : \nabla \boldsymbol{\phi}_i \, \mathrm{d}\Omega &+ \sum_{j=1}^{3n} u_j \int_\Omega (\boldsymbol{u}_{h1} \cdot \nabla)\boldsymbol{\phi}_j \cdot \boldsymbol{\phi}_i \, \mathrm{d}\Omega - \sum_{k=1}^{m} p_k \int_\Omega \psi_k \nabla \cdot \boldsymbol{\phi}_i \, \mathrm{d}\Omega \qquad (2.21) \\
&= -\sum_{j=3n+1}^{3(n+\partial n)} u_j \int_\Omega (\boldsymbol{u}_{h2} \cdot \nabla)\boldsymbol{\phi}_j \cdot \boldsymbol{\phi}_i \, \mathrm{d}\Omega - \nu \sum_{j=3n+1}^{3(n+\partial n)} u_j \int_\Omega \nabla \boldsymbol{\phi}_j : \nabla \boldsymbol{\phi}_i \, \mathrm{d}\Omega \\
\sum_{j=1}^{3n} u_j \int_\Omega \psi_l \nabla \cdot \boldsymbol{\phi}_j \, \mathrm{d}\Omega &= -\sum_{j=3n+1}^{3(n+\partial n)} u_j \int_\Omega \psi_l \nabla \cdot \boldsymbol{\phi}_j \, \mathrm{d}\Omega, \qquad (2.22)
\end{aligned}$$

for $i = 1, \ldots, 3n$ and $l = 1, \ldots, m$. It can be rewritten to the matrix form

$$\begin{bmatrix} \nu\mathbf{A} + \mathbf{N}(\mathbf{u}) & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}, \tag{2.23}$$

where $\mathbf{u}$ is the vector of unknown coefficients of velocity, $\mathbf{p}$ is the vector of unknown coefficients of pressure, $\mathbf{A}$, which correspons with the first term in 2.21, is the matrix of diffusion, $\mathbf{N}(\mathbf{u})$, which correspons with the second term in 2.21, is the matrix of advection which depends on the solution, B, which correspons with the third term in 2.21 and first term in 2.22, is the matrix from the continuity equation, and $\mathbf{f}$ and $\mathbf{g}$ are discrete right-hand side vectors arising from the Dirichlet boundary conditions. Each part of system (2.23) is assembled as (see [26])

$$\mathbf{A} = [a_{ij}],\ a_{ij} = \int_\Omega \nabla \boldsymbol{\phi}_i : \nabla \boldsymbol{\phi}_j \ \mathrm{d}\Omega, \tag{2.24}$$

$$\mathbf{N}(\mathbf{u}) = [n_{ij}],\ n_{ij} = \int_\Omega (\mathbf{u}\cdot\nabla)\boldsymbol{\phi}_j \cdot \boldsymbol{\phi}_i \ \mathrm{d}\Omega, \tag{2.25}$$

$$B = [b_{lj}],\ b_{lj} = -\int_\Omega \psi_l \nabla \cdot \boldsymbol{\phi}_j \ \mathrm{d}\Omega, \tag{2.26}$$

$$\mathbf{f} = [f_i],\ f_i = -\sum_{j=3n_{\boldsymbol{u}}+1}^{3(n_{\boldsymbol{u}}+\partial n_{\boldsymbol{u}})} u_j \int_\Omega (\mathbf{u}\cdot\nabla)\boldsymbol{\phi}_j \cdot \boldsymbol{\phi}_i \ \mathrm{d}\Omega - \nu \sum_{j=3n_{\boldsymbol{u}}+1}^{3(n_{\boldsymbol{u}}+\partial n_{\boldsymbol{u}})} u_j \int_\Omega \nabla\boldsymbol{\phi}_j : \nabla\boldsymbol{\phi}_i \ \mathrm{d}\Omega, \tag{2.27}$$

$$\mathbf{g} = [g_l],\ g_l = \sum_{j=3n_{\boldsymbol{u}}+1}^{3(n_{\boldsymbol{u}}+\partial n_{\boldsymbol{u}})} u_j \int_\Omega \psi_l \nabla\cdot\boldsymbol{\phi}_j \ \mathrm{d}\Omega. \tag{2.28}$$

System (2.23) is nonlinear due to the matrix $\mathbf{N}(\mathbf{u})$, and now we look on the possibilities of its linearization.

2.2.3 Linearization of the nonlinear systems

Now we look at the possibilities of solving nonlinear systems arising from discretization of the Navier-Stokes equations. This requires nonlinear iterations and solving a linearized system at every nonlinear step. Let us look on two classical methods. First method is Newton's iteration. In this approach we first solve system of linear equations

$$\begin{bmatrix} \nu\mathbf{A} + \mathbf{N} + \mathbf{W} & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \delta\mathbf{u}^k \\ \delta\mathbf{p}^k \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}, \tag{2.29}$$

for corrections of velocity $\delta\mathbf{u}$ and pressure $\delta\mathbf{p}$ in current step. Here the matricies $\mathbf{A}$ and B are the same asi in (2.21) and (2.22) respectively. New matricies in (2.29) are the vector-convection matrix $\mathbf{N}$ and the Newton derivative matrix $\mathbf{W}$ which depend on the solution of velocity from previous step $\mathbf{u}^k$, and are defined as

$$\mathbf{N} = [n_{ij}],\ n_{ij} = \int_\Omega (\mathbf{u}^k\cdot\nabla\boldsymbol{\phi}_j)\cdot\boldsymbol{\phi}_i \ \mathrm{d}\Omega, \tag{2.30}$$

$$\mathbf{W} = [w_{ij}],\ w_{ij} = \int_\Omega (\boldsymbol{\phi}_j\cdot\nabla\mathbf{u}^k)\cdot\boldsymbol{\phi}_i \ \mathrm{d}\Omega, \tag{2.31}$$

for i and $j = 1,\dots,3n$. The right-hand side vectors $\mathbf{f}$ and $\mathbf{g}$ also depend on solution of velocity $\mathbf{u}^k$ and $\mathbf{p}^k$ from precious step. They are defined as

$$\mathbf{f} = [f_i],\ f_i = -\int_\Omega \mathbf{u}^k\cdot\nabla\mathbf{u}^k\cdot\boldsymbol{\phi}_i \ \mathrm{d}\Omega - \nu\int_\Omega \nabla\mathbf{u}^k : \nabla\boldsymbol{\phi}_i \ \mathrm{d}\Omega + \int_\Omega \mathbf{p}^k\cdot(\nabla\boldsymbol{\phi}_i) \ \mathrm{d}\Omega, \tag{2.32}$$

$$\mathbf{g} = [g_l],\ g_l = \int_\Omega \psi_l(\nabla\cdot\mathbf{u}^k) \ \mathrm{d}\Omega. \tag{2.33}$$

The system (2.29) is referred to as the *discrete Newton problem*. After solving the corrections $\delta\mathbf{u}$ and $\delta\mathbf{p}$, we get the solution in $(k+1)$th step as $\mathbf{u}^{k+1} = \mathbf{u}^k + \delta\mathbf{u}^k$ and $\mathbf{p}^{k+1} = \mathbf{p}^k + \delta\mathbf{p}^k$, respectively.

Second approach to solving nonlinear system which is used in my computations is Picard's iteration. This simple method use for linearization of matrix $\mathbf{N}$ solution of the velocity from previous step $\mathbf{u}^k$. This leads to solving a sequence of linear systems of equations in the form

$$\begin{bmatrix} \nu\mathbf{A} + \mathbf{N}(\mathbf{u}^k) & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}^{k+1} \\ \mathbf{p}^{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}, \tag{2.34}$$

where the matricies are build in the same way as in (2.21) and (2.22) respectively. For the matrix $\mathbf{N}(\mathbf{u}^k)$ we substitute a solution of velocity from the previous step to the matrix $\mathbf{N}$. This—already linear—nonsymmetric system is solved by means of iterative substructuring using the BDDC method as a preconditioner. For more details of both linearizations see [26].

Chapter 3

Domain decomposition methods

Domain decomposition methods are mathematical algorithms for solving linear and nonlinear systems of algebraic equations which arise from discretization of partial differential equations (PDEs). These methods are suitable for parallel computations and their main idea is to decompose the solved problem into several smaller problems which are easier to solve and subsequently join each of these problems into the global one. This allows us to find the parts of the solution independently on each subdomain, therefore in parallel. It is obvious that the decomposition of the solution domain into completely independent problems is not possible and some communication and exchange of information between subdomains is necessary. The main motivation for usage of these methods is their potential for effective parallelization due to the local utilization of data and the ability to solve PDEs which shows different behaviour on different parts of the solution domain.

In connection with PDEs, the term domain decomposition has three slightly different meanings (see [56]):

- In parallel computing, it often means process of data distribution from computational model among processors. In this case the domain decomposition refers to data structure and can be independent of the numerical solution methods.

- In asymptotic analysis, it means decomposition of a solution domain into parts which could be modeled using different equations. On the interface between subdomains, we have various conditions (e.g. continuity). In this case, domain decomposition determines which PDEs we solve.

- In solving or preconditioning of large linear systems of equations, domain decomposition corresponds with partitioning of a large system into smaller problems, solution of which can be used for preconditioner (or solution) of the system of equations arising from discretization of PDEs on a solution domain. In this case, domain decomposition refers only to the solution method for the algebraic system of equations.

All these three approaches could be used in one program.

In my computations, I use domain decomposition method as a preconditioner, therefore as the third option from those above-mentioned. In the remaining part of this chapter, I will focus on this meaning of domain decomposition. The main part in domain decomposition relates with selection of subproblems to ensure a fast rate of convergence of the iterative method. Therefore domain decomposition methods provide preconditioners which can be solved within Krylov subspace methods very fast. With the development of high speed computers, there is a need for efficient methods that can solve large systems on many processors. This makes it possible to handle simulations in three dimensions with high resolution.

In general, we can divide domain decomposition methods into two basic types:

- Overlapping methods – also known as Schwarz methods, where individual subdomains overlap with at least one other subdomain and therefore share some part of the original solution domain (see Figure 3.1 (left))
- Nonoverlapping methods – also known as substructuring or Schur complement methods where there is no overlap between subdomains and subdomains share only the interface between them (see Figure 3.1 (right))

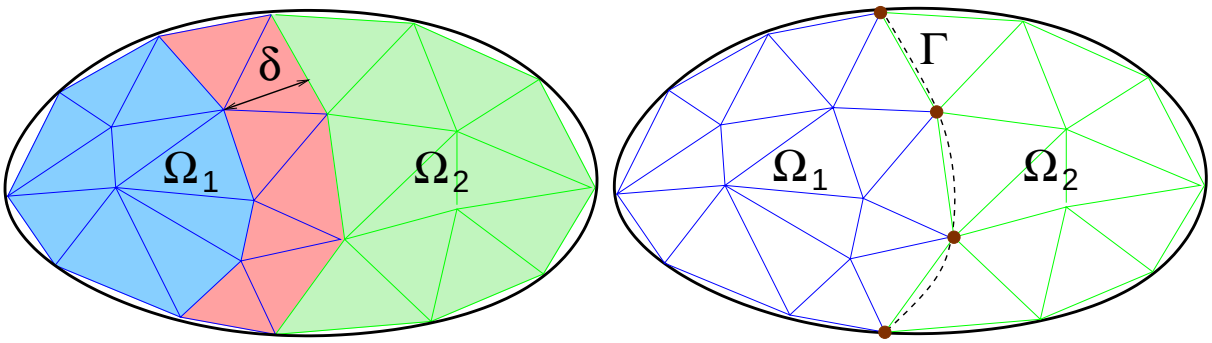

Figure 3.1: Types of domain decomposition methods: overlapping (left) and nonoverlapping (right)

For both types of domain decomposition methods, there exist several kinds of algorithms with different behaviour and suitability for usage (see [56] and [58] for more details). In my calculations, I use the Balancing Domain Decomposition by Constraints (BDDC), which is a method with nonoverlapping domain decomposition of the solution domain used as a preconditioner for solving linear system of algebraic equations. Therefore, in the next part of this chapter, I will aim at the nonoverlapping domain decomposition methods also known as *iterative substructuring*.

3.1 Iterative substructuring

As mentioned before, in my computations, I use a nonoverlapping domain decomposition method. Algorithms using this approach are referred to as iterative substructuring methods or Schur complement methods. In comparison with overlapping domain decomposition methods where the rate of convergence decreases as the overlap is reduced [56], the iterative substructuring algorithms have an additional problem to solve, namely the interface problem. There are many nonoverlapping domain decomposition methods, and I will present the basic idea behind these methods and introduce some of them more closely.

The main idea of iterative substructuring is to reduce the original system arising from the finite element method to the Schur complement system as in [58]. All these methods are based on nonoverlapping decomposition of the solution domain into large set of subdomains also referred to as substructures, as first mentioned in [51]. In general, several levels of nested subdivision can be used in these algorithms. We can see iterative substructuring as a method, in which we stop this subdividing process at some point and solve the remaining linear system by a preconditioned Krylov subspace method. In parallel computing, one or several of these substructures are assigned to one processor and the interior problems can be solved independently, hence in parallel.

In order to reduce the system to the interface (or Schur complement system) we decompose our solution domain Ω into N nonoverlapping subdomains (see Figure 3.2). This decomposition means that the degrees of freedom (dofs) shared by several subdomains are only at the interface of each subdomain while the remaining unknowns are in the interior of the subdomains. For the Taylor-Hood finite elements used in our work, both velocity and pressure unknowns are shared among subdomains and hence become part of the interface Γ.

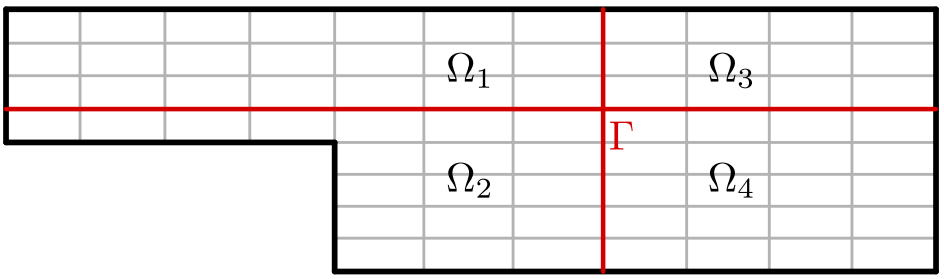

Figure 3.2: Nonoverlapping domain decomposition with interface among subdomains.

The first step in these methods is to formally reorder system (2.34) so that the interface unknowns for velocity and pressure are at the end of the corresponding parts of the vector of unknowns. This leads to the following block system

$$\begin{bmatrix} \nu\mathbf{A}_{11}+\mathbf{N}_{11} & \nu\mathbf{A}_{12}+\mathbf{N}_{12} & B_{11}^T & B_{21}^T \\ \nu\mathbf{A}_{21}+\mathbf{N}_{21} & \nu\mathbf{A}_{22}+\mathbf{N}_{22} & B_{12}^T & B_{22}^T \\ B_{11} & B_{12} & 0 & 0 \\ B_{21} & B_{22} & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u_1} \\ \mathbf{u_2} \\ \mathbf{p_1} \\ \mathbf{p_2} \end{bmatrix} = \begin{bmatrix} \mathbf{f_1} \\ \mathbf{f_2} \\ \mathbf{g_1} \\ \mathbf{g_2} \end{bmatrix}, \tag{3.1}$$

where subscript $_1$ denotes the part with the interior unknowns and subscript $_2$ denotes the part with the interface unknowns. Each matrix on the left-hand side of this system is built from the blocks corresponding to individual subdomains. The next step is to reduce this system to the interface as in [55]. We permute the system (3.1) to

$$\begin{bmatrix} \nu\mathbf{A}_{11}+\mathbf{N}_{11} & \nu\mathbf{A}_{12}+\mathbf{N}_{12} & B_{11}^T & B_{21}^T \\ B_{11} & B_{12} & 0 & 0 \\ \nu\mathbf{A}_{21}+\mathbf{N}_{21} & \nu\mathbf{A}_{22}+\mathbf{N}_{22} & B_{12}^T & B_{22}^T \\ B_{21} & B_{22} & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u_1} \\ \mathbf{p_1} \\ \mathbf{u_2} \\ \mathbf{p_2} \end{bmatrix} = \begin{bmatrix} \mathbf{f_1} \\ \mathbf{g_1} \\ \mathbf{f_2} \\ \mathbf{g_2} \end{bmatrix},$$

and then divide it into two systems

$$\begin{bmatrix} \nu\mathbf{A}_{11}+\mathbf{N}_{11} & B_{11}^T \\ B_{11} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u_1} \\ \mathbf{p_1} \end{bmatrix} + \begin{bmatrix} \nu\mathbf{A}_{12}+\mathbf{N}_{12} & B_{21}^T \\ B_{12} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u_2} \\ \mathbf{p_2} \end{bmatrix} = \begin{bmatrix} \mathbf{f_1} \\ \mathbf{g_1} \end{bmatrix}, \tag{3.2}$$

$$\begin{bmatrix} \nu\mathbf{A}_{21}+\mathbf{N}_{21} & B_{12}^T \\ B_{21} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u_1} \\ \mathbf{p_1} \end{bmatrix} + \begin{bmatrix} \nu\mathbf{A}_{22}+\mathbf{N}_{22} & B_{22}^T \\ B_{22} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u_2} \\ \mathbf{p_2} \end{bmatrix} = \begin{bmatrix} \mathbf{f_2} \\ \mathbf{g_2} \end{bmatrix}. \tag{3.3}$$

Now we express the interior unknowns from system (3.2)

$$\begin{aligned} \begin{bmatrix} \mathbf{u_1} \\ \mathbf{p_1} \end{bmatrix} = & \begin{bmatrix} \nu\mathbf{A}_{11}+\mathbf{N}_{11} & B_{11}^T \\ B_{11} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{f_1} \\ \mathbf{g_1} \end{bmatrix} \\ & - \begin{bmatrix} \nu\mathbf{A}_{11}+\mathbf{N}_{11} & B_{11}^T \\ B_{11} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nu\mathbf{A}_{12}+\mathbf{N}_{12} & B_{21}^T \\ B_{12} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u_2} \\ \mathbf{p_2} \end{bmatrix}, \end{aligned} \tag{3.4}$$

substitute into system (3.3), and we get

$$S \begin{bmatrix} \mathbf{u_2} \\ \mathbf{p_2} \end{bmatrix} = g, \tag{3.5}$$

where

$$g = \begin{bmatrix} \mathbf{f_2} \\ \mathbf{g_2} \end{bmatrix} - \begin{bmatrix} \nu\mathbf{A}_{21} + \mathbf{N}_{21} & B_{12}^T \\ B_{21} & 0 \end{bmatrix} \begin{bmatrix} \nu\mathbf{A}_{11} + \mathbf{N}_{11} & B_{11}^T \\ B_{11} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{f_1} \\ \mathbf{g_1} \end{bmatrix}$$

is the so-called reduced right-hand side and

$$S = \begin{bmatrix} \nu\mathbf{A}_{22} + \mathbf{N}_{22} & B_{22}^T \\ B_{22} & 0 \end{bmatrix} - \begin{bmatrix} \nu\mathbf{A}_{21} + \mathbf{N}_{21} & B_{12}^T \\ B_{21} & 0 \end{bmatrix} \begin{bmatrix} \nu\mathbf{A}_{11} + \mathbf{N}_{11} & B_{11}^T \\ B_{11} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nu\mathbf{A}_{12} + \mathbf{N}_{12} & B_{21}^T \\ B_{12} & 0 \end{bmatrix}$$

is the Schur complement with respect to the interface. In practice, the Schur complement matrix S is obtained by subassembling local Schur complements and often, it is not built explicitly. This operation could be very expensive and therefore only an action of the Schur complement on a vector is computed.

Many iterative substructuring algorithms used as preconditioners can solve problem (3.5). I use one step of two-level and multilevel BDDC methods and in the following part of this chapter, I will introduce this algorithm and also some of its predecessors.

All of these methods first solve the interface problem and then use this solution to calculate the interior unknowns. They also work with the space of discrete harmonic functions which is an important subspace related with Schur complement, and the values at the unknowns at the interface.

3.2 Discrete harmonic functions

The space of discrete harmonic functions is a subspace, in which we find parts of the function of solution on each subdomain. From now on, I denote the matrix of the original linear system as A. All the following theory works with this assumption:

Assumption 1. *Let the matrix A of the original linear problem arising from discretization with finite element method be symmetric and positive definite.*

In general, a function $u \in V_i$ defined on subdomain Ω_i is discrete harmonic, if

$$A_{11_i} u_{1_i} + A_{12_i} u_{2_i} = 0. \tag{3.6}$$

Here the matrix A_{11_i} is related to the subdomain part of the global linear system corresponding to interior unknowns (similarly for the matrix A_{12_i}), and vector u_{1_i} related to the local part of the global vector of unknowns corresponding to interior unknowns on each subdomain (similarly for vector u_{2_i}). The vector $u_i := \mathcal{H}_i(u_{2_i})$ is completely defined by its values on interface $\Omega_i \cap \Gamma$ and it is orthogonal, in the $a_i(\cdot,\cdot)$-inner product, to the space $V_h \cap H_0^1(\Omega_i, \partial\Omega_i \cap \Gamma)$.

Functions from $\widetilde{V}_{h_i}$ are completely defined by values at the interface Γ and by the condition (3.6). Subspace $\widetilde{V}_h \subset V_h$ consists of global piecewise discrete harmonic functions which are discrete harmonic on each subdomain. This subspace is orthogonal to the interior spaces $V_h \cap H_0^1(\Omega_i, \partial\Omega_i \cap \Gamma)$, $i = 1, \ldots, N$, where N is the number of subdomains. We denote the discrete harmonic extension of u_2 by $\mathcal{H}(u_2)$.

In Schur complement systems, the inner product defined by the Schur complement S plays an important role. Preconditioners are defined with respect to this inner product

$$s(u, v) = u_2^T S v_2. \tag{3.7}$$

In order for S to define an inner product, the original matrix A of the global as well as local problem has to be symmetric positive definite. Then $\lambda_{\min}(S) \geq \lambda_{\min}(A)$ where $\lambda_{\min}$ denotes the minimal eigenvalue. For more detail see [58]. I also recall two lemmas from [58] which justify working with the discrete harmonic extension (Lemma 1) and with norms of local discrete harmonic extensions and traces on the subdomain boundaries (Lemma 2).

Lemma 1. *Let u_{2_i} be the restriction of a finite element function to $\partial\Omega_i \cap \Gamma$. Then the discrete harmonic extension $u_i = \mathcal{H}_i(u_{2_i})$ of u_{2_i} into Ω_i satisfies*

$$a_i(u_i, u_i) = \min_{v_i | \partial\Omega_i \cap \Gamma = u_{2_i}} a_i(v_i, v_i)$$

and

$$u_{2_i}^T S_i u_{2_i} = a_i(u_i, u_i).$$

Analogously, if u_2 is the restriction of a finite element function to Γ, the piecewise discrete harmonic extension $u = \mathcal{H}(u_2)$ of u_2 into the interior of the subdomains satisfies

$$a(u, u) = \min_{v | \Gamma = u_2} a(v, v)$$

and

$$s(u, u) = u_2^T S u_2 = a(u, u).$$

Lemma 2. *Let u be discrete harmonic. Then there exist positive constants c and C, independent of h and H, such that*

$$\begin{aligned} c\|u_2\|^2_{H^{1/2}(\partial\Omega_i \cap \Gamma)} \leq \|u\|^2_{H^1(\Omega_i)} \leq C\|u_2\|^2_{H^{1/2}(\partial\Omega_i \cap \Gamma)}, \\ c|u_2|^2_{H^{1/2}(\partial\Omega_i \cap \Gamma)} \leq |u|^2_{H^1(\Omega_i)} \leq C|u_2|^2_{H^{1/2}(\partial\Omega_i \cap \Gamma)}. \end{aligned}$$

Consequently,

$$c\rho_i |u_2|^2_{H^{1/2}(\partial\Omega_i \cap \Gamma)} \leq u_{2_i}^T S_i u_{2_i} \leq C\rho_i |u_2|^2_{H^{1/2}(\partial\Omega_i \cap \Gamma)},$$

with u_{2_i} the restriction of u to $\partial\Omega_i \cap \Gamma$ and the constants independent of h, H, and ρ_i. Here h is a characteristic element size, H is the characteristic subdomain size, and ρ_i is a material constant.

3.3 Schur complement condition number

In this section, I recall an estimate of the condition number of the Schur complement S shown in [58]. According to [12], we must expect the condition number to grow at least as

H^{-2}. For the case with homogeneous Dirichlet boundary condition on the whole boundary $\partial\Omega$, we have an equivalent L^2-norm on interface Γ:

$$\|u_2\|_\Gamma^2 = \sum_{i=1}^{N} \|u\|_{L^2(\partial\Omega_i)}^2.$$

Now we can present a lemma for the estimate of the condition number of the Schur complement.

Lemma 3. *Let u be the trace of a finite element functions in V_h on Γ. Assume that $\rho_i = 1$, $i = 1, \ldots, N$, and that the fine mesh and the coarse partition are quasi uniform. Then, there exist two positive constants c and C, independent of h and H, such that*

$$cH\|u\|_\Gamma^2 \leq s(u,u) \leq Ch^{-1}\|u\|_\Gamma^2.$$

Thus,

$$\kappa(S) \leq \frac{\widetilde{C}}{Hh}.$$

Here $\widetilde{C} = C/c$. For more detail including the proof of this lemma see [58].

3.4 Balancing Neumann-Neumann method

Neumann-Neumann algorithms are among the well-known and most tested domain decomposition methods for elliptic partial differential equations (see [9, 31, 40]). They use local solvers for subdomain problems and subspace related to the entire interface. These methods provide preconditioners for Schur complement systems using flux jumps for Neumann problems and function jumps for Dirichlet problems on each subdomain and then correcting the previous iteration (similarly as in [34, 35]). The Neumann-Neumann methods were first developed without a coarse problem (see [11, 18, 19]), later they were improved by adding a second (coarse) level as in [44], (see also [25, 40, 45]). Similarly to [25], some other approaches like [46, 21, 22] concern additive methods.

In the Balancing Neumann-Neumann method, we denote trace spaces by

$$W_i = W^h(\partial\Omega_i \cap \Gamma),\ i = 1, \ldots, N,$$

where N is the number of subdomains, and by $W = \prod_{i=1}^{N} W_i$ the associated product. Having $u \in W$ and the i-th component denoted by u_i, elements of W are discontinuous across the interface. The finite element approximation is continuous across the interface Γ. We denote this subspace $\widehat{W}$. We also need to introduce the restriction $R_i : \widehat{W} \to W_i$ and interpolation operator $R_i^T : W_i \to \widehat{W}$. This global function $R_i^T u_i \in \widehat{W}$ shares values with u_i on $\partial\Omega_i \cap \Gamma$ and vanishes on the rest of the interface Γ.

As mentioned in [58], in Neumann-Neumann methods, we need to introduce a family of weighting functions $\delta_i \in W_i$ which correspond with the individual subdomains. See [24, 25, 45, 53] for more details. These functions are defined for $\gamma \in [1/2, \infty]$ by a sum of contributions from Ω_i and its neighbors,

$$\delta_i(x) = \frac{\sum_{j \in N_x} \rho_j^\gamma}{\rho_i^\gamma}, \qquad x \in \partial\Omega_i \cap \Gamma.$$

Here N_x denotes the set of indices corresponding to individual boundary of subdomain $\partial\Omega_i$ such as $x \in \partial\Omega_i$. Each node at the interface Γ belongs at least to two subdomains. The pseudoinverse $\delta_i^\times$ is defined as

$$\delta_i^\times(x) = (\delta_i(x))^{-1}, \qquad x \in \partial\Omega_i \cap \Gamma.$$

and provides a partition of unity

$$\sum_i R_i^T \delta_i^\times(x) = 1, \qquad x \in \Gamma.$$

Next we introduce the operator $E_D : W \to \widehat{W}$

$$E_D u := \sum_{i=1}^{N} R_i^T I^h(\delta_i^\times u_i), \tag{3.8}$$

where I_h is identity matrix. Therefore, $E_D u(x)$ produces an average of the value $u_i(x)$ weighted with the $\delta_i^\times$ values, and it is a continuous function. Some choices of δ_i are discussed in Section 4.3.

By means of iterative substructuring, we determine the solution on the interface, and values of any discrete harmonic function are determined from values on $\partial\Omega_i \cap \Gamma$. Having these values, we can compute the interior solution using (3.4).

The Neumann-Neumann method works with a minimal coarse space $W_0 \subset \widehat{W}$ defined as

$$W_0 = \text{span} R_{0i}^T \delta_i^\times, \quad \partial\Omega_i \cap \partial\Omega_D = \emptyset.$$

Here R_0^T is a matrix with columns corresponding with basis functions of W_0 and operator $P_0 : W \to W_0$ is a projection onto a coarse space using the exact solver. Additional functions which can improve performance of these algorithms related to other subdomains can be included as in [25].

Local space W_i is the i-th component of the space W, an element of $R_i^T W_i \subset \widehat{W}$ is a continuous piecewise discrete harmonic function defined by values at $\partial\Omega_i \cap \Gamma$. The symbol $\mathcal{H}_i(\phi u_i)$ means that first the product is mapped to the finite element space by interpolation and then extending the result to a discrete harmonic function.

Recalling the definition of the balancing Neumann-Neumann algorithm from [58] for Poisson problem, the bilinear form $\tilde{s}_i(u, v)$ for subspace W_i is defined as

$$\tilde{s}_i(u, v) := a_{\Omega_i}(\mathcal{H}_i(\delta_i u), \mathcal{H}_i(\delta_i v)) = \rho_i \int_{\Omega_i} \nabla\mathcal{H}_i(\delta_i u) \cdot \nabla\mathcal{H}_i(\delta_i v)\, dx.$$

This scaled Neumann problem defines a projection-like operator $P_i = R_i^T \widetilde{P}_i$ as

$$\tilde{s}_i(\widetilde{P}_i u, v_i) = s(u, R_i^T v_i), \qquad v_i \in W_i. \tag{3.9}$$

The right-hand side of this equation which satisfies these conditions is said to be *balanced*. For floating subdomains, we make the solution of (3.9) unique by adding constraints. For more details on the construction and matrix form of this algorithm, see [58].

3.4.1 Condition number of the Balancing Neumann-Neumann

In this section, I briefly recall the estimate for the condition number of the Schur complement preconditioned by the Balancing Neumann-Neumann method. The local solutions define elements in W_i, but do not match across the interface Γ. They are averaged using $\delta_i^\times$ and then the interface values are extended to interiors as piecewise harmonic functions. We start with $w \in W$ defined as

$$w_i = D_i^{-1} R_i P_i u = D_i^{-1} \widetilde{P}_i u = S_i^\times D_i R_i S u, \qquad u \in \widetilde{W}. \tag{3.10}$$

For w, it also stands that

$$E_D w = \sum_{i=1}^{N} P_i u. \tag{3.11}$$

Now recalling lemmas from [58] we can state the upper and lower bounds for w_i.

Lemma 4. *Let $w \in \mathrm{range}(S)$. Then, if Ω_i is a floating subdomain,*

$$\|w_i\|^2_{L^2(\partial\Omega_i)} \leq C H_i |w_i|^2_{H^{1/2}(\partial\Omega_i)},$$

with a constant that is independent of w_i, H_i, and h_i.

Lemma 5. *Let E_D be the operator defined in formula (3.8). Then,*

$$|E_D w|^2_S \leq C(1 + \log(H/h))^2 |w|^2_S, \qquad w \in \mathrm{range}(S),$$

with a constant that is independent of w_i, H_i, and h_i.

Here the norm

$$|w|^2_S = \langle w, Sw \rangle = \sum_{i=1}^{N} |w_i|^2_{S_i}, \ w \in W,$$

where $|w_i|^2_{S_i}$ provides equivalent seminorms for $w_i \in W_i$, according to Lemmas 1 and 2. More details including proofs of these lemmas can be found in [58].

It is good to note that the theory for the Neumann-Neumann method is built on the same basics as for the FETI (finite element tearing and interconnecting) methods. These methods were first introduced by Farhat and Roux in [30]. FETI algorithms are dual iterative substructuring methods with discontinuity across the interface, which use Lagrange multipliers to enforce continuity. For more information about these methods, see [58].

3.5 BDDC

In this section, I introduce the Balancing Domain Decomposition by Constraints (BDDC) method. First I recall its two-level variant for symmetric problems as first introduced in [21] for the Poisson problem and linear elasticity. The underlying theory was presented in [46] and in [47], it was shown that the BDDC method is spectrally equivalent to the FETI-DP method [28]. In [42], the BDDC method was first applied to a saddle-point system with symmetric indefinite matrix arising from the discretization of the Stokes problem. This

approach uses finite elements with discontinuous approximation of pressure which leaves only unknowns for velocity components at the interface. An approach for the Stokes problem using elements with continuous pressure was investigated in [55], and a different approach was later introduced in [41].

3.5.1 Two-level BDDC for symmetric problems

The BDDC method preconditions the residual obtained in the k-th iteration

$$r^k = g - S \begin{bmatrix} \mathbf{u_2}^k \end{bmatrix}, \tag{3.12}$$

within a Krylov subspace method generating an approximate solution to problem (3.5). This is implemented by one action of the BDDC method.

A crucial component of the BDDC algorithm is the choice of the primal variables, also called *coarse degrees of freedom*. The BDDC preconditioner then solves an approximate interface problem in the space of functions that are continuous in these primal variables.

In each iteration of the BiCGstab algorithm, two actions of the BDDC preconditioner are required. This involves solving a coarse problem and independent subdomain problems on each level. First we look at the considered coarse problem. In [60] and [61], the BDDC preconditioner was used without forming an explicit coarse problem for solving the advection-diffusion problem. Here we use an approach with an explicit coarse problem.

Nevertheless, we look at the subdomain problem first. It takes the residuum r^k and splits it into individual subdomains as

$$r_i = W_i R_i r^k, \tag{3.13}$$

where R_i is the operator restricting total residuum on given subdomain and matrix W_i (defined in Chapter 4) gives it weight. To make connection with the previous sections, we can look at the E_D operator as $E_D = \sum_{i=1}^{N} W_i R_i$. Then we solve a saddle-point problem

$$\begin{bmatrix} S_i & C_i^T \\ C_i & 0 \end{bmatrix} \begin{bmatrix} u_i \\ \lambda \end{bmatrix} = \begin{bmatrix} r_i \\ 0 \end{bmatrix} \tag{3.14}$$

on each subdomain, where λ are Lagrange multipliers, S_i is a Schur complement due the interface of i-th subdomain and C_i is a matrix of coarse degrees of freedom, which has as many rows as is the number of coarse nodes, while it has in every row zeros and ones on those positions, which in the global numbering correspond to coarse nodes. After solving this problem on each subdomain, we get the first part of an approximate solution on the interface. Now we look at the second problem, i.e., *the coarse problem*.

During the process of solving the coarse problem we have to, at first, calculate the basis functions of the coarse problem for each subdomain as the solution of

$$\begin{bmatrix} S_i & C_i^T \\ C_i & 0 \end{bmatrix} \begin{bmatrix} \Psi_i \\ \Lambda_i \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix}, \tag{3.15}$$

where Ψ_i is the matrix of coarse basis functions, in which every column corresponds to one coarse node on the subdomain. These functions are in a corresponding coarse node equal to one and in the rest are equal to zero. After solving this problem, we can determine Schur complements on each subdomain for the coarse problem as

$$S_{Ci} = \Psi_i^T S_i \Psi_i = -\Lambda_i,$$

and then assemble the global matrix of the Schur complement for coarse problem

$$S_C = \sum_{i=1}^{N} R_{Ci}^T S_{Ci} R_{Ci}, \tag{3.16}$$

where operator R_{Ci}^T is distributing individual Schur complements to the global matrix of the coarse problem.

In each action of BDDC, we calculate residuum for the coarse problem as

$$r_C = \sum_{i=1}^{N} R_{Ci}^T \Psi_i^T r_i,$$

and then solve the coarse problem

$$S_C u_C = r_C. \tag{3.17}$$

After solving this problem, we distribute the solution of the coarse problem on individual subdomains

$$u_{Ci} = \Psi_i R_{Ci} u_C,$$

and then we get the final preconditioned residual $M_{BDDC} : r_\Gamma^k \to u_\Gamma^k$ according to

$$u_\Gamma^k = \sum_{i=1}^{N} R_i^T W_i (u_i + u_{Ci}). \tag{3.18}$$

Next we look at the multilevel extension of the BDDC method for symmetric problems.

3.5.2 Multilevel BDDC for symmetric problems

The multilevel extension of the BDDC method was presented first for three dimensions in [59], and later for an arbitrary number of levels in [48]. The multilevel BDDC method was combined with the adaptive selection of coarse unknowns and implemented into an open-source parallel solver *BDDCML* in [57]. A recent overview of adaptive BDDC was provided in [50]. The potential of the multilevel method to scale up to 499 thousand cores was demonstrated in [7]. In this section, I present the main idea of the multilevel BDDC method. The detailed formulation is in Chapter 4 which aims at the multilevel BDDC for Navier-Stokes equations, therefore at nonsymmetric problems.

The main idea of multilevel BDDC stands on solving the coarse problem (3.17). Instead of solving it directly as in the two-level method, we consider subdomains as elements with coarse degrees of freedom as nodes and apply another instance of the BDDC method to it. Therefore, we build a new set of subdomains that are formed from the subdomains on a lower level. For this new set, which we call a *level*, we select the coarse nodes and compute the set of coarse basis functions. This could be done for an arbitrary number of levels and only on the last level, we build the coarse Schur complement and solve the coarse problem. After getting the coarse solution on the last level, we gradually compute the coarse solutions

Algorithm 1 Setup of the BDDC preconditioner with L levels

1: **for** level $\ell = 1, \ldots, L-1$ **do**
2: from subdomain matrix A_i^ℓ get S_i^ℓ
3: solve problems (3.15) to get Ψ_i^ℓ
4: get local coarse matrices A_{Ci}^ℓ
5: build subdomain matrices on the next level $A_j^{\ell+1}$ (3.16)
6: **end for**
7: factorize global coarse matrix A^L

on lower levels. At the end, we have a solution on the first coarse level from which we get the preconditioned residual as in the two-level method. Algorithm 1 briefly describes the setup of the multilevel BDDC method for symmetric systems.

Here ℓ stands for the level of the BDDC method. After setting up the BDDC preconditioner, we can perform its multilevel action described in Algorithm 2.

Algorithm 2 Application of the BDDC preconditioner with L levels

1: distribute residual r_i to each subdomain and build the coarse residual on the first level
2: solve subdomain problems (3.14) on the first level
3: **for** level $\ell = 2, \ldots, L-1$ **do**
4: extract local parts of residual r_i^ℓ (3.13)
5: get pre-corrected residual and solve subdomain problems on current level
6: build the coarse residual $r_C^\ell = r^{\ell+1}$
7: **end for**
8: solve coarse problem on the last level L
9: **for** level $\ell = L-1, \ldots, 2$ **do**
10: distribute coarse solution u_C^ℓ to each subdomain $u_{C_i}^\ell$ and build global interface solution u_2^ℓ
11: distribute interface solution to each subdomain $u_{i_2}^\ell$, get interior correction on each subdomain, and build approximate coarse solution $u_C^{\ell-1}$
12: **end for**
13: distribute the coarse solution on the first level to each subdomain u_{Ci} and get preconditioned residual u^k (3.18)

Once again, the detail of the multilevel method implementation for nonsymmetric systems arising from discretization of the Navier-Stokes equations will be presented in Chapter 4.

3.5.3 Algebraic view on the BDDC preconditioner

In this section, I breifly recall the algebraic theory for the BDDC method from [47] for the Poisson problem. I will also preserve the notation of the spaces and operators as in [47], so some of them might differ from those in the text above, but I will note that and make the connection between them.

The concerned problem is

$$\min_{w \in \widehat{W}} \frac{1}{2} w^T S w - w^T g, \tag{3.19}$$

where $\widehat{W} = \text{range}(R)$, S is a symmetric positive semidefinite block diagonal Schur complement matrix, g is the reduced right-hand side, R is a zero-one matrix of full column rank

such that $R_i R_i^T = I$ for I being identity matrix of appropriate dimension, and w is a vector of degrees of freedom continuous across the subdomain interfaces. We also have Schur complement on i-th subdomain after elimination of interior degrees of freedom S_i, subdomain load vector g_i, vector of the subdomain degrees of freedom w_i, and restriction operator from global degrees of freedom to subdomain degrees of freedom R_i.

The matrices are considered to be operators between given spaces such that

$$R_i : U \to W_i, \qquad R : U \to W = W_1 \times \cdots \times W_N, \tag{3.20}$$

$$S_i : W_i \to W_i, \qquad S : W \to W. \tag{3.21}$$

Here the space W_i is the space of degrees of freedom (DOFs) on the interfaces of the i-th subdomain, the space U is space of global degrees of freedom, and the space $\widehat{W}$ is the space of DOFs continuous across the subdomain interfaces.

In the BDDC method, the condition $w \in \widehat{W}$ is enforced by making $w = Ru$, where u is a vector of global DOFs. Additional constraints are imposed by means of matrix $Q_P : \mathbb{R}^{n_c} \to U$ and for the BDDC method, the coarse degrees of freedom u_C are defined as $u_C = Q_P^T u$. Also restriction operator for coarse DOFs $R_{ci} : \mathbb{R}^{n_c} \to \mathbb{R}^{n_{ci}}$ which defines $C_i = R_{ci} Q_P^T R_i^T$, $R_c = [R_{c1}, \ldots, R_{cN}]$, and C is a diagonal matrix with C_i on diagonal. We assume that R_c has full column rank. Let Φ be a block matrix with columns forming a basis for coarse degrees of freedom

$$\Phi = [\Phi_1, \ldots, \Phi_N], \qquad C\Phi = R_c. \tag{3.22}$$

By minimizing $\Phi^T S \Phi$ we define coarse basis functions Ψ which we need for the BDDC method and correspond with those described in the previous section. These coarse basis functions we get as columns of Ψ by solving

$$\begin{bmatrix} S & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \Psi \\ \Lambda \end{bmatrix} = \begin{bmatrix} 0 \\ R_c \end{bmatrix}. \tag{3.23}$$

Now we define space $\widetilde{W}$ as the subspace of W continuous across subdomain interfaces at coarse degrees of freedom as

$$\widetilde{W} = w \in W : \exists u_c : Cw = R_c u_c = w \in W : Cw = \operatorname{range}(R_c). \tag{3.24}$$

Space $\widetilde{W}_\Delta = \operatorname{null} C$ with coarse DOFs equal to zero is called the *dual space*, and the *primal space* $\widetilde{W}_\Pi = \operatorname{range}(\Phi)$. In [47] it is shown that

$$\widetilde{W} = \widetilde{W}_\Pi \bigoplus \widetilde{W}_\Delta. \tag{3.25}$$

The next assumption is that S is positive definite on $\widetilde{W}$.

Finally, we need to introduce weight matrices, here denoted as $D_{Pi} : W_i \to W_i$, and a global block diagonal matrix D_P with matrices D_{Pi} on the diagonal. These matrices form a partition of unity

$$R^T D_P R = I. \tag{3.26}$$

Averaging $w \in W$ on the interfaces between subdomains gives a global solution $u \in U$,

$$u = R^T D_P w. \tag{3.27}$$

The BDDC method works with is the assembled Schur complement $A = R^T S R$ and the preconditioner P defined as

$$Pr = R^T D_P \left(\Psi u_c + z \right), \tag{3.28}$$

where

$$\Psi^T S \Psi u_c = \Psi^T D_P^T R r, \tag{3.29}$$

and

$$Sz + C^T \nu = D_P^T R r, \qquad Cz = 0. \tag{3.30}$$

For theoretical purposes, we need to introduce another form of the preconditioner in lemma from [47].

Lemma 6. *The preconditioned operator PA satisfies, for any $u \in U$,*

$$PAu = R^T D_P w, \tag{3.31}$$

where w is defined by

$$w \in \widetilde{W}, \qquad \langle Sw, v \rangle = \langle Au, R^T D_P v \rangle \qquad \forall v \in \widetilde{W}. \tag{3.32}$$

For the BDDC preconditioner, the condition number is bounded by

$$\kappa = \frac{\lambda_{\max}(PA)}{\lambda_{\min}}(PA) \leq \omega.$$

I also recall the proof of this inequality from [47]. For the lower bound, we set in (3.32) $v = Ru$ and use the Cauchy inequality to get

$$\langle Au, u \rangle = \langle Au, R^T D_P R u \rangle = \langle Sw, Ru \rangle \leq \langle Sw, w \rangle^{1/2} \langle SRu, Ru \rangle^{1/2}. \tag{3.33}$$

Setting $v = w$ in (3.32) and using (3.31), we find that

$$\langle Sw, w \rangle = \langle Au, R^T D_P w \rangle = \langle Au, PAu \rangle. \tag{3.34}$$

From the definition of A,

$$\langle SRu, Ru \rangle = \langle R^T S R u, u \rangle = \langle Au, u \rangle. \tag{3.35}$$

By substituting (3.34) and (3.35) into (3.33) we obtain

$$\langle Au, u\rangle \leq \langle Au, PAu\rangle^{1/2}\langle Au, u\rangle^{1/2},$$

and

$$\langle Au, u\rangle \leq \langle Au, PAu\rangle,$$

which gives us lower bound $\lambda_{\min}(PA) \geq 1$.

For the upper bound we use (3.31) and (3.34) and get

$$\begin{aligned}\langle APAu, PAu\rangle &= \langle R^T SRPAu, PAu\rangle = \langle SRPAu, RPAu\rangle \\ &= \|RR^T D_P w\|_S^2 \leq \|RR^T D_P\|_S^2 \|w\|_S^2 = \|RR^T D_P\|_S^2 \langle Au, PAu\rangle,\end{aligned}$$

where $\|RR^T D_P\|_S$ is the norm of $RR^T D_P$ as an operator on $\widetilde{W}$. We have

$$\|RR^T D_P\|_S = \|I - RR^T D_P\|_S = \|B_D^T B\|_S,$$

where $B : W \to \Lambda$ is a matrix such that $Bu = 0$, and B_D^T is the generalized inverse of B such that $B_D^T B + RR^T D_P = I$. From assumption

$$\|B_D^T B\|_S^2 \leq \omega \|u\|_S^2 \qquad \forall u \in \widetilde{W}, \tag{3.36}$$

it follows that

$$\langle APAu, PAu\rangle \leq \omega \langle Au, PAu\rangle.$$

Using this and the Cauchy inequality, we get

$$\langle Au, PAu\rangle \leq \langle Au, u\rangle^{1/2}\langle APAu, PAu\rangle^{1/2} \leq \langle Au, u\rangle^{1/2}\omega\langle Au, PAu\rangle^{1/2},$$

and,

$$\langle Au, PAu\rangle \leq \omega \langle Au, u\rangle,$$

which gives us an upper bound $\lambda_{\max}(PA) \leq \omega$.

Finally, we recall an estimate of the constant ω from [49] for a 2D problem. Hence, $\kappa(PA) \leq \omega$, it can be viewed as an operator norm of the weight operator or the S-norm of the averaging operator,

$$\omega \leq C\left(1 + \log\frac{H}{h}\right)^2,$$

where H is the characteristic size of subdomains, h is the characteristic element size, and C is a constant independent of H and h. This estimate was extended to a 3D problem in [39].

3.5.4 Convergence of extended versions of BDDC

Now we look at the convergence properties of the BDDC method for some other problems. In Chapter 4, I will discus the BDDC method for Navier-Stokes equations, which falls beyond the existing theory. Therefore, I recall convergence properties of the two-level BDDC method for the Stokes problem, the advection-diffusion problem, and for the multilevel BDDC for symmetric positive definite problems.

Let us first recall the convergence properties of the bound for BDDC method applied to the Stokes problem without continuous pressure. I recall the main result for condition number bounds from [42]. The lower bound for the condition number is

$$\langle \mathbf{u}, \mathbf{u} \rangle_{\tilde{S}} \leq \langle \mathbf{u}, M^{-1}\tilde{S}\mathbf{u} \rangle_{\tilde{S}},$$

where $\tilde{S}$ is the operator of the interface problem, $\langle \mathbf{a}, \mathbf{a} \rangle_{\tilde{S}}$ denotes $\mathbf{a}^T \tilde{S} \tilde{R}_D^T \tilde{R}\mathbf{a}$ ($\tilde{R}$ is the restriction operator), and $M^{-1}\tilde{S}$ is the preconditioned operator. This shows that 1 is the bound of the minimum eigenvalue of the preconditioned operator.

For the upper bound, we have

$$\langle \mathbf{u}, M^{-1}\tilde{S}\mathbf{u} \rangle_{\tilde{S}} \leq C \frac{1}{\beta^2} \left(1 + \log \frac{H}{h}\right)^2 \langle \mathbf{u}, \mathbf{u} \rangle_{\tilde{S}},$$

where C is a constant which is independent of H and h, and β is the inf-sup stability constant. For more details, see [42].

I also recall a bound for the Stokes problem with continuous pressure approximation from [41]. The lower bound is

$$\langle Gx, x \rangle \geq c\beta^2 \langle Mx, x \rangle,$$

where $\langle \cdot, \cdot \rangle$ is an inner product, G is a Schur complement, M is a preconditioner operator, c is a constant which is independent of H and h, and β is the inf-sup stability constant.

For upper bound, we have

$$\langle Gx, x \rangle \leq \tilde{C}\Phi(H/h) \langle Mx, x \rangle,$$

where C is a constant which is independent of H and h, and $\Phi(H/h) = (1 + \log(H/h))^2$. For more detail see [41].

There are also bounds for the advection-diffusion problem, therefore a problem with a nonsymmetric matrix, in [60]. The lower bound is given by

$$\langle Tu_2, Tu_2 \rangle_{B_\Gamma} \leq C_1 \frac{\tilde{\Phi}^4(H/h)}{\mu^2 \max(\mu, C_m)} \langle u_2, u_2 \rangle_{B_\Gamma},$$

where $\langle a, b \rangle_A$ is a product $a^T Ab$ for a given matrix A and vectors a and b, B_Γ is the symmetric part of the original matrix from the finite element discretization restricted to the interface Γ, C_1 is a positive constant independent of H, h and ν, C_m is a minimum of a positive function defined in [60, Section 3], T is the preconditioned operator, and $\tilde{\Phi}(H/h) = C(1 + \log(H/h))$, where C is a positive constant.

For the upper bound, we have

$$c_0\langle u_2, u_2\rangle_{B\Gamma} \leq \frac{C_2}{\max(\mu, C_m)}\langle u_2, Tu_2\rangle_{B\Gamma},$$

where C_2 is a positive constant independent of H, h and ν.

Now I recall the convergence properties of the multilevel BDDC method for symmetric positive definite problems [59, 48].

The condition number of the Schur complement preconditioned by the multilevel BDDC, κ, is bounded as

$$\kappa \leq \prod_{i=1}^{L-1} C_i \left(1 + \log \frac{H_i}{H_{i-1}}\right)^2, \tag{3.37}$$

where H_i is the characteristic subdomain size on the i-th level, and $h = H_0$, the characteristic element size.

As suggested in [26, Section 2.1.2], the usual bound on the error reduction of the conjugate gradient (CG) method suggests that the number of CG iterations is proportional to $\sqrt{\kappa}$ for large κ and a prescribed tolerance. Although far from describing the complex behaviour of the CG method, it can still suggest that when combined with (3.37), we can expect the number of iterations proportional to

$$k \sim \prod_{i=1}^{L-1} \left(1 + \log \frac{H_i}{H_{i-1}}\right), \tag{3.38}$$

i.e. removing the power from the terms in the product. Finally, note that in the case of the two-level method, bound (3.38) simplifies to

$$k \sim \left(1 + \log \frac{H}{h}\right). \tag{3.39}$$

Neither the BDDC theory for the Stokes problem [42, 41], nor the one for the advection-diffusion problem [60], and the multilevel extension for symmetric positive definite problems [59, 48] apply to our case of the Navier-Stokes equations. Nevertheless, in Section 5.2.3, we perform numerical experiments with varying H/h to investigate whether the behaviour of the method agrees with (3.38).

In the following chapter, I present the BDDC method and its multilevel extension for the Navier-Stokes equations.

Chapter 4

BDDC algorithms for nonsymmetric systems and its building components

In this chapter, I present the novel approach of using BDDC method for nonsymmetric systems arising from discretization of the Navier-Stokes equations and its implementation details, like interface scaling and solution domain partitioners. First, I present two-level BDDC method for Navier-Stokes equations described in [2]. Next I describe its multilevel extension presented in [4]. I also desribe four used interface weights for the BDDC preconditioner asociated with matrix of weights W_i in my calculations and two types of used partitioners for decomposition of the solution domain Ω. Here one of the used weights is our own new type of weight presented in [4]. I also introduced two types of used partitioners for solution domain especially our own, which was first presented in [1]. Finally, I introduce my own mesh builder used for building solution domain for cavity problem, presented in Chapter 5, by individual subdomains.

In my computations, problem (3.5) is solved by the BiCGstab method [62] with one step of two-, three-, and four-level BDDC used as the preconditioner. Domain decomposition allows us to perform the action of the BDDC preconditioner and of the matrix S in parallel in each iteration. The solution is realised by the multilevel BDDC implementation in the *BDDCML* library[1] [57]. First, let us look at the two-level method and its known theory.

4.1 Two-level BDDC

As mentioned in Introduction, the BDDC method was first proposed in [21] for the Poisson problem and linear elasticity. The underlying theory was presented in [46] and in [47], it was shown that the BDDC method is spectrally equivalent to the FETI-DP method [28]. In [42], the BDDC method was first applied to a saddle-point system with symmetric indefinite matrix arising from the discretization of the Stokes problem. This approach uses finite elements with discontinuous approximation of pressure which leaves only unknowns for velocity components at the interface. An approach for the Stokes problem using elements with continuous pressure was investigated in [55], and a different approach was later introduced in [41].

By discretizing and linearizing an approximation of the Navier-Stokes equations, we get saddle-point systems with nonsymmetric matrices. An application of the BDDC method to nonsymmetric matrices arising from the advection-diffusion problems was presented in [60] and [61], where the method was formulated without an explicit coarse problem. An explicit

[1] `http://users.math.cas.cz/~sistek/software/bddcml.html`

coarse problem of BDDC was presented for nonsymmetric problems arising from the Euler equations in [64]. Earlier domain decomposition methods for problems with nonsymmetric matrices include also the Robin-Robin preconditioner [5, 6] for advection-diffusion problems.

In [2], I combined the approaches from [55] and [64] and applied the 2-level BDDC method to the Navier-Stokes equations for the lid-driven cavity. I apply the BDDC preconditioner to the nonsymmetric problems arising from the discretization of the Navier-Stokes equations. It works with the residual in k-th iteration

$$r^k = g - S \begin{bmatrix} \mathbf{u_2}^k \\ \mathbf{p_2}^k \end{bmatrix}, \tag{4.1}$$

obtained from the BiCGstab method generating an approximate solution to problem (3.5). This is implemented by one action of the BDDC method.

A crucial component of a BDDC algorithm is the choice of the primal variables, also called coarse degrees of freedom, on each level. The BDDC preconditioner then solves an approximate interface problem in the space of functions that are continuous in these primal variables.

First, we classify the interface into vertices, edges, and faces of subdomains. For the Taylor-Hood elements, degrees of freedom (dofs) are located at nodes. Hence, faces are defined as subsets of nodes shared by the same two subdomains, edges are subsets of nodes shared by several subdomains, and vertices are degenerate edges with only one node.

Here, the main primal unknowns are arithmetic averages over edges and faces defined independently for each component of velocity and for the pressure unknowns. In addition, pointwise continuity of these components is required at subdomain vertices, also called corners. This means that for subdomains that have in common at least a face (f), an edge (e), or a corner (c), the following functionals are equal

$$L^{e,f}(u) = \sum_j u_{ij},\ L^{e,f}(p) = \sum_j p_j, L^c(u) = u_i,\ L^c(p) = p,$$

where u_i, $i = 1, 2, 3$, are the components of velocity, and index j corresponds to the number of unknowns on the shared edge or face.

In each iteration of the BiCGstab algorithm, two actions of the BDDC preconditioner are required. This involves solving a coarse problem and independent subdomain problems on each level. First we look at the considered coarse problem. In [60] and [61], the BDDC preconditioner was used without an explicit coarse problem for solving the advection-diffusion problem. Here we use an approach with an explicit coarse problem.

First we look at the subdomain problem. It takes total residuum r^k and splits it to individual subdomains as

$$r_i = W_i R_i r^k, \tag{4.2}$$

where R_i is operator restricting total residuum on a given subdomain and matrix W_i gives it weight. Then we solve on each subdomain a saddle-point problem

$$\begin{bmatrix} S_i & C_i^T \\ C_i & 0 \end{bmatrix} \begin{bmatrix} u_i \\ \lambda \end{bmatrix} = \begin{bmatrix} r_i \\ 0 \end{bmatrix}, \tag{4.3}$$

where λ are Lagrange multipliers, S_i is the Schur complement of the interior unknowns, and C_i is the matrix of coarse degrees of freedom, which has as many rows as is the number of

coarse nodes, while it has in every row zeros and ones on that positions, which in global numbering correspond to coarse nodes. After solving this problem on each subdomain, we first get a part of an approximate solution on the interface. Now we look at the second problem, i.e., the coarse problem.

During the solution of the coarse problem, we have to, at first, calculate basis functions of the coarse problem for each subdomain as the solution of

$$\begin{bmatrix} S_i & C_i^T \\ C_i & 0 \end{bmatrix} \begin{bmatrix} \Psi_i \\ \Lambda_i \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix}. \tag{4.4}$$

Here Ψ_i is the matrix of coarse basis functions, in which every column corresponds to one coarse node on the subdomain. These functions are in the corresponding coarse node equal to one and equal to zero in the rest. For nonsymmetric problem, it is needed to compute also a set of adjoint coarse basis functions (see [64]). These we get by solving

$$\begin{bmatrix} S_i^T & C_i^T \\ C_i & 0 \end{bmatrix} \begin{bmatrix} \Psi_i^* \\ \Lambda_i \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix}, \tag{4.5}$$

where Ψ_i^* is the matrix of adjoint basis functions. By solving this problem, we can determine the contribution of the i-th subdomain to the coarse problem

$$S_{Ci} = \Psi_i^{*T} S_i \Psi_i = -\Lambda_i.$$

Then, we can assemble the global matrix of the coarse problem

$$S_C = \sum_{i=1}^{N} R_{Ci}^T S_{Ci} R_{Ci},$$

where N is a number of subdomains, and operator R_{Ci} selects local coarse unknowns from the global coarse vector.

In each action of BDDC, we calculate residuum for the coarse problem as

$$r_C = \sum_{i=1}^{N} R_{Ci}^T \Psi_i^{*T} r_i,$$

and then solve the coarse problem

$$S_C u_C = r_C. \tag{4.6}$$

After solving of problem we distribute the solution to individual subdomains by

$$u_{Ci} = \Psi_i R_{Ci} u_C,$$

and then get the final preconditioning $M_{BDDC} : r_\Gamma^k \to u_\Gamma^k$ as

$$u_\Gamma^k = \sum_{i=1}^{N} R_i^T W_i (u_i + u_{Ci}). \tag{4.7}$$

Next we look at the multilevel extension of the BDDC method.

4.2 Multilevel BDDC for nonsymmetric systems

The main focus of this section is a formulation of the multilevel BDDC preconditioner for nonsymmetric problems and its application to linear systems obtained by Picard linearization of the Navier-Stokes equations. The content of this section is based on [4], we described the multilevel BDDC preconditioner for the nonsymmetric problems arising from the discretization of the Navier-Stokes equations above. In the k-th iteration, the preconditioner is applied to the residual (3.12) obtained from the BiCGstab method generating an approximate solution to problem (3.5).

The multilevel extension of the BDDC method was presented first for three dimensions in [59], and later for an arbitrary number of levels in [48]. The multilevel BDDC method was combined with the adaptive selection of coarse unknowns and implemented into an open-source parallel solver *BDDCML* in [57]. A recent overview of adaptive BDDC was provided in [50]. The potential of the multilevel method to scale up to 499 thousand cores was demonstrated in [7].

Before applying the preconditioner, we have to set it up. First, we find the coarse basis functions independently for each subdomain on each level. For finding the subdomains on the next level, we look at the subdomains on the previous level as elements for the next level, and the coarse dofs will be considered as unknowns. Subdomains on the next level are formed by connecting several subdomains from the previous level. For example, in Fig. 3.2, we can join subdomains Ω_1 and Ω_2 to form the first subdomain and subdomains Ω_3 and Ω_4 to form the second subdomain on the next level. From the subdomain matrix on each level, we build and solve the saddle-point system

$$\begin{bmatrix} S_i^\ell & C_i^{\ell^T} \\ C_i^\ell & 0 \end{bmatrix} \begin{bmatrix} \Psi_i^\ell \\ \Lambda_i^\ell \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix}, \tag{4.8}$$

where ℓ is the level of the BDDC method, S_i^ℓ is the Schur complement with respect to the interface of the i-th subdomain (built as in 3.5), and C_i^ℓ is the matrix defining the coarse dofs, which has as many rows as is the number of coarse dofs defined at the subdomain. The solution Ψ_i^ℓ is the matrix of coarse basis functions with every column corresponding to one coarse unknown on the subdomain. These functions are equal to one in one coarse dof and they are equal to zero in the remaining local coarse unknowns.

Note the presence of the explicit Schur complement S_i^ℓ in (4.8). In BDDC, an analogous problem can be formed for the original subdomain matrix A_i^ℓ, as it was used, e.g., in the original paper [21], considering the discrete harmonic extension of the coarse basis functions to subdomain interiors. To be more specific, dividing the local unknowns into interior (subscript $_1$) and interface ones (subscript $_2$), we introduce a block structure

$$A_i^\ell = \begin{bmatrix} A_{i_{11}}^\ell & A_{i_{12}}^\ell \\ A_{i_{21}}^\ell & A_{i_{22}}^\ell \end{bmatrix},$$

from which the local Schur complement can be expressed as

$$S_i^\ell = A_{i_{22}}^\ell - A_{i_{21}}^\ell (A_{i_{11}}^\ell)^{-1} A_{i_{12}}^\ell.$$

Although we describe the algorithm using the explicit Schur complements S_i^ℓ, the computations are performed without its explicit construction, and only factorizations and back-substitutions with $A_{i_{11}}^\ell$ are required.

As introduced in [64], a set of adjoint coarse basis functions $\Psi_i^{*\ell}$ is also needed for non-symmetric problems. These are obtained as the solution to problem (4.8) with a transposed matrix,

$$\begin{bmatrix} S_i^{\ell^T} & C_i^{\ell^T} \\ C_i^{\ell} & 0 \end{bmatrix} \begin{bmatrix} \Psi_i^{*\ell} \\ \Lambda_i^{\ell^T} \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix}. \tag{4.9}$$

After solving (4.8) and (4.9), we can get, on each level, the local coarse matrix on each subdomain as

$$A_{Ci}^{\ell} = (\Psi_i^{*\ell})^T S_i^{\ell} \Psi_i^{\ell} = -\Lambda_i^{\ell}. \tag{4.10}$$

The local matrices A_{Ci}^{ℓ} resemble the element matrices from the FEM. In the multilevel method, they are assembled into subdomain matrices on the next level

$$A_j^{\ell+1} = \sum_{i=1}^{N_{S_j}} R_{Cij}^{\ell^T} A_{Ci}^{\ell} R_{Cij}^{\ell}, \tag{4.11}$$

where $R_{Cij}^{\ell} = R_j^{\ell+1} R_{Ci}^{\ell^T}$. Here $R_j^{\ell+1}$ is the restriction of the global vector of unknowns to those present on the j-th subdomain on level $\ell+1$, R_{Ci}^{ℓ} is the restriction of the global vector of coarse unknowns to those present at the i-th subdomain on level ℓ and N_{S_j} is the number of subdomains from level ℓ which form the j-th subdomain on the $(\ell+1)$-st level. On the last level, we build the global coarse problem, therefore $A_j^L = A^L$, where L is the number of levels of the BDDC method. The setup of the multilevel BDDC is described in Algorithm 3.

Algorithm 3 Setup of the BDDC preconditioner with L levels

1: **for** level $\ell = 1, \ldots, L-1$ **do**
2: from subdomain matrix A_i^{ℓ} get S_i^{ℓ}
3: solve problems (4.8) and (4.9) to get Ψ_i^{ℓ} and $\Psi_i^{*\ell}$
4: get local coarse matrices A_{Ci}^{ℓ} from (4.10)
5: build subdomain matrices on the next level $A_j^{\ell+1}$ by (4.11)
6: **end for**
7: factorize global coarse matrix A^L

After this setup, the preconditioner can be applied in each iteration. Let us first look at the 2-level method and describe the modification for the multilevel method later. At the beginning, we take the residual vector r^k and extract its local parts on each subdomain as

$$r_i = W_i R_i r^k, \tag{4.12}$$

where R_i is the operator restricting a global interface vector to the i-th subdomain, and matrix W_i applies weights to satisfy the partition of unity. The types of weights we use are described in detail in Section 4.3.

Then we get the coarse residual as

$$r_C = \sum_{i=1}^{N} R_{Ci}^T \Psi_i^{*T} r_i. \tag{4.13}$$

Note that here we use the adjoint coarse basis functions Ψ_i^{*T}. Then we solve the coarse problem

$$A_C u_C = r_C \tag{4.14}$$

and distribute the coarse solution to each subdomain, followed by prolonging it on the fine mesh (using standard coarse basis functions Ψ_i) as

$$u_{Ci} = \Psi_i R_{Ci} u_C. \tag{4.15}$$

Now we look at the subdomain problems. On each subdomain, a saddle-point system

$$\begin{bmatrix} S_i & C_i^T \\ C_i & 0 \end{bmatrix} \begin{bmatrix} u_i \\ \lambda \end{bmatrix} = \begin{bmatrix} r_i \\ 0 \end{bmatrix} \tag{4.16}$$

is solved. Here λ are Lagrange multipliers, and S_i and C_i are the same matrices as in (4.8). After solving this problem on each subdomain, we get the subdomain correction u_i.

At the end of the action of the 2-level BDDC preconditioner, we get the preconditioned residual u^k, $M_{BDDC} : r^k \to u^k$, by combining the subdomain corrections with the subdomain coarse solution as

$$u^k = \sum_{i=1}^{N} R_i^T W_i (u_i + u_{Ci}). \tag{4.17}$$

If we want to add more levels to the method, we have to look at problem (4.14) first. In the 2-level method, we solve (4.14) by a direct method, whereas in the multilevel method, we apply a step of the BDDC method to solve the coarse problem (4.14) only approximately. This means that we build residuals and matrices (in setup) corresponding to the cluster of subdomains, which will form one subdomain on the next level. First, we build global coarse residual on the first level as in the standard 2-level method

$$r_C = \sum_{i=1}^{N} R_{Ci}^T \Psi_i^{*T} r_i, \tag{4.18}$$

and solve subdomain problems on the first level (4.16).

Now starting on the second level and ending on the $(L-1)$-st level we denote

$$u^\ell = u_C^{\ell-1} \quad \text{and} \quad r^\ell = r_C^{\ell-1},$$

extract local parts of the residual on each subdomain

$$r_i^\ell = W_i^\ell R_i^\ell r^\ell, \tag{4.19}$$

and solve the following interior problem on each subdomain

$$A_{i_1 1}^\ell u_{i_1}^\ell = r_{i_1}^\ell, \tag{4.20}$$

where subscript $_1$ again corresponds to the the interior unknowns of the subdomain (subscript $_2$ will correspond to the interface unknowns of the subdomain). After solving this problem we perform the interior pre-correction as

$$\tilde{r}_{i_2}^\ell = r_{i_2}^\ell - A_{i_2 1}^\ell u_{i_1}^\ell. \tag{4.21}$$

Next, we solve subdomain problems

$$\begin{bmatrix} S_i^\ell & C_i^{\ell^T} \\ C_i^\ell & 0 \end{bmatrix} \begin{bmatrix} u_{i_2}^\ell \\ \lambda^\ell \end{bmatrix} = \begin{bmatrix} \tilde{r}_{i_2}^\ell \\ 0 \end{bmatrix} \tag{4.22}$$

and build the coarse residual

$$r_C^\ell = \sum_{i=1}^{N^\ell} R_{Ci}^{\ell T} \Psi_i^{*\ell T} \tilde{r}_{i_2}^\ell. \tag{4.23}$$

Finally, we solve the coarse problem on the last level L to get $u^L = u_C^{L-1}$.

After getting the solution on the highest level, we gradually build the approximate coarse solution on the first level. Starting on level $L-1$ and going down to level 2, we have the coarse solution distributed to each subdomain, and the subdomain solution prepared in (4.22)

$$u_{C_i}^\ell = \Psi_i^\ell R_{C_i}^\ell u_C^\ell \quad \text{and} \quad u_{i_2}^\ell. \tag{4.24}$$

Then we build the BDDC approximation of the global interface solution

$$u_2^\ell = \sum_{i=1}^{N_S^\ell} R_i^{\ell T} W_i^\ell (u_{C_i}^\ell + u_{i_2}^\ell). \tag{4.25}$$

After that, we distribute the interface solution on each subdomain

$$u_{i_2}^\ell = R_i^\ell u_2^\ell, \tag{4.26}$$

and compute the interior correction $\tilde{u}_{1_i}^\ell$ on each subdomain

$$A_{i_1 1}^\ell \tilde{u}_{i_1}^\ell = -A_{i_1 2}^\ell u_{i_2}^\ell. \tag{4.27}$$

Finally, we bring in the interior correction from (4.20) and build the approximate coarse solution as

$$u_C^{\ell-1} = u^\ell = \begin{bmatrix} \sum_{i=1}^{N_S^\ell} R_{1i}^{\ell T} (\tilde{u}_{i_1}^\ell + u_{i_1}^\ell) \\ u_2^\ell \end{bmatrix}, \tag{4.28}$$

where $R_{1i}^{\ell T}$ is the operator mapping interior unknowns from subdomain to global coarse vector on the previous level.

After this, we have the approximate coarse solution on the second level and get the solution of problem (4.17) like in the 2-level method. The process is summarized in Algorithm 4. For the 2-level method, the algorithm follows just steps 1–3, 12, 13, 21, and 22. The rest of the steps are meaningful just in the multilevel case. Another difference is that in steps 5–10 and 15–19 of the multilevel method, we work with unknowns over the whole domain, while in the 2-level method, we are restricted to the interface.

Algorithm 4 Application of the BDDC preconditioner with L levels

distribute residual to each subdomain on the first level r_i (4.12)
build the coarse residual on the first level r_C (4.13)
solve subdomain problems (4.16) on the first level
for level $\ell = 2, \ldots, L-1$ **do**
    $r^\ell = r_C^{\ell-1}$ and $u^\ell = u_C^{\ell-1}$
    extract local parts of residual r_i^ℓ (4.19)
    solve (4.20)
    get pre-corrected residual $\tilde{r}_{i_2}^\ell$ (4.21)
    solve subdomain problems (4.22)
    build the coarse residual r_C^ℓ (4.23)
end for
$r^L = r_C^{L-1}$, $u^L = u_C^{L-1}$ and solve $A^L u^L = r^L$
$u_C^{L-1} = u^L$
for level $\ell = L-1, \ldots, 2$ **do**
    distribute coarse solution u_C^ℓ to each subdomain $u_{C_i}^\ell$ (4.24)
    build global interface solution u_2^ℓ (4.25)
    distribute interface solution to each subdomain $u_{i_2}^\ell$ (4.26)
    get interior correction on each subdomain by solving (4.27)
    build approximate coarse solution $u_C^{\ell-1}$ (4.28)
end for
distribute the coarse solution on the first level to each subdomain u_{Ci} (4.15)
get preconditioned residual u^k (4.17)

4.3 Interface scaling

Let us now look at the used weights in my computations. Several types of interface weights have been developed for the BDDC preconditioner, with each of them having its advantages and disadvantages for certain kinds of problems. In this section, we describe four types of weights used in our calculations presented in Chapter 5.

Before we start with the description of the individual scalings, we recall that the main requirement on the matrix of weights W_i is that it forms a partition of unity [58],

$$\sum_{i=1}^{N} R_i^T W_i R_i = I, \tag{4.29}$$

where I is the identity matrix.

An important class of the interface averaging operators is represented by diagonal matrices

$$W_i = \begin{pmatrix} w_i^1 & & \\ & w_i^2 & \\ & & \ddots \end{pmatrix}, \tag{4.30}$$

where w_i^k denotes the weight for the k-th (with respect to the subdomain interface) dof on the i-th subdomain. In this case, the requirements are fulfilled by the following simple construction: First, every subdomain generates a nonnegative weight $\widetilde{w}_i^k$. These values are then shared with all neighbouring subdomains, and the normalized weight w_i^k satisfying the

partition of unity is obtained by dividing the local weight with the sum of contributions from all neighbours,

$$w_i^k = \frac{\widetilde{w}_i^k}{\sum_{j=1}^{N_S} \widetilde{w}_j^k}, \tag{4.31}$$

where N_S is the number of subdomains sharing the dof.

The first type of weights is perhaps the most standard one in literature. It is based on the cardinality (*card*) of the set of subdomains sharing the dof. Hence, $\widetilde{w}_i^k = 1$, and

$$w_i^k = \frac{1}{N_S}. \tag{4.32}$$

For example, the weight is simply $w_i^k = 1/2$ if the dof is shared by two subdomains.

The second type of weights is also well established. It is derived from diagonal entries (*diag*) of the subdomain matrices A_i^ℓ. The weight in this case is defined as $\widetilde{w}_i^k = a_{ss,i}$, where s is the subdomain index corresponding to the k-th interface dof. The construction is then completed by (4.31). However in our case, a modification of these weights is required for the block of pressure unknowns, where the diagonal of A_i contains zeros. In these dofs, we switch to the cardinality scaling above, $\widetilde{w}_i^k = 1$.

An important generalization of diagonal weights is the *deluxe scaling* [23, 8, 20]. Although relatively costly, deluxe scaling leads to very robust BDDC preconditioners even for complicated problems. However, an important ingredient of this scaling is the change of basis converting primal dofs to particular unknowns. Since our implementation of BDDC is based on the saddle-point systems with constraints (4.16) without changing the basis, this type of weights is not compatible with our current implementation in *BDDCML*.

Instead, we test a type of weights inspired by the deluxe scaling and introduced in [16] as *unit load (ul)* scaling. Within this averaging, we compose a vector $\widetilde{\mathbf{w}}_i = (\widetilde{w}_i^1, \widetilde{w}_i^2, \dots)^T$ from local solutions to the i-th subdomain problem (4.16) under a unit load applied to a face, i.e. with right-hand side vector equal to ones at the unknowns of the face. In the interface dofs corresponding to corners and edges, we again resort to the cardinality scaling, $\widetilde{w}_i^k = 1$. The construction is again completed by (4.31). The main difference from deluxe scaling is that instead of solving the local problems with the current solution in every iteration, these problems are solved just once for specific right-hand sides, and the solutions are then converted into the diagonal matrices W_i.

The final tested type of weights is a recent idea inspired by numerical schemes for flow problems. Loosely speaking, for dominant advection, it should be beneficial to consider the subdomain from which the fluid flows with a higher weight than for the one where the dof is a part of an inflow boundary. In numerical schemes, this is sometimes referred to as upwinding.

More specifically, these *upwind* weights are based on the inner product of the vector of velocity and the unit vector of outer normal to the subdomain boundary (see Fig. 4.1),

$$p_i^k = \frac{\mathbf{v}^k \cdot \mathbf{n}_i^k}{\|\mathbf{v}^k\|_2}.$$

The values of the p_i^k are from the interval $[-1, 1]$. To derive a nonnegative weight, we map these values into the interval $[0, 1]$ by taking $\widetilde{w}_i^k = \frac{p_i^k + 1}{2}$, which is used for all velocity unknowns. For pressure dofs, we again consider $\widetilde{w}_i^k = 1$. The final partition of unity is achieved again by (4.31).

Obviously, this kind of weights would introduce a nonlinearity by the dependence of the weight on the solution itself. We use the velocity field from the previous nonlinear iteration

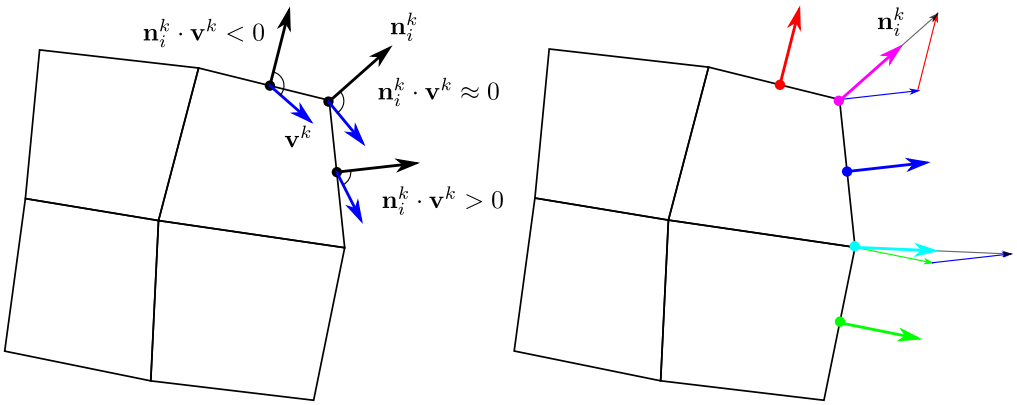

Figure 4.1: Relation of the unit outer normal vectors to the boundary and the velocity vectors in the construction of the *upwind* interface scaling (left), and the construction of the approximate interface normals (right).

as an approximation of the actual velocity vector. Note that the weights are then fixed throughout the BiCGstab iterations.

The final remark is related to constructing the approximation of the unit outer normal vector at positions of the nodes, i.e. element nodes, and edge and face midsides. As illustrated in Fig. 4.1, we take the normals of the element faces as the building block, and define the normals at vertices and edges simply as an arithmetic average of the normals of the adjacent faces.

4.4 Mesh partitioning

Let us now look at the employed partitioners. In this section, I compare two approaches to partitioning the computational domain and mesh into subdomains. A standard approach is based on converting the computational mesh into a graph. In the so called dual graph, finite elements represent vertices of the graph and if two elements share an edge (in 2D) or a face (in 3D), the corresponding graph vertices are connected by a graph edge. The task of partitioning a mesh is translated into a problem of dividing a graph into subgraphs, with the goal that the subgraphs contain approximately the same number of vertices and number of edges connecting the subgraphs is minimized. The first requirement corresponds to balancing load on individual subdomains, while the second is related to the size of the interface between subdomains and amount of communication.

The partitioning of a graph into N subdomains is defined in the following way. The graph $G = (\Omega, E)$, where solution domain Ω with size n and E the number of edges, is partitioned into N subsets $\Omega_1, \Omega_2, \cdots, \Omega_N$ such that $\Omega_i \cap \Omega_j = \emptyset$ for $i \neq j$, size of $\Omega_i \approx n/N$, $\cup_i \Omega_i = \Omega$, and the number of edges from E which belong to different subdomain is minimized. There are established algorithms for partitioning a graph as well as their open-source implementations. We make use of the *METIS* library (version 4.0) [37] for this purpose.

Graph partitioning provides an automated way for dividing computational mesh into subdomains of well-balanced sizes even for complex geometries and meshes. However, information about the geometry of the interface is lost during the conversion into the graph, and the resulting interface can be very irregular. This is a known issue studied mathematically for elliptic problems e.g. by [38]. The structure of the interface and its impact on some Navier-Stokes problems is investigated in Chapter 5.

Another strategy to partitioning the mesh is based on the geometry of the domain and

dividing the subdomains for example by the recursive bisection (RCB) of the longest edge of the geometry or its bounding box. An element belongs to a subdomain, for example, if its center of gravity belongs to the given box. Ideally, these cuts are aligned with layers of elements to prevent irregular interfaces. For simple cuboidal domains and the suitable number of subdomains, it is straightforward to produce a partition that is both balanced and avoids irregular interfaces.

Many geometries, including those of the hydrostatic bearings we aim at, are not completely general and can be decomposed into approximately cuboidal blocks. In order to avoid partitions with irregular subdomains for this type of problems, we have developed a simple partitioner which divides the computational mesh into these cuboidal blocks which are then partitioned into regular subdomains.

Our partitioner works similarly as the RCB partitioner to preserve 2D interfaces between subdomains. For a given problem, we first define cuboidal blocks of the solution domain and divide these blocks by recursive bisection along given axes into balanced subdomains. This comes hand in hand with creating a suitable structural mesh of the solution domain using *GMSH* software [32], and defined cuboidal volumes which also preserve 2D interfaces between each other. In our cases, we only need to create a suitable 2D projection of the solution domain, which we then extrude along the third axis. It also means that for each problem, we may alter the axes along which we use recursive bisection.

In my computation, I use this partitioner for two problems. For a narrowing channel problem described in Section 5.1, I divide the solution domains by straight cuts only along the x-axis. An example of the interface between two subdomains provided by this *geometric partitioner* is in Fig. 4.2.

Figure 4.2: Interfaces between subdomains provided by a *graph* partitioner (left) and by a *geometric* partitioner (right).

In the case of hydrostatic bearing in Section 5.3.1.3, I first define the solution domain in *GMSH* software as three hexahedral volumes and then use recursive bisection along x and y coordinate for the lowest hexahedral, and along all three axes for the remaining two. The resulting division is in Figure 5.18. In Section 5.3.2, I create a mesh in *GMSH* so that its 2D projection can be divided into blocks that preserve 2D interfaces between each other, see Figure 4.3.

Then I decompose blocks of the two lowest layers using recursive bisection along x and y axes, and the upper layer by z coordinate eventually (in my computation, the upper layer is formed only by one subdomain), as in Figure 5.20.

In the rest of the paper, we refer to these two strategies as the *graph* and the *geometric* partitioner. It seems reasonable to prescribe a suitable number of subdomains for each cuboidal block and obtain the total number of subdomains simply as a sum over these blocks for preventing load imbalance.

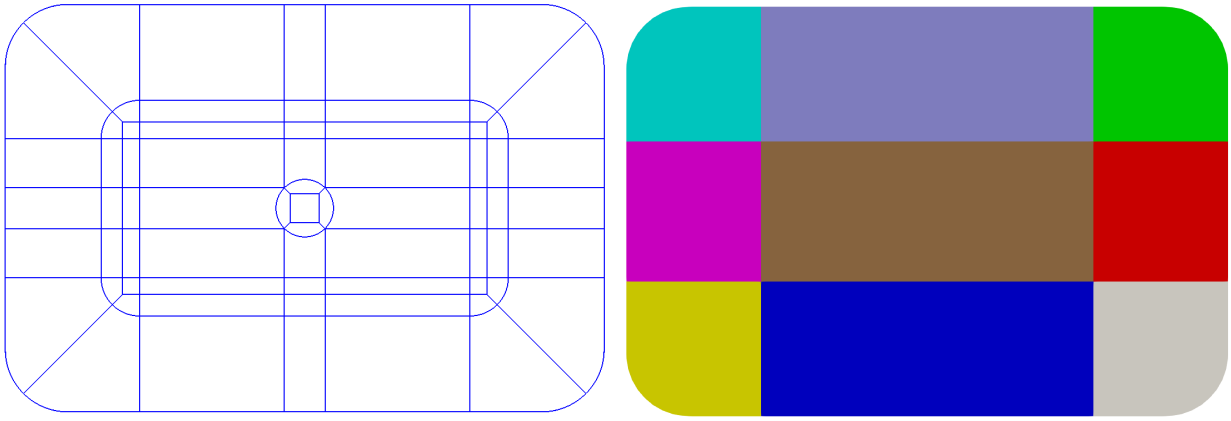

Figure 4.3: 2D projection of the solution domain for hydrostatic bearing in *GMSH* (left) and resulting subdomain in the lowest layer (right).

4.5 Mesh builder

In this section, I introduce my software used for creating subdomains for a large mesh employed in the computation of the cavity problem described in Section 5.2. In these simulations, I needed to solve the problem of dividing a large cubic mesh into smaller cubes as subdomains. Due to the size of these meshes, I was not able to use neither the *graph* nor the *geometric* partitioner. Therefore, I create a program that produces files with individual subdomains and follows the global numbering of the whole solution domain crated by the *GMSH* software.

This *MeshBuilder* software produces a set of cubic subdomains corresponding with unit cube under two parameters. The first is the number of subdomains per cube edge, and the second is the number of hexahedral Taylor-Hood Q2-Q1 elements per subdomain edge. The main part of this software is to make a correct mapping between the local and global degrees of freedom. Both, the global and the local numbering of DOFs are done subsequentially in the x, then in y, and finally in the z coordinate. To see how this map between global and local numbering could look, see Figure 4.4 for a 2D case.

In Figure 4.4, we can see how the numbering of the global DOFs works for 2D case (black numbers), together with a local numbering on individual subdomains for the case of two (red and green numbers) and three (red, green, and blue numbers) subdomains. In the 3D case, the numbering starts at the first layer in the z coordinate like in the 2D case and then moves to the second layer in the z coordinate and continues analogously.

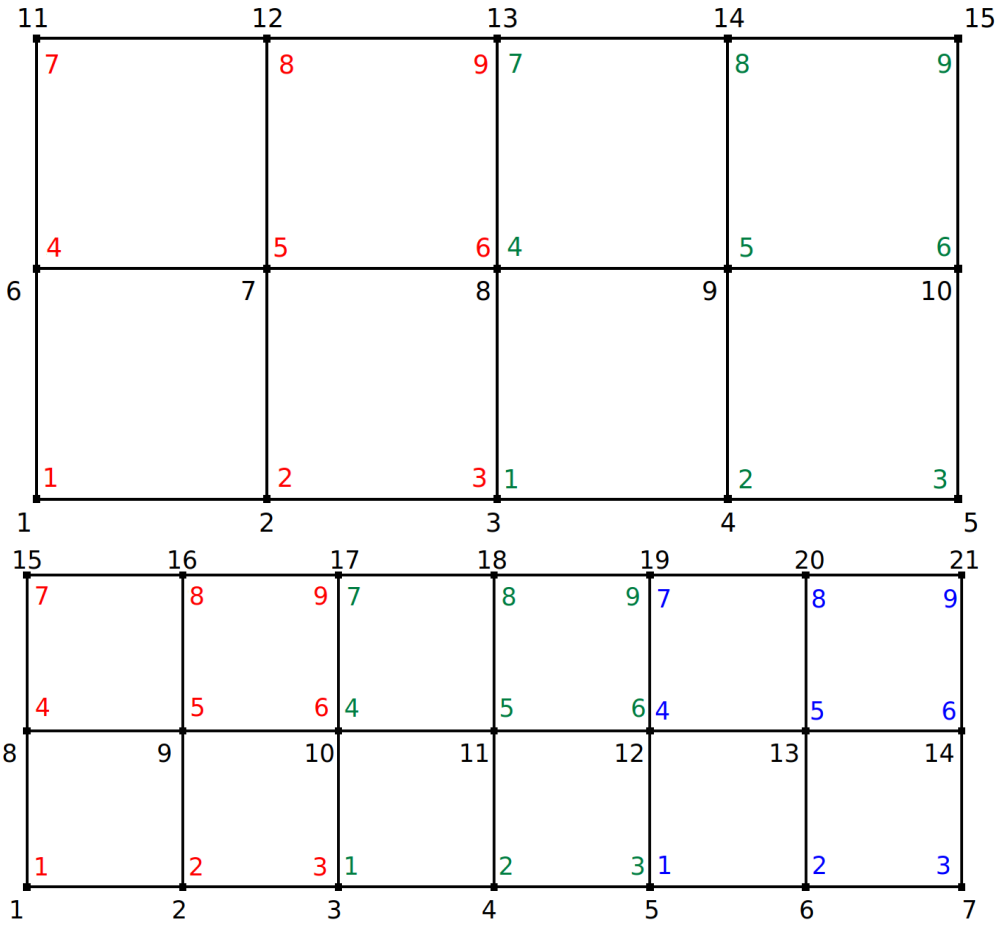

Figure 4.4: Map between global (black) and local (colored) numbering for two (top) and three (bottom) subdomains.

Chapter 5

Numerical results

In this section, we apply the variants of the BDDC method to steady incompressible flow goverened by the Navier-Stokes equations. I sumarize all my results concerning investigation of possibilities of the multilevel BDDC method as well as its applicability to the industrial problem of flow inside the hydrostatic bearings. Namely I investigated the behaviour of the 2-, 3-, and 4-level BDDC method for the benchmark problem of a 3-D lid-driven cavity, behaviour of this method for 4 different weights type (card, diag, unit jump, and upwind) for cavity problem, 2 different partitioners (graph and geometric) for 2-level method for narrowing channel problem, an experimental test of convergence of the BDDC method with respect to changing the H/h ratio. Finally we eployed this method for solving an industrial prloblem of flow of oil inside the hydrostatic bearing. All the computations are performed by a parallel finite element package written in C++ and described in [54], and using the *BDDCML* library [57] for solving the arising system of linear equations. For linearization, we use Picard's iteration with the precision measured as $\epsilon_N = \left\|\mathbf{u}^k - \mathbf{u}^{k-1}\right\|_2$. The BiCGstab method preconditioned by the BDDC preconditioner is used for the linearized system with precision measured as $\epsilon_L = \left\|r^k\right\|_2 / \left\|g\right\|_2$. According to our previous experience from [54], the convergence of the BiCGstab method is comparable to that of GMRES. Simulations were performed on the *Salomon* supercomputer at the IT4Innovations National Supercomputing Center using the cores of Intel Xeon E5-2680v3 12C 2.5GHz processors.

5.1 Narrowing channel

First we look at the problem of the narrowing channel. In our computations we aim at the influence of interface irregularities on the BDDC solver for Navier-Stokes equations. In particular, we investigate the effect of the aspect ratio of the finite elements at the interface on convergence. This is motivated by our target application—simulations of oil flow in hydrostatic bearings with very narrow throttling gaps. In order to study this phenomenon, a benchmark problem suitable for such a study is proposed and the partitioning strategies described in Section 4.3 are compared. Picard's iteration is terminated when $\epsilon_L \leq 10^{-5}$ or after performing 100 iterations. The BiCGstab method is stopped if $\epsilon_N \leq 10^{-6}$, with the limit of 1000 iterations.

As a measure of convergence, we monitor the number of BiCGstab iterations needed in one Picard's iteration. Two matrix-vector multiplications are needed in each iteration of BiCGstab, and after each of them, the terminal condition is evaluated. Correspondingly, inspired by the Matlab `bicgstab` function, termination after the first matrix-vector multiplication is reported by a half iteration in the BiCGstab iteration counts. Numbers of

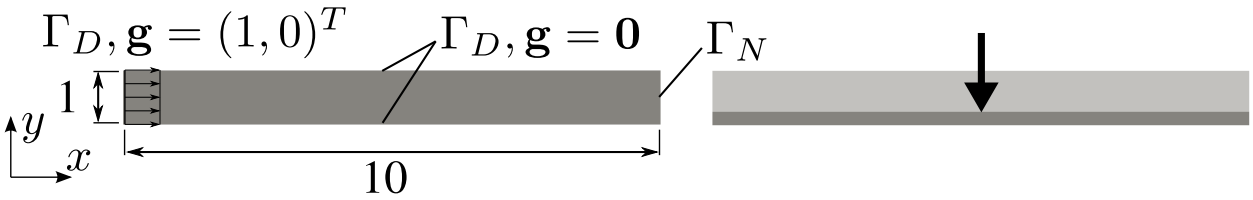

Figure 5.1: The narrowing channel 2-D benchmark; original channel (left) and narrowing along the y-axis (right).

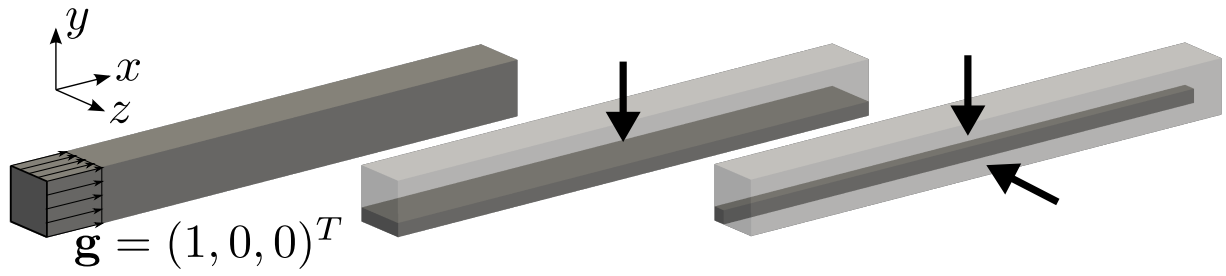

Figure 5.2: The narrowing channel 3-D benchmark; original channel (left), narrowing along the y-axis (centre), and narrowing along both y and z-axes (right).

iterations are presented as minimum, maximum, and mean over all nonlinear iterations for a given case.

The benchmark problem consists of a sequence of simple channels in 2D (Fig. 5.1) and 3D (Fig. 5.2). The dimension of the channels along one or two (in 3D) coordinates is gradually decreased, with the initial dimensions 10×1×1 along the x, y, and z axes. The computational mesh is based on rectangular (2D) or cuboidal (3D) finite elements uniformly distributed along each direction. The number of elements is 100×10×10 along the x, y and z coordinates. In total, the 3-D problem contains 10 000 elements, 88 641 nodes, and 278 144 unknowns.

The *aspect ratio of elements* $\mathcal{AR} = h_{\max}/h_{\min}$ is defined as the ratio of the longest edge of the element $h_{\max}$ to its shortest counterpart $h_{\min}$. The $\mathcal{AR} = 1$ corresponds to square (or cubic) elements. We test the sequence of narrowing channels for $\mathcal{AR} \in \{1, 2, 4, 10, 20, 40, 100\}$.

The velocity at the inlet starts from $\mathbf{g} = (1, 0, 0)^T$ for $x = 0$, the velocity at the walls is fixed to $\mathbf{g} = \mathbf{0}$, and the face of the channel for $x = 10$ corresponds to Γ_N. We have considered two scenarios for the inflow velocity during the narrowing. The first is simply keeping the magnitude of the velocity fixed throughout the sequence. In the second scenario, the magnitude of the velocity is increased proportionally to the decrease of the height, so that the Reynolds number, defined as $Re = \frac{|\mathbf{u}|D}{\nu}$, is kept constant for the decreasing channel height D. However, results for both scenarios of the inlet boundary condition have been almost identical, and we present only the results for fixed Reynolds number for brevity. We use $\nu = 1$ for our computations. The channel is divided into 4 subdomains by the graph and the geometric partitioners described in Section 4.3.

First we look at the two-dimensional problem. For the graph partitioner, the interface contains both long and short edges of elements. On the other hand, the interface is composed solely from short edges for the geometric partitioner (see Fig. 5.3). Corresponding results are in Tables 5.1 and 5.2.

For the 3-D case, we consider two kinds of problems. First we decrease only the y-dimension of the channel, while in the second case, we shrink both y and z dimensions of the cross-section (see Fig. 5.2). The graph partitioner produces rough interface in both cases,

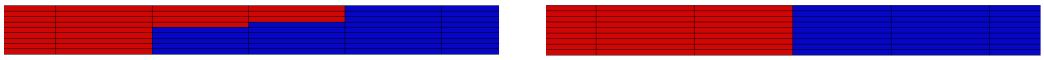

Figure 5.3: Detail of the interface between two subdomains in 2D for graph (left) and geometric (right) partitioner.

partitioner		graph						
$\mathcal{AR}$		1	2	4	10	20	40	100
Picard's its.		4	4	5	5	7	6	40
BiCGstab its.	min	9	10.5	13.5	13.5	15	16.5	17.5
	max	9.5	10.5	13.5	15	16	17.5	19.5
	mean	9.4	10.5	13.5	14.2	15.2	16.7	18.1

Table 5.1: Numbers of iterations for *graph* partitioner for the 2-D narrowing channel.

while the geometric partitioner leads to rectangular faces at the interface in the first case (see Fig. 5.4) and square faces in the second case. Resulting numbers of iterations are presented in Tables 5.3–5.6. Numbers in italic are runs that did not converge due to reaching the maximal number of iterations or time restrictions. A solution of the problem for the initial channel geometry is presented in Fig. 5.5.

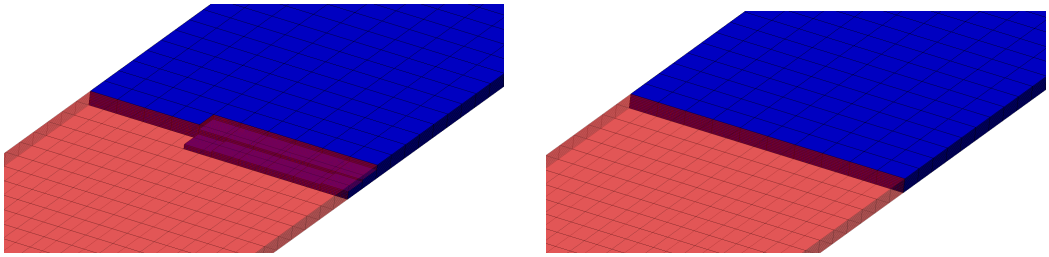

Figure 5.4: Detail of the interface between two subdomains for narrowing along the y-coordinate in 3D for graph (left) and geometric (right) partitioner.

From Tables 5.1–5.6 we can conclude that $\mathcal{AR}$ of faces at the interface has a remarkable influence on the number of BiCGstab iterations in each Picard's iteration. Using the graph partitioner results in a rough interface combining long and short edges. This has a large impact on the efficiency of the BDDC preconditioner and the number of linear iterations increases significantly.

Employing the geometric partitioner leads to straight cuts between subdomains aligned with layers of elements. In 2D, this is sufficient to achieve convergence of the linear solver independent of $\mathcal{AR}$. In 3D, the situation is more subtle. For the case of narrowing the channel only along the y-axis, the aspect ratio of the rectangular element faces at the interface also worsens during contracting the channel. This is translated into a slight growth of the number of BiCGstab iterations in Table 5.4 even in this case, although the convergence is much more favourable than for the graph partitioner. If we narrow the channel along both y and z coordinates, the shape of the element faces at the interface does not deteriorate from squares, and we observe good convergence independent of $\mathcal{AR}$ in Table 5.6.

partitioner		geometric						
$\mathcal{AR}$		1	2	4	10	20	40	100
Picard's its.		3	4	5	5	6	6	5
BiCGstab its.	min	4.5	4.5	4.5	4	3	3	3
	max	4.5	4.5	4.5	4	3	3	3
	mean	4.5	4.5	4.5	4	3	3	3

Table 5.2: Numbers of iterations for *geometric* partitioner for the 2-D narrowing channel.

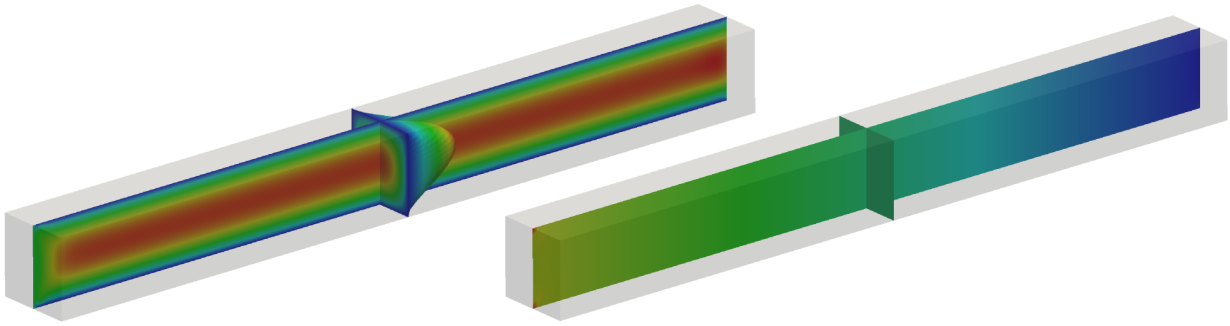

Figure 5.5: Solution in the initial 3-D channel geometry; magnitude of velocity (left) and pressure in the plane of symmetry (right).

partitioner		graph						
$\mathcal{AR}$		1	2	4	10	20	40	100
Picard's its.		4	5	5	42	5	*100*	*100*
BiCGstab its.	min	17.5	20	25.5	44.5	84.5	145	400
	max	18.5	20.5	25.5	51	113.5	858	*1000*
	mean	18.3	20.4	25.5	46.2	93.9	209	761

Table 5.3: Numbers of iterations for *graph* partitioner for the 3-D channel narrowed along the y-coordinate.

partitioner		geometric						
$\mathcal{AR}$		1	2	4	10	20	40	100
Picard's its.		4	5	5	5	5	5	99
BiCGstab its.	min	5.5	6.5	7.5	11.5	16	19.5	19.5
	max	5.5	6.5	7.5	12	17.5	19.5	21
	mean	5.5	6.5	7.5	11.9	17.2	19.5	19.5

Table 5.4: Numbers of iterations for *geometric* partitioner for the 3-D channel narrowed along the y-coordinate.

partitioner		graph						
$Æ$		1	2	4	10	20	40	100
Picard's its.		4	4	4	5	8	19	*28*
BiCGstab its.	min	17.5	19.5	27.5	36	51	80	197
	max	18.5	20.5	28	41.5	53	92.5	*1000*
	mean	18.3	19.8	27.9	39.5	51.8	87.7	590

Table 5.5: Numbers of iterations for *graph* partitioner for the 3-D channel narrowed along both y and z-coordinates.

partitioner		geometric						
$Æ$		1	2	4	10	20	40	100
Picard's its.		4	4	4	5	5	5	4
BiCGstab its.	min	5.5	5.5	6	5	4.5	4.5	4.5
	max	5.5	6	6	5.5	5	5	4.5
	mean	5.5	5.9	6	5.1	4.9	4.6	4.5

Table 5.6: Numbers of iterations for *geometric* partitioner for the 3-D channel narrowed along both y and z-coordinates.

5.2 Lid-driven cavity

In this section we investigate behaviour of the BDDCML solver in application to benchmartk problem of 3-D lid-driven cavity suggested in [63]. The computational domain is a unit cube with Dirichlet boundary conditions. A twisted unit tangential velocity vector is considered on the top wall, $\boldsymbol{u}_{\text{top}} = (1/\sqrt{3}, \sqrt{2}/\sqrt{3}, 0)$. Zero velocity is considered on the remaining 5 walls. The computational mesh is uniform with hexahedral elements, and it is divided into cubic subdomains with a rising number of them per domain edge (see Fig. 5.6) using my *Meshbuilder* software described in Section 4.5. Simulations were performed on the *Salomon* supercomputer at the IT4Innovations National Supercomputing Center using the same number of cores of Intel Xeon E5-2680v3 12C 2.5GHz processors as the number of subdomains. We terminate Picard's iterations after reaching the precision $\epsilon_N \leq 10^{-5}$ or after 100 iterations, while the inner linear iterations are terminated when $\epsilon_L \leq 10^{-6}$ or after reaching the maximum number of 1000 iterations.

We present three performed tests of behaviour of BDDCML method. Namely, weak scalability compared for 2-, 3-, and 4-level method, comparision of four used weights type described in Section 4.3, including our own new upwind weight type, and experimental test of convergence of the BDDC method with respect to changing the H/h ratio.

5.2.1 Weak scalability

In this section, we compare the weak scalability of the multilevel BDDC method in application to 3D lid-driven cavity. Results for the 2-, 3-, and 4-level BDDC method for Reynolds numbers 1 and 100 are presented.

5.2.1.1 2- and 3-level method

First, we compare the 2- and the 3- level BDDC method. The number of subdomains grows from 8 to 4913, and there are 8 elements per subdomain edge. The number of unknowns

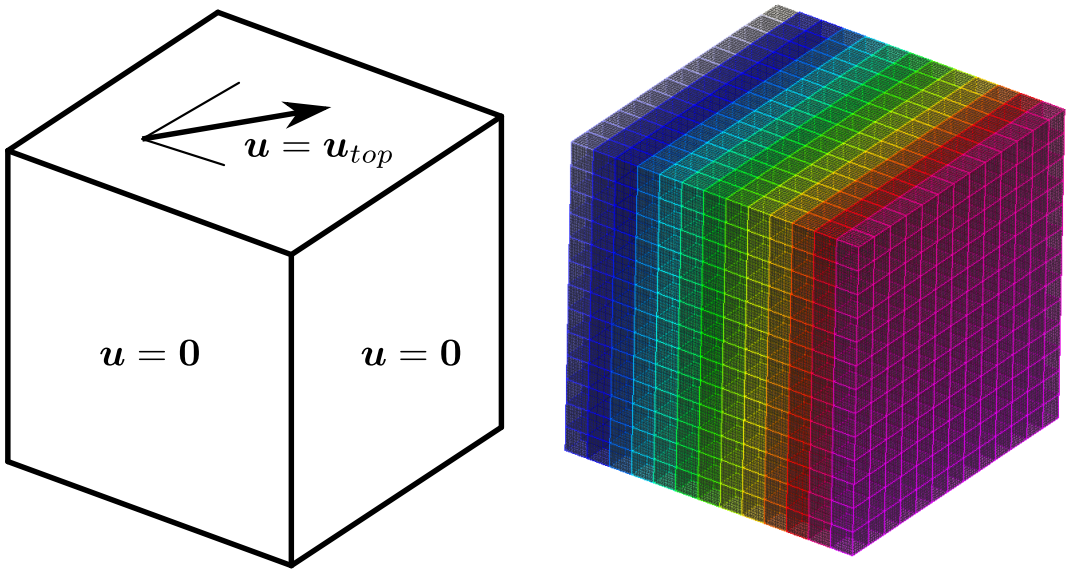

Figure 5.6: Boundary conditions (left) and example of mesh division with 13 subdomains per edge (right) for the lid-driven cavity.

grows from 112 724 to 63 610 604, and the size of the interface problem grows from 10 324 to 10 915 264 unknowns. We consider two values of the Reynolds number, in particular, 1 and 100. We define the Reynolds number as LU/ν, where L is the length of an edge of the solution domain, U is the velocity of the top wall and ν is the kinematic viscosity. With increasing Reynolds number, the significance of the symmetric part of (2.34) is decreasing, and the nonsymmetric part has larger influence (see [62]). For these simulations, we use cubes as subdomains on the first coarse level (see Fig. 5.6) for both cases. To build subdomains on the second level for the 3-level method, we use the METIS graph partitioner [37]. An example of a cluster of subdomains of the first level composing a subdomain on the second level is shown in Fig. 5.7. We monitor the mean, maximal, and minimal number of linear iterations, the number of nonlinear iterations, the mean setup time of the BDDC preconditioner, the mean time for the Krylov subspace method, and the mean time for one iteration for the 2- and the 3-level method for both cases of the Reynolds number. These values are presented in Tables 5.7–5.10.

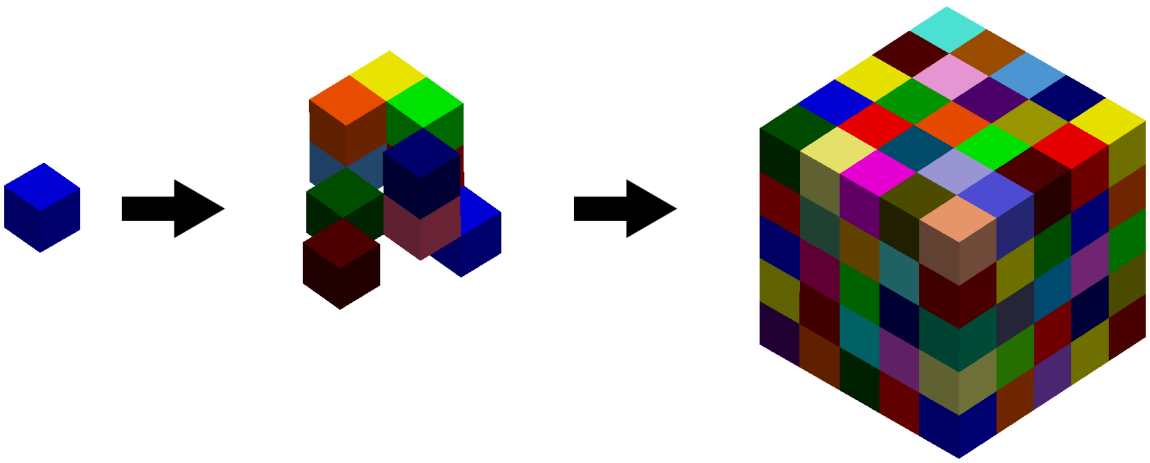

Figure 5.7: Clustering of subdomains on the second level using METIS.

nproc	nonl	linear solve			time [s]		
		min	max	mean	setup	BiCGstab iter (one iter)	total
8	4	8.5	9	8.9	2.33	0.58 (0.07)	2.91
27	4	10.5	11.5	11.3	3.16	0.84 (0.07)	4.00
64	4	11.5	11.5	11.5	3.71	1.24 (0.11)	4.95
125	4	12.5	12.5	12.5	3.98	1.50 (0.12)	5.48
216	4	11.5	12	11.9	4.80	1.91 (0.16)	6.71
343	4	12.5	12.5	12.5	5.39	2.46 (0.20)	7.85
512	4	12.5	12.5	12.5	6.31	3.04 (0.24)	9.35
729	4	13	13	13	8.79	5.45 (0.42)	14.2
1000	4	12.5	12.5	12.5	11.3	7.00 (0.56)	18.3
1331	4	13	13	13	15.1	10.3 (0.79)	25.4
1728	4	12.5	12.5	12.5	20.7	14.0 (1.12)	34.7
2197	4	13	13	13	31.0	22.0 (1.69)	53.0
2744	4	12.5	12.5	12.5	42.8	28.3 (2.26)	71.1
3375	4	13	13	13	56.8	40.9 (3.15)	97.7
4096	4	12.5	12.5	12.5	79.6	21.6 (1.73)	101.2
4913	4	13	13	13	111.8	29.4 (2.26)	141.2

Table 5.7: $Re = 1$, 2-levels. Number of nonlinear iterations, number of linear iterations (minimal, maximal, and mean), mean setup time, time for the BiCGstab iterations, time for one linear iteration, and the total time.

nproc	nonl	linear solve			time [s]		
		min	max	mean	setup	BiCGstab iter (one iter)	total
8	4	10	10	10	2.33	0.65 (0.07)	2.98
27	4	12.5	12.5	12.5	3.18	0.93 (0.07)	4.11
64	4	13.5	13.5	13.5	3.71	1.40 (0.10)	5.11
125	4	17.5	17.5	17.5	3.95	1.94 (0.11)	5.89
216	4	15	16	15.3	4.43	2.05 (0.13)	6.48
343	4	19.5	19.5	19.5	4.41	2.74 (0.14)	7.15
512	4	17.5	17.5	17.5	4.72	2.78 (0.16)	7.50
729	4	21.5	22.5	21.8	5.20	4.03 (0.18)	9.23
1000	4	16.5	18	17.6	5.10	3.78 (0.21)	8.88
1331	5	18	18.5	18.4	5.25	7.87 (0.43)	13.1
1728	4	17.5	20.5	19.8	6.45	6.61 (0.33)	13.1
2197	4	17.5	20.5	18.1	6.70	7.14 (0.39)	13.8
2744	4	18.5	18.5	18.5	8.86	9.25 (0.50)	18.1
3375	5	17.5	19.5	18.7	8.40	10.7 (0.57)	19.4
4096	4	19.5	20.5	20.3	10.8	8.55 (0.42)	19.4
4913	4	19.5	19.5	19.5	13.2	9.80 (0.50)	23.0

Table 5.8: $Re = 1$, 3-levels. Number of nonlinear iterations, number of linear iterations (minimal, maximal, and mean), mean setup time, time for the BiCGstab iterations, time for one linear iteration, and the total time.

nproc	nonl	linear solve			time [s]		
		min	max	mean	setup	BiCGstab iter (one iter)	total
8	18	9.5	14	13.8	2.22	0.86 (0.06)	2.98
27	19	10.5	17.5	15.1	3.16	1.12 (0.07)	4.28
64	20	11.5	18.5	17.2	3.64	1.84 (0.11)	5.48
125	21	13	19	17.0	3.94	2.04 (0.12)	5.98
216	21	12	16.5	15.4	4.84	2.49 (0.16)	7.33
343	21	11	16	15.3	5.76	3.08 (0.20)	8.84
512	22	12.5	15	14.4	6.30	3.56 (0.25)	9.86
729	22	12.5	14.5	14.0	8.58	5.28 (0.38)	13.9
1000	22	12	14	13.0	11.2	7.26 (0.56)	18.5
1331	22	13	14.5	14.4	15.1	11.7 (0.81)	26.8
1728	23	12.5	13.5	13.0	20.7	14.6 (1.12)	35.3
2197	23	13	13.5	13.4	31.7	22.6 (1.69)	54.3
2744	23	12.5	14	13.8	43.3	31.1 (2.25)	74.4

Table 5.9: $Re = 100$, 2-levels. Number of nonlinear iterations, number of linear iterations (minimal, maximal, and mean), mean setup time, time for the BiCGstab iterations, time for one linear iteration, and the total time.

nproc	nonl	linear solve			time [s]		
		min	max	mean	setup	BiCGstab iter (one iter)	total
8	18	10.5	16	14.7	2.30	0.94 (0.06)	3.24
27	19	12.5	18.5	16.4	3.15	1.21 (0.07)	4.36
64	20	13.5	20.5	18.1	3.68	1.89 (0.10)	5.57
125	100	17.5	27.5	25.4	3.93	2.80 (0.11)	6.73
216	26	17.5	22.5	22.2	4.29	2.99 (0.13)	7.28
343	100	19.5	34.5	30.1	4.41	4.25 (0.14)	8.66
512	24	17.5	21.5	21.3	4.82	3.62 (0.17)	8.44
729	100	22.5	31.5	28.2	5.13	5.48 (0.19)	10.6
1000	100	18.5	16	29.7	4.85	6.52 (0.22)	11.4
1331	30	18.5	30.5	23.1	5.34	6.12 (0.26)	11.5
1728	28	17.5	29.5	25.7	6.34	8.50 (0.33)	14.8
2197	100	17.5	32.5	29.2	6.97	11.6 (0.40)	18.6
2744	100	18.5	32.5	27.7	9.07	13.7 (0.49)	22.8

Table 5.10: $Re = 100$, 3-levels. Number of nonlinear iterations, number of linear iterations (minimal, maximal, and mean), mean setup time, time for the BiCGstab iterations, time for one linear iteration, and the total time.

We can see that for the case with Reynolds number $Re = 1$, the numbers of nonlinear iterations stay the same for almost all cases for both 2- and 3-level method. The numbers of linear iterations appear to have a similar, slightly increasing trend. From the setup times, we can see that for the 2- level method, the setup time rapidly increases with the number of processors, whereas for the 3-level method, the increase of the setup time is significantly slower. From the mean times in Tables 5.7 and 5.8 we can see that the 3-level method is faster than the 2-level one, especially for a bigger number of processors. Also, there is not such a rapid increase of computational time for the 3-level method as for the 2-level case.

One can observe that while the 3-level method gets considerably faster than the 2-level method, the computational times are not perfectly weakly scalable even for the 3-level case. While this can be clearly attributed to the coarse problem solves for the 2-level method, it is likely the global communication related to propagating the higher-level solutions and residuals what worsens the weak scalability also for the 3-level method.

Let us look closer at the case of $Re = 100$. We can see that for the 2-level method, the number of nonlinear iterations is slightly increasing with the rising number of processors. For the 3-level method, the number of nonlinear iterations is similar in some cases, and in some cases, it reaches the limit of 100 nonlinear iterations. Looking closer at the output of these simulations, we could see that after the maximum of 20 nonlinear iterations, the residual ϵ_N oscillates between $5 \cdot 10^{-5}$ and $10 \cdot 10^{-5}$, but never drops below the prescribed 10^{-5}. For the setup time, we can see similar behaviour as in the case with $Re = 1$, and the same stands for the time for solving the linear problem and the time for 1 iteration. If we compare the results for $Re = 1$ and $Re = 100$, a significant difference is just in the number of nonlinear iterations. This can be attributed to the bigger influence of the nonsymmetric part of (2.34).

Since we are mainly interested in the efficiency of the BDDC method and the linear solver, we focus on the mean number of linear iterations over all nonlinear iterations, the mean setup time for preparing the BDDC preconditioner, the mean time for solving the linear problem, and the mean time for one linear iteration. The comparison of the behaviour of the 2- and 3-level methods for these parameters with a rising number of processors for both cases of the Reynolds number is presented in Figs. 5.8 and 5.9.

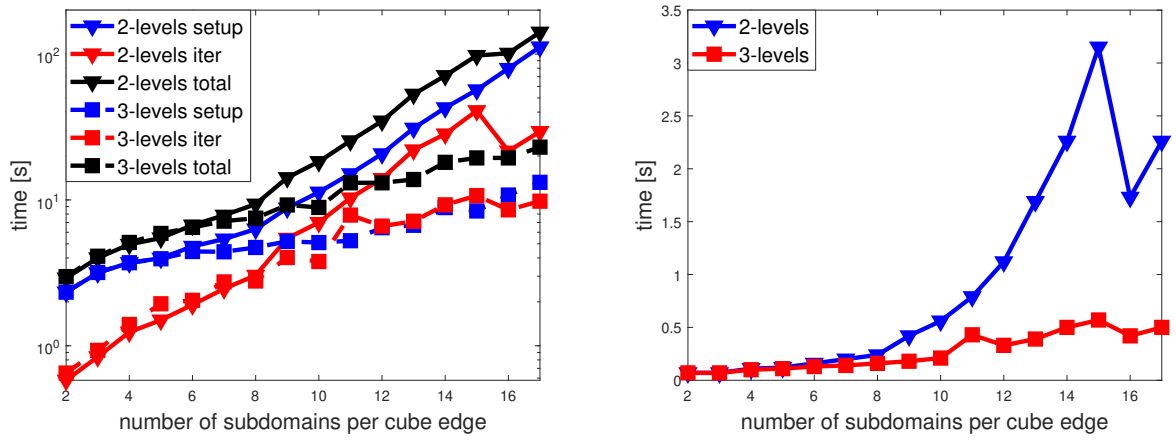

Figure 5.8: $Re = 1$. Mean time for setup, mean time for the BiCGstab iterations and mean total time (left), mean time for one iteration (right).

In Fig. 5.8, there is a large drop of time for one iteration for the 2-level method. While reproducible, we are not able to satisfactorily explain this phenomenon.

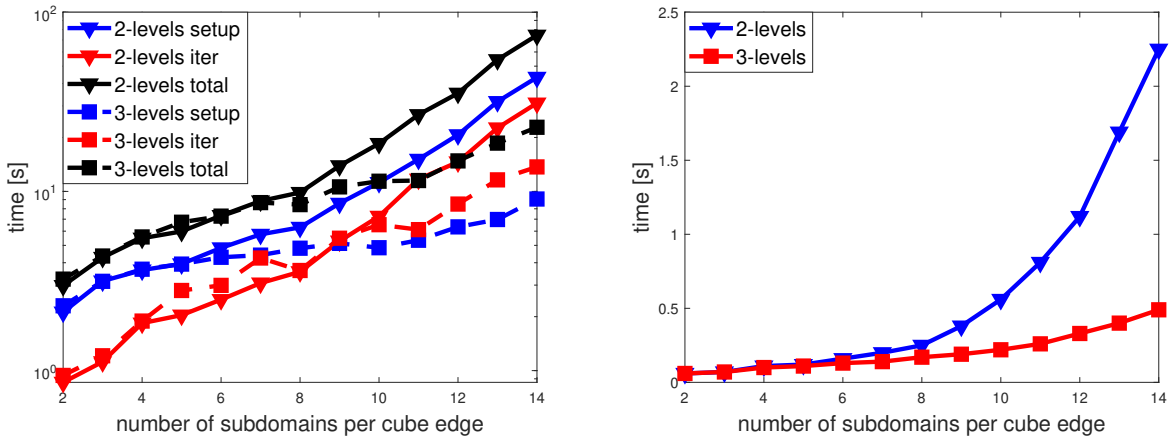

Figure 5.9: $Re = 100$. Mean time for setup, mean time for the BiCGstab iterations and mean total time (left), mean time for one iteration (right).

5.2.1.2 3- and 4-level method

Next, we compare the 3- and the 4-level methods. The number of subdomains goes from 8 up to 1728 with 12 elements per subdomain edge. Because the number of elements per subdomain edge is different, we had to recompute also the results for the 3-level method, and the results in this section are different from those in Section 5.2.1.1. As a consequence, we now have larger local problems. The number of unknowns ranges from 368 572 to 75 461 332, and the size of the interface problem grows from 22 972 to 8 600 372 unknowns. Here we form subdomains on coarse levels for the 3-level method in the same way as in the previous subsection, i.e. using the METIS graph partitioner. Figure 5.10 shows the potentially complex subdomains on higher levels resulting from employing METIS. However, for the 4-level method, we form regular subdomains on all levels. Thus, on the first coarse level, we have cubes as subdomains, on the second coarse level we join a line of cubes into a cuboid, and on the third coarse level, we join plate of cuboids to form the subdomains (see Fig. 5.11). The reason for this explicit creation of the coarse subdomains is that the simulations using METIS did not converge for the 4-level method. This seems to be related to our previous findings about a significantly better convergence of the BDDC method with regular subdomains (see [1] for more details). We set the value of the Reynolds number to 1, and we compare the same parameters as in the previous case. We again report the number of linear iterations (maximal, minimal, and mean) over all nonlinear iterations, the mean setup time of the BDDC preconditioner, the mean time for solving the linear problem, and the mean time for one linear iteration. All these values are in Tables 5.11 and 5.12.

We can see that the numbers of nonlinear iterations remain similar as in the previous subsection for almost all cases for the 3- and the 4- level method. However, the numbers of linear iterations are rapidly increasing for the 4-level method. If we look at the setup times, we can see that the time for the 3- and the 4-level method seems to be slowly increasing with the number of subdomains, with a similar rate. A big difference is in the total time for solving the linear problem. Due to the rapid increase in the mean number of linear iterations, the total time for solving the linear problem by the 4-level method gets much larger than for the 3- and even the 2-level method although each iteration is cheaper for larger numbers of subdomains. Thus further worsening of the approximation of the preconditioner by introducing the fourth level is not beneficial for the problem of our experiment.

Again, we mainly monitor the mean number of linear iterations over all nonlinear it-

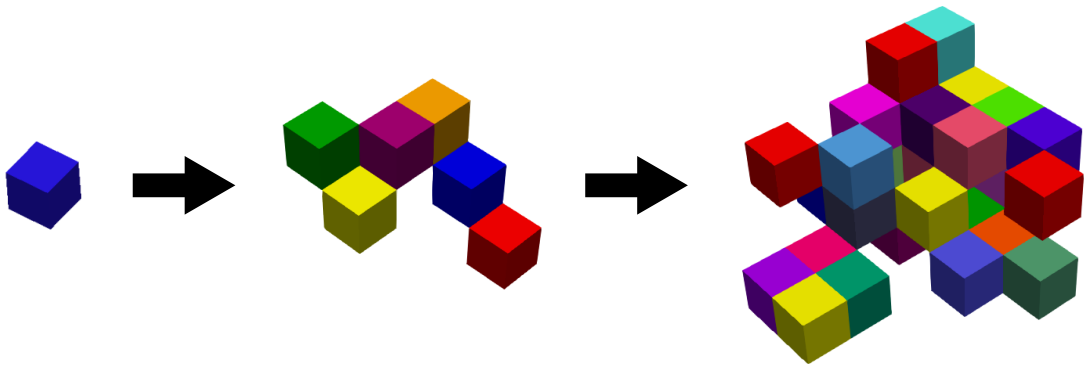

Figure 5.10: Decomposition by METIS for the 4-level BDDC method. First-level subdomains (left), second-level subdomains (centre), and third-level subdomains (right).

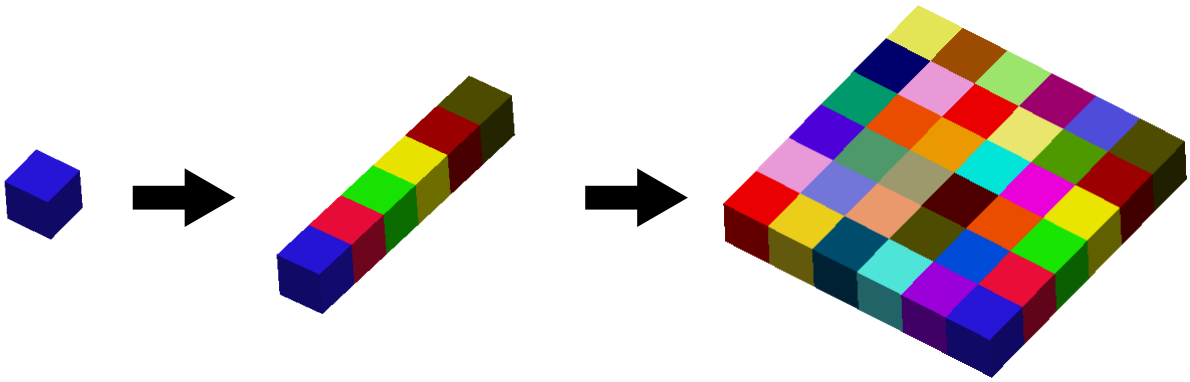

Figure 5.11: Regular decomposition for the 4-level BDDC method. First-level subdomains (left), second-level subdomains (centre), and third-level subdomains (right).

nproc	nonl	linear solve			time [s]		
		min	max	mean	setup	BiCGstab iter (one iter)	total
8	4	14	14	14	13.57	4.06 (0.29)	17.6
27	4	16.5	16.5	16.5	15.92	5.78 (0.35)	21.7
64	4	16	16	16	18.03	8.17 (0.51)	26.2
125	6	28.5	30.5	29.7	18.24	15.06 (0.51)	33.3
216	16	26.5	27.5	27.3	19.77	16.16 (0.59)	35.9
343	8	33.5	36	34	33.46	29.95 (0.88)	63.4
512	5	38.5	41.5	40.3	25.18	41.85 (1.04)	67.0
729	20	32	36	34.1	21.28	28.05 (0.82)	49.3
1000	4	24	24.5	24.4	26.08	26.46 (1.08)	52.5
1331	4	23.5	26	24.1	25.77	27.37 (1.14)	53.1
1728	4	25.5	25.5	25.5	26.36	30.34 (1.19)	56.7

Table 5.11: $Re = 1$, 3-levels. Number of nonlinear iterations, number of linear iterations (minimal, maximal, and mean), mean setup time, time for the BiCGstab iterations, time for one linear iteration, and the total time.

nproc	nonl	linear solve			time [s]		
		min	max	mean	setup	BiCGstab iter (one iter)	total
8	5	13.5	13.5	13.5	13.49	3.90 (0.29)	17.4
27	4	16.5	17.5	16.8	15.92	5.86 (0.35)	21.8
64	4	22.5	23.5	23.3	22.80	20.84 (0.89)	43.6
125	4	33	34.5	34.1	18.16	17.28 (0.51)	35.4
216	5	47.5	49.5	48.3	36.14	48.95 (1.01)	85.1
343	22	66.5	73	67.6	22.04	54.61 (0.81)	76.7
512	12	97	100.5	100.2	24.63	102.49 (1.02)	127.1
729	11	122	145	135.2	24.58	140.49 (1.04)	165.1
1000	4	175.5	189	182.4	31.10	127.39 (0.70)	158.5
1331	12	207.5	238	218	25.14	241.14 (1.11)	266.3
1728	4	250.5	282	267.6	57.55	223.12 (0.83)	280.7

Table 5.12: $Re = 1$, 4-levels. Number of nonlinear iterations, number of linear iterations (minimal, maximal, and mean), mean setup time, time for the BiCGstab iterations, time for one linear iteration, and the total time.

erations, the mean setup time for preparing the BDDC preconditioner, the mean time for solving the linear problem, and the mean time for one linear iteration. The comparison of the behaviour of the 3- and the 4-level methods for these parameters with a rising number of processors is presented in Fig. 5.12.

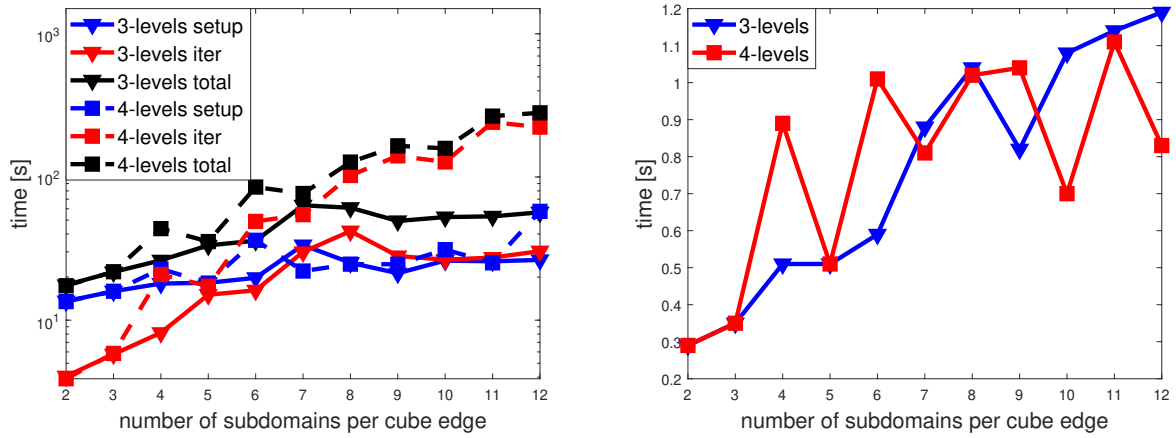

Figure 5.12: $Re = 1$. Mean time for setup, mean time for the BiCGstab iterations and mean total time (left), mean time for one iteration (right).

In Figure 5.12, there are once again some wild drops and ups which we cannot satisfactorily explain, but at least for the 3-level method, it could be caused by certain more suitable decompositions by the METIS graph partitioner on higher levels. We again see a suboptimal weak scalability probably caused by the coarse problem global communication.

5.2.2 Interface scalings

In this section, we compare the behaviour of the 2-level BDDC method for different types of interface weights described in Section 4.3, namely the cardinality scaling (*card*), scaling by diagonal stiffness (*diag*), scaling weights from unit load (*ul*), and the proposed *upwind*

weights. For these simulations, the number of subdomains is 125 with 8 elements per subdomain edge. We consider Reynolds numbers 100 and 200. Also the division into subdomains remains. We once again monitor the mean, maximal, and minimal number of linear iterations, the number of nonlinear iterations, the mean setup time of the BDDC preconditioner, the mean time for the Krylov subspace method, and the mean time for one iteration. These values are presented in Tables 5.13 and 5.14.

weights type	nonl	linear solve			time [s]		
		min	max	mean	setup	BiCGstab iter (one iter)	total
card	23	13	19	17	4.18	1.99 (0.12)	6.17
diag	23	13	18	15.1	3.85	1.78 (0.12)	5.63
ul	23	12.5	14.5	13.5	4.22	1.60 (0.12)	5.82
upwind	23	23	14.5	14.3	4.05	1.69 (0.12)	5.74

Table 5.13: $Re = 100$. Number of nonlinear iterations, number of linear iterations (minimal, maximal, and mean), mean setup time, time for the BiCGstab iterations, time for one linear iteration, and the total time.

weights type	nonl	linear solve			time [s]		
		min	max	mean	setup	BiCGstab iter (one iter)	total
card	100	12	84.5	68.8	3.84	8.03 (0.12)	11.03
diag	33	12	77.5	69.5	3.87	8.09 (0.12)	11.96
ul	100	12	81	54.4	4.19	6.35 (0.12)	10.54
upwind	48	22	80.5	28.2	4.28	3.31 (0.12)	7.59

Table 5.14: $Re = 200$. Number of nonlinear iterations, number of linear iterations (minimal, maximal, and mean), mean setup time, time for the BiCGstab iterations, time for one linear iteration, and the total time.

The maximal number of nonlinear iteration was again set to 100, which was reached for several cases (Table 5.14). For those cases, the error ϵ_N oscillated between $5 \cdot 10^{-5}$ and 10^{-4}, but never dropped below 10^{-5}.

From Tables 5.13 and 5.14, we can see that there is no significant difference in using different weights for Reynolds number 100. However, for Reynolds number 200, we can observe a large difference in the mean number of linear iterations. Consequently, the time for linear iterations is also different, although the time per one iteration stays the same. Based on these tables, we can conclude that the upwind weights may lead to a significant reduction of the number of linear iterations as well as of the computational time.

5.2.3 Experimental H/h dependence

Next we perform an experimental test of convergence of the BDDC method with respect to changing the H/h ratio, i.e. the ratio of the characteristic size of a subdomain to the characteristic size of an element. For cubic subdomains, H/h represents the number of elements along a subdomain edge. In these simulations, we have 64 subdomains and vary the number of elements per subdomain edge from 2 to 24. The Reynolds number was set to 1, and we compare the behaviour of the 2- and the 3-level method. The results are presented in Fig. 5.13.

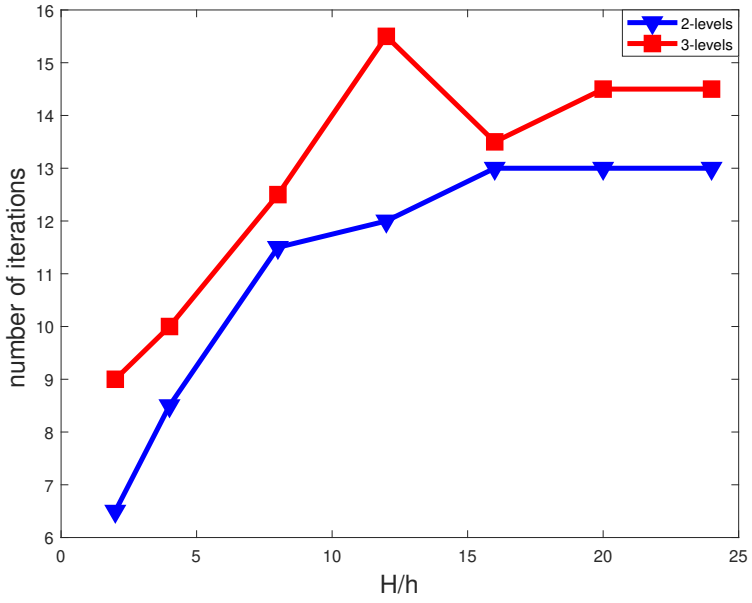

Figure 5.13: Dependence of the number of iterations on the H/h ratio.

From Fig. 5.13, we can see that with the growing H/h ratio, the number of BiCGstab iterations approximately follows the logarithmic dependence (3.38), for both the 2- and, with slight irregularities, the 3-level method. Although a theoretical insight is not available for this class of problems, the dependence seems to resemble the behaviour for problems with a symmetric positive definite matrix.

5.3 Hydrostatic bearings

In this section we finally combine the previous experience and present the results for driving application of our research, namely simulations of oil flow in hydrostatic bearings. These are parts of production machines that keep moving parts of the machines on a thin layer of oil to provide low friction. Oil is pressurized to a few MPa, and it flows through the input of the hydrostatic bearing into a so-called hydrostatic cell. Then it flows out through a very thin (few tens of micrometers) throttling gap.

Hydrostatic bearings have been studied for a long time at the Research Center of Manufacturing Technology at the Faculty of Mechanical Engineering of the Czech Technical University in Prague. These parts are typically designed using analytic solution of flow inside the throttling gap. However, a detailed picture of the 3D flow in the whole bearing was missing. Consequently, performing 3D simulations of oil flow inside several designs and settings of the bearings has been another aim of my reserch and the results are summarized in this section.

Numerical simulation of this problem comes with several challenges, like a large pressure gradient realized in the throttling gap, small thickness of the throttling gap, and moving of the bearing. A major issue is the emergence of finite elements with bad aspect ratio in the throttling gap, see Figure 5.15. During our research, I have gradually addressed most of these issues, at the end being able to simulate real-scale geometry problems of sliding bearings. In the next part of this subsection, I provide an overview of my process of computations. This consist two subsections, one correspond with test problems of special geometry of the

Case	$\lvert\boldsymbol{u}_{input\ mean}\rvert$ [m/s]	h [μm]	μ [Ns/m^2]	$\lvert\boldsymbol{u}_{bottom\ wall}\rvert$ [m/s]
1	0.3	1000	0.809	1
2	0.3	200	0.5	0
3	0.3	50	0.01	1

Table 5.15: Parameters of simulations for each case of the hydrostatic bearings.

bearings, second to realistic geometry of this bearing.

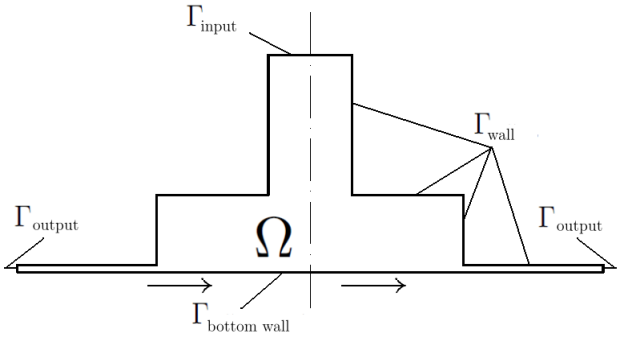

Figure 5.14: A scheme of a cross-section of the solution domain for hydrostatic bearings with boundary conditions.

5.3.1 Test problems

In this section I present simulations of hydrostatic bearings with test geomteries. Input parameters of each simulation, such as the input mean velocity $\boldsymbol{u}_{\text{input mean}}$, the velocity of the bottom wall $\boldsymbol{u}_{\text{bottom wall}}$, the dynamic viscosity μ, and the height of the throttling gap h are summarized in Table 5.15. In all presented simulations, we use the 2-level method with cardinality weights.

5.3.1.1 Case 1

At the beginning of our research, we dealt with simulations of hydrostatic bearings with an artificially magnified throttling gap and geometry axially symmetric along the vertical axis (see Fig. 5.16). However, by considering a linear sliding of the bearing, the boundary conditions lack axial symmetry, and the problem can no longer be solved as axially symmetric. From the beginning, we aimed at reaching the real-scale height of the throttling gap. Unfortunately, we were able to reach only a throttling gap of the height 1 mm, which is 10 times more than the real gap, but it still gave us an initial insight into the flow behaviour during the movement of the bearing. However, convergence deteriorated quickly with decreasing the throttling gap height. For these calculations, we have created meshes in the Gmsh software [32] and divided them into 32 subdomains using the METIS partitioner. The problem contains almost 609 thousand unknowns with 93 thousand unknowns at the interface. Our simulations were performed on the SGI Altix supercomputer at the supercomputing center of the Czech Technical University in Prague using 32 cores of Intel Xeon 2.66 GHz processors. Precision for Picard's iteration was set to $\epsilon_N \leq 10^{-5}$, and the linear iterations were terminated when $\epsilon_L \leq 10^{-6}$ or after reaching the maximum number of 100 iterations. This set of prescribed precisions was used for all presented calculations of hydrostatic bearings.

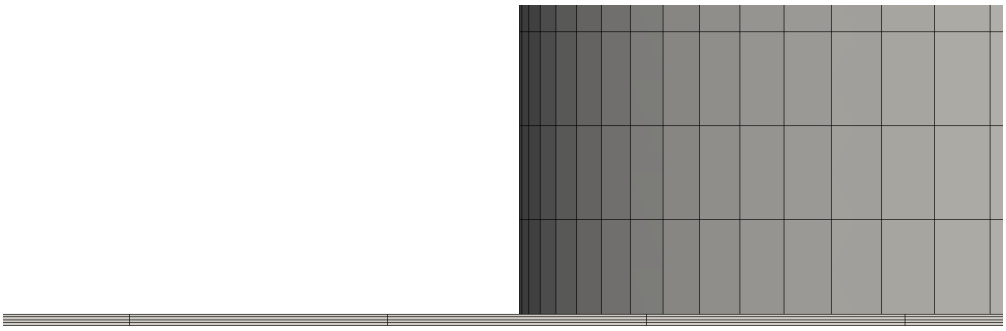

Figure 5.15: Detail of the elements inside the throttling gap.

The solution required 12 nonlinear iterations, each of them performing in average 511 linear iterations. The solution is presented in Fig. 5.16.

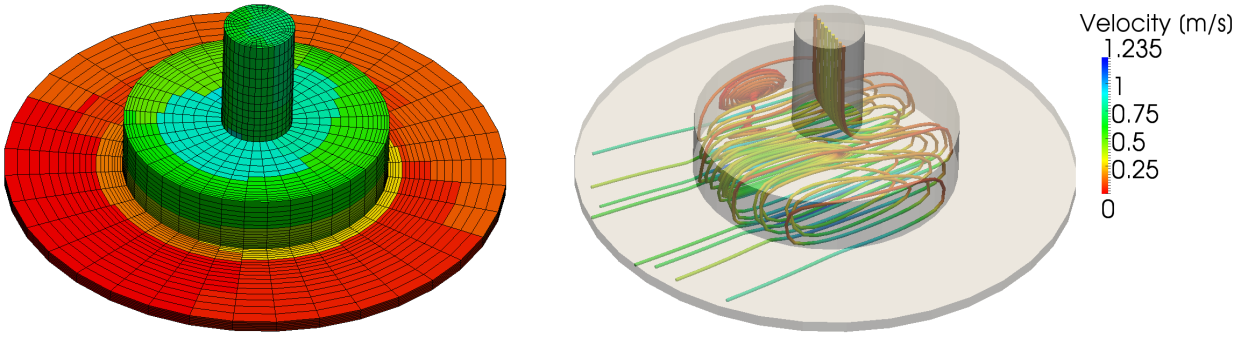

Figure 5.16: Case 1. The solution domain decomposed into 32 subdomains (left) and the streamtraces coloured by the magnitude of velocity (right).

5.3.1.2 Case 2

In the next phase, we have switched to a rectangular geometry of the hydrostatic bearing and tried to solve the throttling-gap-height issue by using a very fine mesh and decomposition into hundreds of subdomains (precisely 1200), still using the METIS partitioner. We were able to reach the height of 100 μm of the throttling gap, which is the real height for certain cases, but only without the motion of the bearing. Hence, this solution has given us a better idea about the pressure values and development inside the hydrostatic bearing. The mesh was also generated in the Gmsh software. It has about 13 million unknowns with 2.7 million unknowns at the interface. The simulations were performed on the *Salomon* supercomputer using 1200 cores. Precisions of the iterations were set as above. The presented calculation converged after 3 nonlinear iterations with an average of 754 linear iterations for each of them. The solution is shown in Fig. 5.17. Recall that there is no sliding considered in this case.

5.3.1.3 Case 3

Next, we studied the influence of the subdomain interface on the convergence of the BDDC solver. The results were presented in detail in [1]. Here, we just briefly recall the main findings for completeness. The study has shown that if we are able to maintain straight interfaces between subdomains, the solver requires significantly less linear iterations, and we

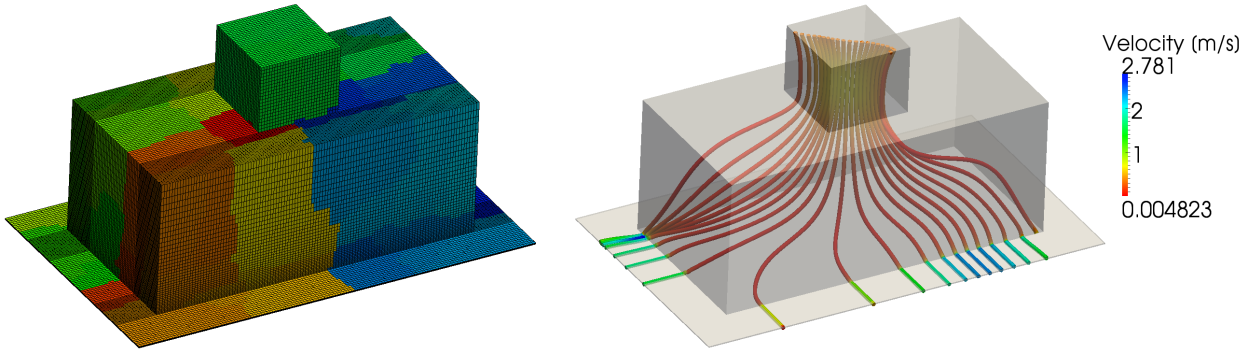

Figure 5.17: Case 2. The solution domain decomposed into 1200 subdomains (left) and the streamtraces coloured by the magnitude of velocity (right).

were able to simulate the problem with the real geometry of the hydrostatic bearing. The stopping criteria were set as in the previous cases. The computational mesh contains about 500 thousand unknowns with 56 thousand unknowns at the interface. The mesh was divided using an in house partitioner maintaining straight cuts as in Fig. 5.18. These computations were performed on the Altix supercomputer using 32 cores. The method converged after 5 nonlinear iterations with an average of 271 linear iterations. The solution is shown in Fig. 5.18.

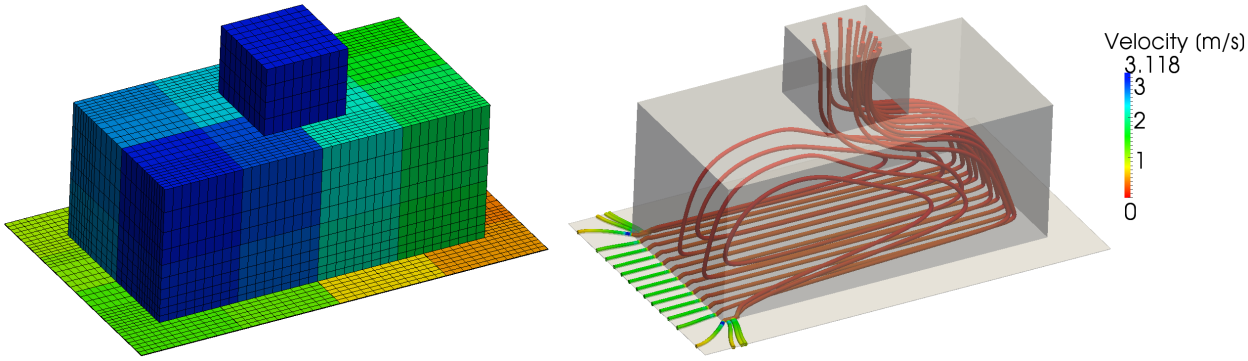

Figure 5.18: Case 3. The computational mesh decomposed into 32 subdomains (left) and streamtraces coloured by the magnitude of velocity (right).

5.3.2 Bearing with realistic geometry

Finally, utilizing the findings from the previous experiments, we were able to perform simulations on a real geometry of the bearing from a production machine. Here I recall our results from [3]. Our calculations aim at an industrial problem of oil flow inside hydrostatic bearings. In this section, I present results for two kinds of problems. The first problem is without considering the motion of the bearing. For this setup, we have performed an experiment to validate the numerical solver. The second problem is a bearing sliding in a straight direction, and this set-up corresponds to a running production machine. The solution domain with its dimensions is depicted in Figure 5.19, and the two studied cases correspond to prescribing zero and non-zero velocity at the bottom side of the domain (see Figure 5.14).

The computational mesh was created and divided into specified structured volumes using the *GMSH* software. These volumes preserve flat interfaces between each other to make it

possible to decompose the solution domain into 21 global subdomains using our 'geometric' partitioner prefers straight cuts between subdomains described in [1]. The mesh consists of 124 828 elements which correspond with 3 269 679 unknowns and 186 834 interface unknowns. The decomposed mesh is presented in Figure 5.20. A detail of the mesh in the throttling gap, where elements with a high aspect ratio occur, is in Figure 5.15.

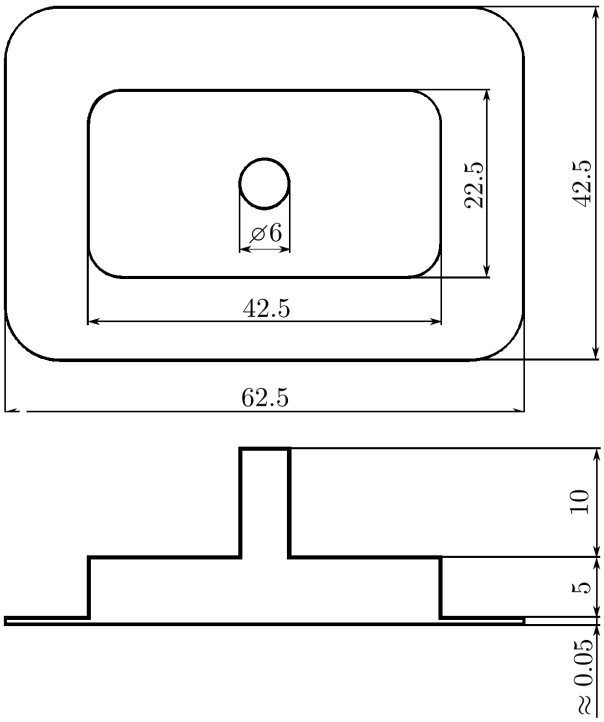

Figure 5.19: Dimensions of the hydrostatic cell in millimeters.

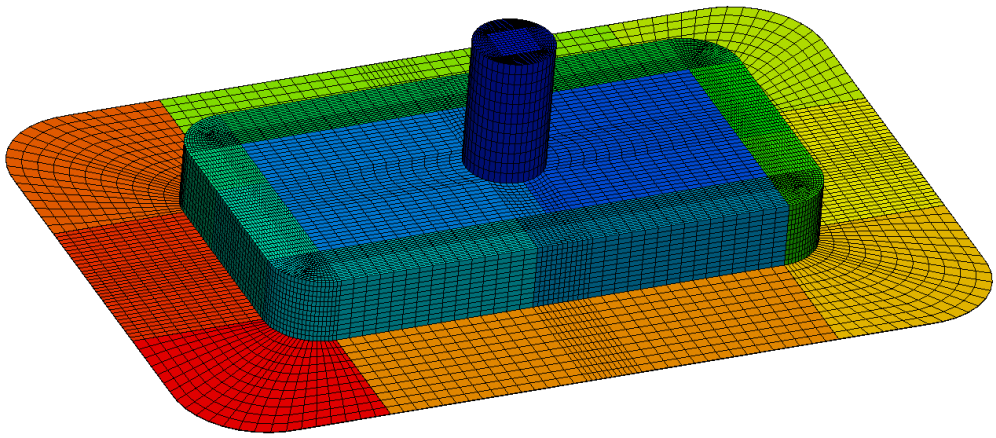

Figure 5.20: Computational mesh decomposed into 21 subdomains.

5.3.2.1 Bearing without motion

At first we dealt with the problem without sliding of the bottom wall. We have been able to validate our calculation for this case. The experiment was performed at the Research Center of Manufacturing Technology at the Faculty of Mechanical Engineering of the Czech Technical University in Prague. A scheme of the measurement is shown in Figure 5.21.

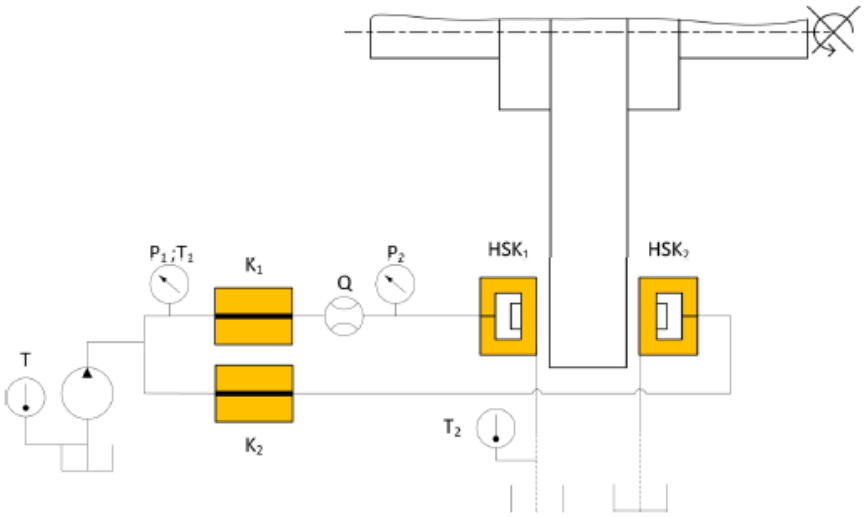

Figure 5.21: Scheme of the measurement.

In this experiment, a throttling gap h of the hydrostatic bearing (HSK), the flow rate Q through the selected branch of the circuit, and the temperature T_2 of the oil Fuchs Renolin B 46 HVI were measured. The mean inflow velocity v_{mid} is then computed from Q. The last input parameter is the dynamic viscosity μ, which we derive from the measured temperature T_2 using tables provided with the oil. For comparing our simulations with the experiment, we use the value of pressure at the entrance of the hydrostatic cell p_2. The values obtained by the experiment are summarized in Table 5.16.

v_{mid} [m/s]	h [μm]	μ [Ns/m^2]	p_2 [MPa]
0.44	61.3	0.0712	2.7124

Table 5.16: Summary of the measured parameters used for the simulation. The measured value of p_2 is compared with the one from the simulation.

I set the parabolic velocity profile with the prescribed mean velocity v_{mid} on the circular input. The velocity of sliding $v_{\text{bottom wall}}$ is set to zero in this case. The oil flows into the atmospheric pressure at the edge of the throttling gap Γ_{output}. Hence, after the computation, we post-process the field of pressure in the whole domain by adding this constant value 100 000 Pa.

For this realistic bearing, I present the detailed results of pressure and velocity fields. In this case, 3 Picard's iterations were needed to reach precision $\epsilon_N \leq 10^{-4}$, and on average 798 BiCGstab iterations were needed for the linearised system to achieve the precision of the relative residual $\epsilon_L \leq 10^{-5}$.

First, we look at the pressure field inside the hydrostatic bearing in Figures 5.22–5.24.

We can see from the results that the pressure inside the hydrostatic cell (and therefore at the input) is almost constant and the pressure drop is realized in the throttling gap and the pressure gradient inside the throttling gap is symmetric. The value of pressure at the entrance of the hydrostatic cell p_2 obtained from our calculation is 2.826 MPa, which presents only 4% difference from the measured value 2.7124 MPa. This is an encouraging agreement within the accuracy of our experimental set-up.

Now we explore the velocity field inside the hydrostatic cell. In Figures 5.25–5.28 there are streamtraces in different parts of the bearing coloured by the velocity magnitude.

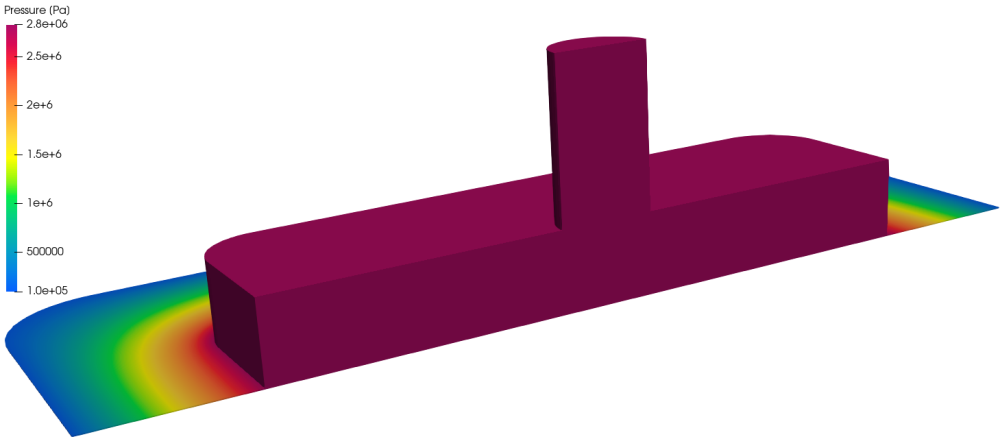

Figure 5.22: Pressure field in half of the hydrostatic bearing without motion.

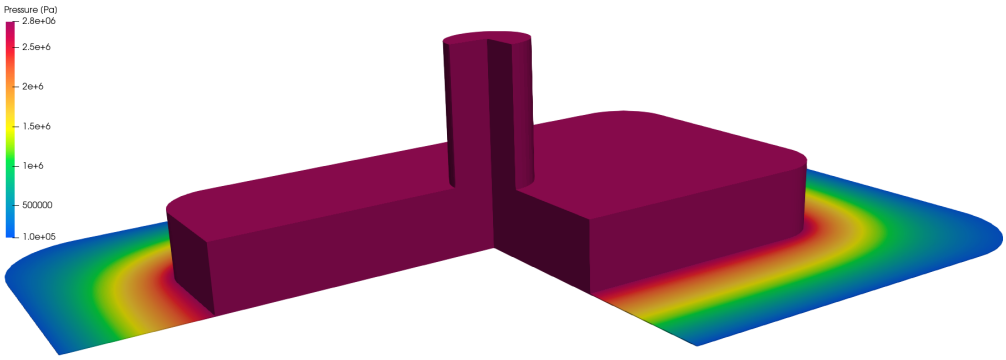

Figure 5.23: Pressure field in three quarters of the hydrostatic bearing without motion.

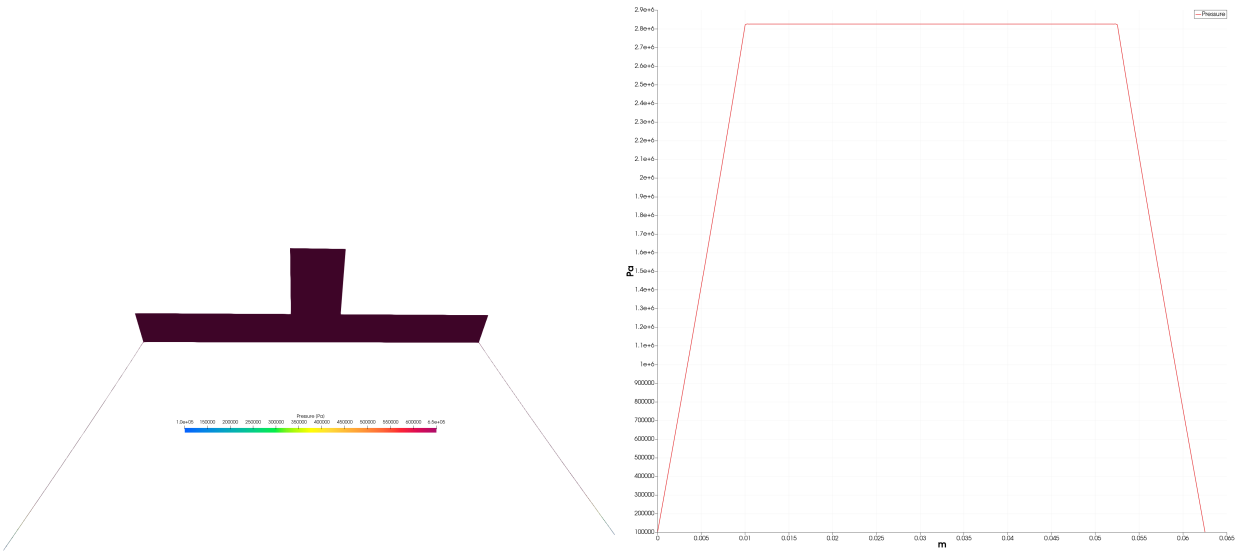

Figure 5.24: The pressure in the hydrostatic bearing without motion. Warped pressure field (left) and plot (right).

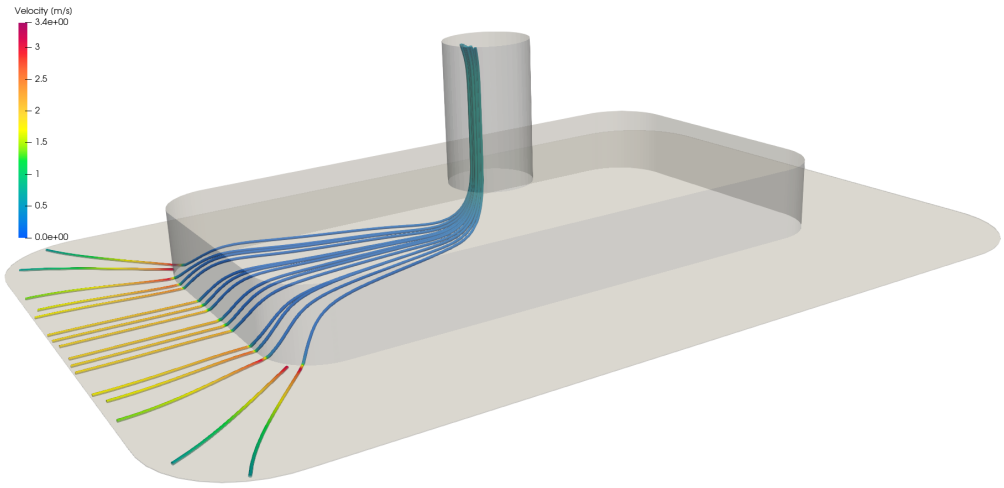

Figure 5.25: Streamtraces coloured by the magnitude of velocity in the hydrostatic bearing without motion along the short edge in negative x direction.

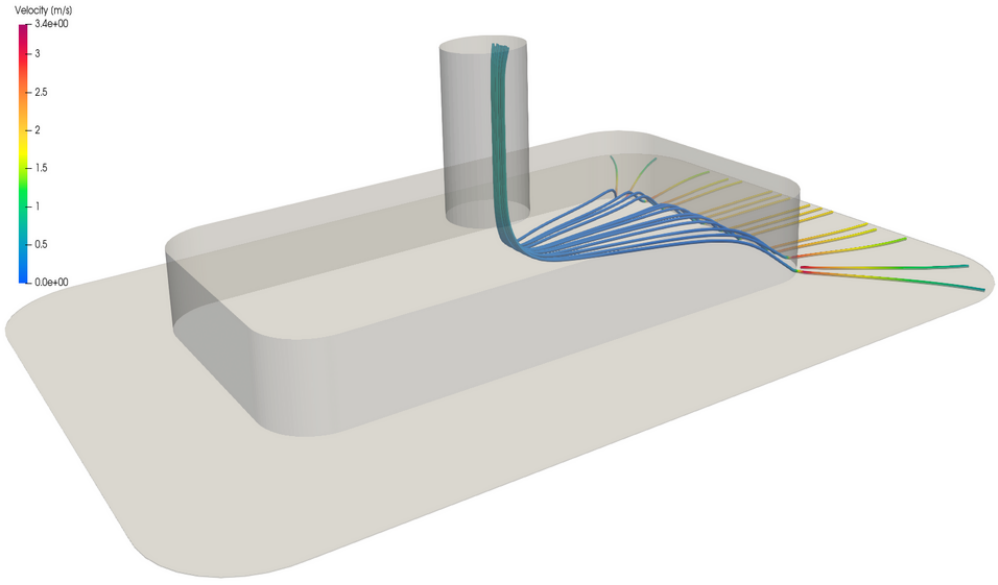

Figure 5.26: Streamtraces coloured by the magnitude of velocity in the hydrostatic bearing without motion along the short edge in positive x direction.

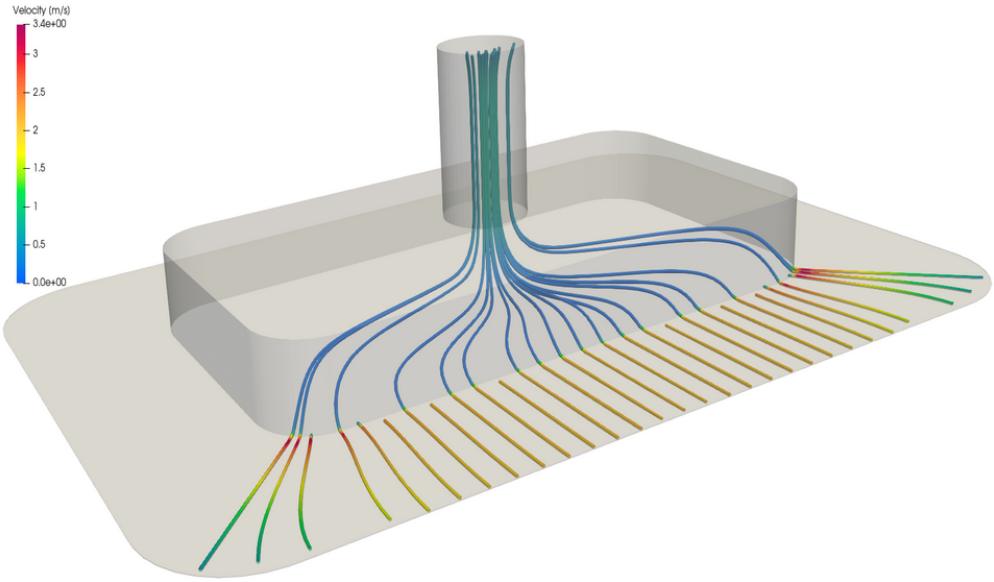

Figure 5.27: Streamtraces coloured by the magnitude of velocity in the hydrostatic bearing without motion along the long edge.

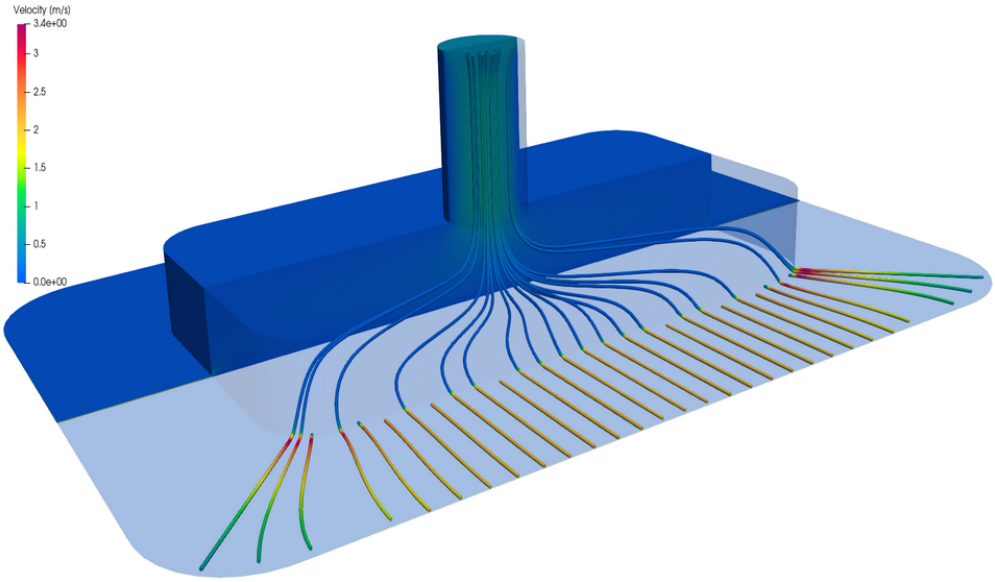

Figure 5.28: Streamtraces coloured by the magnitude of velocity in the hydrostatic bearing without motion along the long edge with velocity field.

We can see that for the case without the motion of the bearing, the oil flows out equally along the edges of the throttling gap and the magnitude of the velocity is maximal at the entrance to the throttling gap and moreover in that part, where is the distance from the entrance to the output also maximal, therefore in the corners of the hydrostatic cell. Next, we at look what is happening inside the hydrostatic cell, namely if there is some vortex generated. In Figures 5.29–5.37, there are slices of the velocity fields inside the bearing with streamtraces going from the center of the hydrostatic cell towards the long edge, therefore with increasing y coordinate, together with joint pictures for better understanding of the positions of individual slices.

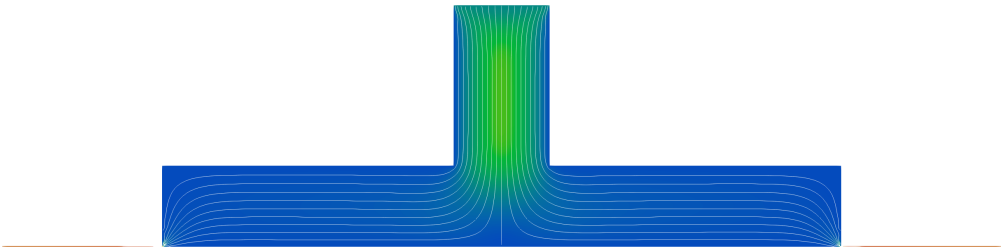

Figure 5.29: Slice of the velocity field with streamtraces for $y = 0$ m for the hydrostatic bearing without motion.

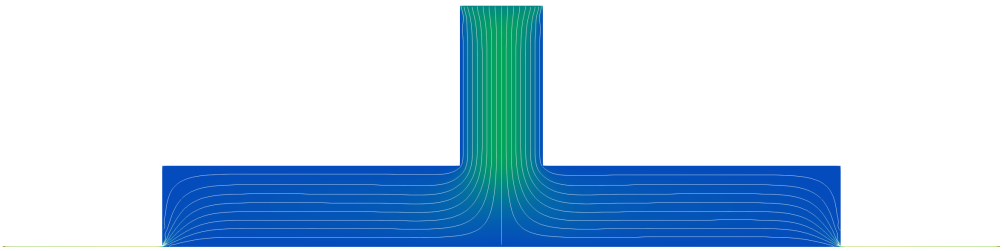

Figure 5.30: Slice of the velocity field with streamtraces for $y = 0.0015$ m for the hydrostatic bearing without motion.

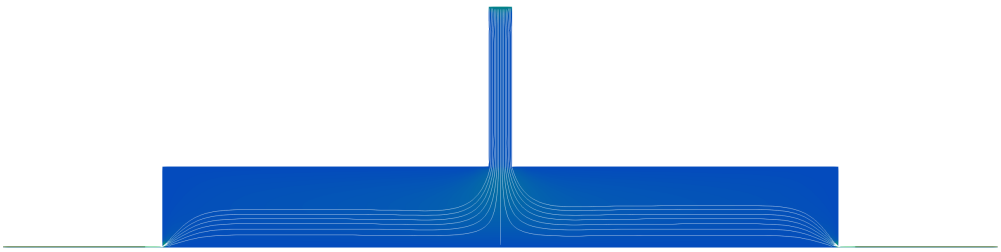

Figure 5.31: Slice of the velocity field with streamtraces for $y = 0.0029$ m for the hydrostatic bearing without motion.

From these results, we can see that there are no vortex generated, flow is distributed equally and symmetrically, and as getting closer to the side-wall of the hydrostatic cell, the flow is pushed more to the upper wall of the bearing.

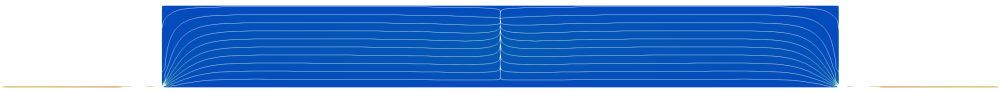

Figure 5.32: Slice of the velocity field with streamtraces for $y = 0.005$ m for the hydrostatic bearing without motion.

Figure 5.33: Slice of the velocity field with streamtraces for $y = 0.007$ m for the hydrostatic bearing without motion.

Figure 5.34: Slice of the velocity field with streamtraces for the $y = 0.009$ m for the hydrostatic bearing without motion

Figure 5.35: Slice of the velocity field with streamtraces for $y = 0.011$ m for the hydrostatic bearing without motion.

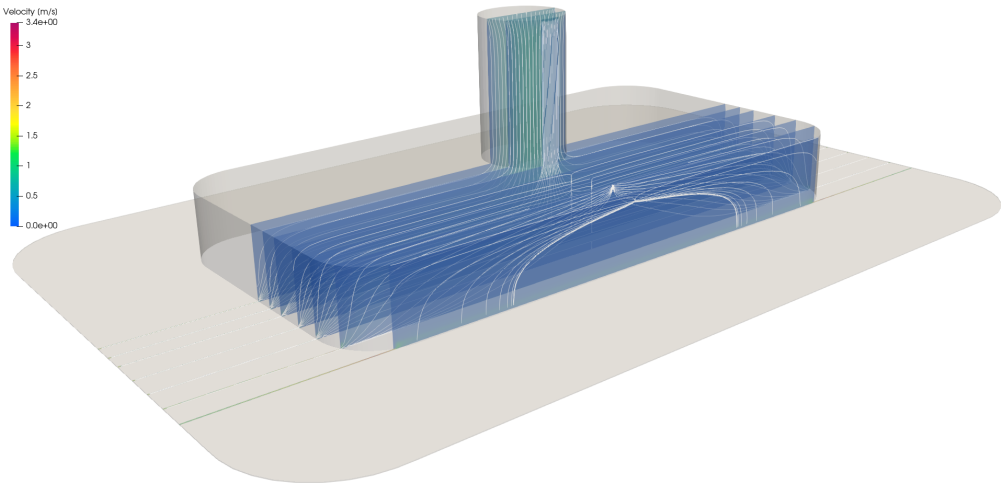

Figure 5.36: Joint slices of the velocity field with streamtraces for the hydrostatic bearing without motion – rear view.

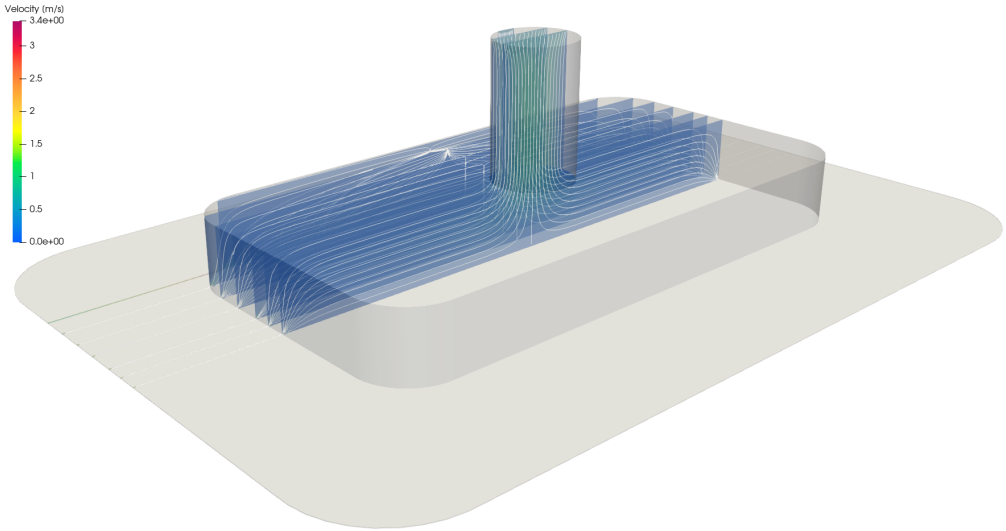

Figure 5.37: Joint slices of the velocity field with streamtraces for the hydrostatic bearing without motion – front view.

5.3.2.2 Bearing with sliding

After a successful validation of the previous problem, we consider the problem of a sliding bearing, which corresponds to the operational conditions of the machine. The boundary conditions are the same as in the previous case except for the velocity of the bottom wall, which is set to $v_{\text{bottom wall}} = 1$ m/s. The input parameters for this computation are summarized in Table 5.17.

v_{mid} [m/s]	h [μm]	μ [Ns/m^2]
0.3	50	0.1

Table 5.17: Parameters for the simulation of the sliding bearing.

For this case, we needed 3 Picard's iterations to reach precision $\epsilon_N \leq 10^{-4}$, while on average 748 BiCGstab iterations were needed for the linearised system to achieve the precision of the relative residual $\epsilon_L \leq 10^{-5}$. Here I also present detailed results of pressure and velocity fields, starting with the pressure field inside moving hydrostatic bearing in Figures 5.38–5.40.

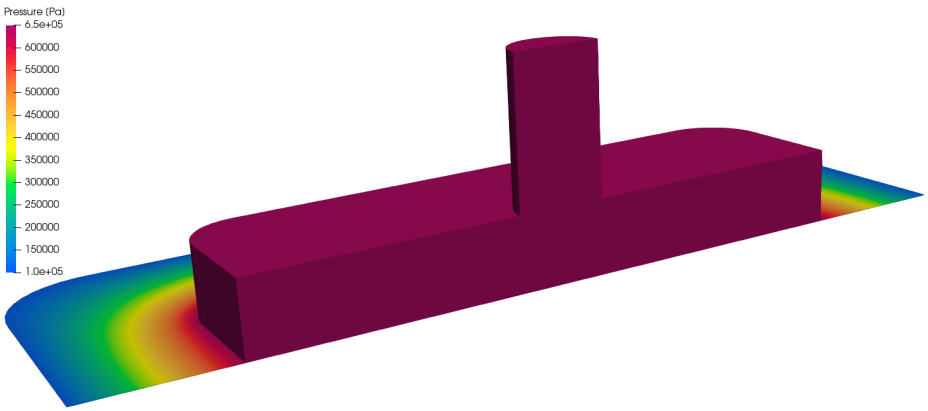

Figure 5.38: Pressure field in half of the sliding hydrostatic bearing.

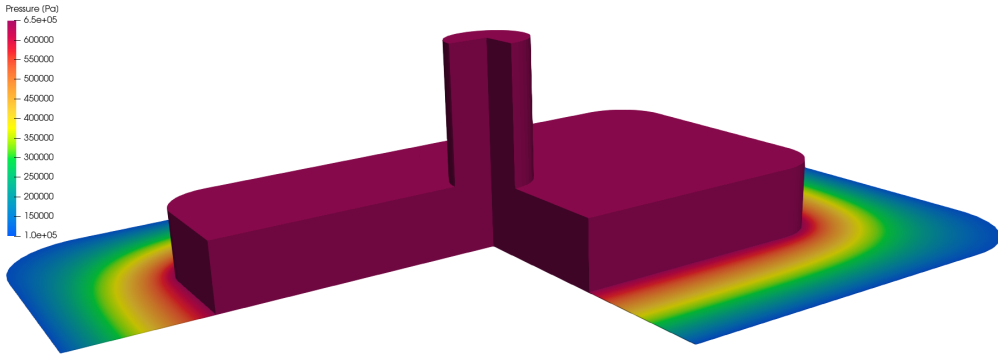

Figure 5.39: Pressure field in three quarters of the sliding hydrostatic bearing.

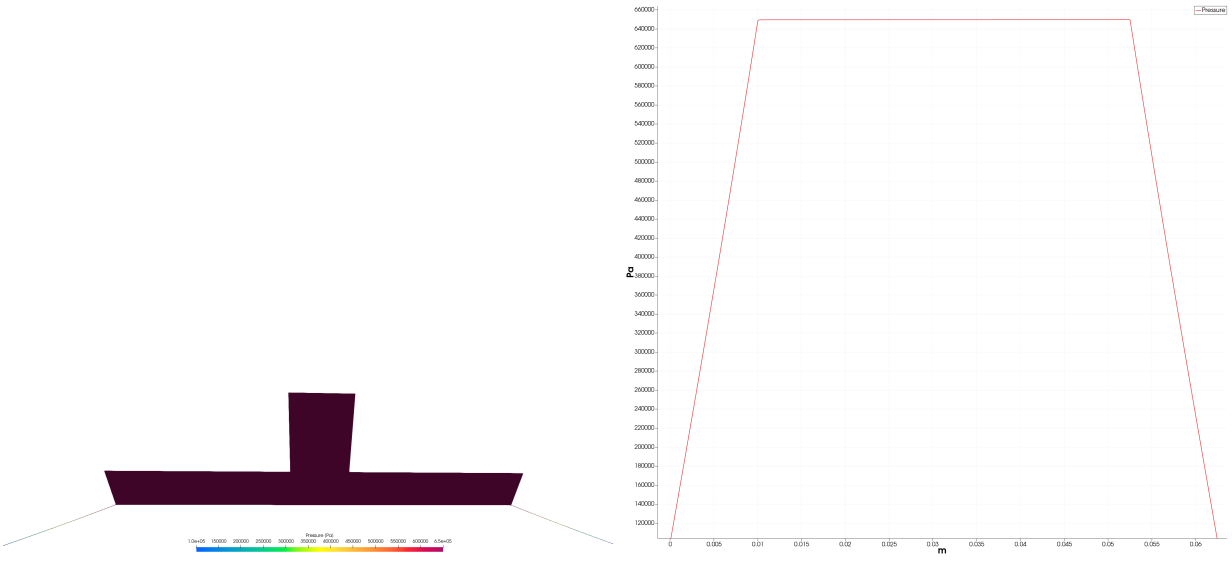

Figure 5.40: The pressure in the sliding hydrostatic bearing. Warped pressure field (left) and plot (right).

We can see from the results that the pressure field is almost indistinguishable from the case without motion, with the pressure drop realized only in the throttling gap, and the almost constant pressure in the rest of the hydrostatic cell. Also, the pressure gradient inside the throttling gap is symmetric.

Now we explore the velocity field inside the hydrostatic cell. In Figures 5.41–5.44 there are streamtraces in different parts of the bearing coloured by the velocity magnitude.

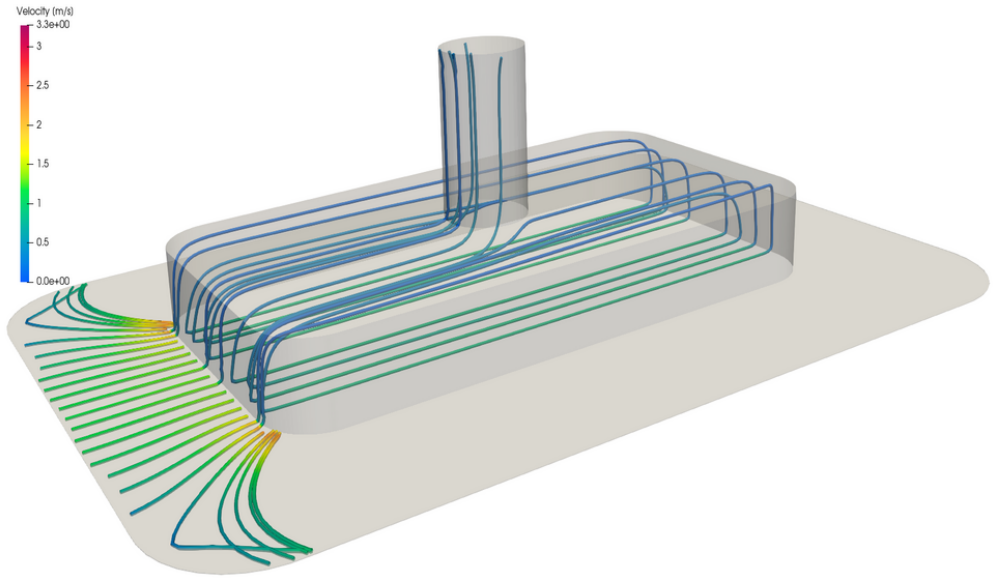

Figure 5.41: Streamtraces coloured by the magnitude of velocity in the sliding hydrostatic bearing along the short edge in negative x direction.

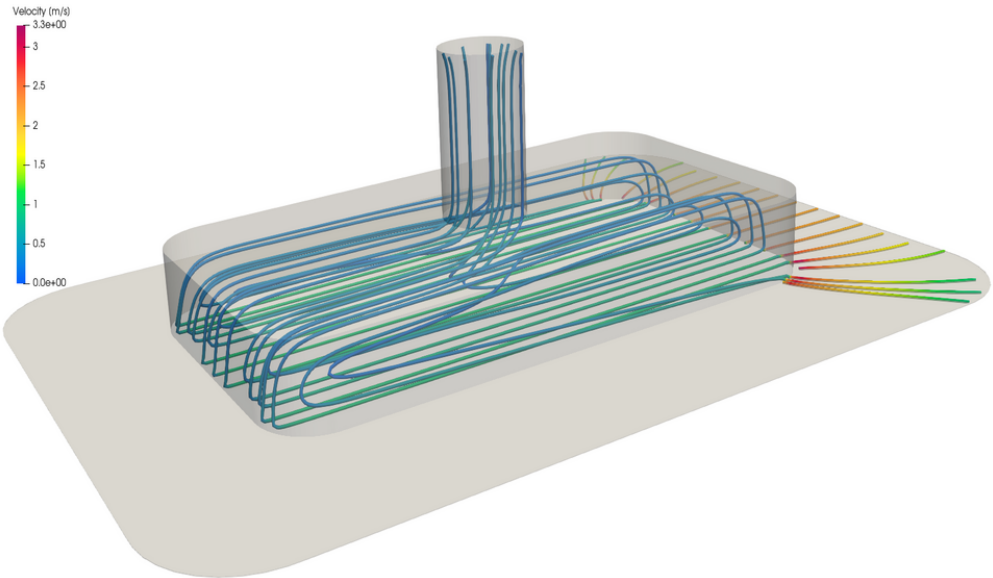

Figure 5.42: Streamtraces coloured by the magnitude of velocity in the sliding hydrostatic bearing along the short edge in positive x direction.

We can see that for the case with the motion of the bearing, the velocity field changes dramatically. We can see that a large vortex rolling inside of the hydrostatic cell is formed, and the oil is pulled by the motion of the bottom wall towards one side of the throttling

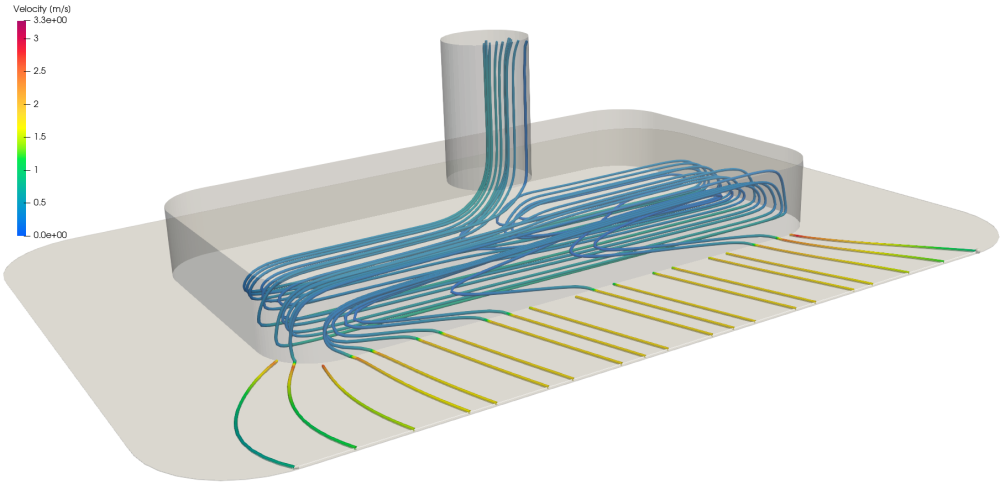

Figure 5.43: Streamtraces coloured by the magnitude of velocity in the sliding hydrostatic bearing along the long edge.

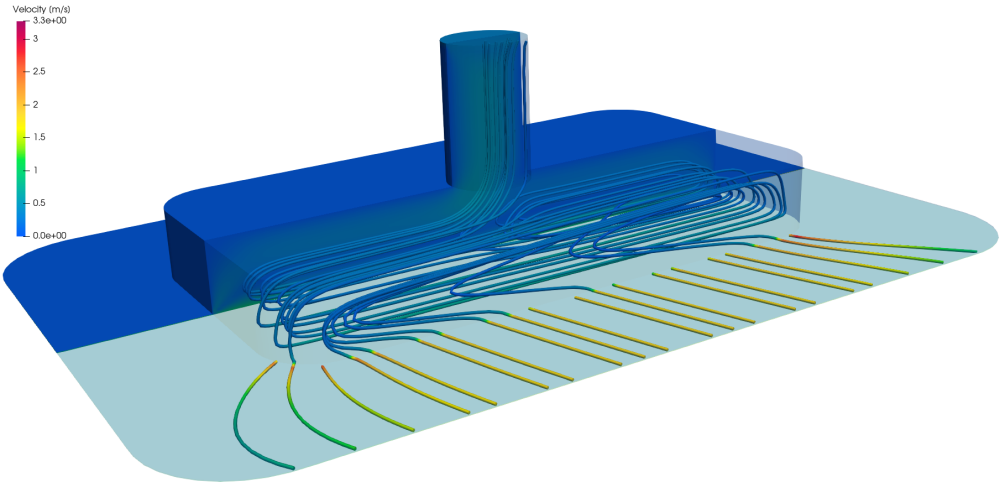

Figure 5.44: Streamtraces coloured by the magnitude of velocity in the sliding hydrostatic bearing along the long edge with velocity field.

gap. This results in the fact that the maximal magnitude of the velocity of the oil is at the entrance to the throttling gap in the side which corresponds with the movement of the hydrostatic bearing. The latter effect is important for setting the operational regime of the machine, especially the maximum velocity of the sliding such that the oil flow towards the front of the bearing is still maintained.

Next, we look, once again, what is happening inside the hydrostatic cell, namely if there are some vortices generated. In Figures 5.45–5.53, there are slices of the velocity fields inside the slicing bearing with streamtraces going from the center of the hydrostatic cell towards the long edge, therefore with increasing y coordinate, together with joint pictures for better understanding of the positions of individual slices.

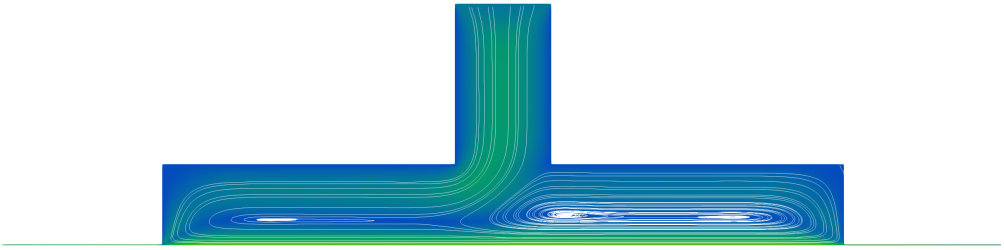

Figure 5.45: Slice of the velocity field with streamtraces for $y = 0$ m for the sliding hydrostatic bearing.

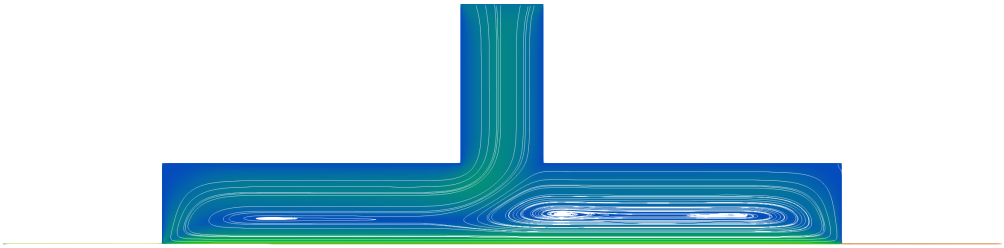

Figure 5.46: Slice of the velocity field with streamtraces for $y = 0.0015$ m for the sliding hydrostatic bearing.

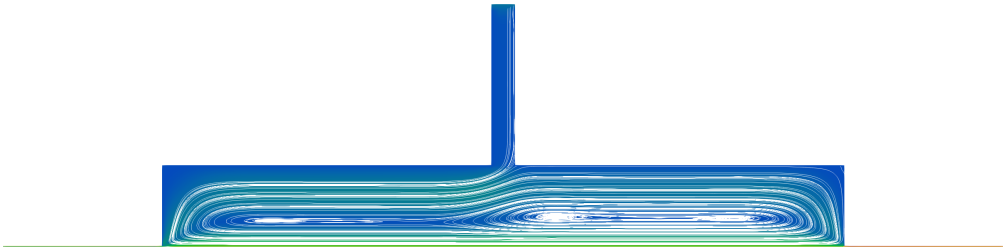

Figure 5.47: Slice of the velocity field with streamtraces for $y = 0.0029$ m for the sliding hydrostatic bearing.

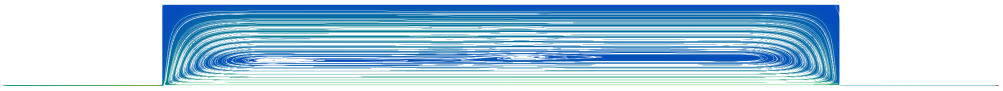

Figure 5.48: Slice of the velocity field with streamtraces for $y = 0.005$ m for the sliding hydrostatic bearing.

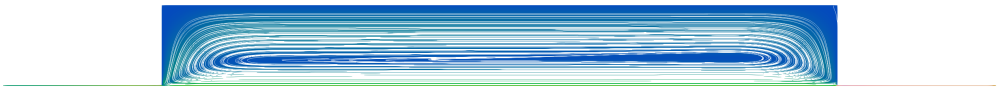

Figure 5.49: Slice of the velocity field with streamtraces for $y = 0.007$ m for the sliding hydrostatic bearing.

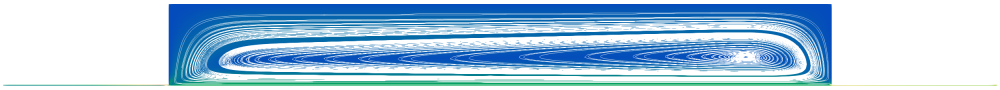

Figure 5.50: Slice of the velocity field with streamtraces for $y = 0.009$ m for the sliding hydrostatic bearing.

Figure 5.51: Slice of the velocity field with streamtraces for $y = 0.011$ m for the sliding hydrostatic bearing.

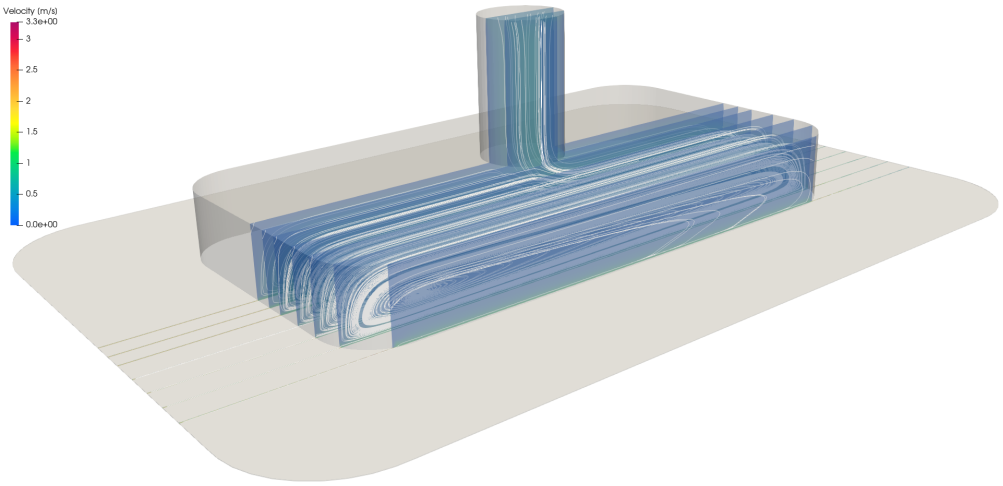

Figure 5.52: Joint slices of the velocity field with streamtraces for the sliding hydrostatic bearing – rear view.

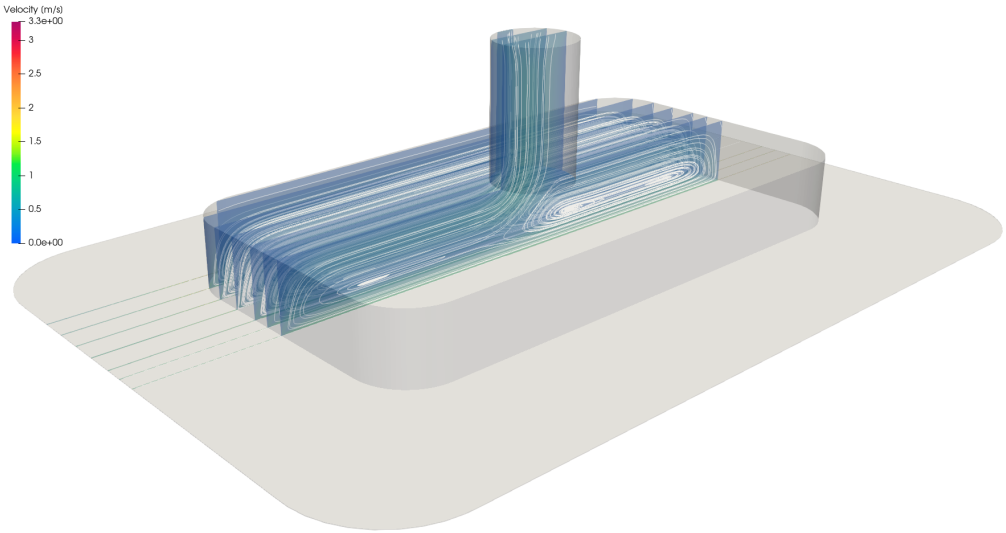

Figure 5.53: Joint slices of the velocity field with streamtraces for the sliding hydrostatic bearing – front view.

From these results, we can see that due to the movement of the hydrostatic bearing, vortices are generated. Moreover, as we move towards the side-wall, the number of vortices is decreasing and in the end vanish. The centers of the vortices are changing with the changing y coordinate. This knowledge and a look inside into the hydrostatic cell is valuable in the design of the hydrostatic bearings.

Chapter 6

Conclusions

In today's computation, a high resolution large-scale simulations are necessary for many engineering applications. This is allowed by a suitable mathematical methods, namely, the domain decomposition methods. Using this approach, it is possible to perform many kinds of parallel simulations. Therefore, fast and effective parallel solvers for computational fluid dynamics play an important role in nowadays research.

In this thesis, the mathematical model of flow of a fluid described by steady incompressible Navier-Stokes equations is considered. These equations are discretized by the finite element method and the arising nonlinear systems are linearized by Picard's iteration. Details of this approach are described in Chapter 2. The system of linear equations is solved using a nonoverlapping domain decomposition method by means of iterative substructuring. An overview of DD methods aimed at nonoverlapping methods, especially at the BDDC method, is presented in Chapter 3. The arising nonsymmetric interface problem is solved by the BiCGstab method preconditioned by one step of BDDC and its multilevel variant. This novel approach of using multilevel BDDC for nonsymmetric systems is described in Chapter 4. In addition, various weight types and used partitioners are discussed in this chapter. Computations are performed for several problems, and they are summarized in Chapter 5. I have compared the weak scalability of 2-, 3-, and 4-level BDDC methods for 3D lid-driven cavity, compared two types of partitioners of the solution domain, compared four different weights on the interface including my own upwind based, and presented the chronology of simulating the flow of oil inside the hydrostatic bearing ending with real-scale geometry simulations. Now I recall the main original results and achievements.

By combining the approaches from [55] and [64], I applied the 2-level BDDC method to the Navier-Stokes equations for the lid-driven cavity benchmark problem in [2]. Next, I have presented a formulation of the multilevel BDDC preconditioner for nonsymmetric problems and its application to linear systems obtained by Picard's linearization of the Navier-Stokes equations in a journal paper [4]. My computations employed the *BDDCML* solver and a parallel finite element package written in C++ described in [54], which I have extended towards the flow problems related to my thesis.

The tests of the influence of the interface between subdomains were performed for two partitions in [1]. The first was created by a partitioner based on the graph partitioning implemented in the *METIS* library. The second one was my own partitioner based on geometry and division of the domain with straight cuts promoting just a 2D interface between individual subdomains. From the comparison of the number of linear iterations with a growing aspect ratio of the finite elements, one can see that using the new geometric partitioner with straight cuts significantly decreases the number of linear iterations with increasing the aspect ratio of the elements.

For a benchmark of 3-D lid-driven cavity problem, I have explored the behaviour of the 2-, 3-, and 4-level BDDC method. I have focused mainly on the number of linear iterations and the mean times for the setup of the preconditioner and for the iterations of the Krylov subspace method. The performance was tested on up to 5 thousand CPU cores. The 3-level method has shown remarkable speedup compared to the 2-level method, especially for large numbers of subdomains where the coarse problem significantly grows. However, switching to the 4-level method has not brought us an overall improvement. Although each iteration was cheaper than in the 3-level case, the method has required a very large number of iterations and thus performed even worse than the 2-level method. For this problem, I also introduced my software for building the computational mesh by individual subdomains which was necessary, especially for large meshes.

A new type of weights inspired by numerical schemes for flow problems has been proposed. This upwind-based scaling has been compared with three other suitable interface scaling types. My results have suggested that with a growing Reynolds number, the importance of the scaling type increases, and the upwind weights provide promising results as presented in [4].

In [4], I have performed experiments investigating the behaviour of the 2-level and 3-level BDDC method with respect to the H/h ratio. Although theoretical estimates are not available for this class of problems, the numbers of BiCGstab iterations seem to confirm the logarithmic behaviour has proven for symmetric positive definite problems.

Finally, I have applied the BDDC methodology to an industrial problem in engineering, namely, the flow of oil in hydrostatic bearings. I have presented the gradual development of my approach to these simulations. By solving a number of issues, I have managed to perform challenging computations on a real geometry of the bearing by means of the 2-level BDDC method presented in [3].

Several topics are left for future investigation. A better understanding of the effect of forming subdomains on higher levels is still required, especially for applying the multilevel BDDC method to the hydrostatic bearing problem. This could also help to explain the rapidly worsening behaviour of the 4-level method for the cavity problem. More experiments are also required for confirming the benefits of the upwind-based interface scaling.

References

Author's publications

[1] M. Hanek, J. Šístek, and P. Burda. The effect of irregular interfaces on the BDDC method for the Navier-Stokes equations. In Ch.-O. Lee, X.-Ch. Cai, D. E. Keyes, H. H. Kim, A. Klawonn, E.-J. Park, and O. B. Widlund, editors, *Domain Decomposition Methods in Science and Engineering XXIII*, pages 171–178. Springer International Publishing AG, 2017.

[2] M. Hanek, J. Šístek, and P. Burda. An application of the BDDC method to the Navier-Stokes equations in 3-D cavity. In J. Chleboun, P. Přikryl, K. Segeth, J. Šístek, and T. Vejchodský, editors, *Programs and algorithms of numerical mathematics 17*, pages 77–85. Institute of Mathematics AS CR, 2015.

[3] M. Hanek, J. Šístek, P. Burda, and E. Stach. Parallel domain decomposition solver for flows in hydrostatic bearings. In D. Šimurda and T. Bodnár, editors, *Topical Problems of Fluid Mechanics 2018*, pages 137–144. Institute of Thermomechanics AS CR, 2018.

[4] M. Hanek, J. Šístek, and P. Burda. Multilevel BDDC for incompressible Navier-Stokes equations. *SIAM J. Sci. Comput.*, 42(6):C359–C383, 2020.

Bibliography

[5] Y. Achdou, P. Le Tallec, F. Nataf, and M. Vidrascu. A domain decomposition preconditioner for an advection–diffusion problem. *Comput. Methods Appl. Mech. Engrg.*, 184(2):145 – 170, 2000.

[6] Y. Achdou and F. Nataf. A Robin-Robin preconditioner for an advection-diffusion problem. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 325(11):1211 – 1216, 1997.

[7] S. Badia, F. Martín, A., and J. Principe. Multilevel balancing domain decomposition at extreme scales. *SIAM J. Sci. Comput.*, 38(1):C22–C52, 2016.

[8] L. Beirão da Veiga, S. Pavarino, L., S. Scacchi, B. Widlund, O., and S. Zampini. Isogeometric BDDC preconditioners with deluxe scaling. *SIAM J. Sci. Comput.*, 36(3):A1118–A1139, 2014.

[9] M. Bhardwaj, D. Day, Ch. Farhat, M. Lesoinne, K. Pierson, and D. Rixen. Application of the feti method to ASCI problems: Scalability results on the one thousand processors and discussion of higly heterogenous problems. *Internat. J. Numer. Methods Engrg.*, 47:513–535, 2000.

[10] D. Boffi, F. Brezzi, and M. Fortin. *Mixed Finite Element Methods and Applications*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2013.

[11] J.-F. Bourgat, R. Glowinski, P. Le Tallec, and M. Vidrascu. Variational formulation and algorithm for trace operator in domain decomposition calculations. *Domain Decomposition Methods. Second International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 3–16, 1989.

[12] S. C. Brenner. The condition number of the Schur complement in domain decomposition. *Numer. Math.*, 83:187–203, 1999.

[13] S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Method*. Springer-Verlag. Springer-Verlag New York, 1994.

[14] S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Methods*, volume vol. 15 of Texts in Applied Mathematics of *second ed.* Springer-Verlag, New York, 2002.

[15] F. Brezzi and M. Fortin. *Mixed and Hybrid Finite Element Method*. Springer-Verlag. Springer-Verlag New York, 1991.

[16] M. Čertíková, J. Šístek, and P. Burda. Different approaches to interface weights in the BDDC method in 3D. In J. Chleboun, P. Přikryl, K. Segeth, J. Šístek, and T. Vejchodský, editors, *Programs and algorithms of numerical mathematics 17*, pages 47–57. Institute of Mathematics AS CR, 2015.

[17] P. Concus, G. H. Golub, and D. P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic PDE. *Sparse Matrix Computations*, pages 309–332, 1976.

[18] Y.-H. De Roeck. Resolution sur ordinateurs multi-processeurs de probleme d'elasticite par decomposition de domaines. *PhD thesis*, 1991.

[19] Y.-H. De Roeck and P. Le Tallec. Analysis and test of a local domain decomposition preconditioner. *Four International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 112–128, 1991.

[20] C. R. Dohrmann and O. B. Widlund. A BDDC algorithm with deluxe scaling for three-dimensional H(curl) problems. *Comm. Pure Appl. Math.*, 69(4):745–770, 2016.

[21] C. R. Dohrmann. A preconditioner for substructuring based on constrained energy minimization. *SIAM J. Sci. Comput.*, 25(1):246–258, 2003.

[22] C. R. Dohrmann. A study of domain decomposition preconditioners. *Technical Report SAND2003-4391*, 2003.

[23] C. R. Dohrmann and O. B. Widlund. Some recent tools and a BDDC algorithm for 3D problems in H(curl). In Randolph Bank, Michael Holst, Olof Widlund, and Jinchao Xu, editors, *Domain Decomposition Methods in Science and Engineering XX*, volume 91 of *Lecture Notes in Computational Science and Engineering*, pages 15–25. Springer, 2013.

[24] M. Dryja, M. V. Sarkis, and O. B. Widlund. Multilevel Schwarz methods for elliptic problems with discontinuous coefficients in three dimensions. *Numer. Math.*, 72(3):313–348, 1996.

[25] M. Dryja and O. B. Widlund. Schwarz methods of Neumann-Neumann type for three-dimensional elliptic finite element problems. *Comm. Pure Appl. Math.*, 48(2):121–155, 1995.

[26] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics.* Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 2005.

[27] C. Farhat, M. Lesoinne, P. Le Tallec, K. Pierson, and D. Rixen. FETI-DP: a dual-primal unified FETI method. I. A faster alternative to the two-level FETI method. *Internat. J. Numer. Methods Engrg.*, 50(7):1523–1544, 2001.

[28] Ch. Farhat, M. Lesoinne, and K. Pierson. A scalable dual-primal domain decomposition method. *Numer. Linear Algebra Appl.*, 7:687–714, 2000.

[29] Ch. Farhat and F.-X. Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *Internat. J. Numer. Methods Engrg.*, 32(6):1205–1227, 1991.

[30] Ch. Farhat and F.-X. Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *Internat. J. Numer. Methods Engrg.*, 32:1205–1227, 1991.

[31] Ch. Farhat and F.-X. Roux. Implicit parallel processing in structural mechanics. *Computational Mechanics Advances*, 2(1):1–124, 1994.

[32] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Internat. J. Numer. Methods Engrg.*, 79:1309–1331, 2009.

[33] V. Girault and P.-A. Raviart. *Finite element methods for Navier-Stokes equations.* Springer-Verlag, Berlin, 1986.

[34] R. Glowinski and M. F. Wheeler. Domain decomposition and mixed finite element methods for elliptic problems. *Technical Report 87-11*, 1987.

[35] R. Glowinski and M. F. Wheeler. Domain decomposition and mixed finite element methods for elliptic problems. *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 144–172, 1988.

[36] P. M. Gresho and R. L. Sani. *Incompressible Flow and the Finite Element Method.* Chichester. John Wiley & Sons Ltd, 2000.

[37] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[38] A. Klawonn, O. Rheinbach, and O. B. Widlund. An analysis of a FETI-DP algorithm on irregular subdomains in the plane. *SIAM J. Numer. Anal.*, 46(5):2484–2504, 2008.

[39] A. Klawonn, O. B. Widlund, and M. Dryja. Dual-primal FETI methods for three-dimensional elliptic problems with heterogeneous coefficients. *SIAM J. Numer. Anal.*, 40(1):159–179, 2002.

[40] P. Le Tallec. Domain decomposition methods in computational mechanics. *Computational Mechanics Advances*, 1(2):121–220, 1994.

[41] J. Li and X. Tu. A nonoverlapping domain decomposition method for incompressible Stokes equations with continuous pressures. *SIAM J. Numer. Anal.*, 51(2):1235–1253, 2013.

[42] J. Li and O. B. Widlund. BDDC algorithms for incompressible Stokes equations. *SIAM J. Numer. Anal.*, 44(6):2432–2455, 2006.

[43] J. Li and O. B. Widlund. FETI-DP, BDDC, and block Cholesky methods. *Internat. J. Numer. Methods Engrg.*, 66(2):250–271, 2006.

[44] J. Mandel. Balancing domain decomposition. *Comm. Numer. Methods Engrg.*, 9(3):233–241, 1993.

[45] J. Mandel and M. Brezina. Balancing domain decomposition for problems with large jumps in coefficients. *Math. Comp.*, 65:1387–1401, 1996.

[46] J. Mandel and C. R. Dohrmann. Convergence of a balancing domain decomposition by constraints and energy minimization. *Numer. Linear Algebra Appl.*, 10(7):639–659, 2003.

[47] J. Mandel, C. R. Dohrmann, and R. Tezaur. An algebraic theory for primal and dual substructuring methods by constraints. *Appl. Numer. Math.*, 54(2):167–193, 2005.

[48] J. Mandel, B. Sousedík, and C. R. Dohrmann. Multispace and multilevel BDDC. *Computing*, 83(2-3):55–85, 2008.

[49] J. Mandel and R. Tezaur. On the convergence of a dual-primal substructuring method. *Numer. Math.*, 88:543–558, 2001.

[50] C. Pechstein and C. R. Dohrmann. A unified framework for adaptive BDDC. *Electron. Trans. Numer. Anal.*, 46:273–336, 2017.

[51] S. Przemieniecki, J. Matrix structural analysis of substructures. *Am. Inst. Aero. Astro. J.*, 1:138–147, 1963.

[52] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 1991.

[53] M. V. Sarkis. Schwarz preconditioners for elliptic problems with discontinuous coefficients using conforming and non-conforming elements. *PhD thesis*, 1994.

[54] J. Šístek and F. Cirak. Parallel iterative solution of the incompressible Navier-Stokes equations with application to rotating wings. *Computers & Fluids*, 122:165–183, 2015.

[55] J. Šístek, B. Sousedík, P. Burda, J. Mandel, and J. Novotný. Application of the parallel BDDC preconditioner to the Stokes flow. *Computers & Fluids*, 46:429–435, 2011.

[56] B. F. Smith. *Domain decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Numerical Mathematics and Scientific Computation. Cambridge University Press, Cambridge, 1996.

[57] B. Sousedík, J. Šístek, and J. Mandel. Adaptive-Multilevel BDDC and its parallel implementation. *Computing*, 95(12):1087–1119, 2013.

[58] A. Toselli and O. B. Widlund. *Domain Decomposition Methods—Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2005.

[59] X. Tu. Three-level BDDC in three dimensions. *SIAM J. Sci. Comput.*, 29(4):1759–1780, 2007.

[60] X. Tu and J. Li. A balancing domain decomposition method by constraints for advection-diffusion problems. *Commun. Appl. Math. Comput. Sci*, 3(1):25–60, 2008.

[61] X. Tu and J. Li. BDDC for nonsymmetric positive definite and symmetric indefinite problems. In M. Bercovie, M. Gander, R. Kornhuber, and O. Widlund, editors, *Domain Decomposition Methods in Science and Engineering XVIII*, pages 75–86. Springer International Publishing AG, 2009.

[62] H. A. van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13(2):631–644, 1992.

[63] A. J. Wathen, D. Loghin, D. A. Kay, H. C. Elman, and D. J. Silvester. A new preconditioner for the Oseen equations. In F. Brezzi, A. Buffa, S. Corsaro, and A. Murli, editors, *Numerical mathematics and advanced applications*, pages 979–988, Milano, 2003. Springer-Verlag Italia. Proceedings of ENUMATH 2001, Ischia, Italy.

[64] M. Yano. Massively parallel solver for the high-order Galerkin least-squares method. Master's thesis, Massachusests Institute of Technology, 2009.