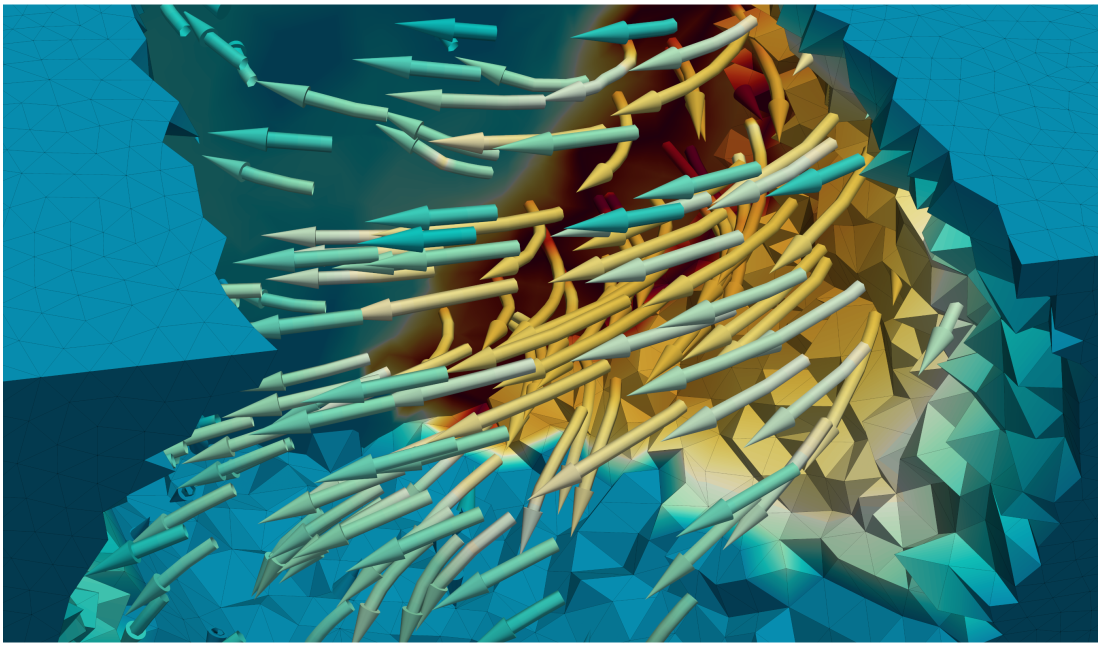
Parallel Overlapping Schwarz Preconditioners for Incompressible Fluid Flow and Fluid-Structure Interaction Problems

Christian Hochmuth



Parallel Overlapping Schwarz Preconditioners for Incompressible Fluid Flow and Fluid-Structure Interaction Problems

inaugural-dissertation

zur
Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität zu Köln

vorgelegt von
Christian Hochmuth
aus Werne

Köln, 2020

Berichterstatter: Prof. Dr. Axel Klawonn
(Universität zu Köln)

Prof. Dr. Oliver Rheinbach
(TU Bergakademie Freiberg)

Tag der mündlichen Prüfung: 18. Mai 2020

Abstract

Efficient methods for the approximation of solutions to incompressible fluid flow and fluid-structure interaction problems are presented. In particular, partial differential equations (PDEs) are derived from basic conservation principles. First, the incompressible Navier–Stokes equations for Newtonian fluids are introduced. This is followed by a consideration of solid mechanical problems. Both, the fluid equations and the equation for solid problems are then coupled and a fluid-structure interaction problem is constructed. Furthermore, a discretization by the finite element method for weak formulations of these problems is described. This spatial discretization of variables is followed by a discretization of the remaining time-dependent parts. An implementation of the discretizations and problems in a parallel C++ software environment is described. This implementation is based on the software package Trilinos[1].

The parallel execution of a program is the essence of *High Performance Computing* (HPC). HPC clusters are, in general, machines with several tens of thousands of cores. The fastest current machine, as of the TOP500 list[2] from November 2019, has over 2.4 million cores, while the largest machine possesses over 10 million cores.

To achieve sufficient accuracy of the approximate solutions, a fine spatial discretization must be used. In particular, fine spatial discretizations lead to systems with large sparse matrices that have to be solved. Iterative preconditioned Krylov methods are among the most widely used and efficient solution strategies for these systems. Robust and efficient preconditioners which possess good scaling behavior for a parallel execution on several thousand cores are the main component. In this thesis, the focus is on parallel algebraic preconditioners for fluid and fluid-structure interaction problems. Therefore, monolithic overlapping Schwarz preconditioners for saddle point problems of Stokes and Navier–Stokes problems are presented. Monolithic preconditioners for incompressible fluid flow problems can significantly improve the convergence speed compared to preconditioners based on block factorizations. In order to obtain numerically scalable algorithms, coarse spaces obtained from the *Generalized Dryja–Smith–Widlund* (GDSW) and the *Reduced dimension GDSW* (RGDSW) approach are used. These coarse spaces

[1] trilinos.github.io; see also [90].

[2] www.top500.org.

can be constructed in an essentially algebraic way. Numerical results of the parallel implementation are presented for various incompressible fluid flow problems. Good scalability for up to 11 979 MPI ranks, which corresponds to the largest problem configuration fitting on the employed supercomputer, were achieved. A comparison of these monolithic approaches and commonly used block preconditioners with respect to time-to-solution is made. Similarly, the most efficient construction of two-level overlapping Schwarz preconditioners with GDSW and RGDSW coarse spaces for solid problems is reported. These techniques are then combined to efficiently solve fully coupled monolithic fluid-strucuture interaction problems.

Zusammenfassung

Effiziente Methoden zur approximativen Berechnung von Lösungen inkompressibler Fluid- und Fluid-Struktur-Interaktionsprobleme werden vorgestellt. Aus grundlegenden physikalischen Erhaltungsprinzipien werden partielle Differentialgleichungen (PDGLs) für diese Probleme hergeleitet. Zunächst werden dabei die inkompressiblen Navier–Stokes-Gleichungen für newtonsche Fluide eingeführt. Im Anschluss folgt eine Betrachtung von Problemen der Festkörpermechanik. Die Fluidgleichungen und die Gleichungen der Festkörpermechanik werden gekoppelt, um so ein Fluid-Struktur-Interaktionsproblem zu konstruieren. Schwache Formulierungen dieser Probleme werden zunächst mit der Finite-Elemente-Methode diskretisiert. Auf diese räumliche Diskretisierung folgt eine Diskretisierung der verbleibenden zeitabhängigen Variablen. Darüber hinaus wird eine Implementierung dieser Probleme in einer parallelen C++-Softwareumgebung beschrieben. Diese parallele Implementierung basieren auf dem Softwarepaket Trilinos[3]

Das parallele Ausführen eines Programms ist der Kernbestandteil von *High Performance Computing* (HPC). HPC-Cluster sind Hochleistungscomputer mit in der Regel mehreren zehntausend Rechenkernen. Der aktuell schnellste Hochleistungscomputer besitzt über 2,4 Millionen Rechenkerne, wobei der größte Cluster sogar über 10 Millionen Kerne besitzt; Stand November 2019 der TOP500-Liste[4].

Um eine ausreichende Genauigkeit einer approximativen Lösung zu erreichen, muss in der Regel eine feine räumliche Diskretisierung verwendet werden. Diese feinen räumlichen Diskretisierungen führen zu großen dünnbesetzten Matrizen, die es zu lösen gilt. Iterative, vorkonditionierte Krylov-Methoden gehören dabei zu den weitverbreitetsten und effizientesten Lösungsmethoden für diese Probleme. Insbesondere die Entwicklung und Implementierung von robusten und effizienten Vorkonditionierern, die ein gutes Skalierungsverhalten in der parallelen Ausführung auf mehreren tausend Kernen besitzen, sind für die Effizienz von großer Bedeutung. Der Schwerpunkt dieser Dissertation liegt auf parallelen und algebraischen Vorkonditionierern für Fluid- und Fluid-Struktur-Interaktionsprobleme, die nur wenig Informationen des eigentlichen Problems benötigen. Es werden monolithische überlappende Schwarz-Vorkonditionierer

[3] trilinos.github.io; siehe auch [90].

[4] www.top500.org.

für Sattelpunktprobleme von Stokes- und Navier–Stokes-Problemen vorgestellt. Monolithische Vorkonditionierer für inkompressible Fluidprobleme können die Konvergenzgeschwindigkeit im Vergleich zu Vorkonditionierern, die auf Blockfaktorisierungen basieren, deutlich verbessern. Um numerisch skalierbare Algorithmen zu konstruieren, werden *Generalized–Dryja–Smith–Widlund-* (GDSW) und *Reduced-Dimension-GDSW*-Grobgitterräume (RGDSW) genutzt. Diese Grobgitterräume können im Wesentlichen algebraisch konstruiert werden. Es werden numerische Ergebnisse der parallelen Implementierung für verschiedene inkompressible Fluidströmungsprobleme vorgestellt. Hierfür wird eine gute Skalierbarkeit auf bis zu 11 979 MPI Rängen erreicht, was der größten Problemkonfiguration für den genutzten Supercomputer entspricht. Des Weiteren wird ein Vergleich dieser monolithischen Ansätze und häufig genutzter Block-Vorkonditionierer für inkompressible Navier–Stokes-Probleme vorgenommen. Ebenso wird die effiziente Konstruktion von zweistufigen überlappenden Schwarz-Vorkonditionierern mit GDSW- und RGDSW-Grobgitterräumen für Probleme der Festkörpermechanik vorgestellt. Alle diese Techniken werden dann kombiniert, um vollstndig gekoppelte monolithische Fluid-Struktur-Interaktionsprobleme effizient zu lösen.

Acknowledgements

I would like to thank my doctoral advisor Professor Axel Klawonn for the great opportunity to join his team and to work with him on the topics of this thesis. I am also very grateful for his support and guidance throughout the last couple of years, especially during the difficult times. Furthermore, I express my gratitude to Professor Oliver Rheinbach for being a referee of my thesis and for being part of my thesis committee as well as for several interesting discussions, and Professor Ulrich Lang for chairing my thesis committee.

I am very thankful to everyone in my team including Viktor Grimm, Alexander Heinlein, Jascha Knepper, Martin Lanser, Sabine Musielack-Erle, Matthias Uran and Janine Weber as well as my former colleagues Stephanie Friedhoff, Martin Kühn, and Patrick Radtke. All of them have encouraged, motivated, and supported me during my doctoral studies. A special thanks goes to Alexander Heinlein, Jascha Knepper, and Janine Weber for reading drafts of this thesis and making helpful comments. I specifically thank Alexander Heinlein for many fruitful discussions and much valuable advice, for reading my entire thesis and for being part of my thesis committee. Without his work on FROSch and his continuous assistance none of this would have been possible. Another special thanks goes to Jascha Knepper with whom I share an office and who also read my entire thesis. His kindness and calmness helped me through many difficult periods, especially in the last couple of months. I also thank him for constructing an FSI geometry of a real artery.

I gratefully acknowledge financial support by the German Science Foundation (DFG), project no. 214421492 and the computing time granted by the Center for Computational Sciences and Simulation (CCSS) of the University of Duisburg-Essen and provided on the supercomputer magnitUDE (DFG grants INST 20876/209-1 FUGG, INST 20876/243-1 FUGG) at the Zentrum für Informations- und Mediendienste (ZIM). I further thank the authors of [26] for providing a realistic coronary artery geometry.

Last but not least, I thank my girlfriend Antonia for reading and commenting on the entire thesis and the emotional support as well as my mother Ute and sister Talli for always being there for me.

Contents

List of Tables

List of Figures

1 Introduction

Computational fluid dynamics is a major field of research in applied mathematics. Enhanced modeling of the physical behavior, new methods for the spatial and temporal discretizations, and fast and efficient solvers are some of the relevant research topics. In this thesis, we will concentrate on the last aspect: fast and efficient solvers for incompressible fluid flow problems.

Fluid mechanics is a branch of the broader field of continuum mechanics. A second branch of continuum mechanics is solid mechanics. We want to extend the purely fluid based problems to the more complex *fluid-structure interaction* (FSI) problems. As the name already suggests, we have an additional structural problem which is coupled with the fluid equations. An example for an FSI problem is the flow of blood in an artery. There, the blood is modeled as a fluid problem and the artery itself is the corresponding structural problem. The book *Cardiovascular Mathematics* [70] gives a great overview of the mathematical aspect of fluid-structure interaction and applications to cardiovascular systems. The first main principle in fluid dynamics is the conservation of mass which can be modeled in two different ways. Therefore, we must distinguish between compressible fluids, e.g., a gas, and incompressible fluids, e.g., water. Consider a cavity filled with a fluid and covered by a lid. If we press down on the lid and prevent any leakage, the volume which is occupied by the fluid particles is decreased. As a consequence, the density of the fluid increases, due to the conservation of mass. This behavior is modeled by equations for compressible fluids, e.g, the Euler equations. In contrast, fluids with a constant density are called incompressible fluids. The above example of a closed cavity with decreasing volume does not apply for these fluids. We need to modify the above setup and allow for leaking, or use a dedicated outflow region to satisfy the conservation of mass for incompressible fluids. Our efficient solution methods for fluid flow and fluid-structure interaction problems will be used in the simulation of blood flow in arteries, whose physical behavior is captured very well by incompressible fluid equations. Therefore, only incompressible fluids will be considered in this thesis. The second main principle of continuum mechanics, and therefore fluids, is the conservation of momentum. Together with the conservation of mass, we will arrive at a system of two coupled equations, the quantities of interest are the velocity and pressure.

Simulations of incompressible fluids are used to approximate, e.g, the flow of blood through arteries [70], the drag and lift forces created by an aerofoil in a wind tunnel [114], or even velocities of larger scale problems like earth mantle convection [112]. Incompressible fluid flow problems are modeled by the time-dependent Navier–Stokes equations, which consist of two *partial differential equations* (PDEs). In general, these PDEs cannot be solved analytically and we need suitable discretizations. A widely used approach is the *finite element method* (FEM). To approximate solutions with the FEM, we generally start with a triangulation of the given computational domain. Usually, triangles or quadrilaterals are used in two dimensions and tetrahedra or hexahedra are used in the three-dimensional case. Based on this, the solution is approximated with a suitable basis representation. In a next step, and without considering the nonlinearities of the Navier–Stokes problem, a matrix and right-hand side are assembled from the weak formulations of the given PDEs with an elementwise procedure. This yields a linear system of equations for the unknown coefficients of the basis representation; cf., e.g., [22]. In particular, finite element discretizations of the underlying PDEs typically result in large, sparse, and ill-conditioned linear systems.

Special care has to be taken when constructing preconditioners for the discrete problems. In particular, the block structure and the coupling blocks have to be handled appropriately to guarantee fast convergence of iterative methods. We will focus on the construction of monolithic two-level overlapping Schwarz preconditioners with coarse spaces built from discrete (saddle point) harmonic extensions, which was already proposed in [79]. Specifically, the extensions of *Generalized Dryja–Smith–Widlund* (GDSW) coarse spaces, which were originally introduced for linear second- and fourth-order elliptic partial differential equations in [45,49], to general saddle point problems will be presented in Chapter 6. This approach was inspired by the work on monolithic overlapping Schwarz preconditioners with Lagrangian coarse spaces for saddle point problems by Klawonn and Pavarino in [104, 105]. For elliptic problems, the GDSW coarse basis functions are energy minimal extensions that are able to represent the nullspace of the elliptic operator. One significant advantage of our new method is that it can be applied to arbitrary geometries and domain decompositions, whereas the use of Lagrangian coarse basis functions requires a coarse triangulation. This limits the use of Schwarz methods with Lagrangian coarse spaces for arbitrary geometries.

In [51, 53, 54], different approaches for reducing the dimension of GDSW coarse spaces have been introduced and some of them have been proven to significantly improve the parallel scalability compared to standard GDSW coarse spaces; cf. [89]. These *Reduced dimension GDSW* (or simply *Reduced GDSW*) (RGDSW) coarse spaces were also used for the monolithic preconditioners in [80] and will be presented along the GDSW coarse space in this thesis. Alternatively, the parallel scalability could be improved by solving the coarse problem inexactly using another GDSW preconditioner resulting in a three-level GDSW method; cf. [88]. For highly heterogeneous multiscale problems, GDSW coarse spaces were enhanced using local generalized eigenvalue problems in [82], resulting in a method that is robust with respect to (w.r.t.) the contrast of the coefficients.

As described in [85], in a parallel implementation for elliptic problems, GDSW coarse spaces can be constructed in an algebraic fashion directly from the fully assembled system matrix. The parallel implementation of Schwarz preconditioners with GDSW coarse spaces for saddle point problems is based on the *FROSch* (Fast and Robust Overlapping Schwarz) software [83], which is part of the package ShyLU of the Trilinos library [90]. FROSch is a framework for parallel Schwarz preconditioners in Trilinos, and one of its main contributions is a parallel and algebraic implementation of GDSW preconditioners for elliptic problems; see also [77, 85–87, 89]. Therefore, the parallel implementation described in this thesis is also algebraic, i.e., the preconditioner can be easily built requiring only few input parameters; the construction will be described in more detail in sections 6.3.5 and 7.5. Parts of these new implementations have already partly been added to FROSch and are therefore available as open source as part of Trilinos. The remaining features will be added in the future and will therefore also be available, as part of FROSch.

In our approach, we solve saddle point problems using the *Generalized Minimal Residual method* (GMRES) [128]. We mainly consider the construction of monolithic preconditioners for Stokes and Navier–Stokes problems in two and three dimensions. An extensive overview of numerical methods for these saddle point problems is given in [16].

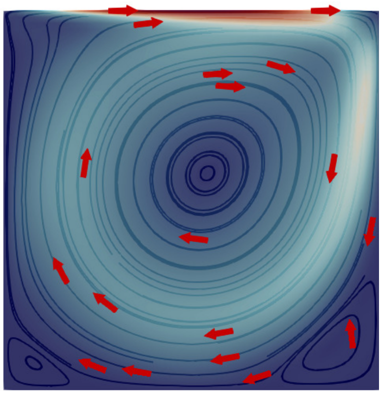
Older approaches for the iterative solution of saddle point problems are *exact and inexact Uzawa algorithms* [5, 12, 24, 62, 126, 150], where velocity and pressure are decoupled and solved in a segregated approach. Other physics-based approaches are the *Semi-Implicit Method for Pressure Linked Equations* (SIMPLE) and its generalizations, i.e., SIMPLEC and SIMPLER; cf. [39, 59, 121, 122]. Block preconditioners for GMRES, the *Minimal Residual method* (MINRES), and the *conjugate residual method* are presented in [61, 102, 103, 105, 108, 127, 131, 133, 146]. Further block preconditioners are the *Pressure Convection-Diffusion* (PCD) preconditioner [64, 101, 132], the *Least-Squares Commutator* (LSC) preconditioner [60, 63, 64], *Yosida's method* [123, 124], the *Relaxed Dimensional Factorization* (RDF) preconditioner [18] and the *Dimensional Splitting* (DS) preconditioner [17, 44]. Early studies of domain decomposition methods for Stokes problems were given in [23]. Schwarz preconditioners for saddle point problems have already been used for the approximation of the inverse matrices of blocks in [11, 37, 43, 85] and as monolithic preconditioners in [13, 14, 31, 140, 149]. Alternative solvers for saddle point problems are, e.g., multigrid methods; cf. [21, 25, 72, 96, 141, 142, 147]. Let us note that there are several other publications on iterative solvers for saddle point problems.

In addition to the incompressible fluid flow problems, we will model, discretize, and efficiently solve solid dynamics problems. The two-level GDSW and RGDSW Schwarz preconditioners, which will be employed to solve the fluid flow problems, can also be used to efficiently solve solid dynamics problems in a parallel environment. The monolithic methods and implementations for the 2×2 fluid flow block problems can be used for the general 1×1 block systems of solid dynamics problems. In addition, we will also present results for the almost incompressible limit of a linear elastic material. There, we will arrive at the same 2×2 saddle point structure which is obtained for incompressible fluid flow problems. Both the fluid flow and solid problems are then coupled to model a fully coupled monolithic FSI problem.

This thesis is organized as follows, in Chapter 2 we will derive the incompressible Navier–Stokes and Stokes equations. In Chapter 3, we consider solid dynamic problems. We combine the fluid and the solid problems to a fluid-structure interaction problem in Chapter 4. To do so, we first derive an alternative formulation for the Navier–Stokes equations based on an *Arbitrary Lagrangian Eulerian* (ALE) mapping. In FSI, ALE mappings are used to match the solid and fluid problems, which are typically studied in different coordinate systems. A detailed discussion of these different coordinate systems will be given in Chapters 2 to 4. In Chapter 5, we will define model problems and consider spatial finite element discretizations as well as temporal discretizations. For the temporal discretizations we will use implicit single-step *Runge–Kutta* (RK) methods and multi-step *Backward Differentiation Formulas* (BDF).

The two-level overlapping Schwarz domain decomposition preconditioners with GDSW and RGDSW coarse spaces will be presented in Chapter 6. We will focus on the construction of monolithic preconditioners for fluid flow problems. Furthermore, we will discuss standard block-diagonal and block-triangular preconditioners as well as SIMPLE and LSC block preconditioners for Stokes and Navier–Stokes problems. The chapter closes with a presentation of the block factorization-based *FaCSI* preconditioner for FSI problems; cf. [42]. For the approximation of block inverses of the above mentioned block methods, we will use both one-level and two-level overlapping Schwarz (R)GDSW preconditioners.
The discrete problems are implemented using our C++ library FEDDLib, which is based on the software package Trilinos [90]. Some basic information on Trilinos, the general structure of the FEDDLib, and details on the parallel finite element assembly are given in Chapter 7. Furthermore, we will present the setup of specific discrete problems based on the underlying weak formulations described in Chapter 5 and discuss the solution phase, which makes use of several Trilinos packages. In Chapter 8, numerical results for the previously introduced preconditioners and discretized problems are presented. We will focus on discussing the parallel efficiency of different preconditioners for fluid, solid, and FSI problems. The concluding Chapter 9 summarizes the findings of the previous chapters.

2 Stokes and Navier–Stokes Equations



This chapter is dedicated to the derivation of the two central sets of equations for incompressible fluid flow problems which are studied in this thesis, the Stokes and Navier–Stokes equations. Incompressible fluid flow problems are part of the field of fluid dynamics, which belongs to the continuum mechanics. In Chapter 4, we will describe another branch of continuum mechanics, the solid dynamics. A main difference between the dynamics of fluids and solids is the coordinate system, in which they are typically studied. For fluids the standard choice is the *Eulerian* coordinate system while the description of solids is normally defined in the *Lagrangian* coordinate system. In general, the displacement of material (point) from an external view is the main quantity of interest in solid dynamics. Therefore, a problem in the Lagrangian coordinate system uses a reference domain and the quantities of interest describe, in general, a relation between this reference domain and the current state; e.g., the displacement of a material point from its coordinates in the reference system to the new coordinates after a force is applied. In contrast, in the Eulerian framework we are inside the system and move with the material. Here, the observation is made at a fixed point and the quantity of interest is not the deformation or displacement but rather the velocity.

In the next section, we will present some basic principles for both frameworks and discuss the transformation of variables to different coordinate systems, which builds the basis of continuum mechanics. The presentation follows the description in [125].

2.1 Continuum Mechanics

Let $\hat{\Omega} \subset \mathbb{R}^3$ be a closed domain with a smooth boundary $\partial\hat{\Omega}$. Further, let $\hat{V} \subset \hat{\Omega}$ be an arbitrary closed volume in $\hat{\Omega}$ at time $t = 0$. Consider a point $\hat{\mathbf{x}} \in \hat{V}$, which can be associated with a single particle. Its position at time $t > 0$ is $\mathbf{x} := \mathbf{x}(\hat{\mathbf{x}}, t) = \hat{T}(\hat{\mathbf{x}}, t)$. In general, we will use a "hat"-symbol for variables in the reference configuration. Furthermore, we make the following assumption.

Assumption. $\hat{T}(\cdot, t) : \hat{\Omega} \to \Omega$ *is an orientation preserving* C^∞*–diffeomorphism with* $\det\left(\hat{T}\,(\cdot, t)\right) > 0$ *for* $t > 0$ *and* $\hat{T}(\cdot, 0)$ *is the identity.*

Since we are not interested in a single particle, we will make this observation for every point $\hat{\mathbf{x}} \in \hat{V}$, which results in the volume $V(t) = \hat{T}(\hat{V}, t)$ at time t. In the Lagrangian framework we directly compute the deformation from $\hat{\mathbf{x}}$ to $\mathbf{x}$ as an external observer. The *deformation* $\hat{\mathbf{u}}$ of a point or particle $\hat{\mathbf{x}} \in \hat{V}$ is defined as

$$\hat{\mathbf{u}}(\hat{\mathbf{x}}, t) = \mathbf{x}(\hat{\mathbf{x}}, t) - \hat{\mathbf{x}}. \tag{2.1}$$

Furthermore, the material velocity is defined as

$$\hat{\mathbf{v}}(\hat{\mathbf{x}}, t) := \frac{d}{dt}\mathbf{x}(\hat{\mathbf{x}}, t) = \frac{d}{dt}\hat{\mathbf{u}}(\hat{\mathbf{x}}, t). \tag{2.2}$$

We can reverse this view and study the quantities of interest in the current system. Then, we use

$$\mathbf{x} = \hat{\mathbf{x}} + \hat{\mathbf{u}}(\hat{\mathbf{x}}, t) \quad \text{and} \quad \mathbf{u}(\mathbf{x}, t) := \hat{\mathbf{u}}(\hat{\mathbf{x}}, t) = \mathbf{x} - \hat{\mathbf{x}}. \tag{2.3}$$

The Eulerian velocity in a point $\mathbf{x} \in V(t)$ is

$$\mathbf{v}(\mathbf{x}, t) := \frac{\partial \mathbf{x}}{\partial t} = \frac{\partial}{\partial t}\hat{T}(\hat{\mathbf{x}}, t) = \frac{\partial}{\partial t}\hat{\mathbf{u}}(\hat{\mathbf{x}}, t) = \hat{\mathbf{v}}(\hat{\mathbf{x}}, t). \tag{2.4}$$

Moreover, in structural mechanics the following basic quantities are used; cf. [125].

Definition 1 (Deformation Gradient)**.** *With a differential deformation field* $\hat{\mathbf{u}}(\hat{\mathbf{x}}, t)$ *on the material domain* $\hat{V}$ *the deformation gradient is defined as* $\hat{\mathbf{F}}(\hat{\mathbf{x}}, t) := I + \hat{\nabla}\hat{\mathbf{u}}(\hat{\mathbf{x}}, t)$.

Additionally, the determinant of the deformation gradient

$$\hat{J} := \det\left(\hat{\mathbf{F}}(\hat{\mathbf{x}}, t)\right) \tag{2.5}$$

denotes the local changes of volume; cf. [125] For the volume $V(t)$ it holds that

$$|V(t)| = \int_{\hat{V}} \hat{J}\, d\hat{\mathbf{x}}. \tag{2.6}$$

In the following, we will derive the equations of incompressible fluid flow in the Eulerian coordinate system. In theory, it is possible to model them in the Lagrangian coordinate system. There, we would need to be able to determine the position of every particle of the initial system $\hat{V}$ at every time throughout our observation. This can only be done if we study a closed system where no particle enters or leaves the domain of interest. However, in fluid dynamics we typically have an inflow and outflow region, which prevents us from using the Lagrangian coordinate system.

2.2 Incompressible Fluids

Only isothermal flows, i.e., flows with a constant temperature, are considered in this thesis. Therefore, we neglect the energy and our fluid dynamics model only needs to satisfy

- the conservation of mass and
- the conservation of momentum.

The first quantity of interest is, as already mentioned, the velocity $\mathbf{v}(\mathbf{x}, t)$. Furthermore, we need to determine the pressure $p(\mathbf{x}, t)$ and the density $\rho(\mathbf{x}, t)$. In the following two sections, we will discuss the conservation of mass and momentum for incompressible fluids. This discussion is along the line of [97, 125].

2.3 Conservation of Mass

We define the mass as of $V(t)$ as

$$m\left(V(t)\right) := \int_{V(t)} \rho(\mathbf{x}, t)\, d\mathbf{x}. \tag{2.7}$$

Hence, the conservation of mass yields

$$\frac{d}{dt}\int_{V(t)} \rho(\mathbf{x}, t)\, d\mathbf{x} = 0. \tag{2.8}$$

The time dependence of the integral is resolved with the following theorem; cf. [125].

Theorem 1 (Reynold's Transport Theorem)**.** *Let $f(\mathbf{x}, t) : \ \Omega \times (0, \infty) \to \mathbb{R}$ be a differentiable function. Then, for each volume $V(t) \subset \Omega$ it holds*

$$\frac{d}{dt}\int_{V(t)} f(\mathbf{x}, t)\, d\mathbf{x} = \int_{V(t)} \frac{\partial}{\partial t} f + \operatorname{div}(f\mathbf{v})\, d\mathbf{x}. \tag{2.9}$$

Now the conservation of mass of can be transformed to a point-wise condition. Applying theorem 1 to eq. (2.8) yields

$$\frac{d}{dt}\int_{V(t)} \rho(\mathbf{x}, t)\, d\mathbf{x} = \int_{V(t)} \frac{\partial}{\partial t} \rho + \operatorname{div}(\rho\mathbf{v})\, d\mathbf{x}. \tag{2.10}$$

We chose $V(t)$ as an arbitrary volume and can therefore use the following point-wise equation for the conservation of mass

$$\frac{\partial}{\partial t} \rho + \operatorname{div}(\rho\mathbf{v}) = 0, \tag{2.11}$$

assuming continuity of eq. (2.11) in $V(t)$.

Since we only consider incompressible fluids, the density is constant in time and space, i.e., in particular $\frac{\partial}{\partial t}\rho = 0$. Thus, eq. (2.11) reduces to the incompressibility condition

$$\operatorname{div}(\mathbf{v}) = 0. \tag{2.12}$$

Similarly, this condition can be obtained by directly using the incompressibility condition on the volume $V(t)$, i.e., the mass of $V(t)$ does not change with respect to time:

$$0 = \frac{d}{dt}|V(t)| = \frac{d}{dt}\int_{V(t)} 1\,d\mathbf{x}. \tag{2.13}$$

Applying theorem 1 to eq. (2.13) yields

$$\int_{V(t)} \operatorname{div}(\mathbf{v}) d\,\mathbf{x} = 0,$$

and eq. (2.12) follows with a point-wise consideration.

2.4 Conservation of Momentum

In this section we will describe the conservation of linear momentum. Linear momentum for flows is equivalent to *Newton's second law of motion*; cf. [97]. Newton's second law of motion states that

$$\text{net force} = \text{mass} \times \text{acceleration}.$$

Therefore, we define the linear momentum as

$$\mathbf{p} := \int_{V(t)} \rho(\mathbf{x},t)\mathbf{v}(\mathbf{x},t)\,d\mathbf{x}.$$

Then, the law of conservation of linear momentum reads

$$\frac{d}{dt}\mathbf{p} = F\left(V(t)\right) + F\left(\partial V(t)\right) \tag{2.14}$$

with the external volume force

$$F\left(V(t)\right) := \int_{V(t)} \rho(\mathbf{x},t)\mathbf{f}(\mathbf{x},t)\,d\mathbf{x}, \tag{2.15}$$

and the internal surface force

$$F\left(\partial V(t)\right) := \int_{\partial V(t)} \mathbf{t}\, d\mathbf{x}, \tag{2.16}$$

where $\mathbf{f}$ is an external volume force, such as gravitiy, and $\mathbf{t} = \mathbf{t}(\mathbf{x}, \mathbf{n})$ is a surface stress vector known as *Cauchy stress vector*. In general, a stress describes the internal force which acts on an imaginary surface inside $V(t)$.

Next, we state *Cauchy's Stress Theorem*, cf. [22], which provides the linear dependency of the stress vector $\mathbf{t}$ on the normal direction $\mathbf{n} \in S$. In the following, we will use $S \subset \Omega$ as the surface of an arbitrary cut through Ω; cf. [125].

Theorem 2 (Cauchy's Stress Theorem)**.** *Let* $\mathbf{t}(\cdot, \mathbf{n}) \in C^1(\Omega)$ *for fixed* $\mathbf{n}$*,* $\mathbf{t}(\mathbf{x}, \cdot) \in C^0(\Omega)$ *for fixed* $\mathbf{x}$*, and* $\mathbf{g}(\mathbf{x}, t) := \rho(\mathbf{x}, t)\mathbf{f}(\mathbf{x}, t) \in C^0(\Omega)$ *in equilibrium for a fixed time* t*. Then, there exists a symmetric tensor* $\boldsymbol{\sigma} \in C^1(\Omega)$*, such that*

$$\mathbf{t}(\mathbf{x}, \mathbf{n}) = \boldsymbol{\sigma}(\mathbf{x})\mathbf{n}, \quad \mathbf{x} \in \Omega, \ \mathbf{n} \in S.$$

For a proof see [34] or [97]; the latter uses the intuitively accessible tetrahedron argument. Cauchy's Stress Theorem or Principle builds the foundation of continuum mechanics. In particular, the stress tensor represents all internal forces.

Using the above theorem, we can write the surface stress in normal direction as $\mathbf{t} = \boldsymbol{\sigma}\mathbf{n}$ with *Cauchy stress tensor* $\boldsymbol{\sigma} \in \mathbb{R}^{3\times 3}$. Futhermore, using the divergence theorem for eq. (2.16) we obtain

$$F(\partial V(t)) = \int_{V(t)} \operatorname{div}(\boldsymbol{\sigma})\, d\mathbf{x}. \tag{2.17}$$

We apply Reynold's transport theorem to the left side of the conservation of momentum eq. (2.14) and obtain

$$\frac{d}{dt}\mathbf{p} = \int_{V(t)} \frac{\partial}{\partial t}(\rho\mathbf{v}) + \operatorname{div}(\rho\mathbf{v} \otimes \mathbf{v})\, d\mathbf{x}. \tag{2.18}$$

Similarly to the conservation of mass, the conservation of momentum must hold point-wise for every particle in $V(t)$. We can now combine eq. (2.18), eq. (2.15), and eq. (2.17) to the conservation of momentum in conservative form

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + \operatorname{div}(\rho \mathbf{v} \otimes \mathbf{v}) = \rho \mathbf{f} + \operatorname{div}(\boldsymbol{\sigma}).$$

For a transition from the conservative to the following nonconservative formulation we refer to [125]. The nonconservative formulation reads

$$\rho \frac{\partial}{\partial t} \mathbf{v} + \rho \left(\mathbf{v} \cdot \nabla\right) \mathbf{v} = \rho \mathbf{f} + \operatorname{div}(\boldsymbol{\sigma}). \tag{2.19}$$

Furthermore, Hooke's law for the stress tensor $\boldsymbol{\sigma}$ is

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\epsilon} + \lambda \operatorname{trace}(\boldsymbol{\epsilon}) I, \tag{2.20}$$

where $\boldsymbol{\epsilon}$ is the linearized strain tensor, cf. section 3.2, and λ and μ are the first and second Lamé constants, respectively. In fluid mechanics μ is known as the dynamic viscosity. In Chapter 3, we will consider Hooke's law in the context of solid mechanics. However, for incompressible fluids Hooke's law reduces to the Navier–Stokes material law

$$\boldsymbol{\sigma} = \mu(\nabla \mathbf{v} + \nabla \mathbf{v}^T) + \lambda I, \tag{2.21}$$

since $\operatorname{trace}(\boldsymbol{\epsilon}) = \operatorname{div}(\mathbf{v}) = 0$. Furthermore, we introduce the pressure $p = -\lambda I$, which will act as a Lagrangian multiplier for the incompressibility condition of the fluid.

2.5 Stokes and Navier–Stokes Equations

Using the Navier–Stokes material law of eq. (2.21) and the incompressibility condition $\operatorname{div}(\mathbf{v}) = 0$ we obtain the incompressible (unsteady or time-dependent) Navier–Stokes equations

$$\begin{aligned} \rho \left(\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} \right) - \mu \nabla \cdot (\nabla \mathbf{v} + \nabla \mathbf{v}^T) + \nabla p = \rho \mathbf{f}, \\ \operatorname{div}(\mathbf{v}) = 0, \end{aligned}$$

This form is often referred to as the stress-divergence form; cf. [74]. We obtain the conventional form by rewriting

$$\nabla \cdot (\nabla \mathbf{v} + \nabla \mathbf{v}^T) = \nabla^2 \mathbf{v} + \nabla(\nabla \cdot \mathbf{v}), \tag{2.22}$$

and using $\operatorname{div}(\mathbf{v}) = \nabla \cdot \mathbf{v} = 0$. With $\nabla^2 \mathbf{v} = \Delta \mathbf{v}$ we obtain the time-dependent Navier–Stokes equations in conventional form.

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} \right) - \mu \Delta \mathbf{v} + \nabla p = \rho \mathbf{f}, \tag{2.23}$$

$$\operatorname{div}(\mathbf{v}) = 0. \tag{2.24}$$

In the following, we will consider the conventional form if we consider pure fluid problems. Next, we can scale the pressure with the density; i.e., we replace p with p/ρ. Furthermore, we introduce the kinematic viscosity $\nu = \mu/\rho$. If we assume that eqs. (2.23) and (2.24) reach a stationary limit at $t \in (0, \infty)$, we can use $\frac{\partial u}{\partial t} = 0$ and obtain the steady Navier–Stokes equations

$$(\mathbf{v} \cdot \nabla)\mathbf{v} - \nu \Delta \mathbf{v} + \nabla p = \mathbf{f}, \tag{2.25}$$

$$\operatorname{div}(\mathbf{v}) = 0. \tag{2.26}$$

For fluids with negligible advective forces, i.e., if $(\mathbf{v} \cdot \nabla)$ is comparatively small, we can further remove the advective term from the Navier–Stokes equations and obtain the Stokes equations

$$-\mu \Delta \mathbf{v} + \nabla p = \mathbf{f}, \tag{2.27}$$

$$\operatorname{div}(\mathbf{v}) = 0. \tag{2.28}$$

The above momentum equation of the Stokes problem is typically stated with the dynamic viscosity μ instead of the kinematic viscosity. We need to enclose all of the above problems, i.e., the time-dependent and stationary Navier–Stokes as well as the Stokes problem, with appropriate boundary conditions in order to have well-posed problems. In general, a fluid domain Ω possesses an inflow region or boundary $\partial\Omega_{\text{in}}$ where velocities are prescribed with Dirichlet boundary conditions:

$$\mathbf{v} = \mathbf{g} \quad \text{on} \quad \partial\Omega_{\text{in}}.$$

Furthermore, if particles can enter the computational domain, particles must also be able to leave the domain in the case of incompressible fluids. Therefore, we define an outflow region $\partial\Omega_{\text{out}}$. The corresponding boundary condition is often stated as a Neumann boundary condition.

$$\boldsymbol{\sigma}\mathbf{n} = 0 \quad \text{on} \quad \partial\Omega_{\text{out}}. \tag{2.29}$$

This boundary condition is typically denoted as *do–nothing* boundary condition. We will see in sections 5.1.1 and 5.1.2 that, after the variational formulation is derived with Green's formula and the resulting integral over the Neumann boundary $\partial\Omega_{\text{out}}$ is dropped, it is immediately satisfied. If we use the conventional form eq. (2.23) the do-nothing boundary condition reads

$$\nu\frac{\partial \mathbf{v}}{\partial \mathbf{n}} - p\mathbf{n} = 0 \quad \text{on} \quad \partial\Omega_{\text{out}}.$$

We highlight that it is a different condition than the stress-free condition of eq. (2.29) which is used for the stress-divergence form eq. (2.22) of the Navier–Stokes equations. The stress-divergence form of the Navier–Stokes equations is the correct form for physically meaningful natural boundary conditions; cf. [74]. Therefore, it is mandatory to use this form in a fluid-structure interaction problem, where fluid and solid stresses are coupled at the fluid-solid interface. This coupling is further described in Chapter 4. However, for a separated fluid problem the conventional form of the Navier–Stokes equations is sufficient. All other boundaries of a fluid problem are generally walls $\partial\Omega_{\text{wall}}$ which prevent the fluid from leaking out of the computational domain. The most commonly used boundary conditions for walls are so-called no-slip conditions, where a fluid velocity of zero is prescribed:

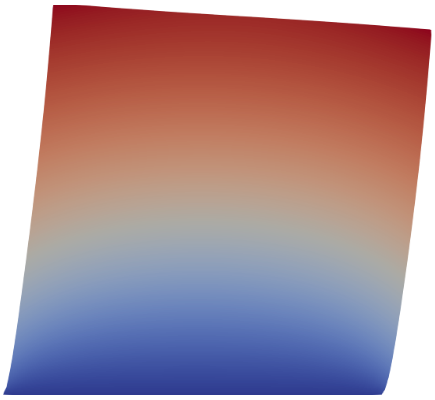
$$\mathbf{v} = 0 \quad \text{on} \quad \partial\Omega_{\text{wall}}.$$

Later, we will also consider a specific problem setup without an inflow or outflow region. There, the fluid domain is closed, no particle enters or leaves the computational domain and the fluid is driven by a “moving” lid. Therefore, this problem is called *lid-driven cavity*. There are several indication values which characterize different properties of a specific Navier–Stokes problem. The most widely used is the *Reynolds number* (Re). It describes the relative contributions of advection and diffusion and gives a general measurement of how advection dominated a specific problem is. The Reynolds number is defined as, cf. [125],

$$\mathrm{Re} := \frac{\rho \bar{u} L}{\mu} = \frac{\bar{u} L}{\nu} \tag{2.30}$$

where $\bar{u}$ is some known velocity and L is a characteristic length scale. Generally, $\bar{u}$ is the average or maximum inflow velocity and L is equal to the height or length of an obstacle, inflow region, or outflow region. Two- and three-dimensional flows possess quite different characteristics for a given Reynolds number; i.e., a two-dimensional problem with a stable solution might be unstable for the same Reynolds number and an equivalent domain in the three-dimensional case. Moreover, there is no general upper limit to the Reynolds number above which a flow becomes unstable. Additionally, problems with high Reynolds numbers can suffer from instabilities due to highly advective regions which are not properly resolved with the spatial discretization. In section 5.1.6 we will comment on stabilization techniques for Navier–Stokes problems which exhibit such stability issues.

3 Solid Mechanics



In this chapter we want to present the basic principles and concepts of solid mechanics. Similar to the previous chapter about incompressible fluid mechanics, solid mechanics builds on the same basic concepts of conservation laws. Although both fields belong to the broader area of continuum mechanics, the usual view, i.e, Lagrangian or Eulerian, on the relevant variables or quantities is quite different. We saw in the last chapter that it is much more convenient and less restrictive to consider fluids in the Eulerian framework, which means that the quantity of interest is velocity. For solid mechanics a more natural viewpoint is the Lagrangian framework. Here, the primary quantity of interest will be the displacement of a solid body. For convenience, we will restate some of the considerations which were made at the beginning of Chapter 2. The presentation follows the description in [125].

Each material point $\hat{\mathbf{x}} \in \hat{V}$ will be at $\mathbf{x} = \hat{T}(\hat{\mathbf{x}}, t) \in V(t)$ at time $t > 0$. Similarly, we have defined the invertible mapping $\hat{T}(t) : \hat{V} \to V(t)$ for every material point of the initial reference system $\hat{V} := V_0 = V(0)$. Furthermore, we define a second (arbitrary) reference system $\hat{W} \subset \hat{\Omega}_W \subset \mathbb{R}^3$. We will work with different reference systems $\hat{V}$ and $\hat{W}$, since for fluid-structure interaction problems two different mappings will be considered.

For a mapping from the arbitrary system $\hat{W}$ to the current configuration, we will define the mapping $\hat{T}_W(t) : \hat{W} \to V(t)$. A generic configuration of the three systems $\hat{V}$, $V(t)$, and $\hat{W}$ and the mappings $\hat{T}$ and $\hat{T}_W$ is depicted in fig. 3.1.

In the next section, we will transform the balance of momentum equation for the current system $V(t)$ to the arbitrary reference system $\hat{W}$. In section 3.2, we will then define the general reference system $\hat{W}$ to be identical to the initial reference configuration $\hat{V}$ and state the equation of our solid problem. We use the arbitrary system $\hat{W}$ since we can reuse many of the results in Chapter 4, where we present a coupled fluid–structure interaction problem in the Arbitrary Lagrangian Eulerian formulation.

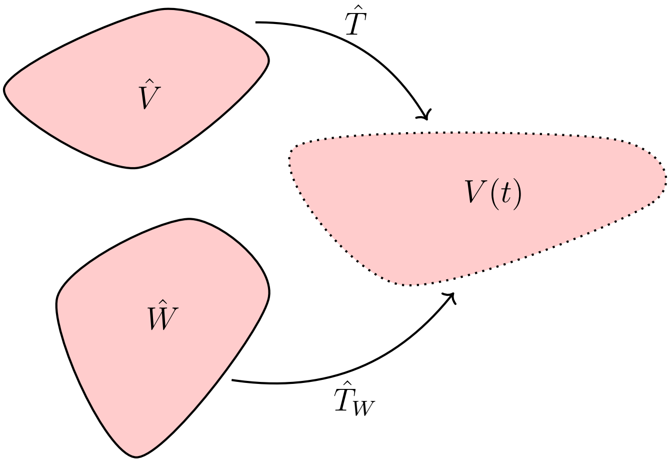


Figure 3.1: Generic mapping of initial reference domain $\hat{V}$ and arbitrary reference domain $\hat{W}$ to the current domain $V(t)$

3.1 Conservation of Momentum in a Reference Configuration

At the beginning of this section, we will state two important lemmata for the transformation to an arbitrary system; cf. [125]. Let $\hat{W}$ be the arbitrary system which is fixed in time and $\hat{T}_W(t) : \hat{W} \to V(t)$ be an invertible mapping with gradient $\hat{\mathbf{F}}_W := \hat{\nabla}\hat{T}_W$ and determinant $\hat{J}_W := \det(\hat{\mathbf{F}}_W) > 0$.

Lemma 1 (Inverse Mapping)**.** *Let $T_W(t) : V(t) \to \hat{W}$ be the inverse mapping, let $\mathbf{F}_W := \nabla T_W$ be its gradient and let $J_W := \det(\mathbf{F}_W)$ be its determinant. With sufficient regularity, it holds that*

$$\mathbf{F}_W = \hat{\mathbf{F}}_W^{-1}, \quad J_W = \hat{J}_W^{-1}, \quad \frac{\partial}{\partial t} T_W = -\hat{\mathbf{F}}_W^{-1} \frac{\partial}{\partial t} \hat{T}_W.$$

The following relations for a scalar function $f : V(t) \to \mathbb{R}$ and a vector field $\mathbf{w} : V(t) \to \mathbb{R}^d$ follow from lemma 1.

$$\nabla f = \hat{\mathbf{F}}_W^{-T} \hat{\nabla} \hat{f}, \quad \nabla \mathbf{w} = \hat{\nabla} \hat{\mathbf{w}} \hat{\mathbf{F}}_W^{-1}. \tag{3.1}$$

Lemma 2 (Transformation of Temporal Derivatives)**.** *Let $f : V(t) \to \mathbb{R}$ be a function with sufficient regularity and $\hat{f}(\hat{\mathbf{x}}_W, t) = f(\mathbf{x}, t)$ its counterpart. It holds that*

$$\frac{\partial}{\partial t} f = \frac{\partial}{\partial t} \hat{f} - \left(\hat{\mathbf{F}}_W^{-1} \frac{\partial}{\partial t} \hat{T}_W \cdot \hat{\nabla} \right) \hat{f}, \quad \frac{d}{dt} f = \frac{\partial}{\partial t} \hat{f} + \left(\hat{\mathbf{F}}_W^{-1} \left(\hat{\mathbf{v}} - \frac{\partial}{\partial t} \hat{T}_W \right) \cdot \hat{\nabla} \right) \hat{f},$$

where $\hat{\mathbf{v}}$ is the particle velocity.

We can now transform the first parts of the conservation of momentum equation from $V(t)$ to $\hat{W}$. For convenience, we restate the conservation of momentum in nonconservative form, which was introduced with eq. (2.19):

$$\rho \frac{\partial}{\partial t} (\mathbf{v}) + \rho (\mathbf{v} \cdot \nabla) \mathbf{v} = \rho \mathbf{f} + \operatorname{div}(\boldsymbol{\sigma}) \quad \text{in } V(t).$$

Furthermore, we express the following quantities for the reference configuration $\hat{W}$: Density $\hat{\rho}(\hat{\mathbf{x}}_W, t) = \rho(\mathbf{x}, t)$, velocity $\hat{\mathbf{v}}(\hat{\mathbf{x}}_W, t) = \mathbf{v}(\mathbf{x}, t)$, volume force $\hat{\mathbf{f}}(\hat{\mathbf{x}}_W, t) = \mathbf{f}(\mathbf{x}, t)$ and stress tensor $\hat{\boldsymbol{\sigma}}(\hat{\mathbf{x}}_W, t) = \boldsymbol{\sigma}(\mathbf{x}, t)$.

Using eq. (3.1) and lemma 2 for the partial temporal derivative of the velocity and the convective part, we obtain

$$\frac{\partial}{\partial t} \mathbf{v} = \frac{\partial}{\partial t} \hat{\mathbf{v}} - \left(\hat{\mathbf{F}}_W^{-1} \frac{\partial}{\partial t} \hat{T}_W \cdot \hat{\nabla} \right) \hat{\mathbf{v}},$$
$$(\mathbf{v} \cdot \nabla \mathbf{v}) = \nabla \mathbf{v} \mathbf{v} = \hat{\nabla} \hat{\mathbf{v}} \hat{\mathbf{F}}_W^{-1} \hat{\mathbf{v}} = \left(\hat{\mathbf{F}}_W^{-1} \hat{\mathbf{v}} \cdot \hat{\nabla} \right) \hat{\mathbf{v}},$$

which yields

$$\frac{\partial}{\partial t} \mathbf{v} + (\mathbf{v} \cdot \nabla \mathbf{v}) = \frac{\partial}{\partial t} \hat{\mathbf{v}} + \left(\hat{\mathbf{F}}_W^{-1} \left(\hat{\mathbf{v}} - \frac{\partial}{\partial t} \hat{T}_W \right) \cdot \hat{\nabla} \right) \hat{\mathbf{v}}. \tag{3.2}$$

Choosing $\hat{W} = \hat{V}$ the temporal derivative of the mapping is $\frac{\partial}{\partial t}\hat{T}_W = \frac{\partial}{\partial t}\hat{T} = \hat{\mathbf{v}}$ and we obtain

$$\frac{\partial}{\partial t}\mathbf{v} + (\mathbf{v} \cdot \nabla \mathbf{v}) = \frac{\partial}{\partial t}\hat{\mathbf{v}}.$$

Finally, we need to transform $\operatorname{div}(\boldsymbol{\sigma})$ to the reference configuration $\hat{W}$.

Lemma 3 (Piola Transformation)**.** *Let the vector field $\mathbf{w} : V(t) \to \mathbb{R}^d$ be differentiable and $\hat{\mathbf{w}}$ the corresponding field in the reference system $\hat{W}$. The Piola transformation of $\mathbf{w}$ is*

$$\hat{J}_W \hat{\mathbf{F}}_W^{-1} \hat{\mathbf{w}}.$$

For every volume $V(t)$ with reference system $\hat{W}$ we have

$$\int_{\partial V(t)} \mathbf{n} \cdot \mathbf{w} \, ds = \int_{\partial \hat{W}} \hat{\mathbf{n}} \cdot \left(\hat{J}_W \hat{\mathbf{F}}_W^{-1} \hat{\mathbf{w}} \right) \, d\hat{s},$$
$$\int_{V(t)} \operatorname{div}(\mathbf{w}) \, d\mathbf{x} = \int_{\hat{W}} \widehat{\operatorname{div}}(\hat{J}_W \hat{\mathbf{F}}_W^{-1} \hat{\mathbf{w}}) \, d\hat{\mathbf{x}}.$$

In our continuous setting, this relation must also hold point-wise:

$$\hat{J}_W \operatorname{div}(\mathbf{w}) = \widehat{\operatorname{div}}(\hat{J}_W \hat{\mathbf{F}}_W^{-1} \hat{\mathbf{w}}). \tag{3.3}$$

For a proof see [125]; eq. (3.3) must be seen as an equality for all points in $\hat{W}$ with $\mathbf{x} = \hat{T}_W(\hat{\mathbf{x}}_W, t)$.

We obtain the following surface forces in the reference system by using a column-wise consideration of the stress tensor $\boldsymbol{\sigma} = (\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3)$, by applying lemma 3 to each $\boldsymbol{\sigma}_i$, $i = 1, 2, 3$, and by combining them afterwards:

$$\begin{aligned} F(\partial V(t)) = \int_{\partial V(t)} \boldsymbol{\sigma}\mathbf{n} \, ds &= \int_{V(t)} \operatorname{div}(\boldsymbol{\sigma}) \, d\mathbf{x} \\ &= \int_{\hat{W}} \widehat{\operatorname{div}}(\hat{J}_W \hat{\boldsymbol{\sigma}} \hat{\mathbf{F}}_W^{-T}) \, d\hat{\mathbf{x}}. \qquad (3.4) \\ &= \int_{\partial \hat{W}} (\hat{J}_W \hat{\boldsymbol{\sigma}} \hat{\mathbf{F}}_W^{-T}) \hat{\mathbf{n}} \, d\hat{s} \end{aligned}$$

Definition 2. *The first Piola–Kirchhoff stress tensor is defined as*

$$\hat{\mathbf{P}} := \hat{J}_W \hat{\boldsymbol{\sigma}} \hat{\mathbf{F}}_W^{-T}.$$

The second Piola–Kirchhoff stress tensor is defined as

$$\hat{\mathbf{\Sigma}} := \hat{\mathbf{F}}_W^{-1} \hat{\mathbf{P}} = \hat{J}_W \hat{\mathbf{F}}_W^{-1} \hat{\boldsymbol{\sigma}} \hat{\mathbf{F}}_W^{-T}.$$

The above stress tensors give a relation between stresses in the Eulerian system $V(t)$ and the reference system $\hat{W}$. Note that for small deformations $\hat{J}_W \approx 1$ and $\hat{\mathbf{F}}_W \approx I$ are generally used. Then, the first and second Piola–Kirchhoff stress tensors are equal.
With the first Piola–Kirchhoff stress tensor and the transformation of temporal derivatives, which was used in eq. (3.2), we obtain the momentum equation on the reference system $\hat{W}$:

$$\hat{J}_W \hat{\rho} \left(\frac{\partial}{\partial t} \hat{\mathbf{v}} + \left(\hat{\mathbf{F}}_W^{-1} \left(\hat{\mathbf{v}} - \frac{\partial}{\partial t} \hat{T}_W \right) \cdot \hat{\nabla} \right) \hat{\mathbf{v}} \right) = \hat{J}_W \hat{\rho} \hat{\mathbf{f}} + \widehat{\operatorname{div}} \left(\hat{J}_W \hat{\boldsymbol{\sigma}} \hat{\mathbf{F}}_W^{-T} \right). \tag{3.5}$$

In the following section, the arbitrary reference domain $\hat{W}$ will be specified, which will yield the solid equation. Furthermore, we will define material laws, which will be used to describe the material under consideration. These material laws are for isotropic and isothemeral materials. Isotropic materials are independent of the orientation, which means that responses to strain and strain rates are the same in every direction. In contrast, anisotropic materials generally depended on a fiber orientation. Moreover, isothermal materials are independent of temperature. All materials are independent of the point-of-view, which is denoted as objectivity. Materials are hyperelastic if the stresses and strains are related via

$$\hat{\mathbf{\Sigma}} = \frac{\partial W(\hat{\mathbf{E}})}{\partial \hat{\mathbf{E}}} \quad \text{or} \quad \hat{\mathbf{P}} = \frac{\partial W(\hat{\mathbf{F}})}{\partial \hat{\mathbf{F}}},$$

where W is an energy density function and $\hat{\mathbf{E}}$ is the *Green–Lagrange tensor*, which is a measure for the *strain*. It is related to the deformation gradient $\hat{\mathbf{F}}$ via the *right Cauchy–Green* tensor $\hat{\mathbf{C}}$ which is defined as

$$\hat{\mathbf{C}} := \hat{\mathbf{F}}^T \hat{\mathbf{F}}.$$

Then, we can define the Green–Lagrange tensor as

$$\hat{\mathbf{E}} := \frac{1}{2}(\hat{\mathbf{C}} - I).$$

It follows that $\hat{\mathbf{E}}$ is a nonlinear function w.r.t. the deformation $\hat{\mathbf{u}}$:

$$\hat{\mathbf{E}} = \frac{1}{2}(\hat{\nabla}\hat{\mathbf{u}} + \hat{\nabla}\hat{\mathbf{u}}^T + \hat{\nabla}\hat{\mathbf{u}}^T\hat{\nabla}\hat{\mathbf{u}}).$$

3.2 Solid Equation

The conservation of momentum of eq. (3.5) for the reference system $\hat{W} = \hat{V}$ and therefore $\frac{\partial}{\partial t}\hat{T}_V = \hat{\mathbf{v}}$ reduces to

$$\hat{J}\hat{\rho}\frac{\partial^2}{\partial t^2}\hat{\mathbf{u}} = \hat{J}\hat{\rho}\hat{\mathbf{f}} + \widehat{\operatorname{div}}(\hat{\mathbf{F}}\hat{\mathbf{\Sigma}}), \tag{3.6}$$

where we used $\frac{\partial}{\partial t}\hat{\mathbf{u}} = \hat{\mathbf{v}}$ and the second Piola–Kirchhoff stress tensor $\hat{\mathbf{\Sigma}}$. Here, we consider compressible materials which possess the following relation between the initial density $\rho(\mathbf{x}, 0) = \hat{\rho}_0(\hat{\mathbf{x}})$ and the density $\rho(\mathbf{x}, t)$ at time $t \geq 0$.

$$m(\hat{V}) = \int_{\hat{V}} \hat{\rho}_0(\hat{\mathbf{x}})\; d\hat{\mathbf{x}} = \int_{V(t)} \rho(\mathbf{x}, t)\; d\mathbf{x} = \int_{\hat{V}} \hat{J}\hat{\rho}(\hat{\mathbf{x}}, t)\; d\hat{\mathbf{x}} \;= m(V(t)).$$

With this relation between density and initial density we obtain the solid equation for an elastic material in the initial reference system $\hat{V}$:

$$\hat{\rho}_0\frac{\partial^2}{\partial t^2}\hat{\mathbf{u}} - \widehat{\operatorname{div}}(\hat{\mathbf{F}}\hat{\mathbf{\Sigma}}) = \hat{\rho}_0\hat{f}. \tag{3.7}$$

We need to enclose eq. (3.7) with boundary and initial conditions. Let $\hat{\Omega}$ be the reference system and computational domain and let $\partial\hat{\Omega}$ be the boundary or surface of $\hat{\Omega}$. We partition the boundary into a Neumann boundary part $\partial\hat{\Omega}_N$ and a Dirichlet boundary part $\partial\hat{\Omega}_D$ with $\partial\hat{\Omega} = \partial\hat{\Omega}_N \cup \partial\hat{\Omega}_D$. On the Neumann boundary we prescribe stresses in normal direction:

$$\mathbf{n} \cdot \hat{\mathbf{F}}\hat{\mathbf{\Sigma}} = \mathbf{n} \cdot \hat{J}\hat{\boldsymbol{\sigma}}_s\hat{\mathbf{F}}^{-T} = \mathbf{g}_N \quad \text{on} \quad \partial\hat{\Omega}_N \times [0, T], \tag{3.8}$$

with final time T. For the Dirichlet boundary we prescribe the displacements

$$\hat{\mathbf{u}} = \mathbf{g}_D \;\text{ on }\; \partial\hat{\Omega}_D \times [0,T]. \tag{3.9}$$

Furthermore, we need initial conditions for the displacement, velocity, and density:

$$\hat{\mathbf{u}}(\cdot,0) = \hat{\mathbf{u}}_0, \;\; \frac{\partial}{\partial t}\hat{\mathbf{u}}(\cdot,0) = \hat{\mathbf{v}}_0, \;\; \hat{\rho}(\cdot,0) = \hat{\rho}_0 \;\text{ in }\; \hat{\Omega} \times \{0\}.$$

If the boundary conditions in eq. (3.8) and eq. (3.9) are constant w.r.t. time, the solid equation can run into a steady-state. In this case, the velocity is zero and consequently $\partial^2\hat{\mathbf{u}}/\partial t^2 = 0$. Therefore it is sufficient to solve the stationary solid problem

$$\begin{aligned}
-\widehat{\operatorname{div}}(\hat{\mathbf{F}}\hat{\boldsymbol{\Sigma}}) &= \hat{\rho}\hat{\mathbf{f}} && \text{in } \hat{\Omega}, && (3.10)\\
\hat{\mathbf{u}} &= \mathbf{g}_D && \text{on } \partial\hat{\Omega}_D, && (3.11)\\
\mathbf{n}\cdot\hat{\mathbf{F}}\hat{\boldsymbol{\Sigma}} = \mathbf{n}\cdot\hat{J}\hat{\boldsymbol{\sigma}}_s\hat{\mathbf{F}}^{-T} &= \mathbf{g}_N && \text{on } \partial\hat{\Omega}_N. && (3.12)
\end{aligned}$$

We now need to define material laws to model the specific physical behavior of elastic and hyperelastic materials. The following nonlinear materials will be used for later simulations.

St. Venant–Kirchhoff material: The *St. Venant–Kirchhoff* material law uses the following linear dependency of the stress tensor $\hat{\boldsymbol{\Sigma}}$ on the nonlinear stresses:

$$\hat{\boldsymbol{\Sigma}} = 2\mu\hat{\mathbf{E}} + \lambda\,\mathrm{trace}(\hat{\mathbf{E}})I.$$

Neo–Hooke material: The *Neo–Hooke* material is defined with the stored energy function

$$W\left(\hat{\mathbf{C}}\right) = \frac{\mu}{2}\left(I_1 - 3\right) - \mu\log\left(\hat{J}\right) + \frac{\lambda}{2}\log\left(\hat{J}\right)^2,$$

where $I_1 := \mathrm{trace}\left(\hat{\mathbf{C}}\right)$ is the first principal invariant; cf. [137].

Linear Elasticity (Navier–Lamé): With the following assumptions we can make simplifications that result in a fully linear problem.

- Only small deformations are considered and the deformation gradient $\hat{\mathbf{F}}$ is small too. Therefore, we can use the approximations $\hat{\mathbf{F}} \approx I$ and $\hat{J} \approx 1$. Furthermore, we can drop all hats, because the Lagrangian and Eulerian reference systems are equivalent.

- The strains are so small that we can linearize the Green–Lagrange tensor

$$\hat{\mathbf{E}} = \frac{1}{2}(\hat{\nabla}\hat{\mathbf{u}} + \hat{\nabla}\hat{\mathbf{u}}^T + \hat{\nabla}\hat{\mathbf{u}}^T\hat{\nabla}\hat{\mathbf{u}}) \approx \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T) =: \boldsymbol{\epsilon}$$

The stationary Navier–Lamé or linear elasticity problem for a reference domain Ω with boundary $\partial\Omega = \partial\Omega_N \cup \partial\Omega_D$ and density $\rho = 1$ is

$$-\operatorname{div}(\boldsymbol{\sigma}) = \mathbf{f} \quad \text{in } \Omega, \tag{3.13}$$

$$\mathbf{u} = \mathbf{g}_D \quad \text{on } \partial\Omega_D, \tag{3.14}$$

$$\mathbf{n} \cdot \boldsymbol{\sigma} = \mathbf{g}_N \quad \text{on } \partial\Omega_N, \tag{3.15}$$

with the linear material law $\boldsymbol{\sigma} = 2\mu\boldsymbol{\epsilon} + \lambda\,\mathrm{trace}(\boldsymbol{\epsilon})I$. For linear material laws the first Lamé constant λ and second Lamé constant μ can be computed from the *Poisson ratio* ν and *Young's modulus* E:

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)},$$

$$\mu = \frac{E}{2(1+\nu)}.$$

Similarly, this linear elastic material model for $\boldsymbol{\sigma}$ can be inserted into the time-dependent eq. (3.6), which results in the following time-dependent linear elasticity problem for the density $\rho = 1$:

$$\frac{\partial^2}{\partial t^2}\mathbf{u} = f + \operatorname{div}(\boldsymbol{\sigma}) \quad \text{in } \ \Omega \times [0, T], \tag{3.16}$$

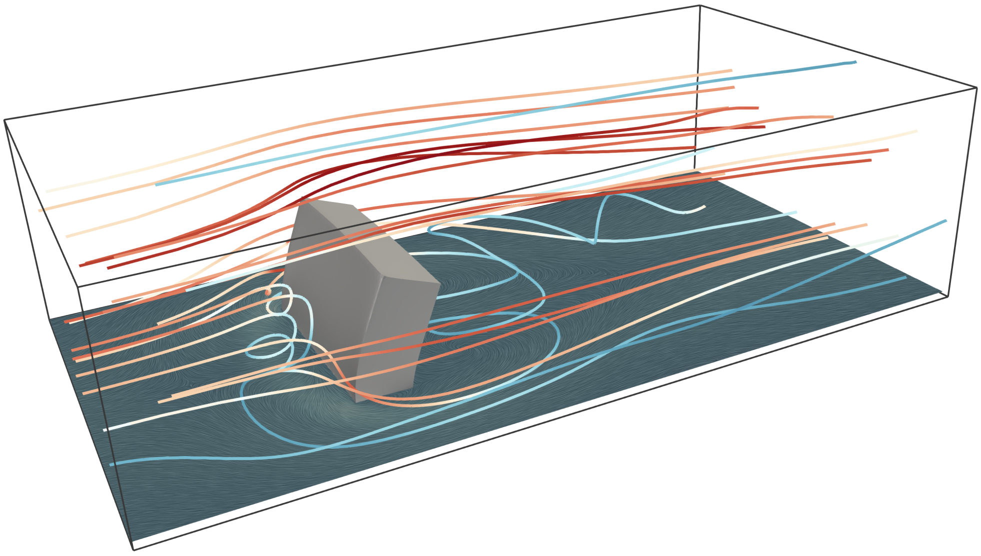
$$\mathbf{u} = \mathbf{g}_D \quad \text{on } \ \partial\Omega_D \times [0, T], \tag{3.17}$$

$$\mathbf{n} \cdot \boldsymbol{\sigma} = \mathbf{g}_N \quad \text{on } \ \partial\Omega_N \times [0, T], \tag{3.18}$$

$$\mathbf{u}(\cdot, 0) = \mathbf{u}_0 \quad \text{in } \ \Omega \times \{0\}, \tag{3.19}$$

$$\frac{\partial}{\partial t}\mathbf{u}(\cdot, 0) = \mathbf{v}_0 \quad \text{in } \ \Omega \times \{0\}. \tag{3.20}$$

4 Fluid-Structure Interaction



The monolithic system of partial differential equations for the fluid-structure interaction problem couples the incompressible Navier–Stokes equations with the solid equation. The discussion of this chapter follows the derivation of FSI systems in [70] and [125]. We want to determine the fluid velocity $\mathbf{v}_f$, the fluid pressure p_f, and the solid displacement $\hat{\mathbf{u}}_s$. The coupling is enforced by three equalities. Firstly, we want matching velocities of fluid and solid at the interface $\Gamma(t)$. This is the kinematic coupling condition. The second condition relates the stresses in normal direction of the fluid with the stresses in normal direction of the solid on the interface $\hat{\Gamma}$, which is the dynamic coupling condition. Thirdly, equal displacements of fluid mesh and solid on the interface $\hat{\Gamma}$ or $\Gamma(t)$ are necessary to limit the movement of both. Otherwise, the solid domain could enter and intersect the fluid domain and vice versa. For a detailed description of the coupling conditions we refer to [70] and [125]. The notation on its own already hints at the issue that the coupled system uses both the current system in the Eulerian framework for the fluid and the Lagrangian framework for the solid.

The fully coupled FSI system reads

$$\rho_f \left(\frac{\partial}{\partial t} \mathbf{v}_f + (\mathbf{v}_f \cdot \nabla) \mathbf{v}_f \right) - \operatorname{div} (\boldsymbol{\sigma}_f) = \rho_f \mathbf{f} \qquad \text{in } \Omega_f(t), \tag{4.1}$$

$$\operatorname{div} (\mathbf{v}_f) = 0 \qquad \text{in } \Omega_f(t), \tag{4.2}$$

$$\hat{\rho}_0 \frac{\partial^2}{\partial t^2} \hat{\mathbf{u}}_s - \widehat{\operatorname{div}} \left(\hat{\mathbf{F}}_s \hat{\boldsymbol{\Sigma}}_s \right) = \hat{\rho}_0 \hat{\mathbf{f}}_s \qquad \text{in } \hat{\Omega}_s, \tag{4.3}$$

$$\mathbf{v}_f = \hat{\mathbf{v}}_s \circ \hat{T}_s^{-1} \qquad \text{on } \Gamma(t), \tag{4.4}$$

$$\hat{\mathbf{n}} \cdot \left(\hat{J}_f \hat{\boldsymbol{\sigma}}_f \hat{\mathbf{F}}_f^{-1} \right) = \hat{\mathbf{n}} \cdot \hat{\mathbf{F}} \hat{\boldsymbol{\Sigma}}_s \qquad \text{on } \hat{\Gamma}, \tag{4.5}$$

$$\hat{\mathbf{u}}_f = \hat{\mathbf{u}}_s \qquad \text{on } \hat{\Gamma}, \tag{4.6}$$

with the current fluid domain $\Omega_f(t)$, reference solid domain $\hat{\Omega}_s$, and time $t \geq 0$. For statements on existence and uniqueness of the solution of simplified systems, we refer to [70, 125]. We note that many of the results for uniqueness and existence of a solution are made for a coupled problem which is defined for a unified framework; i.e., we either use the reference domains $\hat{\Omega}_*$ or the current domains Ω_* for solid and fluid. A major difficulty arises due to the moving fluid domain, which depends on the deformation of the solid. Let $\hat{\Omega}_f$ be the initial configuration of the fluid domain at time $t = 0$. In general, we define a function

$$\hat{T}_f(t) : \hat{\Omega}_f \to \Omega_f(t), \tag{4.7}$$

which maps the initial fluid domain to the current fluid domain. Similarly, we define a mapping for the solid domain:

$$\hat{T}_s(t) : \hat{\Omega}_s \to \Omega_s(t), \tag{4.8}$$

which is used in the coupling of fluid and solid interface velocities. In the next section, we will focus on the representation of the incompressible Navier–Stokes equations on a moving domain with the ALE formulation.

4.1 Arbitrary Lagrangian Eulerian Description of the Navier–Stokes Equations

Firstly, we use the mapping of eq. (4.7) with sufficient regularity to transform the velocity and pressure variables from the reference domain $\hat{\Omega}_f$ to the current domain $\Omega_f(t)$:

$$\hat{\mathbf{v}}(\hat{\mathbf{x}},t) = \mathbf{v}(\hat{T}_f(\hat{\mathbf{x}}),t) = \mathbf{v}(\mathbf{x},t), \;\; \hat{p}(\hat{\mathbf{x}},t) = p(\hat{T}_f(\hat{\mathbf{x}}),t) = p(\mathbf{x},t), \;\; \forall \hat{\mathbf{x}} \in \hat{\Omega}_f. \tag{4.9}$$

For convenience, we restate eq. (3.5) which is the conservation of linear momentum in a reference system $\hat{W}$.

$$\hat{J}_W \hat{\rho} \left(\frac{\partial}{\partial t}\hat{\mathbf{v}} + \left(\hat{\mathbf{F}}_W^{-1} \left(\hat{\mathbf{v}} - \frac{\partial}{\partial t}\hat{T}_W \right) \cdot \hat{\nabla} \right) \hat{\mathbf{v}} \right) = \hat{J}_W \hat{\rho} \hat{\mathbf{f}} + \widehat{\operatorname{div}} \left(\hat{J}_W \hat{\boldsymbol{\sigma}} \hat{\mathbf{F}}_W^{-T} \right).$$

Similarly, we can state the conservation of mass $\operatorname{div}(\mathbf{w}) = 0$ for a reference domain $\hat{W}$ and a vector field $\mathbf{w}$ with the Piola transformation of eq. (3.3):

$$\hat{J}_W \operatorname{div}(\mathbf{w}) = \widehat{\operatorname{div}}(\hat{J}_W \hat{\mathbf{F}}_W^{-1} \hat{\mathbf{w}}) = 0.$$

If we replace the mapping to the arbitrary domain $\hat{W}$ with the mapping to the reference fluid domain $\hat{\Omega}_f$, this results in the following equivalence (4.10) and (4.11) due to the conservations of momentum

$$\begin{aligned} \rho_f \left(\frac{\partial}{\partial t}\mathbf{v}_f + (\mathbf{v}_f \cdot \nabla)\, \mathbf{v}_f \right) |_{\Omega_f(t)} = \\ \rho_f \left(\hat{J}_f \left(\frac{\partial}{\partial t}\hat{\mathbf{v}}_f + \left(\hat{\mathbf{F}}_f^{-1} \left(\hat{\mathbf{v}}_f - \frac{\partial}{\partial t}\hat{T}_f \right) \cdot \nabla \right) \hat{\mathbf{v}}_f \right) \right) |_{\hat{\Omega}_f}, \end{aligned} \tag{4.10}$$

$$\operatorname{div}(\boldsymbol{\sigma}_f) + \rho_f f|_{\Omega_f(t)} = \widehat{\operatorname{div}} \left(\hat{J}_f \hat{\boldsymbol{\sigma}}_f \hat{\mathbf{F}}_f^{-T} \right) + \hat{J}_f \rho_f \hat{f}|_{\hat{\Omega}_f}, \tag{4.11}$$

and the following equality (4.12) due to the conservation of mass

$$\operatorname{div}(\mathbf{v})\,|_{\Omega_f(t)} = \widehat{\operatorname{div}} \left(\hat{J}_f \hat{\mathbf{F}}_f^{-1} \hat{\mathbf{v}}_f \right) |_{\hat{\Omega}_f}. \tag{4.12}$$

Here, the Cauchy stress tensor of the fluid in the reference configuration is

$$\hat{\boldsymbol{\sigma}}_f(\hat{\mathbf{v}},\hat{p}) = -\hat{p}I + \rho_f \nu_f \left(\hat{\nabla}\hat{\mathbf{v}}\hat{\mathbf{F}}_f^{-1} + \hat{\mathbf{F}}_f^{-T}\hat{\nabla}\hat{\mathbf{v}}^T \right).$$

In eq. (4.10), the additional transport term $-\partial_t \hat{T}_f$, which originates from the moving mesh, enters the momentum equation. Next, we will map the Navier–Stokes equations back into the current system $\Omega_f(t)$, which results in a formulation that is easier to discretize. However, in contrast to the ALE formulation which uses the reference domain $\hat{\Omega}_f$ we now need to explicitly move our fluid domain with the map $\hat{T}_f$. Further, we define the material time derivative to circumvent some further issues with time-dependent integrals. Again, we use an arbitrary volume $V(t) \subset \Omega_f(t)$. We define the *material time derivative*, cf. [70], for an Eulerian field $\mathbf{q}$ in the current configuration with the Lagrangian representation $\hat{\mathbf{q}}$ as

$$\frac{D}{Dt}\mathbf{q}\left(\cdot, t\right) := \frac{\partial}{\partial t}\hat{\mathbf{q}}\left(\cdot, t\right) \circ \hat{T}^{-1}. \tag{4.13}$$

Furthermore, using the following definition of a partial derivative yields

$$\begin{aligned} \frac{\partial \hat{\mathbf{q}}}{\partial t}(\hat{\mathbf{x}}, t) &= \lim_{h \to 0} \frac{\hat{\mathbf{q}}(\hat{\mathbf{x}}, t+h) - \hat{\mathbf{q}}(\hat{\mathbf{x}}, t)}{h} \\ &= \lim_{h \to 0} \frac{\mathbf{q}(\hat{T}(\hat{\mathbf{x}}, t+h), t+h) - \mathbf{q}(\hat{T}(\hat{\mathbf{x}}, t), t)}{h} \\ &= \frac{d}{dt}\mathbf{q}(\hat{T}(\hat{\mathbf{x}}, t), t). \end{aligned} \tag{4.14}$$

Combining eq. (4.13) and eq. (4.14) we obtain

$$\frac{D}{Dt}\mathbf{q}\left(\mathbf{x}, t\right) = \frac{d}{dt}\mathbf{q}\left(\hat{T}\left(\hat{\mathbf{x}}, t\right), t\right), \quad \text{with } \mathbf{x} = \hat{T}\left(\hat{\mathbf{x}}, t\right). \tag{4.15}$$

Analogously, we define the *ALE time derivative* with ALE mapping $\hat{T}_f$

$$\frac{\partial \mathbf{q}}{\partial t}|_{\hat{T}_f} = \frac{d}{dt}\mathbf{q}\left(\hat{T}_f\left(\hat{\mathbf{x}}, t\right), t\right), \quad \text{with } \mathbf{x} = \hat{T}_f\left(\hat{\mathbf{x}}, t\right). \tag{4.16}$$

This ALE time derivative is needed, since it is not clear that a material point $\mathbf{x} \in V(t)$ is still in the domain $V(t + \Delta t)$ for the next time step $t + \Delta t$ in a purely Eulerian formulation; cf. [70].

We can relate the ALE time derivative with the partial time derivative on the reference domain $\hat{\Omega}_f$. Furthermore, we account for the change in volume with

$d\mathbf{x} = \hat{J}_f d\hat{\mathbf{x}}$ and obtain

$$\frac{\partial \mathbf{v}_f}{\partial t}|_{\hat{T}_f} = \hat{J}_f \frac{\partial \hat{\mathbf{v}}_f}{\partial t}. \tag{4.17}$$

Therefore, we can use eq. (4.17) to replace $\hat{J}_f \frac{\partial}{\partial t}\hat{\mathbf{v}}_f$ in eq. (4.10) with the ALE time derivative. The first part of the momentum equation on the reference domain now reads

$$\rho_f \left(\frac{\partial}{\partial t}\mathbf{v}_f|_{\hat{T}_f} + \hat{J}_f \hat{\mathbf{F}}_f^{-1} \left(\left(\hat{\mathbf{v}}_f - \frac{\partial}{\partial t}\hat{T}_f \right) \cdot \nabla \right) \hat{\mathbf{v}}_f \right) |_{\hat{\Omega}_f}. \tag{4.18}$$

Note that the above representation must be seen as a symbolic or intermediate formulation, since $\mathbf{v}_f$ is clearly a variable of the current configuration $\Omega_f(t)$ and not the reference configuration $\hat{\Omega}_f$. Therefore, we further map the remaining parts of the incompressible Navier–Stokes equations back to the current domain and obtain an alternative ALE formulation of the Navier–Stokes equations:

$$\begin{aligned} \rho_f \left(\frac{\partial}{\partial t}\mathbf{v}_f|_{\hat{T}_f} + \left(\left(\mathbf{v}_f - \mathbf{w} \right) \cdot \nabla \right) \mathbf{v}_f \right) - \operatorname{div}\left(\boldsymbol{\sigma}_f \right) &= \rho_f \mathbf{f}, \\ \operatorname{div}\left(\mathbf{v} \right) &= 0, \end{aligned} \tag{4.19}$$

with fluid mesh velocity $\mathbf{w} = \frac{\partial}{\partial t}\hat{T}_f$. Through the introduction of the ALE time derivative the correction of the transport term with the fluid mesh velocity is kept in the representation on the current configuration $\Omega_f(t)$. We will see a second major difference to the fully Eulerian description without the ALE time derivative in section 5.1. There, we introduce the finite element discretization of the monolithic fluid-structure interaction problem with Navier–Stokes equations in ALE form as in eq. (4.19). The fully Eulerian description would introduce additional nonlinearities through the time derivative $\partial \mathbf{v}/\partial t$.

Note that we could have directly used the ALE time derivative of eq. (4.16) for eq. (4.1) to obtain eq. (4.19). Then, we could have neglected the ALE mapping to the reference fluid domain $\hat{\Omega}_f$. However, it is important to recognize that there is more than one ALE formulation of a coupled fluid-structure interaction problem. We have seen the ALE Navier–Stokes problem which is fully mapped to its reference domain $\hat{\Omega}_f$, i.e., without moving the mesh. In contrast, we can evaluated the Navier–Stokes equations on the current domain $\Omega_f(t)$ and only transform the temporal derivative with the ALE time derivative.

4.2 Monolithic Fluid-Structure Interaction System in Arbitrary Lagrangian Eulerian Form

The ALE mapping of the Navier–Stokes equations provides us the representation of the following fully coupled FSI system in ALE form:

$$
\begin{aligned}
\rho_f \left(\frac{\partial}{\partial t} \mathbf{v}_f|_{\hat{T}_f} + \left((\mathbf{v}_f - \mathbf{w}) \cdot \nabla \right) \mathbf{v}_f \right) - \operatorname{div}\left(\boldsymbol{\sigma}_f\right) &= \rho_f \mathbf{f} && \text{in } \Omega_f(t), \qquad (4.20)\\
\operatorname{div}\left(\mathbf{v}_f\right) &= 0 && \text{in } \Omega_f(t), \qquad (4.21)\\
\hat{\rho}_0 \frac{\partial^2}{\partial t^2} \hat{\mathbf{u}}_s - \widehat{\operatorname{div}} \left(\hat{\mathbf{F}}_s \hat{\boldsymbol{\Sigma}}_s \right) &= \hat{\rho}_0 \hat{\mathbf{f}}_s && \text{in } \hat{\Omega}_s, \qquad (4.22)\\
\mathbf{v}_f &= \hat{\mathbf{v}}_s \circ \hat{T}_s^{-1} && \text{on } \Gamma(t), \qquad (4.23)\\
\hat{\mathbf{n}} \cdot (\hat{J}_f \hat{\boldsymbol{\sigma}}_f \hat{\mathbf{F}}_f^{-1})) &= \hat{\mathbf{n}} \cdot \hat{\mathbf{F}}_s \hat{\boldsymbol{\Sigma}}_s && \text{on } \hat{\Gamma}, \qquad (4.24)\\
\hat{\mathbf{u}}_f &= \hat{\mathbf{u}}_s && \text{on } \hat{\Gamma}. \qquad (4.25)
\end{aligned}
$$

We need to enclose this system with appropriate boundary and initial conditions. Similar to the separated fluid and solid problems, we have, in general, an inflow region $\partial\Omega_{\text{in}}$, an outflow region $\partial\Omega_{\text{out}}$, and no-slip boundaries $\partial\Omega_{\text{wall}}$ for the fluid. Typically, the solid part of the FSI problem has Neumann and Dirichlet boundary regions $\partial\hat{\Omega}_{s,N}$ and $\partial\hat{\Omega}_{s,D}$, respectively. In general, the solid is clamped at $\partial\hat{\Omega}_{s,D}$. If the solid is, apart from the Dirichlet boundary, fully surrounded by the fluid domain, the Neumann boundary of the solid and the fluid-solid interface are identical: $\partial\hat{\Omega}_{s,N} = \hat{\Gamma}$. In general, we use the same boundary conditions for the coupled FSI problem as for the fluid subproblem, cf. section 2.5, and the solid subproblem, cf. section 3.2. In fig. 4.1 a two-dimensional FSI problem with suitable boundary regions is depicted. In the next section we will present a concrete ALE maps which can be used in an implementation.

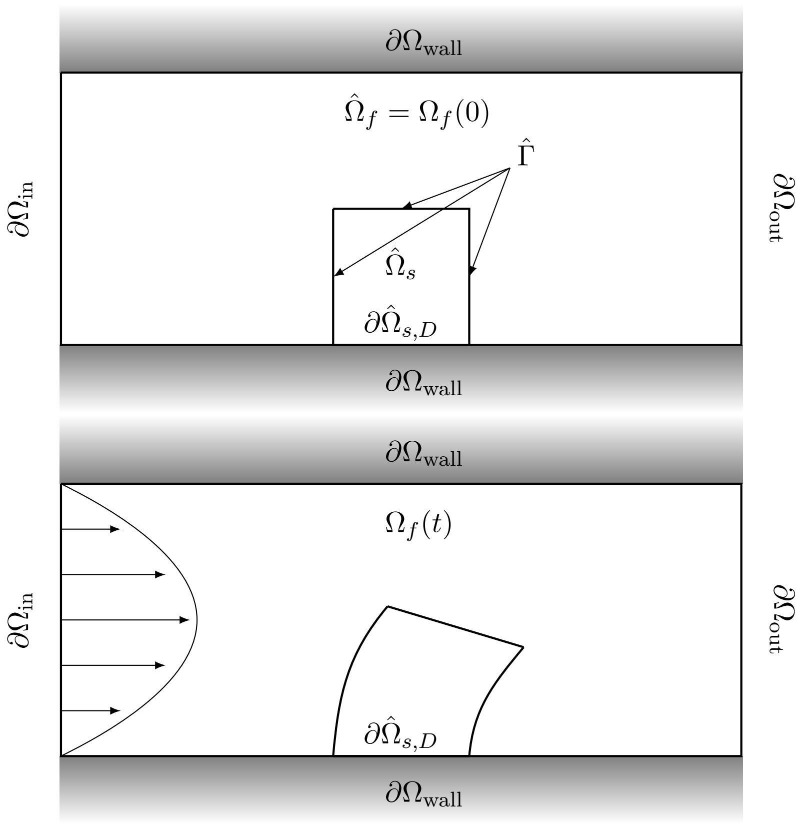

$\partial\Omega_{\text{wall}}$
$\hat{\Omega}_f = \Omega_f(0)$
$\hat{\Gamma}$
$\partial\Omega_{\text{in}}$
$\partial\Omega_{\text{out}}$
$\hat{\Omega}_s$
$\partial\hat{\Omega}_{s,D}$
$\partial\Omega_{\text{wall}}$
$\partial\Omega_{\text{wall}}$
$\Omega_f(t)$
$\partial\Omega_{\text{in}}$
$\partial\Omega_{\text{out}}$
$\partial\hat{\Omega}_{s,D}$
$\partial\Omega_{\text{wall}}$

Figure 4.1: Initial two-dimensional FSI domains (top) and generic deformed domains at time t (bottom) with the boundary regions highlighted.

4.3 Arbitrary Lagrangian Eulerian Map

In our setting, the ALE map enters the monolithic system as an additional PDE. There are several ways to define the PDE of the ALE map. However, only few of the equations used in practice will result in solutions which theoretically possess sufficient regularity. For a detailed discussion of the regularity and related issues we refer to [125]. In the following, we will refer to the PDE of the ALE map as the *geometry problem*. The solution of the ALE mapping will be uniquely defined by the boundary conditions. We assume that the boundary for the moving domain $\Omega_f(t)$ is known. Furthermore, we need to define the reference domain $\hat{\Omega}_f$. For practical reasons, this will be the initial domain of the fluid problem $\hat{\Omega}_f = \Omega_f(0)$.

The ALE map is defined as

$$\hat{T}_f(\hat{x},t) = \hat{x}_f + \hat{\mathbf{u}}_f(\hat{x},t),$$

where $\hat{\mathbf{u}}_f$ is the deformation of the fluid domain. The boundary position of $\partial\Omega_f(t)$ prescribes boundary values for the geometry problem. Further, we need to compute the interior values of $\Omega_f(t)$. The easiest choice is an harmonic extension of the boundary values to the interior. For this particular geometry problem we have to solve

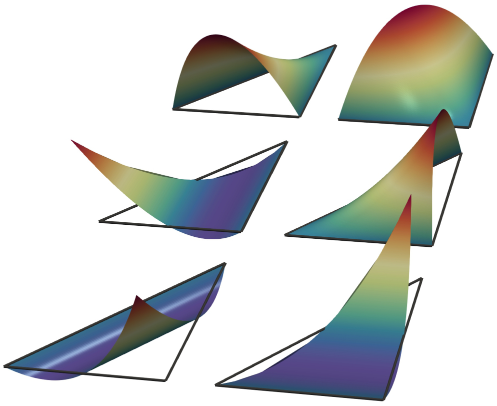
$$\begin{aligned} -\Delta\hat{\mathbf{u}}_f &= 0 \quad \text{in } \hat{\Omega}_f \text{ and} \\ \hat{\mathbf{u}}_f &= \mathbf{g}(t) \quad \text{on } \partial\hat{\Omega}_f \end{aligned}$$

with given displacements $\mathbf{g}(t)$ on the boundary. An issue with this geometry problem is the low regularity of the solution. If only a few edges of the triangulation of the solid domain are deformed alot, we can run into corner singularities, since the resulting elements of the fluid domain might be degenerated; cf. [125]. Furthermore, the prescribed deformation on the boundary might be large and the solution of the geometry problem in areas close to the large deformations can be too small, due to a fast decay of the harmonic extension. Then, the boundary of the solid domain might enter the fluid domain and we get self-penetrating elements. To circumvent this issue, we add a smoothing function to the harmonic geometry problem. The scaling parameter α which is constant for an element of the triangulation is added to the problem. The value of α is defined as the distance of the center of an element to the interface [125] and should be chosen in accordance with the underlying domain. We will fully define the smoothing function in section 5.1.7, where we discuss the finite element discretization of fluid-structure interaction problems for specific geometries. An alternative approach consists in the computationally more expensive pseudo elasticity problem. Here, the geometry problem is defined similar to the stationary linear elasticity problem of eqs. (3.13) to (3.15):

$$\begin{aligned} -\operatorname{div}(\boldsymbol{\sigma}) &= 0 \qquad \text{in } \hat{\Omega}_f \text{ and} \\ \hat{\mathbf{u}}_f &= \mathbf{g}(t) \qquad \text{on } \partial\hat{\Omega}_f. \end{aligned}$$

Different choices of geometry problems are presented in [125]. There, in addition to the harmonic operator and a pseudo elasticity problem, a biharmonic operator is discussed and a study of the three different choices is presented.

5 Discretizations in Space and Time



We want to compute approximate solutions for the previously presented Stokes and Navier–Stokes problems, for solid problems, as well as for fully coupled FSI problems. In this chapter, we present the discretization methods for the problems of Chapters 2 to 4. We use the *method of lines* for all time-dependent problems, i.e., we first discretize the spatial dimension and obtain a semi-discretized system. Only then, we apply a suitable discretization in time. Therefore, we present different methods to discretize *ordinary differential equations* (ODEs). Then, we apply these methods to our semi-discretized and time-depenent problems. In general, the methods for temporal discretizations can be separated in two classes. Firstly, we use one-step, multi-stage Runge–Kutta methods. Secondly, we employ multistep methods which are based on Backward Differentiation Formula schemes. Furthermore, we will present *Newmark methods*, which will be used to discretize the second-order systems of our solid problems. It is also possible to reverse this process and start with a temporal discretization, followed by a discretization in space. This approach is generally known as *Rothe's method.* Furthermore, there are methods which apply both discretizations at the same time and with the same discretization method; e.g., a problem can be discretized in space and time with the Galerkin approach and finite elements. Therefore, these approaches are denoted as *space-time methods.*

In the following section, we will start with the spatial discretization using the *Finite Element Method* (FEM). For an introduction to the FEM, see, e.g., [22]. Two other popular methods for spatial discretizations are the *Finite Difference Method* (FDM) [135] and the *Finite Volume Method* (FVM) [118].
In section 5.2, the discretization in time will be discussed. Furthermore, we will combine all previous discretizations in section 5.3 for the coupled FSI problem. We conclude this chapter with section 5.4, where we will present general solution strategies for nonlinear problems.

5.1 Model Problems and Spatial Discretization with Finite Elements

In the following, we assume that $\Omega \subset \mathbb{R}^d$, $d = 2, 3$ is fixed in time. This is sufficient for all problems except for the FSI problem. We will start by giving an overview of the weak formulations of our problems. Then, we will further specify different model problems with specific boundary conditions and geometries.

In Chapter 6 we will present the construction of our preconditioners for these model problems. Furthermore, we will discuss numerical results for our preconditioners used in the solution of these model problems in Chapter 8. There, the performance and efficiency of different preconditioners is studied for these model problems. Moreover, we will consider other problems in Chapter 8, which are not introduced in the following sections, since these problems are not used repeatedly throughout the numerical results of this thesis. However, the following weak formulations and the construction of preconditioners will be the same. The only differences are due to different geometries and boundary conditions.

When deriving weak formulations of PDEs, *Green's formula* is often used, which leads to boundary integrals. Dirichlet boundary values will be directly built into the space of trial and test functions. For the following weak formulations it will be assumed that the integrals over all Neumann boundaries $\partial\Omega_{\mathrm{N}}$ are zero if not stated otherwise. All function values for these boundaries, which lead to vanishing integrals, will be specified in the description of the specific problems. Furthermore, we use a slight change of notation in comparison to the fluid, solid, and FSI PDEs of Chapters 2 to 4. The solution we want to compute, will be generally noted as u in the scalar case or $\mathbf{u}$ if we seek to determine a vector field. Moreover, our space of test functions will generally be v or $\mathbf{v}$ for scalars and vector fields, respectively. As a reminder, in Chapters 2 to 4 we used $\mathbf{u}$ for displacements and $\mathbf{v}$ for velocities.

5.1.1 Stokes Problems

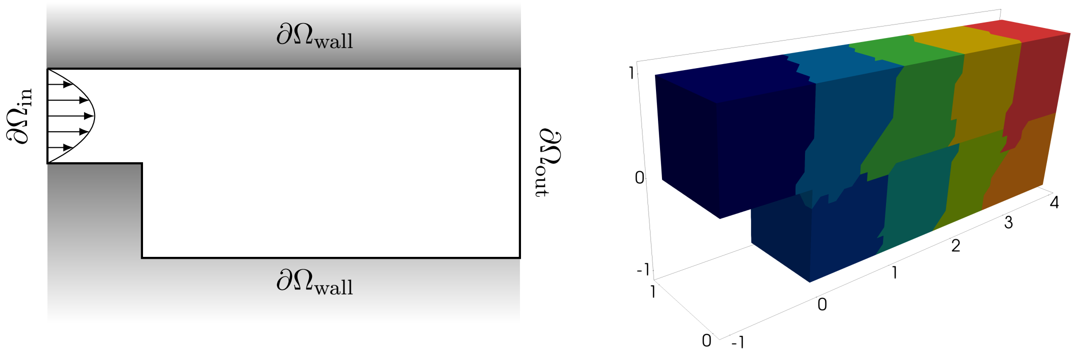

$\partial\Omega_{\text{wall}}$
$\partial\Omega_{\text{in}}$
$\partial\Omega_{\text{out}}$
$\partial\Omega_{\text{wall}}$
1
0
-1
1
0
-1
0
1
2
3
4

Figure 5.1: Cross-section (left) and unstructured domain decomposition into nine subdomains of the three-dimensional backward facing step geometry (right). The Dirichlet boundary $\partial\Omega_D$ consists of the inlet $\partial\Omega_{\text{in}}$ and the walls $\partial\Omega_{\text{wall}}$, the outlet $\partial\Omega_{\text{out}}$ is a Neumann boundary $\partial\Omega_N$; see fig. 5.2 for the resulting streamline solution of a Navier–Stokes problem. Taken from [80].

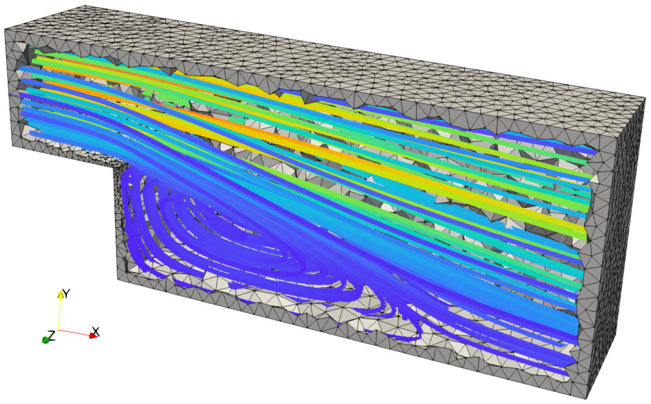


Figure 5.2: Streamline solution of a three-dimensional backward facing step Navier–Stokes problem. Taken from [80].

Our first model problem is given by the Stokes equations in two and three dimensions; cf. eqs. (2.27) and (2.28). We seek to determine the velocity $\mathbf{u} \in V_{\mathbf{g}}$ with $V_{\mathbf{g}} = \{\mathbf{v} \in (H^1(\Omega))^d : \mathbf{v}|_{\partial\Omega_D} = \mathbf{g}\}$ and the pressure $p \in Q$, with $\mathbf{g}$ and Q defined in the following, of an incompressible fluid with negligible advective forces by solving the variational formulation: find $(\mathbf{u}, p)$, such that

$$\mu \int_\Omega \nabla\mathbf{u} : \nabla\mathbf{v}\, d\mathbf{x} - \int_\Omega \operatorname{div}(\mathbf{v})\, p\, d\mathbf{x} = \int_\Omega \mathbf{f} \cdot \mathbf{v}\, d\mathbf{x} \quad \forall \mathbf{v} \in V_0, \tag{5.1}$$

$$-\int_\Omega \operatorname{div}(\mathbf{u})\, q\, d\mathbf{x} = 0 \quad \forall q \in L^2(\Omega), \tag{5.2}$$

with $V_0 \subset (H^1(\Omega))^d$, dynamic viscosity μ, and $\partial\Omega_D = \partial\Omega$. For two matrices $A = (a_{ij})_{i,j=1}^n$ and $B = (b_{ij})_{i,j=1}^n$ we have $A : B := \sum_{i,j=1}^n a_{ij}b_{ij}$.

In our numerical tests, we consider the (leaky) lid-driven cavity Stokes problem; cf. [63], which we will denote as *LDC Stokes problem*: let Ω be the unit square or cube in two or three dimensions, respectively. We prescribe the velocity boundary conditions by

$$d = 2: \ \mathbf{g} = (1,0)^T \ \text{ if } \ x_2 = 1, \ \mathbf{g} = (0,0)^T \ \text{ if } \ x_2 < 1,$$
$$d = 3: \ \mathbf{g} = (1,0,0)^T \text{ if } x_3 = 1, \ \mathbf{g} = (0,0,0)^T \text{ if } x_3 < 1,$$

where $Q = L_0^2(\Omega)$, and we choose $\mu = 1$ and $\mathbf{f} \equiv 0$ for the Stokes problems of this thesis. Further, we consider a problem which we will denote as *channel Stokes problem*: let Ω be a channel with $\Omega = [0,4] \times [0,1]$ and $\Omega = [0,4] \times [0,1]^2$ in two and three dimensions, respectively. The inflow and outflow boundary conditions are

$$\begin{aligned} &d = 2: \ \mathbf{g} = (4x_2(1-x_2), 0)^T && \text{on } \partial\Omega_{\text{in}}, \\ &d = 3: \ \mathbf{g} = (16x_2(1-x_2)x_3(1-x_3), 0, 0)^T && \text{on } \partial\Omega_{\text{in}}, \\ &\frac{\partial\mathbf{u}}{\partial\mathbf{n}} - p\mathbf{n} = 0 && \text{on } \partial\Omega_{\text{out}}, \end{aligned}$$

with the outward pointing normal vector $\mathbf{n}$ and Neumann boundary $\partial\Omega_{\text{out}}$. In two and three dimensions the Neumann boundary is $\partial\Omega_{\text{out}} = \{(4, x_2)^T \in \mathbb{R}^2 : 0 < x_2 < 1\}$ and $\partial\Omega_{\text{out}} = \{(4, x_2, x_3) \in \mathbb{R}^3 : 0 < x_2, x_3 < 1\}$, respectively. On the remainder of the boundary, we set no-slip boundary conditions and $Q = L^2(\Omega)$ is used.

Furthermore, we consider the following three-dimensional Stokes problem. We seek to determine the velocity $\mathbf{u} \in V_{\mathbf{g}}$ and the pressure $Q = L^2(\Omega)$, such that eq. (5.1) holds. We consider the three-dimensional *backward facing step* (BFS) geometry shown in fig. 5.1; cf. [63] for the two-dimensional geometry. The Dirichlet boundary conditions at the inflow and the walls are given by

$$\mathbf{g} = \begin{cases} (16u_{\max}x_2(1-x_2)x_3(1-x_3), 0, 0)^T & \text{for } x \in \partial\Omega_{\text{in}}, \\ (0,0,0)^T & \text{for } x \in \partial\Omega_{\text{wall}}. \end{cases}$$

At the outlet, we prescribe the do-nothing boundary condition, i.e.,

$$\frac{\partial \mathbf{u}}{\partial \mathbf{n}} - p\mathbf{n} = 0 \quad \text{on } \partial\Omega_{\text{out}}.$$

Furthermore, we will use $u_{\max} = 1.0$ and denote this problem as the *BFS Stokes problem.*

5.1.2 Navier–Stokes Problems

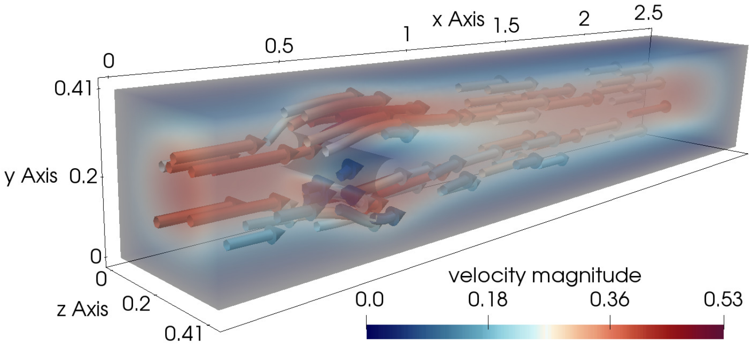


Figure 5.3: Velocity solution of the steady Navier–Stokes benchmark problem; $Re = 20$, $(x, y, z) = (x_1, x_2, x_3)$. Taken from [79].

Next, we consider the steady-state Navier–Stokes equations modeling the flow of an incompressible Newtonian fluid with kinematic viscosity $\nu > 0$; cf. eqs. (2.25) and (2.26). We seek to determine the velocity $\mathbf{u} \in V_{\mathbf{g}}$ and the pressure $p \in Q$, with $\mathbf{g}$ and Q defined in the following, by solving the variational formulation: find $(\mathbf{u}, p)$, such that

$$\begin{aligned}\nu \int_\Omega \nabla \mathbf{u} : \nabla \mathbf{v} \, d\mathbf{x} \;+\; \int_\Omega (\mathbf{u} \cdot \nabla \mathbf{u}) \cdot \mathbf{v} \, d\mathbf{x} - \int_\Omega \operatorname{div}(\mathbf{v}) \, p \, d\mathbf{x} &= \int_\Omega \mathbf{f} \cdot \mathbf{v} \, d\mathbf{x} \quad \forall \mathbf{v} \in V_0, \\ - \int_\Omega \operatorname{div}(\mathbf{u}) \, q \, d\mathbf{x} &= 0 \qquad\qquad\quad \forall q \in L^2(\Omega).\end{aligned}$$

The term $(\mathbf{u} \cdot \nabla \mathbf{u})$ leads to a nonlinear system. Solution strategies for nonlinear systems are discussed in section 5.4. We will consider two different steady Navier–Stokes model problems for the numerical scaling experiments. For all Navier–Stokes problems, the source function is $\mathbf{f} \equiv 0$.

The first model problem is a regularized lid-driven cavity Navier–Stokes problem, similar to the two-dimensional problem in [63], with $\partial\Omega_D = \partial\Omega$ and $Q = L_0^2(\Omega)$. We refer to it as *LDC Navier–Stokes problem*. The boundary values of the problem are given by

$$\begin{aligned}&d = 2: \; \mathbf{g} = (4x_1(1-x_1), 0)^T \;\text{ if }\; x_2 = 1, \; \mathbf{g} = (0,0)^T \;\text{ if }\; x_2 < 1, \\ &d = 3: \; \mathbf{g} = (16x_1(1-x_1)x_2(1-x_2), 0, 0)^T \text{ if } x_3 = 1, \; \mathbf{g} = (0,0,0)^T \text{ if } x_3 < 1.\end{aligned}$$

As a second steady-state Navier–Stokes problem, we use the domain and boundary conditions of the backward facing step Stokes problem and choose the kinematic viscosity $\nu = 0.01$. This problem will be denoted *BFS Navier–Stokes problem*.

Moreover, we consider the time-dependent Navier–Stokes equations, which model the flow of an incompressible Newtonian fluid with kinematic viscosity $\nu > 0$; cf. eq. (2.23) and eq. (2.24). We seek to determine the velocity $\mathbf{u}(\mathbf{x}, t) \in V_\mathbf{g}$ and the pressure $p(\mathbf{x}, t) \in Q \subset L^2(\Omega)$ by solving the variational formulation: find $(\mathbf{u}, p)$, such that

$$\begin{aligned}\int_\Omega \frac{\partial \mathbf{u}}{\partial t} \cdot \mathbf{v} \, d\mathbf{x} \;+\; \nu \int_\Omega \nabla \mathbf{u} : \nabla \mathbf{v} \, d\mathbf{x} \;+\; \int_\Omega (\mathbf{u} \cdot \nabla \mathbf{u}) \cdot \mathbf{v} \, d\mathbf{x} & \\ - \int_\Omega \operatorname{div}(\mathbf{v}) \, p \, d\mathbf{x} = \int_\Omega \mathbf{f} \cdot \mathbf{v} \, d\mathbf{x} & \qquad \forall \mathbf{v} \in V_0, \\ - \int_\Omega \operatorname{div}(\mathbf{u}) \, q \, d\mathbf{x} = \; 0 \qquad\qquad & \qquad \forall q \in L^2(\Omega).\end{aligned}$$

As we have seen previously for the steady Navier–Stokes problem, the presence of the convection term $\mathbf{u} \cdot \nabla \mathbf{u}$ leads to a nonlinear system. In the steady case, we solve the system using *Newton's method* or *Picard iterations* (fixed point system), cf. [63], whereas, in the time-dependent case, we can also use an extrapolation $\mathbf{u}^*$ to linearize the convective part, i.e.,

$$\mathbf{u} \cdot \nabla \mathbf{u} \approx \mathbf{u}^* \cdot \nabla \mathbf{u}.$$

The next model problem is the three-dimensional Navier–Stokes benchmark for the flow around a cylinder with circular cross-section, to which we will refer as the *Navier–Stokes benchmark*; cf. [130], where a detailed description of the simulation setup is given. See fig. 5.3 for the benchmark geometry and the velocity of the solution of the Navier–Stokes benchmark. The length of the domain is 2.5 m, $A = 0.41\,\mathrm{m}$ is the height and width of the domain. Furthermore, we choose $\nu = 10^{-3}\,\mathrm{m}^2/\mathrm{s}$, and we define $V_{\mathbf{g}} = \{\mathbf{v} \in H^1(\Omega)^d : \mathbf{v}|_{\partial\Omega_D} = \mathbf{g}\}$, $V_0 = \{\mathbf{v} \in H^1(\Omega)^d : v|_{\partial\Omega_D} = 0\}$, and $Q = L^2(\Omega)$. The inflow and outflow boundary conditions are

$$\begin{aligned} \mathbf{u} = \mathbf{g} &= (16 u_{max} x_2 x_3 (A - x_2)(A - x_3)/A^4, 0, 0)^T && \text{on } \partial\Omega_{\text{in}}, \\ \nu \frac{\partial \mathbf{u}}{\partial \mathbf{n}} - p\mathbf{n} &= 0 && \text{on } \partial\Omega_{\text{out}}, \end{aligned}$$

respectively. The Dirichlet inflow boundary is $\partial\Omega_{\text{in}} = \{(0, x_2, x_3) \in \mathbb{R}^3 : 0 < x_2, x_3 < A\}$ and the Neumann outflow boundary is $\partial\Omega_{\text{out}} = \{(2.5, x_2, x_3) \in \mathbb{R}^3 : 0 < x_2, x_3 < A\}$. We use the maximum velocity across the inflow $u_{max} = 0.45 m/s$. On the remainder of the boundary, we set no-slip boundary conditions.

In the dimensionless reformulation of the Navier–Stokes equations, the Reynolds number Re specifies the relative contributions of convection and diffusion; cf. section 2.4. We obtain $\mathrm{Re} = L\bar{u}/\nu$ with the characteristic length scale L and maximum inflow velocity $\bar{u}$. For the LDC Navier–Stokes problem in two and three dimensions, we obtain $\mathrm{Re} = 1/\nu$. The Reynolds number of the benchmark problem with a circular obstacle is $Re = 20$. In our numerical tests for the steady BFS Navier–Stokes problem, we set $L = 2$ as the height of the outlet and choose the maximum inflow velocity $\bar{u} = 1.0$, and $\nu = 0.01$; i.e., $\mathrm{Re} = 200$. When dealing with higher Reynolds numbers instabilities in the numerical simulation can appear. In particular, the advective term or, more precisely, the gradient cannot be approximated properly with the given mesh since the velocity difference between of two neighbouring nodes might be too large. In some cases a finer mesh could be used to resolve these steep gradients. However, this is, in general, not possible and suitable stabilization techniques should be applied instead; cf., e.g, [27]. The problems considered in this thesis do not need a stabilization.

5.1.3 Elasticity Problems

Next, we consider stationary and time-dependent nonlinear elasticity problems for different material laws of eqs. (3.10) to (3.12) and eqs. (3.7) to (3.9), respectively. We seek to determine the displacement $\mathbf{u}(\mathbf{x}, t) \in V_{\mathbf{g}}$ by solving the variational formulation: find $\mathbf{u}$, such that

$$\int_\Omega \frac{\partial^2 \mathbf{u}}{\partial t^2} \cdot \mathbf{v}\, d\mathbf{x} + \int_\Omega \mathbf{F}\boldsymbol{\Sigma}(\mathbf{u}) : \nabla \mathbf{v}\, d\mathbf{x} = \int_\Omega \mathbf{f} \cdot \mathbf{v}\, d\mathbf{x}, \quad \forall \mathbf{v} \in V_0, \tag{5.3}$$

where $V_{\mathbf{g}} = \{\mathbf{v} \in (H^1(\Omega))^d : \mathbf{v}_{|\partial\Omega_D} = \mathbf{g}\}$. For the stationary problem, we use $\frac{\partial^2 \mathbf{u}}{\partial t^2} = 0$ and the first integral vanishes.

For time-dependent and stationary nonlinear elasticity problems, Newton's method is used to linearize $\boldsymbol{\Sigma}(\mathbf{u})$. We mainly consider the hyperelastic St. Venant–Kirchhoff material for our numerical experiments and restate the linear material law for convenience:

$$\boldsymbol{\Sigma}(\mathbf{u}) = 2\mu \mathbf{E} + \lambda\, \mathrm{trace}(\mathbf{E}) I,$$

where the nonlinear strain tensor is given by $\mathbf{E} := \frac{1}{2}\left(\mathbf{C} - I\right)$, and $\mathbf{C}$ is the right Cauchy–Green tensor.

We consider the unit cube $\Omega = [0,1]^3$, the boundary conditions

$$\begin{aligned} \mathbf{u} &= 0 && \text{on } \partial\Omega_D := \{0\} \times [0,1]^2, \\ \mathbf{n} \cdot \mathbf{F}\boldsymbol{\Sigma} &= 0 && \text{on } \partial\Omega_N := \partial\Omega \setminus \partial\Omega_D, \end{aligned}$$

and the body force $\mathbf{f} = (0, -100, 0)^T$. We denote this problem as the *Cube problem*. In addition, we also consider a time-dependent problem where we use a body force $\mathbf{f} = (-20, 0, 0)^T$, for $t < 5 \cdot 10^{-3}$, and $\mathbf{f} = 0$, afterwards. This problem will be computed on a foam-like structure with a softer surrounding tissue. Therefore, we will denote it as *Foam problem*; cf. fig. 8.12 in section 8.5 of the numerical results, for a visualization of the foam structure.

5.1.4 Almost Incompressible Linear Elasticity Problems

Next, we consider mixed linear elasticity problems describing the displacement $\mathbf{u} \in V$ and the pressure $p = -\lambda \operatorname{div} \mathbf{u} \in Q$ of a body consisting of an almost incompressible material. We fix the body along $\Gamma_0 \subset \partial\Omega$ and let the body be subject to the external body force $\mathbf{f}$: find $(\mathbf{u}, p)$, such that

$$\begin{aligned} 2\mu \int_\Omega \boldsymbol{\epsilon}(\mathbf{u}) : \boldsymbol{\epsilon}(\mathbf{v})\, d\mathbf{x} &- \int_\Omega \operatorname{div}(\mathbf{v})\, p\, d\mathbf{x} = \int_\Omega \mathbf{f} \cdot \mathbf{v}\, d\mathbf{x} \quad && \forall \mathbf{v} \in V, \\ - \int_\Omega \operatorname{div}(\mathbf{u})\, q\, dx &- \frac{1}{\lambda} \int_\Omega p\, q\, d\mathbf{x} \quad = 0 && \forall q \in L^2(\Omega), \end{aligned}$$

with $V = \{\mathbf{v} \in H^1(\Omega)^d : \mathbf{v}_{|\Gamma_0} = 0\}$. The components of the linearized strain tensor $\boldsymbol{\epsilon}(\mathbf{v})$ are $\boldsymbol{\epsilon}(\mathbf{v})_{ij} = \frac{1}{2}(\partial v_i/\partial x_j + \partial v_j/\partial x_i)$; cf. section 3.2. While a pure displacement formulation for the linear elasticity problem may suffer from volume locking in the incompressible limit, the above mixed formulation is a good remedy; cf., e.g., [22]. The incompressibility of the material is modeled by λ approaching infinity or by the Poisson ratio $\nu = \lambda/2(\lambda + \mu)$ approaching 0.5. In particular, we fix $\mu = 1.0$ and increase λ accordingly for our numerical tests.

Firstly, we consider a three-dimensional model problem: let Ω be the unit cube and $\Gamma_0 = \partial\Omega$. Further, we choose a uniformly distributed random right-hand side and $Q = L_0^2(\Omega)$. We denote this problem as *mixed linear elasticity (MLE) cube*. Moreover, we will study a second MLE problem on a beam. This problem is denoted as *MLE beam* problem. In particular, we use the following domain and boundary conditions: let $\Omega = [0, 4] \times [0, 1] \times [0, 1]$ be the computational domain which is clamped at $\Gamma_0 = \{(0, x_2, x_3) \in \mathbb{R}^3 : 0 \leq x_2, x_3 \leq 1\}$. Furthermore, we use the external force $\mathbf{f} = (0, 0, -0.01)^T$ and $Q = L^2(\Omega)$.

5.1.5 Finite Elements for Elliptic Problems

The finite element discretization for elliptic problems, such as the stationary elasticity problems, or the elliptic part of our hyperbolic problems, such as the time-dependent elasticity problems, is straightforward. Therefore, we first introduce a triangulation τ_h of Ω into triangles or tetrahedra with characteristic mesh size h, which can be nonuniform. We use the following spaces for our finite element discretization

$$V^h(\Omega) = \{\mathbf{v}_h \in (C(\Omega))^d \cap (H^1(\Omega))^d : \mathbf{v}_h|_T \in P_i \, \forall \, T \in \tau_h\},$$

with $i = 1$ for piecewise linear and $i = 2$ for piecewise quadratic elements on triangular or tetrahedral meshes; $C(\Omega)$ is the space of continuous functions on Ω. Again, we refer to [22] for an introduction to the FEM. In particular, we obtain the following semi-discretized equation for a time-dependent linear elasticity problem:

$$M\mathbf{u}_{tt} + K\mathbf{u} = F, \tag{5.4}$$

where $\mathbf{u}$ are the displacements and $\mathbf{u}_{tt}$ is the acceleration. Here, $M \in \mathbb{R}^{n \times n}$ is the solid mass matrix, $K \in \mathbb{R}^{n \times n}$ is the stiffness matrix, and F is the discretized vector of applied forces. In the case of a nonlinear elasticity problem, we further need a suitable linearization. In general, we will use Newton's method to solve nonlinear problems. Then, we need to replace K with the Jacobian matrix. We will discuss the temporal discretization in section 5.2.4.

5.1.6 Finite Elements for Saddle Point Problems

For the spatial discretization of the incompressible fluid flow problems and mixed elasticity problems, we use mixed finite elements. In the following, we will consider three different mixed finite element pairs. Again, we use a triangulation with characteristic mesh size h into triangles, tetrahedra, and/or hexahedra.

Then, we introduce the conforming discrete piecewise quadratic velocity and piecewise linear pressure spaces

$$\begin{aligned} V^h(\Omega) &= \{\mathbf{v}_h \in (C(\Omega))^d \cap (H^1(\Omega))^3 : \mathbf{v}_h|_T \in P_2 \, \forall \, T \in \tau_h\} \text{ and} \\ Q^h(\Omega) &= \{q_h \in C(\Omega) \cap L^2(\Omega) : q_h|_T \in P_1 \, \forall \, T \in \tau_h\}, \end{aligned}$$

respectively, of Taylor–Hood (P2–P1) mixed finite elements.
Secondly, we introduce the discrete piecewise linear velocity space

$$V^h(\Omega) = \{v_h \in (C(\Omega))^d \cap (H^1(\Omega))^d : \mathbf{v}_h\big|_T \in P_1 \, \forall \, T \in \tau_h\},$$

which, together with the piecewise linear pressure space Q^h, forms the pair of equal-order P1–P1 mixed finite elements. Thirdly, we introduce the pair of Q2–P1disc mixed finite elements which consists of the conforming discrete piecewise biquadratic velocity and discontinuous piecewise linear pressure spaces

$$\begin{aligned} &V^h(\Omega) = \{\mathbf{v}_h \in (C(\Omega))^d \cap (H^1(\Omega))^d : \mathbf{v}_h\big|_T \in Q_2 \, \forall \ \in \tau_h\} \text{ and} \\ &Q^h(\Omega) = \{q_h \in L^2(\Omega) : q_h\big|_T \in P_1 \, \forall \, T \in \tau_h\}. \end{aligned}$$

The mapped version of the pressure space is used; cf. [19]. We only consider structured meshes for the Q2–P1disc discretization, where all hexahedra are cubes. Finite element spaces for velocity and pressure are not independent. In order to guarantee the existence of a unique solution, pairs of finite element spaces for saddle point problems must satisfy the famous discrete *inf–sup conditon*; cf., e.g, [20, 73]. Let $(\mathbf{u}_h, p) \in V^h \times Q^h$ be solutions of the system

$$\begin{aligned} a^h(\mathbf{u}_h, \mathbf{v}_h) + b^h(p_h, \mathbf{v}_h) &= \langle \mathbf{f}, \mathbf{v}_h \rangle \quad \forall \, \mathbf{v}_h \in V^h, \\ b^h(q_h, \mathbf{u}_h) &= \langle g, q_h \rangle \quad \ \forall \, q_h \in Q^h, \end{aligned}$$

where $a^h(\cdot, \cdot)$ and $b^h(\cdot, \cdot)$ are the bilinear forms of our incompressible fluid flow problems. The discrete inf–sup condition is satisfied if

$$\inf_{\forall q_h \in Q^h \setminus \{0\}} \sup_{\forall \mathbf{v}_h \in V^h \setminus \{0\}} \frac{b^h(\mathbf{v}_h, q_h)}{\|\mathbf{v}_h\|_{V^h} \|q_h\|_{Q^h}} \geq \gamma_h,$$

with $\gamma_h > 0$ being independent of the characteristic mesh size h and the dimension of the problem. The inf–sup condition is satisfied for the P2–P1 and Q2–P1disc pairs in two and three dimensions; cf. [20]. However, the P1–P1 pair is not inf–sup stable and we need a suitable stabilization. Therefore, a stabilizing matrix C is added to the following discrete systems; we will specify C shortly hereafter.

The resulting discrete Stokes systems, linearized steady Navier–Stokes systems, and mixed linear elasticity systems have the generic form

$$\mathcal{F}x = \begin{bmatrix} F & B^T \\ B & -C \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} = b, \tag{5.5}$$

with $\mathcal{F} \in \mathbb{R}^{n\times n}$ and $x, b \in \mathbb{R}^n$. The derivation of block and monolithic preconditioners as well as the numerical results for saddle point problems are focused on Stokes and Navier–Stokes problems. Therefore, we denote our general saddle point problem as $\mathcal{F}$ for fluid. Nevertheless, many of the preconditioners can be built and applied nearly equivalently to a mixed elasticity problem.

In general, we obtain the following semi-discretized time-dependent Navier–Stokes problem:

$$\begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_t \\ p_t \end{bmatrix} + \begin{bmatrix} F & B^T \\ B & -C \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix}. \tag{5.6}$$

In section 5.2.3, we will describe a discretization in time of this problem. With a slight misuse of notation, we will always use the notation of system (5.5) for the stationary problem when describing our preconditioners in Chapter 6. However, this will also include the consideration of the fully discretized time-dependent problems, which result from eq. (5.6).

When using P2–P1 or Q2–P1disc mixed finite elements, for incompressible fluid flow problems, we choose $C = 0$, and for the unstable P1–P1 pair, we use the *Bochev–Dohrmann stabilization* [48]. This pressure stabilization is inserted in the conservation of mass equation and penalizes unphysical pressure oscillations. The stabilization term reads

$$C = \frac{1}{\nu} \int_\Omega (p - \rho_0 p)(q - \rho_0 q)\, d\mathbf{x} = \ 0 \quad \forall q \in L^2(\Omega), \tag{5.7}$$

where ρ_0 is an L^2-projection onto the space of discontinuous constant functions $P_0(\tau_h)$. Thus, the projection can be computed locally on each element as

$$\rho_0 q_{|T} = \frac{1}{\int_T d\mathbf{x}} \int_T q\, d\mathbf{x}. \tag{5.8}$$

In case of a mixed linear elasticity system, we always consider a block $C \neq 0$.

For the model problems considered here, the matrix B^T arises from the discretization of the term

$$\int_\Omega \operatorname{div}(\mathbf{v}_h)\, p_h \, d\mathbf{x}\,.$$

after Green's formula was used to transform the integral over $\mathbf{v}_h \cdot \nabla p$. Therefore, the nullspace of B^T consists of all constant pressure functions if no pressure value is prescribed with boundary conditions; i.e, we consider Dirichlet boundary conditions for all velocity *degrees of freedom* (d.o.f.) and $\partial\Omega_D = \partial\Omega$. If the matrix C is symmetric positive definite, the pressure is uniquely determined. Therefore, we restrict the pressure to the space

$$\overline{Q}^h = Q^h \cap L_0^2(\Omega),$$

i.e., all functions must have zero mean value. In order to do so, we can introduce a Lagrange multiplier λ to enforce

$$\int_\Omega p_h \, d\mathbf{x} = 0. \tag{5.9}$$

This results in the block system

$$\overline{\mathcal{F}}\overline{x} = \begin{bmatrix} F & B^T & 0 \\ B & -C & a^T \\ 0 & a & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ 0 \\ 0 \end{bmatrix}, \tag{5.10}$$

where the vector a arises from the finite element discretization of the integral in eq. (5.9). Alternatively, we can use the projection

$$\overline{P} = I_p - a^T(aa^T)^{-1}a, \tag{5.11}$$

onto the space $\overline{Q}^h$, where I_p is the pressure identity matrix.

In our derivations of the new monolithic preconditioners, we first concentrate on fluid flow problems where $\partial\Omega_D = \partial\Omega$, such that the pressure is normalized by (5.9). In section 6.3.4, we will give a brief overview of the projection method to enforce a zero mean value for the pressure. In case of a global problem with natural boundary conditions, i.e., $\partial\Omega_N \neq \emptyset$, the pressure generally does not belong to $\overline{Q}^h$, and we use a saddle point system formulation as in eq. (5.5). As mentioned, the construction of preconditioners in Chapter 6 focuses on fluid flow problems. Therefore, we will use the terms velocity and pressure in the description of our preconditioners. The basic construction of a preconditioner for a mixed linear elasticity problem does not change significantly. However, the nullspaces of Neumann problem matrices, i.e., matrices without Dirichlet boundary conditions, for a Stokes and a mixed linear elasticity problem are different. The nullspace of a Stokes problem consists of all translations, while the nullspace of an elasticity problem additionally contains rotations.

5.1.7 Fluid-Structure Interaction Problems

To discretize the monolithic FSI problems, we use the discretizations which were presented in section 5.1.6 for the time-dependent Navier–Stokes equations and in section 5.1.3 for elasticity problems. We only use conforming finite elements for the fluid and solid subproblems, which results in a one-to-one matching of nodes on the interface $\hat{\Omega}_s$ and $\hat{\Omega}_f$. From now on we will use $\hat{\boldsymbol{\sigma}}_s$ instead of $\hat{\mathbf{F}}\hat{\boldsymbol{\Sigma}}_s$ for a simplified notation of the solid equations and corresponding coupling conditions. Furthermore, we set the source terms of fluid and solid to zero. A slight change in notation is made w.r.t. the fluid, solid and geometry variables compared to Chapter 4. We denote the fluid velocity as $\mathbf{u}_f$, the solid displacement as $\mathbf{d}_s$ and the fluid mesh displacement as $\mathbf{d}_g$. The weak formulation of the FSI problem in eqs. (4.20) to (4.25) with Eulerian coordinates and the ALE time derivative for the fluid reads: find $\mathbf{u}_f \in V_{\mathbf{g}}^{(f)} = \{\mathbf{v}_f \in (H^1(\Omega_f(t)))^d : \mathbf{v}_f|_{\partial\Omega_f(t),D} = \mathbf{g}_f\}$, $p \in L_2(\Omega_f(t))$, $\hat{\mathbf{d}}_s \in \hat{V}_{\mathbf{g}}^{(s)} = \{\hat{\mathbf{v}}_s \in (H^1(\hat{\Omega}_s))^d : \hat{\mathbf{v}}_s|_{\partial\hat{\Omega}_{s,D}} = \mathbf{g}_s\}$, and

$\hat{\mathbf{d}}_f \in \hat{V}_{\mathbf{g}}^{(g)} = \{\hat{\mathbf{v}}_g \in (H^1(\hat{\Omega}_f))^d : \mathbf{v}|_{\partial\hat{\Omega}_{f,D}} = \mathbf{g}_g\}$, such that

$$\begin{aligned}
\int_{\Omega_f(t)} \rho_f \frac{\partial \mathbf{u}_f}{\partial t}|_{\hat{T}} \cdot \mathbf{v}_f \, d\mathbf{x} \, + \int_{\Omega_f(t)} \boldsymbol{\sigma}_f : \nabla \mathbf{v}_f \, d\mathbf{x} & \\
+ \int_{\Omega_f(t)} \rho_f((\mathbf{u}_f - \mathbf{w}) \cdot \nabla \mathbf{u}_f) \cdot \mathbf{v}_f \, d\mathbf{x} - \int_{\Gamma(t)} \boldsymbol{\sigma}_f \mathbf{n}_f \cdot \mathbf{v}_f \, ds = 0 & \qquad \forall \mathbf{v}_f \in V_0^f, \\
- \int_{\Omega_f(t)} \operatorname{div}(\mathbf{u}_f) \, q \, d\mathbf{x} = 0 & \qquad \forall q \in L^2(\Omega_f), \\
\int_{\hat{\Omega}_s} \hat{\rho}_s \frac{\partial^2 \hat{\mathbf{d}}_s}{\partial t^2} \cdot \hat{\mathbf{v}}_s \, + \, \hat{\boldsymbol{\sigma}}_s : \hat{\nabla} \hat{\mathbf{v}}_s \, d\hat{\mathbf{x}} \, - \int_{\hat{\Gamma}} \hat{\boldsymbol{\sigma}}_s \hat{\mathbf{n}} \cdot \hat{\mathbf{v}}_s \, d\hat{s} = 0 & \qquad \forall \hat{\mathbf{v}}_s \in \hat{V}_0^{(s)}, \\
\int_{\hat{\Omega}_f} \alpha \hat{\nabla} \hat{\mathbf{d}}_f : \hat{\nabla} \hat{\mathbf{v}}_g \, d\hat{\mathbf{x}} = 0 & \qquad \forall \hat{\mathbf{v}}_g \in \hat{V}_0^{(g)} \\
-\hat{\boldsymbol{\sigma}}_f \hat{\mathbf{n}}_f = \hat{\boldsymbol{\sigma}}_s \hat{\mathbf{n}}_s, & \quad \text{on } \hat{\Gamma}, \\
\mathbf{u}_f \circ \hat{T} = \frac{\partial \hat{\mathbf{d}}_s}{\partial t}, & \quad \text{on } \hat{\Gamma}, \\
\hat{\mathbf{d}}_f = \hat{\mathbf{d}}_s & \quad \text{on } \hat{\Gamma}.
\end{aligned} \tag{5.12}$$

For the first time in the derivation of our weak formulations, we have not used a vanishing integral over the Neumann boundaries of fluid and solid. We rather keep the two integrals

$$\begin{aligned}
&\int_{\Gamma(t)} \boldsymbol{\sigma}_f \mathbf{n}_f \cdot \mathbf{v}_f \, ds \quad \text{and} \\
&\int_{\hat{\Gamma}} \hat{\boldsymbol{\sigma}}_s \hat{\mathbf{n}} \cdot \hat{\mathbf{v}}_s \, d\hat{s}
\end{aligned}$$

in our formulation. One of the coupling conditions is the equality of stresses in normal direction of fluid and solid: $-\hat{\boldsymbol{\sigma}}_f \hat{\mathbf{n}}_f = \hat{\boldsymbol{\sigma}}_s \hat{\mathbf{n}}_s$. Clearly, for the normal vectors on the fluid-solid interface $-\hat{\mathbf{n}}_f = \hat{\mathbf{n}}_s$ holds. We will use the two interface integrals above and a Lagrangian multiplier to enforce this coupling in a weak sense. Details on the fully discretized system will be given in section 5.3.1. Again, we want to use the method of lines to discretize the above system of equations. Therefore, we first discretize the spatial variables with the FEM, followed by a discretization of the time derivatives. To do so, we need to transform the partial temporal derivative of the fluid momentum equation to a total derivative of the whole integral; cf. [70].

$$
\begin{aligned}
\int_{\Omega_f(t)} \rho_f \frac{\partial \mathbf{u}_f}{\partial t}|_{\hat{T}} \cdot \mathbf{v}_f \, d\mathbf{x} &= \int_{\hat{\Omega}_f} \rho_f \hat{J}_f \frac{\partial \hat{\mathbf{u}}_f}{\partial t} \cdot \hat{\mathbf{v}}_f \, d\hat{\mathbf{x}} \\
&= \frac{d}{dt} \int_{\hat{\Omega}_f} \hat{J}_f \rho_f \hat{\mathbf{u}}_f \cdot \hat{\mathbf{v}}_f \, d\hat{\mathbf{x}} - \int_{\hat{\Omega}_f} \hat{J}_f \rho_f \widehat{\mathrm{div}}(\mathbf{w}) \hat{\mathbf{u}}_f \cdot \hat{\mathbf{v}}_f \, d\hat{\mathbf{x}} \\
&= \frac{d}{dt} \int_{\Omega_f(t)} \rho_f \mathbf{u}_f \cdot \mathbf{v}_f \, d\mathbf{x} - \int_{\Omega_f(t)} \rho_f \,\mathrm{div}(\mathbf{w}) \mathbf{u}_f \cdot \mathbf{v}_f \, d\mathbf{x}.
\end{aligned}
\tag{5.13}
$$

As a model problem, we will consider the following benchmark problem of Richter [125] for the numerical results. The problem will be denoted as the *FSI benchmark problem*. Note that we discretize the full domain: in [125] the problem is cut in half and a symmetry condition is enforced. In fig. 5.4, the computational domains of fluid and structure are shown. We prescribe the velocity in x-direction

$$
u_x\,(y,z) = \frac{y(H-y)(H^2-z^2)}{(H/2)^2 H^2} \frac{9}{8} u_{\mathrm{mean}}.
$$

Moreover, the following boundary conditions are used:

$$
\begin{aligned}
\mathbf{u} = \mathbf{g} = (\beta(t)\, u_x(y,z),\, 0,\, 0)^T \quad &\text{on } \partial\Omega_{f,\mathrm{in}} \\
\nu_f \frac{\partial \mathbf{u}}{\partial \mathbf{n}} - p\mathbf{n} = 0 \text{ on } \partial\Omega_{f,\mathrm{out}}.&
\end{aligned}
$$

The mean velocity is $u_{\mathrm{mean}} = 1\,\mathrm{m/s}$ and $\beta(t)$ is the scaling of the inflow velocity, which allows for a gradual increase of the velocity from 0 at $t = 0$ to the maximum velocity at $t = 2$:

$$
\beta(t) = \begin{cases} \frac{1}{2}(1 - \cos(\pi t/2)), & \text{for } t < 2, \\ 1, & \text{for } t \geq 2. \end{cases}
$$

The following material parameters are used for the FSI benchmark problem: fluid density $\rho_f = 10^3\,\mathrm{kg/m^3}$, kinematic viscosity $\nu_f = 10^{-3}\,\mathrm{m^2/s}$, solid density $\rho_s = 10^3\,\mathrm{kg/m^3}$, shear modulus $\mu_s = 5{\cdot}10^5\,\mathrm{kg/(m{\cdot}s^2)}$ and Poisson's ratio $\nu_s = 0.4$. Furthermore, we employ a nonlinear elasticity problem with the St. Venant–Kirchhoff material law. As already mentioned in section 4.3, we use a scaling

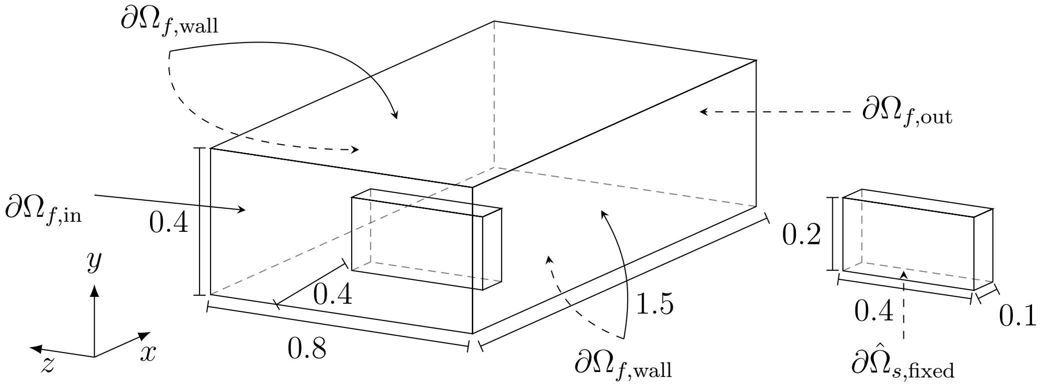

$\partial\Omega_{f,\text{wall}}$
$\partial\Omega_{f,\text{out}}$
$\partial\Omega_{f,\text{in}}$
0.4
0.2
y
0.4
1.5
0.4
0.1
z
x
0.8
$\partial\Omega_{f,\text{wall}}$
$\partial\hat{\Omega}_{s,\text{fixed}}$

Figure 5.4: Full three-dimensional FSI benchmark with fluid and structure domain (left). Separated structure domain for better visibility (right). All values are in meters.

parameter α to smooth the ALE map. We define the distance of a node $\hat{\mathbf{x}} \in \hat{\Omega}_f$ to the interface $\hat{\Gamma}$ as

$$d_{\hat{\Gamma}}(\hat{\mathbf{x}}) = \min_{\forall \hat{\mathbf{y}} \in \hat{\Gamma}} ||\hat{\mathbf{x}} - \hat{\mathbf{y}}||. \tag{5.14}$$

We use the following elementwise definition of the scaling parameter α on an element T of the finite element discretization:

$$\alpha|_T = \begin{cases} 10^6, & \text{if } d_{\hat{\Gamma}}(\mathbf{a}) < dist, \\ 1, & \text{else,} \end{cases}$$

with $\mathbf{a}$ being the center of gravity of and element T and *dist* a suitable distance. We could also use a function with a more gradual decay. However, for the problems under consideration the above scaling gives good results with a critical distance $dist = 0.03$ for the FSI benchmark problem. The geometry problem possesses a homogeneous Dirichlet boundary on $\partial\Omega_{f,\text{wall}}$, $\partial\Omega_{f,\text{in}}$, and $\partial\Omega_{f,\text{out}}$.

Furthermore, we will consider an FSI problem for a realistic artery. Details of this problem will be given together with the results in section 8.6.

5.2 Temporal Discretization

In this section, we describe discretizations of the previously presented semi-discretized and time-dependent problems. However, we will start with a simple ODE and explain the single-step and multi-step methods for this problem. The transfer to more complex block systems, such as the semi-discretized Navier–Stokes problem, will be done in a second step. There, the general structure of the methods is not changed, although it is important to note that the incompressibility equation of our fluid flow problems leads to a *differential algebraic equation* (DAE). This section mainly follows the presentation of the book *Solving Ordinary Differential Equations II* [76]. Since we deal with stiff problems, we are restricted to implicit methods to circumvent stability issues. For a detailed discussion on stability, we refer to [76]. We will start this section by introducing implicit *Runge-Kutta* (RK) methods for ODEs. More precisely, we will focus on *diagonally implicit RK* (DIRK) methods. Furthermore, we will shortly describe BDF methods. In contrast to RK methods, these BDF methods rely on several past solutions, instead of only one. Then, BDF and RK methods will be applied to our time-dependent, semi-discretized Navier–Stokes problem. After that, we will present the family of *Newmark–β* methods. They will be used to discretize our second-order differential equations which arise after the discretization in space of our time-dependent elasticity problems. We conclude this section by combining the temporal discretizations of the fluid and the solid for our monolithic FSI systems.

5.2.1 Implicit Runge–Kutta Methods

Consider the following first-order initial value problem:

$$y' = f(x,y), \quad y(x_0) = y_0,$$

where $x, y \in \mathbb{R}$ and $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. The above problem has the solution

$$y(x_1) = y_0 + \int_{x_0}^{x_1} f(x, y(x))\, dx. \tag{5.15}$$

Here, we use h as the characteristic parameter of our discretization and set $x_1 = x_0 + h$. Using a quadrature rule to approximate the integral in eq. (5.15) leads to the following definition of the RK methods which is taken from [75] in a slightly reduced form.

Definition 3. *Let b_i, a_{ij} $(i, j = 1, \ldots, s)$ be real numbers and let c_i be defined by*

$$c_i = \sum_{j=1}^{s} a_{ij}.$$

The method

$$\begin{aligned} k_i &= f(x_0 + c_i h, y_0 + h \sum_{j=1}^{s} a_{ij} k_j), \quad i = 1, \ldots, s, \\ y_1 &= y_0 + h \sum_{i=1}^{s} b_i k_i, \end{aligned}$$

is called an s-stage Runge–Kutta method. If $a_{ij} = 0$ for $i < j$ and at least one $a_{ii} \neq 0$, we have a diagonal implicit Runge–Kutta (DIRK) method.

RK methods can be summarized with *Butcher tables*, which possess the following form:

$$\begin{array}{c|c} \mathbf{c} & \mathrm{A} \\ \hline & \mathbf{b}^T \end{array} = \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array}$$

In the construction of RK methods, the three following conditions generally need to be satisfied; cf. [76].

$$\begin{aligned} B(p) &: \sum_{i=1}^{s} b_i c_i^{q-1} = \frac{1}{q}, && q = 1, ..., p; \\ C(\eta) &: \sum_{j=1}^{s} a_{ij} c_j^{q-1} = \frac{c_i^q}{q}, && i = 1, ..., s, \quad q = 1, ..., \eta; \\ D(\zeta) &: \sum_{i=1}^{s} b_i c_i^{q-1} a_{ij} = \frac{b_j}{q}(1 - c_j^q), && j = 1, ..., s, \quad q = 1, ..., \zeta; \end{aligned}$$

If $B(p)$ is satisfied, then the resulting quadrature formula (b_i, c_i) is of order p. The essence of $C(\eta)$ and $D(\zeta)$ is captured by the following theorem of Butcher; cf. [28].

Theorem 3. *If the coefficients b_i, c_i, and a_{ij} of a Runge–Kutta method satisfy $B(p), C(\eta)$, and $D(\zeta)$ with $p \leq \eta + \zeta + 1$ and $p \leq 2\eta + 2$ then the method is of order p, with $p, \eta, \zeta \in \mathbb{N}$.*

Some widely used implicit methods can be written in the form of Butcher tables, among them are the first-order *implicit Euler method* and the θ-methods. With $\theta = 1/2$, we obtain the *Crank–Nicolson* method, which is a second-order method. **Implicit Euler method:**

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 0 & 1 \\ \hline & 0 & 1 \end{array}$$

θ-method:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1-\theta & \theta \\ \hline & 1-\theta & \theta \end{array}$$

Both methods are stiffly accurate. We call a method *stiffly accurate* if $c_s = 1$ and $b_j = a_{sj},\ j = 1, \ldots, s$. These methods have the advantage that we can directly use the last computed k_s to obtain the new solution

$$y_1 = y_0 + hk_s.$$

A third order method which is implemented in our software is $DIRK34$; cf. [98] and the literature therein for a derivation of this method. It is a four stage, diagonally implicit, and stiffly accurate method which possesses the following nonzero coefficients:

$$a_{21} = a_{22} = a_{33} = a_{44} = 0.1558983899988677,$$
$$a_{31} = 1 - a_{32} - a_{22}, a_{32} = 1.072486270734370,$$
$$a_{42} = 0.7685298292769537, a_{43} = 0.09666483609791597.$$

By definition 3, it follows that $c_i = \sum_{j=1}^{i} a_{ij}$, $i = 1, \dots, 4$. This concludes the short overview of Runge–Kutta methods. In section 5.2.3, we will apply these methods to Navier–Stokes problems. Other closely related methods are the Rosenbrock–Wanner methods; cf. [76] for a general introduction and [98] for an application to Navier–Stokes problems.

5.2.2 Multi-Step Methods – Backward Differentiation Formulas

In general, a k-step multi-step methods is defined as

$$\alpha_k y_{m+k} + \alpha_{k-1} y_{m+k-1} + \cdots + \alpha_0 y_m = h(\beta_k f_{m+k} + \cdots + \beta_0 f_m);$$

cf. [76]. We will consider the following linear BDF multi-step methods. The *BDF1* method is identical to the implicit Euler method and only makes use of the last solution. In contrast, the *BDF2* method uses the last two solutions

$$y_{m+2} - \frac{4}{3} y_{m+1} + \frac{1}{3} y_m = \frac{2}{3} h f_{m+2},$$

to compute the new solution y_{m+2}. BDF2 is a second-order A-stable method. Moreover, there are no linear multi-step methods which are A-stable and have an order greater than two. Again, we refer to [76] for detailed introduction to the stability theory of stiff ODEs.

5.2.3 Discretization of the Time-Dependent Navier–Stokes Problems

In this section, we apply the previously presented Runge–Kutta and BDF methods to the time-dependent Navier–Stokes problem. We will only consider stable spatial discretizations. Stabilized Navier–Stokes problems can be treated analogously. The application of Runge–Kutta methods follows the presentation in [98]. Let M be a mass matrix, $\tau_m = t_{m+1} - t_m$ be the time step length, and $(\mathbf{u}_{m+1}, p_{m+1})^T$ the new solution which we want to compute. It is possible to use an embedded method for the computation of a lower-order solution. Furthermore, the higher-order and lower-order solution can be used to compute an error and consequently choose a more appropriate time step. Therefore, the time length τ_m can change from one time step to the next.

Let $G(t, \mathbf{u}, p)$ be the operator of all terms of the conservation of momentum equation except the temporal derivate and let $H(t, \mathbf{u})$ be the operator of the incompressibility condition; cf. eq. (5.6) Then, a semi-discretized Navier–Stokes problem can be written as

$$\begin{aligned} M\mathbf{u}_t &= G(t, \mathbf{u}, p), \\ 0 &= H(t, \mathbf{u}). \end{aligned}$$

We obtain the following update rules for the above DAE of index 2 after applying a DIRK scheme.

$$Mk_i = G(t_m + c_i\tau_m, \mathbf{U}_i, P_i), \quad \mathbf{U}_i = \mathbf{u}_m + \tau_m \sum_{j=1}^{i} a_{ij}\mathbf{k}_j, \ i = 1, \ldots, s, \tag{5.16}$$

$$0 = H(t_m + c_i\tau_m, \mathbf{U}_i), \quad P_i = p_m + \tau_m \sum_{j=1}^{i} a_{ij}l_j, \ i = 1, \ldots, s, \tag{5.17}$$

$$\mathbf{u}_{m+1} = \mathbf{u}_m + \sum_{i=1}^{s} b_i\mathbf{k}_i, \quad p_{m+1} = p_m + \sum_{i=1}^{s} b_i l_i. \tag{5.18}$$

Next, we multiply the second equation of (5.16) with the fluid mass matrix M and make use of the first equation of eq. (5.16) accordingly.

$$M\mathbf{U}_i = M\mathbf{u}_m + \tau_m \sum_{j=1}^{i} a_{ij}G(t_m + c_j\tau_m, \mathbf{U}_j, P_j), \tag{5.19}$$

$$0 = H(t_m + c_i\tau_m, \mathbf{U}_i). \tag{5.20}$$

Let $F(\cdot)$ be a suitable linearization of the $(1, 1)$-block of the Navier–Stokes problem. We can now apply eq. (5.19) and eq. (5.20) to our semi-discretized time-dependent Navier–Stokes problem which yields

$$\begin{bmatrix} M + \tau_m a_{ii} F(\mathbf{U}_i) & \tau_m a_{ii} B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \mathbf{U}_i \\ P_i \end{bmatrix} = \\ \begin{bmatrix} M\mathbf{u}_m + \tau_m \sum_{j=1}^{i-1} a_{ij}(\mathbf{f}(t_m + c_j\tau_m) - F(\mathbf{U}_j)\mathbf{U}_j - B^T P_j) + a_{ii}\tau_m \mathbf{f}(t_m + c_i\tau_m) \\ 0 \end{bmatrix}, \tag{5.21}$$

for $i = 1, \ldots, s$. Since we only consider methods with $a_{11} = 0$, eq. (5.21) reduces to $(\mathbf{U}_1, P_1)^T = (\mathbf{u}_m, p_m)^T$ for the first stage. We can now solve all s-stages of eq. (5.21) to compute the new solution $(\mathbf{u}_{m+1}, p_{m+1})^T$.

The application of the BDF2 method of section 5.2.2 to time Navier–Stokes problem with source term $\mathbf{f} = 0$ yields the following system:

$$\begin{bmatrix} \frac{1}{\tau_m} M + F(\mathbf{u}_{m+1}) & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_{m+1} \\ p_{m+1} \end{bmatrix} = \begin{bmatrix} \frac{4}{3\tau_m} M\mathbf{u}_m + \frac{1}{3\tau_m} M\mathbf{u}_{m-1} \\ 0 \end{bmatrix}$$

Here, we scaled the mass matrix with the time step length τ_m in order to get a consistent scaling for our FSI problem; cf. section 5.3.1. The temporal discretization with BDF2 is used in simulations of FSI problems; see section 8.6 for the simulation results.

We need to highlight an important issue for discretizations of Navier–Stokes problems with Runge–Kutta methods. In eq. (5.21) we use the scaled matrix $\tau_m a_{ii} B^T$ which is then multiplied with the pressure p in the same way as the part $M + \tau_m a_{ii} F(\mathbf{U}_i)$ which is multiplied with the velocity. However, in the derivation of the Navier–Stokes equations in Chapter 2, we used the pressure as a Lagrange multiplier for the divergence condition $\operatorname{div} \mathbf{u} = 0$. The above approach leads to an unphysical behavior of the pressure solution. This unphysical behavior can be denoted as *ringing* since the pressure solution changes rapidly from one time step to the next. In fig. 5.5, velocity and pressure solutions from consecutive time steps are depicted. There, the Crank–Nicolson method is used. The solutions are computed for a the two-dimensional Navier–Stokes benchmark, cf. [130], with constant parabolic inflow (maximum inflow velocity $u_{\max} = 1.5$), viscosity $\nu = 10^{-3}$, and Reynolds number $Re = 100$. Additionally, the pressure solution does not only suffer from the ringing phenomenon, but also from an instantaneous start of the simulation. The maximum inflow velocity is applied from the first time step with a Dirichlet boundary condition. However, there is no suitable initial pressure solution p_0. A realistic flow field is only obtained after a few time steps, but only the velocity solution can recover from the instantaneous start. In contrast, the pressure solution cannot recover due to the ringing phenomenon. A remedy is a fully implicit treatment of the pressure in the Crank-Nicolson time discretization. The resulting Navier–Stokes system is

$$\begin{bmatrix} M + \Delta t\theta F(\mathbf{u}_{m+1}) & \Delta t B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_{m+1} \\ p_{m+1} \end{bmatrix} = \begin{bmatrix} M\mathbf{u}_m - \Delta t(1-\theta)F(\mathbf{u}_m)\mathbf{u}_m \\ 0 \end{bmatrix}, \tag{5.22}$$

and the solutions are depicted in fig. 5.6. In eq. (5.22), the source term $\mathbf{f} = 0$ was used. In our implementation of these methods, we will treat the pressure fully implicit if the ringing phenomenon would occur otherwise.

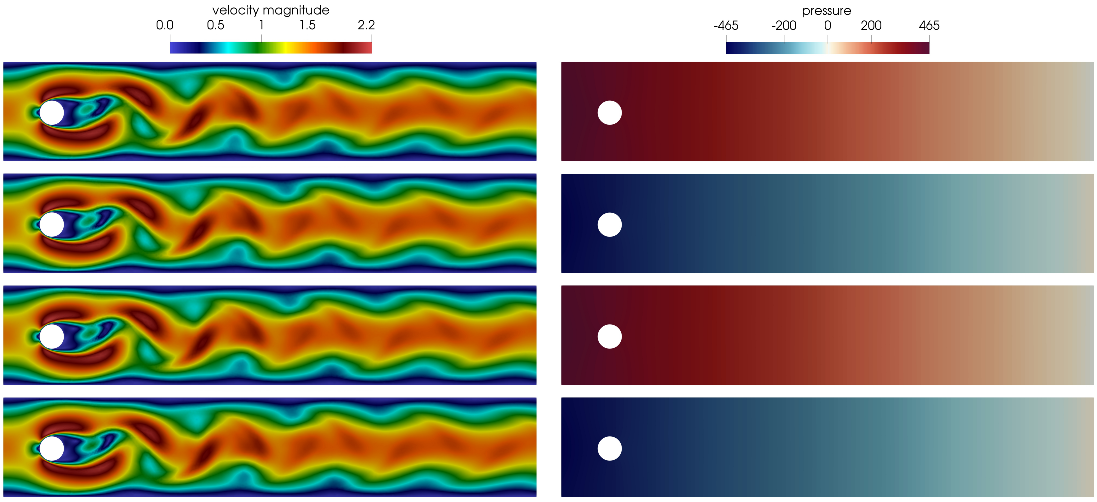


Figure 5.5: Velocity solution (left) and pressure solution (right). Ringing phenomenon of the pressure for the Crank–Nicolson time discretization. Depicted are four consecutive time steps with $\Delta t = 0.01$ after the flow field is fully developed.

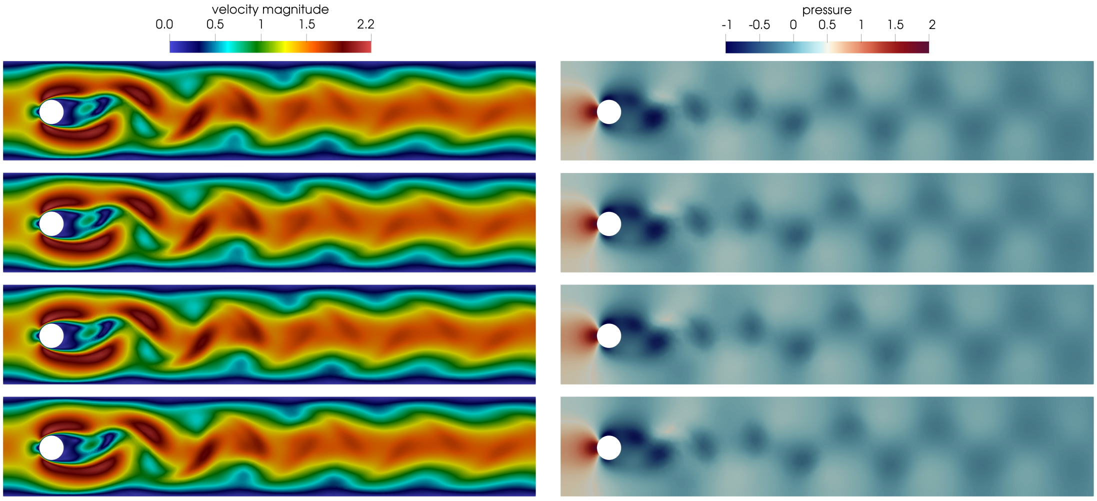


Figure 5.6: Velocity solution (left) and pressure solution (right). Fully implicit treatment of the pressure for the Crank–Nicolson time discretization. Depicted are four consecutive time steps with $\Delta t = 0.01$ after the flow field is fully developed.

5.2.4 Newmark Schemes for Solid Dynamic Problems

In sections 5.1.3 and 5.1.5, we derived the spatial discretization of our time-dependent elasticity equation. We will begin the presentation of Newmark methods by stating the linear semi-discretized equation of motion, which is similar in structure to eq. (5.4) of the semi-discretized elasticity problem. However, we consider an additional viscous damping matrix $C \in \mathbb{R}^{n \times n}$. Furthermore, let M be the mass matrix and K the stiffness matrix. The semi-discretized equation of motion is given by

$$M\mathbf{u}_{tt} + C\mathbf{u}_t + K\mathbf{u} = F, \tag{5.23}$$

where $\mathbf{u}_t$ is the velocity and $\mathbf{u}_{tt}$ the acceleration. Clearly, the equation above is a second-order system and there are two ways to proceed.

Firstly, we can rewrite eq. (5.23) to a first-order system and use a Runge–Kutta or BDF method. However, this approach would increase the dimension of the resulting first-order system by a factor of two compared to the second-order system. Therefore, the computational costs to compute a solution would increase significantly. Secondly, we can directly discretize the second-order problem. The most widely used methods for solid dynamic problems are Newmark schemes [120]. The following description and details on Newmark schemes can be found in [15]. In general, one-step methods of the Newmark family possess the following form:

$$\begin{aligned} M\mathbf{a}_{m+1} + C\mathbf{v}_{m+1} + K\mathbf{u}_{m+1} &= F_{m+1}, \\ \mathbf{u}_{m+1} &= \mathbf{u}_m + \Delta t \mathbf{v}_m + \frac{\Delta t^2}{2}\left(\left(1-2\beta\right)\mathbf{a}_m + 2\beta\mathbf{a}_{m+1}\right), \\ \mathbf{v}_{m+1} &= \mathbf{v}_m + \Delta t\left(1-\gamma\right)\mathbf{a}_m, \end{aligned} \tag{5.24}$$

where $\mathbf{a}_m$, $\mathbf{v}_m$, and $\mathbf{u}_m$ are the known acceleration, velocity, and displacement at the time step m, respectively. We can rewrite the first equation of (5.24) to a system for the unknown displacement $\mathbf{u}_{m+1}$ and update the unknown velocity $\mathbf{v}_{m+1}$ and acceleration $\mathbf{a}_{m+1}$ afterwards, if $\beta \neq 0$. In the following, we will always use a Newmark scheme with $\beta = 1/4$ and $\gamma = 1/2$, which is known as *constant average acceleration method*. It is implicit and unconditionally stable. Moreover, Newmark schemes with $\gamma = 1/2$ are second-order accurate; cf. [15]. We will now apply the Newmark scheme to our nonlinear time-dependent elasticity problem. Furthermore, we linearize eq. (5.3) with Newton's method. Let $J(\mathbf{u}_m^{(k)}) \in \mathbb{R}^{n\times n}$ be the Jacobian matrix of the stationary part of the elasticity equation. In each iteration of Newton's method, we have to solve the following problem for the displacement and update the velocity and acceleration afterwards. In particular, for a nonlinear time-dependent elasticity problem the system reads

$$M\mathbf{a}_{m+1}^{(k+1)} + J(\mathbf{u}_{m+1}^{(k)})\mathbf{u}_{m+1}^{(k+1)} = F_{m+1}, \tag{5.25}$$

where the updates for velocity and acceleration are analogously to (5.24). Again, eq. (5.25) is rewritten to a system for the displacements.

5.3 Discretization of the Fluid-Structure Interaction Problem

In this section, we will present the discretization of the FSI problem. In particular, the temporal and spatial discretizations of the fluid and solid subproblems are combined. Furthermore, the coupling conditions on the interface for the stresses, velocities, and (mesh) displacements are enforced. In section 5.3.1, discrete representations of the coupling conditions are presented for the nonlinear FSI systems. This is followed by a discussion of linearization strategies in section 5.3.2. In particular, we will derive two FSI systems. A fully monolithic system, which includes the geometry problem and a second FSI problem, where the geometry problem is treated separately.

5.3.1 Coupling of the Fluid-Structure Interaction Problem

The coupling conditions of an FSI problem are essential to the interaction of fluid and solid. We use the coupling matrices C_i, $i = 1, \ldots, 5$, which arise due to the coupling of the velocities of fluid and solid as well as due to the Lagrangian multiplier λ which is introduced to satisfy the continuity of normal stress at the interface.

The matrices C_1 and C_2 are used to couple the fluid velocity and the solid velocity on the interface $\Gamma(t)$:

$$\mathbf{u}_f = \hat{\mathbf{u}}_s \circ \hat{T}_s^{-1};$$

cf. eq. (4.23). We can define an equivalent coupling on the interface $\hat{\Gamma}$ for the reference configuration, which was already used in section 5.1.7 for the weak formulation of the FSI system.

$$\mathbf{u}_f \circ \hat{T}_s = \hat{\mathbf{u}}_s$$

Therefore, C_1 is a restriction matrix from all fluid d.o.f. to the velocity interface d.o.f.. Similarly, $\bar{C}_2$ is the restriction of displacement d.o.f. to the interface. Additionally, we need to scale $\bar{C}_2$ with the negative inverse of the time step length, i.e., $C_2 = -1/\tau_m \bar{C}_2$, to enforce the equality of velocities. Furthermore, the coupling of stresses in normal direction is enforced with a Lagrangian multiplier. There, the prolongation $\bar{C}_3 = C_1^T$ from the velocity interface d.o.f. to all fluid d.o.f. and a second prolongation operator $C_4 = \bar{C}_2^T$ from the displacement interface d.o.f. to all displacement d.o.f. is used. It is important to note that the matrix C_3, which is used for the weakly coupled fluid stresses, depends on the time stepping scheme. Suppose, we have the following semi-discretized equation of linear momentum conservation

$$M\mathbf{u}_t + F\mathbf{u} + B^T p = 0,$$

where we ignore the nonlinearity of F. If we use a θ-scheme (without fully implicit treatment of the pressure) and time step length Δt. Thus, we obtain the two equivalent equations:

$$\begin{aligned}
&M\mathbf{u}_{n+1} + \theta\Delta t(F\mathbf{u}_{n+1} + B^T p_{n+1}) = M\mathbf{u}_n - (1-\theta)\Delta t(F\mathbf{u}_n + B^T p_n),\\
&\frac{1}{\theta\Delta t} M\mathbf{u}_{n+1} + \left(F\mathbf{u}_{n+1} + B^T p_{n+1}\right) = \\
&\qquad \frac{1}{\theta\Delta t}\left(M\mathbf{u}_n - (1-\theta)\,\Delta t\left(F\mathbf{u}_n + \theta\Delta t B^T p_n\right)\right).
\end{aligned}$$

We can directly use the coupling matrix $C_3 = \bar{C}_3 = C_1^T$ if we use the second equation. However, if we use the first equation we have to account for the scaling $\theta\Delta t$ of F and B^T. In particular, we must use the scaled coupling matrix $C_3 = \theta\Delta t\bar{C}_3$, because it will add the fluid stresses on the interface to the conservation of momentum equation. In general, similar considerations must be made for the coupling matrix C_4 of the solid stresses. However, we exclusively use Newmark schemes for the time-dependent solid problem and therefore no scaling is needed. The final coupling matrix C_5 is used to model the coupling condition

$$\hat{\mathbf{d}}_f = \hat{\mathbf{d}}_s \quad \text{on } \hat{\Gamma}.$$

Since, we restrict the solid displacement d.o.f. to the interface, we obtain $C_5 = \bar{C}_2$.

After a finite element discretization in space, cf. section 5.1.2 for the fluid subproblem and section 5.1.3 for the solid subproblem, we use the BDF2 method for the fluid, cf. section 5.2.3, and the Newmark method for the solid, cf. section 5.2.4, to discretize the temporal derivatives. We can now simply combine the subproblems and coupling conditions to the monolithic FSI problem. In the following, we denote the discretized time-dependent Navier–Stokes problem as $\mathcal{F}$ and the corresponding vector of velocity and pressure as $\mathcal{U} = (\mathbf{u}^T, p^T)^T$. Furthermore, the discretized time-dependent solid problem is denoted as S and the geometry problem as G. Note that the fluid equations must be assembled in accordance with the mesh movement, which is due to the ALE setting. In particular, the mesh velocity must be subtracted for the advective term; cf. eq. (5.13). Thus $\mathcal{F}$ slightly changes in comparison to the separated Navier–Stokes problem. Moreover, we need to assemble an additional matrix for our Navier–Stokes problem, which is due to the ALE time derivative:

$$\int_{\Omega_f(t)} \rho_f \operatorname{div}(\mathbf{w})\mathbf{u}_f \cdot \mathbf{v}_f \, d\mathbf{x},$$

where $\mathbf{w}$ is the mesh velocity. This term was derived in eq. (5.13) and we need to account for it in our linearization; cf. section 5.3.2. Moreover, we need to account for the mesh movement in each iteration, i.e., we need to reassemble all matrices of the fluid subproblem, because, in general, the underlying domain $\Omega_f(t)$ has changed between two consecutive time steps. We obtain the fully coupled nonlinear FSI problem

$$\begin{pmatrix} \mathcal{F}(\mathcal{U}^{n+1}, \hat{\mathbf{d}}_f^{\,n+1}) & + & 0 & + & C_3\lambda^{n+1} & + & 0 \\ 0 & + & S(\hat{\mathbf{d}}_s^{\,n+1}) & + & C_4\lambda^{n+1} & + & 0 \\ C_1\mathcal{U}^{n+1} & + & C_2\hat{\mathbf{d}}_s^{\,n+1} & + & 0 & + & 0 \\ 0 & + & C_5\hat{\mathbf{d}}_s^{\,n+1} & + & 0 & + & G\hat{\mathbf{d}}_f^{\,n+1} \end{pmatrix} = \begin{pmatrix} b_f^n \\ b_s^n \\ C_2\hat{\mathbf{d}}_s^{\,n} \\ 0 \end{pmatrix}, \tag{5.26}$$

where $b_f = (b_u^T, b_p^T)^T$ and b_s are the fluid and the solid right-hand side, respectively. Note that the following approximation was used for the strongly coupled solid velocity on the interface:

$$\frac{d}{dt}\hat{\mathbf{d}}_s \approx \frac{\hat{\mathbf{d}}_s^{n+1} - \hat{\mathbf{d}}_s^n}{\Delta t} \quad \text{on } \hat{\Gamma},$$

with the equivalent matrix representation

$$C_2\hat{\mathbf{d}}_s^{\,n+1} - C_2\hat{\mathbf{d}}_s^{\,n}.$$

Therefore, the coupling of velocities is now enforced with

$$C_1\mathcal{U}^{n+1} + (C_2\hat{\mathbf{d}}_s^{\,n+1} - C_2\hat{\mathbf{d}}_s^{\,n}) = 0.$$

We call the system in eq. (5.26) *geometry implicit* (GI) since the geometry problem is a part of the monolithic system. In contrast, we define the following *geometry explicit* (GE) system:

$$\begin{pmatrix} \mathcal{F}(\mathcal{U}^{n+1}, \hat{\mathbf{d}_f}^{n}) & + & 0 & + & C_3\lambda^{n+1} \\ 0 & + & S(\hat{\mathbf{d}_s}^{n+1}) & + & C_4\lambda^{n+1} \\ C_1\mathcal{U}^{n+1} & + & C_2\hat{\mathbf{d}_s}^{n+1} & + & 0 \end{pmatrix} = \begin{pmatrix} b_f^n \\ b_s^n \\ C_2\hat{\mathbf{d}_s}^{n} \end{pmatrix}. \tag{5.27}$$

Here, the geometry problem $G\hat{\mathbf{d}_f}^{n+1} = 0$ is decoupled and can therefore be solved separately from eq. (5.27). Furthermore, we do not need to reassemble all matrices of the fluid subproblem in each nonlinear iteration, since the underlying fluid mesh is only changed after the geometry problem is solved. In the next section, we discuss different linearization techniques for the GI system and GE system.

5.3.2 Linearization of the Fluid-Structure Interaction Problem

We begin with the linearization of the GE system of eq. (5.27), which is essentially a combination of the linearized Navier–Stokes and solid subproblems. Additionally, we account for the fluid mesh velocity and use the known fluid mesh displacements $\hat{\mathbf{d}_f}^{n}$ and $\hat{\mathbf{d}_f}^{n-1}$ of the last two time steps for the assembly of the fluid problem. For simplicity, only the dependence on the last solution $\hat{\mathbf{d}_f}^{n}$ is highlighted in eq. (5.27). The mesh velocity is defined as

$$\mathbf{w}^n = \frac{\hat{\mathbf{d}_f}^{n} - \hat{\mathbf{d}_f}^{n-1}}{\Delta t}.$$

If we discretize the GI system using Newton's method we need to assemble partial derivatives of the Navier–Stokes equations w.r.t. the fluid mesh deformation $\hat{\mathbf{d}_f}^{n+1}$. These derivatives are called *shape derivatives*. For a derivation of shape derivatives, we refer to [68]. For a detailed discussion on shape derivatives in a fully coupled monolithic FSI system, we refer to [38]. In Chapter 7, we present a C++ software library which provides the implementation of the FSI problems. T his implementation in based on the master's thesis [33]. The Jacobian matrix of the GI system for Newton's method has the following form:

$$\mathcal{J}(\mathbf{X}^{n+1,k}) = \begin{pmatrix} \mathcal{D}_{\mathcal{U}}(\mathcal{F}\mathcal{U}) & 0 & C_3 & \mathcal{D}_{\hat{\mathbf{d}}_f}(\mathcal{F}\hat{\mathbf{d}}_f) \\ 0 & D_{\hat{\mathbf{d}}_s}(S\hat{\mathbf{d}}_s) & C_4 & 0 \\ C_1 & C_2 & 0 & 0 \\ 0 & C_5 & 0 & G \end{pmatrix} |_{\mathbf{X}^{n+1,k}} \tag{5.28}$$

The term $\mathcal{D}_{\hat{\mathbf{d}}_f}(\mathcal{F}\hat{\mathbf{d}}_f)$ corresponds to the shape derivatives. Apart from the Newton system, we can use an extrapolation or the Oseen system (fixed point system) for the fluid block. In both cases, we simply use a system with the form of (5.28) with $\mathcal{D}_{\hat{\mathbf{d}}_f}(\mathcal{F}\hat{\mathbf{d}}_f) = 0$.

We want to highlight the structure and coupling of the discretized and linearized FSI system. Therefore, we distinguish between interface d.o.f. Γ and interior d.o.f. I. We further distinguish between interior fluid d.o.f. I_f, interior solid d.o.f. I_s, and interior geometry d.o.f. I_g. Note that, with the above definition, an interior d.o.f. can still be a boundary d.o.f. of the corresponding domain. However, it is only an inner d.o.f. w.r.t the fluid-solid interface Γ. Furthermore, we will not distinguish between different linearized systems, e.g., the Jacobian matrix of eq. (5.28), a fixed point system, or a system resulting from an extrapolation. Therefore, we will use the variables F, B, B^T, C for the fluid problem in block form, S for the solid problem, G for the geometry problem, and $D_{\hat{\mathbf{d}}_s,u}$, $D_{\hat{\mathbf{d}}_s,p}$ with $\mathcal{D}_{\hat{\mathbf{d}}_s} = \left(D^T_{\hat{\mathbf{d}}_s,u}, D^T_{\hat{\mathbf{d}}_s,p}\right)^T$ for the shape derivatives. Then, the fully coupled GI FSI system matrix has the form

$$\left(\begin{array}{ccc:cc:c|cc} F_{I_f I_f} & F_{I_f\Gamma} & B^T_{I_f} & 0 & 0 & 0 & D_{\hat{\mathbf{d}}_s,u,I_f I_f} & D_{\hat{\mathbf{d}}_s,u,I_f\Gamma} \\ F_{\Gamma I_f} & F_{\Gamma\Gamma} & B^T_{\Gamma} & 0 & 0 & I & D_{\hat{\mathbf{d}}_s,u,\Gamma I_f} & D_{\hat{\mathbf{d}}_s,u,\Gamma\Gamma} \\ B_{I_f} & B_{\Gamma} & -C & 0 & 0 & 0 & D_{\hat{\mathbf{d}}_s,p,I_f} & D_{\hat{\mathbf{d}}_s,p,\Gamma} \\ \hdashline 0 & 0 & 0 & S_{I_s I_s} & S_{I_s\Gamma} & 0 & 0 & 0 \\ 0 & 0 & 0 & S_{\Gamma I_s} & S_{\Gamma\Gamma} & I & 0 & 0 \\ \hdashline 0 & I & 0 & 0 & -\frac{1}{\Delta t}I & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & G_{I_f I_f} & G_{I_f\Gamma} \\ 0 & 0 & 0 & 0 & -I & 0 & 0 & I \end{array}\right),$$

where $I \in \mathbb{R}^{|\Gamma|\times|\Gamma|}$ is the identity matrix, τ is the time step length, and we assume a temporal discretization which does not require a scaling of the coupling matrices. In case of a GE system, we drop all matrices right of and under the solid line:

$$\left(\begin{array}{ccc:cc:c}
F_{I_f I_f} & F_{I_f \Gamma} & B_{I_f}^T & 0 & 0 & 0 \\
F_{\Gamma I_f} & F_{\Gamma\Gamma} & B_{\Gamma}^T & 0 & 0 & I \\
B_{I_f} & B_{\Gamma} & -C & 0 & 0 & 0 \\
\hdashline
0 & 0 & 0 & S_{I_s I_s} & S_{I_s \Gamma} & 0 \\
0 & 0 & 0 & S_{\Gamma I_s} & S_{\Gamma\Gamma} & I \\
\hdashline
0 & I & 0 & 0 & -\frac{1}{\Delta t} I & 0
\end{array}\right).$$

5.4 Solution Methods for Nonlinear Equations

In this section, we give a short description of two widely used methods for the solution of nonlinear PDEs.

Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a sufficiently smooth function. In general, we seek to determine a root of f, i.e.,

$$f(u) = 0.$$

Firstly, we consider the *fixed point iteration*, where we seek to determine the fixed point u of

$$g(u) = u,$$

by evaluating the following expression until convergence

$$u^{k+1} = g(u^k),$$

with given initial guess u^0. The convergence rate of the fixed point methods is, in general, only linear. However, speedups can be achieved by using *Anderson Acceleration*; cf. [145] or the original work [4].

Secondly, we consider Newton's method. There, a second-order approximation of the underlying function is used. Let $J_{ij} = \frac{\partial f_i}{\partial u_j}$ be the Jacobian matrix of f. We

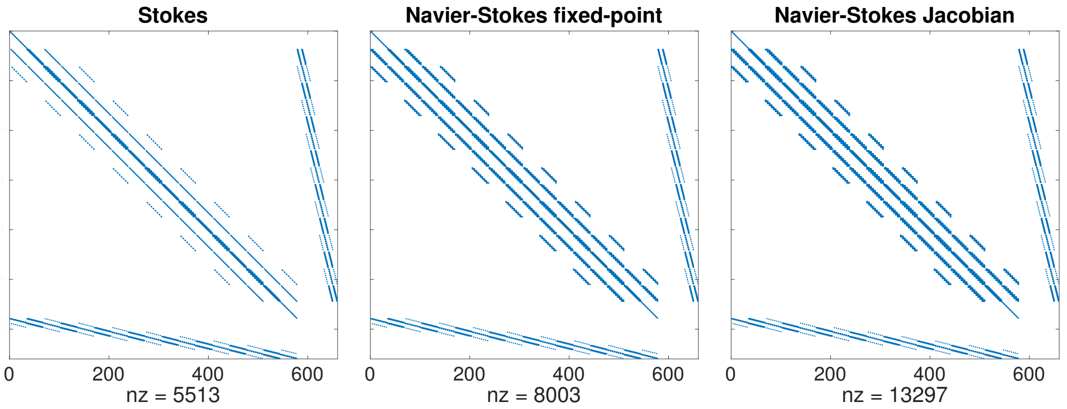


Figure 5.7: Sparsity patterns and numbers of nonzeros (nz) of finite element matrices of a Stokes problem (left), a Navier–Stokes fixed point system (center), and a Jacobian of a Navier–Stokes problem (right). The two-dimensional problems on a unit square are discretized with P2–P1 Taylor–Hood elements. Velocity components are ordered node-wise.

evaluate J in the last iterate u^k and solve the following systems until convergence:

$$\text{Solve } \; J(u^k)\delta u^k = r(u^k) \; \text{ for } \; \delta u^k, \tag{5.29}$$

$$\text{and update } \; u^{k+1} = u^k + \delta u^k. \tag{5.30}$$

Here, $r(u^k) = -f(u^k)$ is the nonlinear residual. Different convergence criteria can be used, e.g., the norm of the update $\|\delta u^k\|$, or the norm of the nonlinear residual $\|r(u^k)\|$. In the numerical simulations in Chapter 8, we will use the relative nonlinear residual norm: $\|r(u^k)\|/\|r(u^0)\|$. As discussed before, we use GMRES for the iterative solution of large sparse systems. For fluid flow problems, finite element discretizations of Jacobian matrices are, in general, much denser than their comparable linear counterparts or the corresponding fixed point systems. However, they are still sparse and can therefore be solved efficiently with (preconditioned) GMRES. In fig. 5.7 we present sparsity patterns of a Stokes problem, a fixed point Navier–Stokes system, and a Jacobian matrix of a Navier–Stokes problem.

We call Newton's method with inexact solutions of the linearized systems an *inexact Newton method*; cf. [41]. The most general form of an inexact Newton method is presented in algorithm 1; cf. [57, 58].

Algorithm 1 Inexact Newton method

```
For given initial guess $u^0$
for $k = 0$ do until convergence
    find some $\delta u^k \in [0, 1)$ and $\eta_k$ such that
        $\left\| f(u^k) + J(u^k)\delta u^k \right\| \leq \eta_k \left\| r(u^k) \right\|$
    update $u^{k+1} = u^k + \delta u^k$
end for
```

For a fixed $\eta_k = 0$, we recover the exact Newton method. If we compute a solution iteratively which satisfies $\left\| f(u^k) + J(u^k)\delta u^k \right\| \leq \eta_k \left\| r(u^k) \right\|$ with GMRES, then η_k must be chosen before the linear system is solved. Then, η_k is simply the tolerance for GMRES which is used to compute an approximate solution of δu in eq. (5.29). An approach with an outer Newton iteration and an inner Krylov subspace method is often referred to as a *Newton–Krylov method*. In the following, we will only allow for changing η_k between successive Newton iterations. The η_k are often called *forcing terms*; cf. [58]. We assume that the initial guess u^0 is close enough to the solution u^* to guarantee the following convergence rate; cf. [41, 58]. The series u^k converges to the solution u^* *q-quadratically* in the norm $\|\cdot\|_*$, if $\lim_{x\to\infty} \eta_k = 0$ and $\eta_k = \mathcal{O}\left(\left\| f(u^k) \right\|\right)$. The norm $\|\cdot\|_*$ is defined as $\|J(u^*)x\|$ for $x \in \mathbb{R}^n$; see [41] and [58] for more convergence rate results.
Still, the local linear model J evaluated in u^0 and the function f might disagree considerably. For instance, the first Jacobian matrix with zero initial guess of a Navier–Stokes problem is only a Stokes problem. If we consider advection dominated problems, this is only a rough approximation of the final converged Navier–Stokes system. It might not be beneficial to choose a small tolerance η_0 as the norm of the residual $\|r(u^1)\|$ might not change much. This phenomenon is called *oversolving* and, apart from a bad approximation, it is costly to iteratively compute solutions with GMRES which satisfy a small tolerance. Therefore, we want to adaptively choose a forcing term to minimize oversolving. We will use

the adaptive forcing term which is denoted *choice 2* in [58]:

$$\eta_k = \gamma \left(\frac{\left\| r(u^k) \right\|_2}{\left\| r(u^{k-1}) \right\|_2} \right)^{\alpha}, \tag{5.31}$$

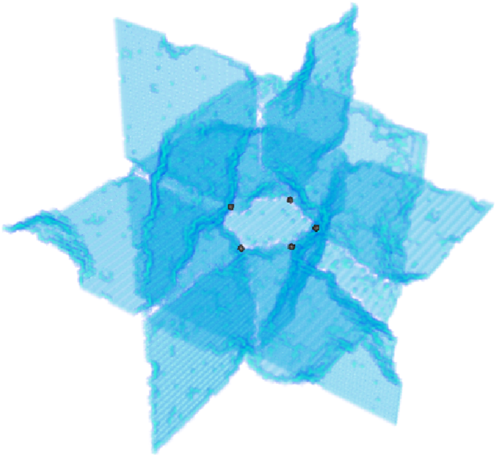
with given $\gamma \in [0,1]$, $\alpha \in (1,2]$, and $\eta_0 \in (0,1)$. The implementation of adaptive forcing terms and globalization techniques of the Trilinos package NOX will be used in our simulations. In NOX, the above computation is denoted as *type 2*. We use this adaptive forcing term since it only requires the computation of two vectors and their norms, which have been already evaluated during the Newton iterations. In contrast, there is another option given in [58] called *choice 1*. There, an additional matrix-vector product has to be computed.

Additionally, we want to prescribe upper and lower bounds for the forcing term: $\eta_k \in [\eta_{\min}, \eta_{\max}]$. Above, we assumed that the initial guess is already close to a solution. In practice, this assumption may not be fulfilled for challenging, highly nonlinear problems. Therefore, we introduce globalization strategies to facilitate the global convergence. The following algorithm is a globalized inexact Newton method with *backtracking*; backtracking is a *line search method*. Another popular class of globalization techniques are so-called *Trust Region methods*. We refer the reader to [57] for a detailed derivation of different globalization techniques and their specific implementations. The Newton–Krylov method with backtracking is summarized in algorithm 2.

Algorithm 2 Inexact Newton method with backtracking

```
For given initial guess $u^0$, $\eta_0 \in [\eta_{\min}, \eta_{max}]$, $t \in (0,1)$ and $0 < \theta_{\min} < \theta_{\max} < 1$
for $k = 0$ do until convergence
    solve $J(u^k)\delta u^k = r(u^k)$ with GMRES and tolerance $\eta_k$
    while $\left\| f(u^k + \delta u^k) \right\|_2 > [1 - t(1 - \eta_k)] \left\| f(u^k) \right\|_2$ do
        choose $\theta \in [\theta_{\min}, \theta_{\max}]$
        update $\delta u^k = \theta \delta u^k$
    end while
    update $u^{k+1} = u^k + \delta u^k$.
    compute $\eta_{k+1}$ with (5.31)
end for
```

6 Domain Decomposition - Two-Level Overlapping Schwarz Methods



In the previous chapter, we discussed finite element discretizations of different PDEs. In order to compute accurate approximations of the solutions, we, in general, need a fine discretization and therefore a high mesh resolution. When dealing with systems with millions of unknowns direct solvers become less efficient. The computational costs for an LU decomposition of a matrix $A \in \mathbb{R}^{n \times n}$ is $\mathcal{O}(n^3)$. Additionally, the high memory requirement a of direct solver prevents the efficient solution of large problems. Therefore, inexact iterative solvers are often employed for the solution of large sparse systems. Since the main focus of this thesis is the solution of incompressible fluid flow and associated FSI problems, which are unsymmetric and indefinite, we will employ the Generalized Minimal Residual method [128]. To facilitate the convergence of GMRES we need to construct efficient preconditioners. Moreover, if we use a preconditioner which changes in every iteration we will use *Flexible GMRES* (FGMRES) [129].

In the following construction of parallel preconditioners, we aim at two things. On the one hand, we want to localize computations, where each local problem is independent of the other local problems. On the other hand, we need to exchange global information, which is lost in the localization process. Therefore, we will construct a (global) coarse problem. For this two-step approach we use domain decomposition based *overlapping Schwarz preconditioners* [134, 139]. Another widely used domain decomposition methods is the *Finite Element Tearing and Interconnecting* method (FETI) [65, 67]. Furthermore, a significant improvement to FETI is the *FETI Dual–Primal* method (FETI–DP) [66, 106, 107, 109, 116]. The *Balancing Domain Decomposition* (BDD) method [139] and the *Balancing Domain Decomposition by Constraints* (BDDC) method [36, 47, 115] are a third class of widely used and efficient approaches. However, they are similar to FETI methods. There are two main differences between the four last mentioned methods and an overlapping Schwarz method. While an overlapping Schwarz method uses overlapping local problems, all other methods use nonoverlapping local problems. Furthermore, different global matrix data that is needed. Schwarz preconditioners can be built from the fully assembled stiffness matrix while FETI, FETI–DP, BDD, and BDDC need the unassembled subdomain Neumann matrices. A one-level Schwarz preconditioner without a coarse problem can be seen as more algebraic than the other four methods. On the contrary, a one-level Schwarz method does, in general, not scale w.r.t. iterations counts. Therefore, a coarse level is needed.

The scalability is one of the most important factors in the construction of preconditioners. In a parallel setting, scalability has two sides. We first consider the numerical scalability, where we are only interested in the iteration counts. Let N be the number of subdomains of our domain decomposition for a mesh with characteristic mesh size h and let H be the characteristic size or diameter of a subdomain. If we increase the number of subdomains, each subdomain gets smaller and therefore H decreases. Next, if we decrease h at the same rate as H decreases we obtain a constant subdomain size in the sense that subdomains possess the same number of nodes; cf. fig. 6.1. The size of a subdomain is typically expressed by the ratio H/h. This consideration is well suited for a structured mesh and decomposition. If we consider a constant subdomain size H/h and an increasing number of subdomains N, an iterative solver with a numerical scalable preconditioner should solve the problem in a (almost) constant number of iterations.

The second aspect of scalability is the parallel scalability. Here, we compare the amount of time which is needed for the construction of a preconditioner and the

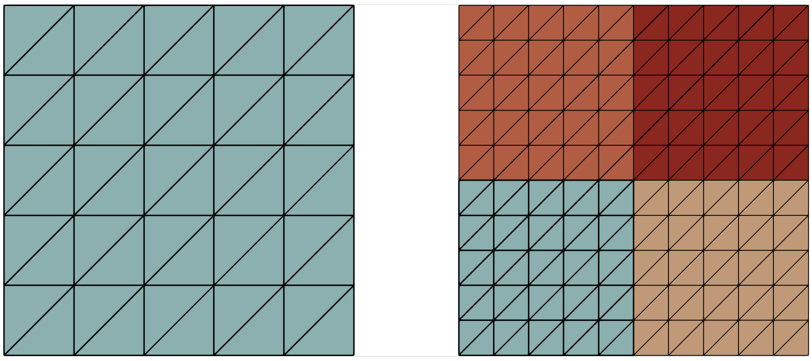


Figure 6.1: Two domains with constant subdomain size $H/h = 5$. Structured decomposition into 1 subdomain (left) and 4 subdomains (right).

application in the solution phase. In particular, we need a parallel environment with many computational units or cores. We can further distinguish between weak and strong scalability. A perfectly weakly scalable preconditioner computes the solution in a constant time for a problem with constant subdomain size H/h and increasing number of subdomains, if the number of cores is increased at the same rate as the number of subdomains.

On the other hand, if we solve the same problem (constant h) on an increasing number of cores and subdomains, e.g., n and $2n$ cores, a perfectly strong scaling preconditioner needs only half of the time.

It is important to note that the scalability in the iterative solution phase greatly depends on the iterative solver. However, GMRES mainly uses matrix-vector products which scale very well for sparse matrices. A minor drawback is the orthogonalization process in GMRES, which gets more expensive with each additional iteration. For a given problem $Ax = b$, initial guess x_0, and unpreconditioned initial residual $r_0 = b - Ax_0$ the m-th *Krylov subspace* is $x_0 + \mathcal{K}_m(A, r_0) = x_0 + \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}$. In each iteration of GMRES an orthonormal basis $\{v_1, \ldots, v_m\}$ of the Krylov subspace is constructed. We refer the reader to [128] for a full description and detailed derivation of the GMRES method. A remedy for the increasing computational costs is the restarted GMRES method; this version is often denoted as *GMRES(r)*. The existing Krylov subspace is simply omitted after r iterations and the last solution

x_r is used as a new initial guess. However, the convergence results which were derived for GMRES cannot be guaranteed for GMRES(r). In particular, this became evident in the solution of some problems of the previous chapter with the preconditioners of this chapter. There, GMRES without restarts was generally more efficient than GMRES(r). Different values for r from 10 to 100 were selected for Stokes and steady Navier–Stokes problems, which were preconditioned with a monolithic two-level overlapping Schwarz preconditioner with a GDSW coarse space. However, the increase in iterations was to large when using GMRES(r) and therefore we only consider GMRES without restarts in the numerical results of Chapter 8.

In the section 6.1, we will describe two-level overlapping Schwarz preconditioners for elliptic problems. In particular, we will focus on the construction of GDSW coarse spaces. Basis functions resulting from harmonic extensions of interface values are constructed, which are then used to compute a coarse problem matrix. This is followed by a discussion of block preconditioners for saddle point problems in section 6.2. We will then describe the construction of monolithic two-level overlapping Schwarz preconditioners for saddle point problems in section 6.3 and focus on preconditioners for incompressible fluid flow problems with GDSW and RGDSW coarse spaces. Fully algebraic GDSW and RGDSW preconditioners based on approximated interfaces are described in section 6.4. In the last section 6.5, we will consider preconditioners for FSI problems.

6.1 Two-Level Overlapping Schwarz Methods with GDSW Coarse Spaces for Elliptic Problems

We begin the construction of overlapping Schwarz preconditioners for an elliptic model problem, which was previously published in [79]. We consider the linear equation system

$$Ax = b$$

arising from a finite element discretization of an elliptic problem, e.g., a Poisson or elasticity problem on $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, with sufficient Dirichlet boundary conditions.

Let Ω be decomposed into nonoverlapping subdomains $\{\Omega_i\}_{i=1}^N$ with typical di-

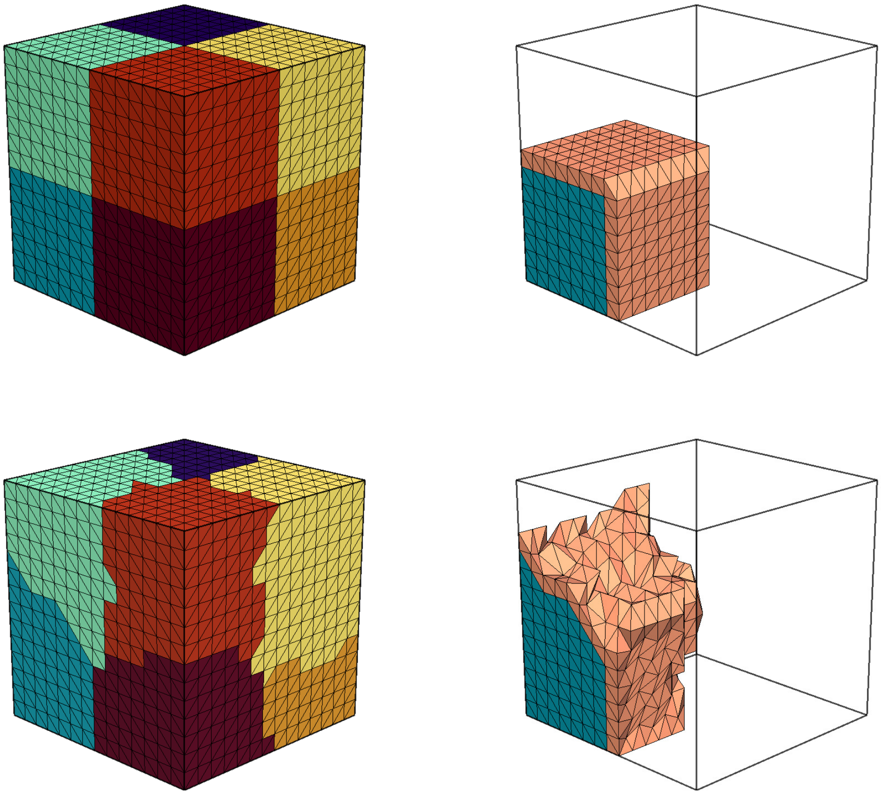


Figure 6.2: Structured mesh and domain decomposition (top) and unstructured mesh and domain decomposition (bottom) of a unit cube: different colors for each subdomain (left) and one subdomain with one highlighted layer of overlap (right). The unstructured mesh is generated as presented in fig. 8.1 and the unstructured domain decomposition is performed using `ParMETIS` [100]. Taken from [79].

ameter H and corresponding overlapping subdomains $\{\Omega_i'\}_{i=1}^N$ with k layers of overlap, i.e., $\delta = kh$; cf. fig. 6.2 for a visualization of a cubic sample domain. The overlapping subdomains can be constructed from the nonoverlapping subdomains by recursively adding one layer of elements after another to the subdomains. Even if no geometric information is given, this can be performed based on the graph of the stiffness matrix A. Furthermore, let

$$\Gamma = \left\{ x \in (\overline{\Omega}_i \cap \overline{\Omega}_j) \setminus \partial\Omega_D : i \neq j, 1 \leq i, j \leq N \right\}$$

be the discrete interface of the nonoverlapping domain decomposition. Note that Γ was previously used as the fluid-solid interface of the FSI problem. However, using Γ as the interface of a domain decomposition is the standard notation from which we do not want to deviate. It should be clear from the context which type of interface is considered.

We define $R_i : V^h \rightarrow V_i^h$, $i = 1, ..., N$, as the restriction from the global finite element space $V^h = V^h(\Omega)$ to the local finite element space $V_i^h := V^h(\Omega_i')$ on the overlapping subdomain Ω_i'; R_i^T is the corresponding prolongation from V_i^h to V^h. In addition to that, let V_0 be a global coarse space and $R_0^T : V_0^h \rightarrow V^h$ the corresponding coarse interpolation.

For the time being, we will use exact local solvers; in section 8.1.1.3, we will also present results for inexact coarse solvers. For exact local and coarse solvers, the additive Schwarz preconditioner in matrix-form can be written as

$$B_{\mathrm{OS-2}}^{-1} = \underbrace{R_0^T A_0^{-1} R_0}_{\text{coarse level}} + \underbrace{\sum_{i=1}^{N} R_i^T A_i^{-1} R_i}_{\text{first level}}, \tag{6.1}$$

where the local and coarse stiffness matrices A_i are given by

$$A_i = R_i A R_i^T, \text{ for } i = 0, ..., N.$$

One typical choice for the coarse basis functions are Lagrangian basis functions on a coarse triangulation. However, a coarse triangulation may not be available for arbitrary geometries and domain decompositions. Therefore, we consider GDSW coarse spaces, which do not require a coarse triangulation.

GDSW Coarse Spaces: The GDSW preconditioner, which was introduced by Dohrmann, Klawonn, and Widlund in [45, 49], is a two-level additive overlapping Schwarz preconditioner with energy minimizing coarse space. Thus, the preconditioner can be written in the form

$$B_{\mathrm{GDSW}}^{-1} = \Phi A_0^{-1} \Phi^T + \sum_{i=1}^{N} R_i^T A_i^{-1} R_i,$$

where A_i and R_i are defined as before and

$$A_0 = R_0 A R_0^T = \Phi^T A \Phi$$

is the matrix of the coarse problem. This corresponds to eq. (6.1), where we replace R_0^T by Φ, as is standard in the context of GDSW preconditioners.

For the GDSW preconditioner, the choice of the matrix Φ is the main ingredient. In order to define the columns of Φ, i.e., the coarse basis functions, a partition of unity of energy-minimizing functions and the nullspace of the operator A are employed.

In particular, the interface Γ is divided into M connected components Γ_j which are common to the same set of subdomains, i.e., into vertices, edges, and, in three dimensions, faces. Now, let Z be the null space of the global Neumann matrix and Z_{Γ_j} the restriction of Z to the degrees of freedom corresponding to the interface component Γ_j. Then, we construct corresponding matrices Φ_{Γ_j}, such that their columns form a basis of the space Z_{Γ_j}. Let $R_{\Gamma j}$ be the restriction from Γ onto Γ_j. Then, the values of the GDSW basis functions Φ on Γ can be written as

$$\Phi_\Gamma = \left[\begin{array}{ccc} R_{\Gamma_1}^T \Phi_{\Gamma_1} & \dots & R_{\Gamma_M}^T \Phi_{\Gamma_M} \end{array} \right]. \tag{6.2}$$

After partitioning the d.o.f. into interface (Γ) and interior (I) ones, the matrix A can be written as

$$A = \begin{bmatrix} A_{II} & A_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} \end{bmatrix}.$$

Then, the basis functions of the GDSW coarse space can be written as discrete harmonic extensions of Φ_Γ to the interior degrees of freedom:

$$\Phi = \left[\begin{array}{c} \Phi_I \\ \Phi_\Gamma \end{array} \right] = \left[\begin{array}{c} -A_{II}^{-1} A_{I\Gamma} \Phi_\Gamma \\ \Phi_\Gamma \end{array} \right]. \tag{6.3}$$

Note that $A_{II} = \operatorname{diag}_{i=1}^N (A_{II}^{(i)})$ is a block diagonal matrix containing the local matrices $A_{II}^{(i)}$ from the nonoverlapping subdomains. Its factorization can thus be computed block by block and in parallel.

The condition number estimate for the GDSW preconditioner

$$\kappa\left(B_{\mathrm{GDSW}}^{-1}A\right) \leq C\left(1+\frac{H}{\delta}\right)\left(1+\log\left(\frac{H}{h}\right)\right)^2,$$

holds for polygonal/polyhedral Lipschitz domains and certain assumptions on the subdomains. In two dimensions, it also holds for the general case of Ω decomposed into John domains; cf. [45, 49]. For certain variants of the GDSW coarse space, the factor $\left(1+\log\left(\frac{H}{h}\right)\right)^2$ can be improved to $\left(1+\log\left(\frac{H}{h}\right)\right)$; see, [50, 52].

6.2 Block Preconditioners for Saddle Point Problems

In this section we want to present several block preconditioners for saddle point problems. For convenience, we restate our two saddle point systems. The general block matrix with has the form

$$\mathcal{F}x = \begin{bmatrix} F & B^T \\ B & -C \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix},$$

and for a problem where the pressure is fixed to zero mean value using a Lagrangian multiplier, it reads

$$\overline{\mathcal{F}}\overline{x} = \begin{bmatrix} F & B^T & 0 \\ B & -C & a^T \\ 0 & a & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ 0 \\ 0 \end{bmatrix}.$$

Both systems were derived in section 5.1.6.

6.2.1 Overlapping Schwarz Methods for Block Preconditioners

The use of overlapping Schwarz techniques for the approximation of block inverses, resulting from an LDU block factorization, was proposed in [105] for indefinite saddle point problems. For the (standard) saddle point system, we obtain the factorization

$$\begin{bmatrix} F & B^T \\ B & -C \end{bmatrix} = \begin{bmatrix} I & 0 \\ BF^{-1} & I \end{bmatrix} \begin{bmatrix} F & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & F^{-1}B^T \\ 0 & I \end{bmatrix},$$

with the Schur complement $S = -C - BF^{-1}B^T$. The resulting block-diagonal preconditioner reads

$$\hat{\mathcal{B}}_D^{-1} = \begin{bmatrix} \hat{F} & 0 \\ 0 & S \end{bmatrix}^{-1} = \begin{bmatrix} \hat{F}^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix}.$$

The upper block-triangular preconditioner is given by

$$\begin{bmatrix} F & B^T \\ 0 & S \end{bmatrix}^{-1} = \begin{bmatrix} F^{-1} & -S^{-1}B^TF^{-1} \\ 0 & S^{-1} \end{bmatrix}.$$

Since the computation of the Schur complement S is very expensive, we seek to replace it with a suitable approximation. For a Navier–Stokes problem with kinematic viscosity ν and a low Reynolds number, the Schur complement can be replaced with a spectrally equivalent scaled pressure mass matrix $1/\nu M_p$; cf. [146]. Similarly, we can directly replace S with M_p for our Stokes problems. In numerical studies for a mixed linear elasticity problem in [103], it was observed that overlapping Schwarz methods with a minimal overlap for the pressure block, using the pressure mass matrix, are a good choice and should be preferred to a scaled identity operator. A condition number estimate as well as a proof on the convergence for GMRES are given in [105] for the block-diagonal and the block-triangular preconditioner, respectively.

For the system with Lagrangian multiplier, we obtain the LDU block factorization

$$\begin{aligned} \begin{bmatrix} F & B^T & 0 \\ B & -C & a^T \\ 0 & a & 0 \end{bmatrix} &= \begin{bmatrix} F & 0 & 0 \\ B & S & 0 \\ 0 & a & s_a \end{bmatrix} \begin{bmatrix} I & F^{-1}B^T & 0 \\ 0 & I & S^{-1}a^T \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} I & 0 & 0 \\ BF^{-1} & I & 0 \\ 0 & aS^{-1} & 1 \end{bmatrix} \begin{bmatrix} F & 0 & 0 \\ 0 & S & 0 \\ 0 & 0 & s_a \end{bmatrix} \begin{bmatrix} I & F^{-1}B^T & 0 \\ 0 & I & S^{-1}a^T \\ 0 & 0 & 1 \end{bmatrix}, \end{aligned}$$

with $s_a = aS^{-1}a^T$. Again, the spectral equivalence of the Schur complement and the pressure mass matrix yields the approximation $S \approx M_p$. Furthermore, we replace s_a by $m_a = aM_p^{-1}a^T$. The solution of $M_p x = a^T$ is the vector $x = \underline{1}$ with mean value 1. Consequently, $a \cdot x = 1$, and therefore, $m_a = 1$.

Then, the inverse of our lower block-triangular preconditioner with Lagrangian multiplier is

$$\begin{bmatrix} F & 0 & 0 \\ B & M_p & 0 \\ 0 & a & 1 \end{bmatrix}^{-1} = \begin{bmatrix} F^{-1} & 0 & 0 \\ -M_p^{-1}BF^{-1} & M_p^{-1} & 0 \\ aM_p^{-1}BF^{-1} & -aM_p^{-1} & 1 \end{bmatrix}.$$

This preconditioner can be analogously written as an upper block-triangular preconditioner.

The above approximation of the Schur complement with a pressure mass matrix is suitable for low Reynolds numbers. However, for higher Reynolds numbers the pressure mass matrix does not account for the dominating advective forces in F. Therefore, for larger advective forces a different approximation of the Schur complement must be used. We will present the SIMPLE method and LSC block-triangular preconditioner in section 6.2.2 and section 6.2.3, respectively.
However, we will first describe the approximation of block inverses for block-diagonal and general block-triangular preconditioners with overlapping Schwarz methods. In particular, we will denote the generic Schur complement approximation with S_{gen} and do not distinguish between SIMPLE, LSC and the above pressure mass matrix approach.

We can immediately use the two-level overlapping Schwarz preconditioners for elliptic problems, which were presented in section 6.1, for the approximation of the blocks F^{-1} and S_{gen}^{-1}. Therefore, we decompose the spaces V^h and Q^h into local spaces

$$\begin{aligned} V_i^h &= V^h \cap (H_0^1(\Omega_i'))^d \text{ and} \\ Q_i^h &= Q^h \cap H_0^1(\Omega_i'), \end{aligned}$$

$i = 1, ..., N$, respectively, defined on the overlapping subdomains Ω_i'. We define restriction operators

$$\begin{aligned} R_{u,i} &: V^h \longrightarrow V_i^h \text{ and} \\ R_{p,i} &: Q^h \longrightarrow Q_i^h, \end{aligned}$$

to the overlapping subdomains Ω_i', $i = 1, ..., N$, for velocity and pressure degrees of freedom, respectively. Consequently, $R_{i,u}^T$ and $R_{i,p}^T$ are extension operators from the local spaces to the global spaces. With these operators, we can restrict our global problems F and S_{gen} to local overlapping problems

$$F_i = R_{u,i} F R_{u,i}^T, \quad S_{\text{gen},i} = R_{p,i} S_{\text{gen}} R_{p,i}^T, \quad i = 1, ..., N.$$

In these problems, homogeneous Dirichlet boundary conditions are implicitly imposed on the boundary of the overlapping subdomains $\partial\Omega_i'$ by construction. We define the two-level overlapping Schwarz preconditioners

$$\begin{aligned} \hat{F}^{-1} &= \Phi_u F_0^{-1} \Phi_u^T + \sum_{i=1}^{N} R_{u,i}^T F_i^{-1} R_{u,i} \quad \text{and} \\ \hat{S}_{\text{gen}}^{-1} &= \Phi_p S_{\text{gen},0}^{-1} \Phi_p^T + \sum_{i=1}^{N} R_{p,i}^T S_{\text{gen},i}^{-1} R_{p,i}, \end{aligned}$$

which are approximate inverses of the matrices F and S_{gen}, respectively. Here, the coarse problems $F_0 = \Phi_u^T F \Phi_u$ and $S_{\text{gen},0} = \Phi_p^T S_{\text{gen}} \Phi_p$ are built with GDSW coarse velocity basis functions Φ_u and pressure basis functions Φ_p, respectively. For the construction of Φ_u, we restrict the nullspace of the subdomain Neumann matrices of F, which consists of all translations, to the interface. Therefore, we set each interface node to

$$r_1 := \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad r_2 := \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

for the two-dimensional case. In three dimensions, each interface node is set to

$$r_1 := \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad r_2 := \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad r_3 := \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Similarly, in two and three dimensions the interface nodes Φ_p are set to the constant function

$$r := \begin{bmatrix} 1 \end{bmatrix}.$$

6.2.2 SIMPLE

The following presentation of SIMPLE and LSC follows the description in [39]. In particular, we will later use implementations of these block preconditioners which are provided by the Trilinos package Teko. The package Teko was developed used in [39]. The SIMPLE block preconditioner was originally derived in [121], and it can be written as

$$\mathcal{B}_S = \begin{bmatrix} F & 0 \\ B & S_{\text{SIMPLE}} \end{bmatrix} \begin{bmatrix} I & \frac{1}{\alpha} H B^T \\ 0 & \frac{1}{\alpha} I \end{bmatrix}.$$

The Schur complement is approximated as $S_{\text{SIMPLE}} = -C - BHB^T$. We use the under-relaxation parameter $\alpha = 0.9$; cf. [39]. The diagonal operator $H \approx F^{-1}$ is defined as

- $H = \text{diag}(F)^{-1}$ for SIMPLE, or
- $H_{ij} = \delta_{ij}(\sum_{k=1}^{N} |F_{ik}|)^{-1}$ for SIMPLEC, where δ_{ij} is the Kronecker delta function.

6.2.3 LSC - Least-Squares Commutator

The LSC block-triangular preconditioner can be written as

$$\mathcal{B}_{LSC} = \begin{bmatrix} F & B^T \\ 0 & S_{LSC} \end{bmatrix}$$

Let M_u be the velocity mass matrix. Then, the approximate inverse of the Schur complement for the LSC block preconditioner reads

$$S_{\text{LSC}}^{-1} = -(BM_u^{-1}B^T + \gamma C)^{-1}(BM_u^{-1}FM_u^{-1}B^T)(BM_u^{-1}B^T + \gamma C)^{-1} + \alpha D^{-1}.$$

We refer to [60] and [63] for a detailed derivation of the LSC preconditioner. In particular, D, α, and γ are used for the stabilized P1–P1 discretization since the operator $BM_u^{-1}B^T$ is singular for this mixed finite element pair.

$$\begin{aligned}\gamma &= \frac{\nu\left(M_u^{-1}F\right)}{3},\\ D &= \operatorname{diag}\left(B\operatorname{diag}\left(F\right)^{-1}B^T + C\right),\\ \alpha &= \frac{1}{\rho\left(B\operatorname{diag}\left(F\right)^{-1}B^TD^{-1}\right)}.\end{aligned}$$

In particular, the inverse of the velocity mass matrix M_u^{-1} is approximated by the inverse of its diagonal $\text{diag}(M_u)^{-1}$, which yields no additional computational costs. However, we can even use a fully algebraic version for the Schur complement S_{LSC}, i.e, we further replace the scaling $\text{diag}(M_u)^{-1}$ with the scaling $\text{diag}(F)^{-1}$. This version is used for our numerical experiments.

6.3 Monolithic Two-Level Overlapping Schwarz Preconditioners for Saddle Point Problems

Monolithic two-level Schwarz preconditioners for saddle point problems are characterized by the fact that the local problems and the coarse problems possess the same block structure as the global saddle point problem. In contrast, the block preconditioners of the previous sections typically omit and/or replace some of the blocks. Consequently, the convergence of monolithic preconditioners is typically significantly faster compared to block preconditioners; cf., e.g., [78, 105]. The following construction of different monolithic preconditioners for fluid flow problems was previously published in [79, 80].

The finite element discretizations of the Stokes and Navier–Stokes equations lead to a symmetric indefinite and nonsymmetric indefinite system, respectively. Therefore, the resulting monolithic two-level overlapping Schwarz preconditioners posses the same properties. In particular, the preconditioned system is not symmetric positive-definite and the standard Schwarz theory which was developed for symmetric positive-definite systems cannot be applied here; cf. [139]. Moreover, convergence estimates for GMRES based on eigenvalue estimates are not valid. Here, a field-of-value analysis could be used instead; cf. [108, 136] and the references therein. However, a field-of-value analysis has not been carried out for monolithic two-level overlapping Schwarz preconditioners for Stokes and Navier–Stokes systems, to the best of our knowledge. This is still a challenging open problem.

In the previous works [50, 51] by Dohrmann and Widlund on GDSW preconditioners for saddle point problems, only discontinuous pressure spaces were considered. Consequently, the saddle point problems could be reduced to elliptic problems by static condensation of the pressure. In contrast, the method presented in this section is inspired by the monolithic Schwarz preconditioners with Lagrangian coarse basis functions introduced by Klawonn and Pavarino in [104, 105], which operate on the full saddle point problem. The coarse basis functions of our new monolithic GDSW preconditioner are discrete saddle point harmonic extensions of interface functions. Preliminary results without algorithmic and implementation details for a related monolithic GDSW preconditioner for continuous pressure spaces, based on ideas of Klawonn and Pavarino in [104, 105], were presented by Dohrmann [46].

Moreover, we will consider restricted and scaled Schwarz operators, introduced by Cai and Sarkis [32], in the first level of our monolithic preconditioners. These first level variants can be equivalently used for the previously presented non-monolithic preconditioners.

In order to improve the parallel performance of monolithic GDSW preconditioners, we will reduce the dimension of the coarse spaces following the work by Dohrmann and Widlund [54] on RGDSW coarse spaces; the smaller dimension typically results in a significantly better parallel performance; cf. [89]. Other earlier approaches to reduce the dimension of GDSW coarse spaces can be found in, e.g., [51, 53].

Furthermore, we employ two alternative strategies to improve the additive and sequential coupling of the two levels which was used in [79]: multiplicative but sequential coupling of the levels and additive coupling combined with the concurrent computation of the levels. Finally, for nonlinear or time-dependent problems, we will make use of different recycling strategies ranging from the reuse of symbolic factorizations of the local overlapping and nonoverlapping matrices to the complete reuse of the coarse basis and matrix. As we will show, recycling of the coarse basis functions can eliminate the drawback of the expensive setup phase of GDSW coarse spaces.

6.3.1 Monolithic Schwarz Preconditioners with Lagrangian Coarse Spaces

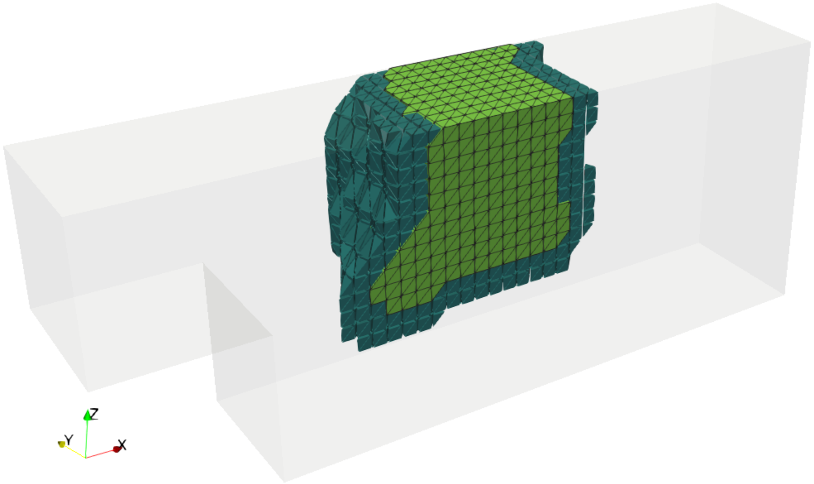


Figure 6.3: A nonoverlapping subdomain (light green) of the three-dimensional backward facing step for an unstructured decomposition with overlap $\delta = 2h$ (dark green).

We first introduce a monolithic two-level overlapping Schwarz preconditioner, where the coarse basis consists of Taylor–Hood or P1–P1 finite elements. This approach was introduced by Klawonn and Pavarino in [104, 105] for monolithic preconditioners. Again, we decompose the spaces V^h and Q^h into local spaces

$$V_i^h = V^h \cap (H_0^1(\Omega_i'))^d \text{ and}$$
$$Q_i^h = Q^h \cap L^2(\Omega_i'),$$

$i = 1, ..., N$, respectively, defined on the overlapping subdomains Ω_i'; an example of an overlapping and nonoverlapping subdomain of three-dimensional BFS geometry is depicted in fig. 6.3.

The restriction operators are defined analogously to section 6.2.1, where approximations to the inverses of separated velocity and pressure blocks were constructed:

$$R_{u,i} : V^h \longrightarrow V_i^h \text{ and}$$
$$R_{p,i} : Q^h \longrightarrow Q_i^h,$$

are restriction operators to overlapping subdomains Ω_i', $i = 1, ..., N$, for velocity and pressure degrees of freedom, respectively. Consequently, $R_{i,u}^T$ and $R_{i,p}^T$ are prolongation operators from local spaces to the global spaces. The monolithic restriction operators

$$\mathcal{R}_i : V^h \times Q^h \longrightarrow V_i^h \times Q_i^h,$$

$i = 1, ..., N$, have the form

$$\mathcal{R}_i := \begin{bmatrix} R_{u,i} & 0 \\ 0 & R_{p,i} \end{bmatrix}.$$

In addition to that, we introduce local projections

$$\overline{\mathcal{P}}_i : V_i^h \times Q_i^h \longrightarrow V_i^h \times \overline{Q}_i^h,$$

where $\overline{Q}_i^h$ is the local pressure space with zero mean value:

$$\overline{Q}_i^h = \{q_h \in Q^h \cap L_0^2(\Omega_i') : \text{supp}(q_h) \subset \Omega_i'\} \subset Q_i^h.$$

Similar to the global projection defined in eq. (5.11), we define local projections

$$\overline{P}_i = I_{p,i} - a_i^T (a_i a_i^T)^{-1} a_i, \; i = 1, ..., N, \tag{6.4}$$

where $I_{p,i}$ is the local pressure identity operator and a_i corresponds to the discretization of eq. (5.9) on the subdomain Ω_i'.

Furthermore, the local velocity identity operators are $I_{u,i}$. Then, we can enforce the zero mean value of the local pressure using the projection

$$\overline{\mathcal{P}}_i := \begin{bmatrix} I_{u,i} & 0 \\ 0 & \overline{P}_i \end{bmatrix}.$$

The monolithic two-level overlapping Schwarz preconditioner with Lagrangian coarse space reads

$$\hat{\mathcal{B}}_{\mathrm{P2-P1}}^{-1} = \mathcal{R}_0^T \mathcal{F}_0^{+} \mathcal{R}_0 + \sum_{i=1}^{N} \mathcal{R}_i^T \overline{\mathcal{P}}_i \mathcal{F}_i^{-1} \mathcal{R}_i, \tag{6.5}$$

where $\mathcal{F}_0^+$ is a pseudo-inverse of the coarse matrix $\mathcal{F}_0 = \mathcal{R}_0 \mathcal{F} \mathcal{R}_0^T$. The construction of the coarse matrix $\mathcal{F}_0$ with corresponding restriction operator $\mathcal{R}_0$ and interpolation operator $\mathcal{R}_0^T$ is described at the end of this section. Since the pressure may not be uniquely determined in the coarse problem, in general, we have to use a pseudo-inverse. In practice, we restrict the coarse pressure to $\overline{Q}^H$ for the solution of the coarse problem. Otherwise, if $\mathcal{F}$ is nonsingular, $\mathcal{F}_0^+ = \mathcal{F}_0^{-1}$. Also note that, due to the projections $\overline{\mathcal{P}}_i$, the preconditioner $\hat{\mathcal{B}}_{\mathrm{P2-P1}}$ is not symmetric for a Stokes problem.

The local matrices

$$\mathcal{F}_i = \mathcal{R}_i \mathcal{F} \mathcal{R}_i^T, i = 1, ..., N,$$

are extracted from the global matrix $\mathcal{F}$ and have homogeneous Dirichlet boundary conditions for both, velocity and pressure.

As we will describe in section 6.3.4, there are several ways to treat the pressure in the local overlapping problems. Here, we ensure zero mean value for the pressure of the local contributions using projections $\overline{\mathcal{P}}_i$ in addition to the Dirichlet boundary data in the pressure.

Note that the local overlapping problems and the coarse problem are saddle point problems with a structure as in eq. (5.5). Similar to the local problems, the coarse problem with Lagrangian basis functions is defined on a corresponding product space $V_0^h \times Q_0^h$, resulting in the coarse interpolation matrix

$$\mathcal{R}_0^T = \begin{bmatrix} R_{0,u}^T & 0 \\ 0 & R_{0,p}^T \end{bmatrix}, \tag{6.6}$$

which contains the interpolations of the corresponding coarse basis functions. Therefore, a coarse triangulation is needed for the construction of the coarse level. This is problematic for arbitrary geometries and, in particular, cannot be performed in an algebraic fashion. Since the coarse pressure has zero mean value due to our specific choice of the pseudo-inverse $\mathcal{F}_0$, we can omit a projection of the coarse solution onto $\overline{Q}^H$.

6.3.2 Restricted and Scaled First-Level Operators

Using restricted or scaled first-level extension operators may improve the convergence of the iterative solver. This results in the *Restricted Additive Schwarz* (RAS) or *Scaled Additive Schwarz* (SAS) method; cf., e.g., [32, 51]. For both approaches, alternative extension operators $\tilde{\mathcal{R}}_i^T$ are introduced, which satisfy

$$\sum_{i=1}^{N} \tilde{\mathcal{R}}_i^T \mathcal{R}_i \underline{1} = \underline{1},$$

where $\underline{1} \in \mathbb{R}^n$ is the vector of ones. The resulting one-level preconditioner reads

$$\hat{\mathcal{B}}_{RAS/SAS}^{-1} = \sum_{i=1}^{N} \tilde{\mathcal{R}}_i^T \mathcal{A}_i^{-1} \mathcal{R}_i.$$

We obtain the RAS method from a unique distribution of d.o.f. among the nonoverlapping subdomains; cf. [56]. Therefore, we apply $\tilde{\mathcal{R}}_i^T$ without communication in a parallel implementation of the RAS method.

In contrast, we construct the extension operators $\tilde{\mathcal{R}}_i^T$ for SAS from the standard operators $\mathcal{R}_i^T$ with an inverse multiplicity scaling, i.e.,

$$\tilde{\mathcal{R}}_i^T = \operatorname{diag}\left(\sum_{i=1}^{N} \mathcal{R}_i^T \mathcal{R}_i \underline{1} \right)^{-1} \mathcal{R}_i^T.$$

The convergence of the Schwarz method can be improved by applying the scaled $\tilde{\mathcal{R}}_i^T$, and requires the same communication as the application of $\mathcal{R}_i^T$.

For the description of SAS and RAS, the projection $\overline{\mathcal{P}}_i$, $i = 1, \ldots, n$, was ignored but can be used analogously to (6.5). Both of these first level combination techniques can be applied analogously to the preconditioners for elliptic problems and block approximations of the previous sections 6.1 and 6.2, respectively.

In this thesis, we always restrict a linear or linearized problem. An approach to facilitate the nonlinear convergence is the restriction to nonlinear local problems; cf. [30]. Nonlinear domain decomposition methods have been successfully applied to Navier–Stokes and FSI problems in [92, 94, 113].

6.3.3 Monolithic Schwarz Preconditioners with GDSW Coarse Spaces

The monolithic GDSW preconditioner for saddle point problems, which was presented in [79], is a two-level Schwarz preconditioner with discrete saddle point harmonic coarse space. Therefore, it can be seen as an extension of GDSW for elliptic problems, as described in section 6.1, to saddle point problems. Its first level is defined as in the previous two sections, and therefore, the preconditioner can be written as

$$\hat{\mathcal{B}}_{\mathrm{GDSW}}^{-1} = \phi \mathcal{F}_0^{+} \phi^T + \sum_{i=1}^{N} \mathcal{R}_i^T \overline{\mathcal{P}}_i \mathcal{F}_i^{-1} \mathcal{R}_i. \tag{6.7}$$

The coarse operator reads

$$\mathcal{F}_0 = \phi^T \mathcal{F} \phi, \tag{6.8}$$

with the coarse basis functions being the columns of the matrix ϕ.

The coarse space is constructed in a similar way as in section 6.1. As for elliptic problems, the problem is first partitioned into interface (Γ) and interior (I) d.o.f.. Correspondingly, the matrix $\mathcal{F}$ can be written as

$$\mathcal{F} = \begin{bmatrix} \mathcal{F}_{II} & \mathcal{F}_{I\Gamma} \\ \mathcal{F}_{\Gamma I} & \mathcal{F}_{\Gamma\Gamma} \end{bmatrix}.$$

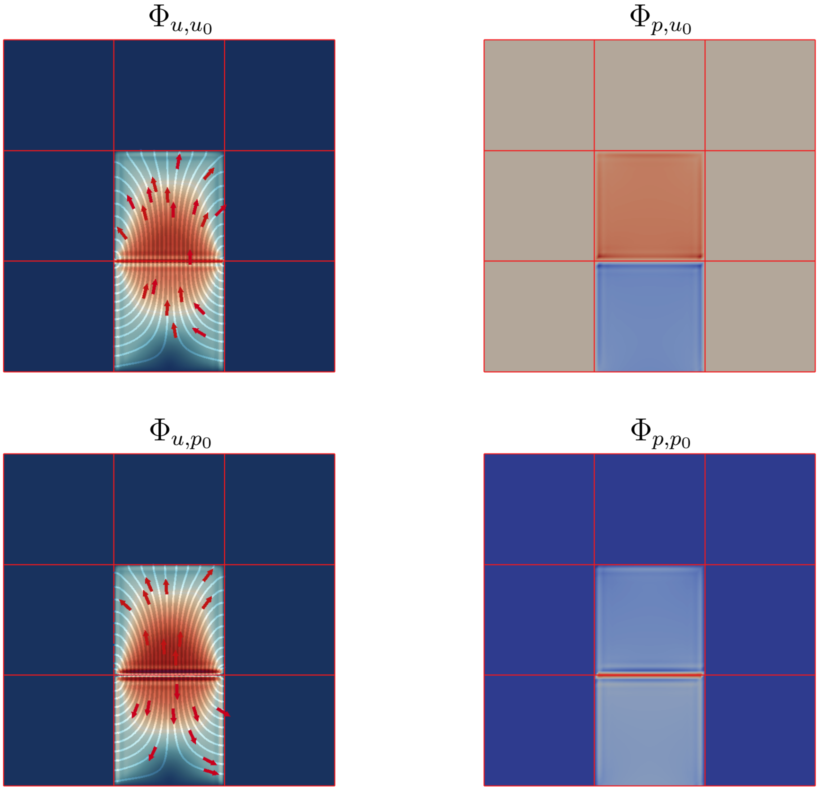


Figure 6.4: Saddle point harmonic extension for the Stokes equations in two dimensions with 9 subdomains. Velocity Φ_{u,u_0} (top left) and pressure Φ_{p,u_0} (top right) components of a velocity edge basis function in y-direction. Velocity Φ_{u,p_0} (bottom left) and pressure Φ_{p,p_0} (bottom right) components of the pressure basis function corresponding to the same edge; see eq. (6.12) for the block structure of ϕ. Taken from [79].

Each of the submatrices $\mathcal{F}_{**}$ is a block matrix of the (5.5). The coarse basis functions are then constructed as discrete saddle point harmonic extensions of the interface values ϕ_Γ, i.e., as the solutions of the linear equation system

$$\begin{bmatrix} \mathcal{F}_{II} & \mathcal{F}_{I\Gamma} \\ 0 & I \end{bmatrix} \begin{bmatrix} \phi_I \\ \phi_\Gamma \end{bmatrix} = \begin{bmatrix} 0 \\ \phi_\Gamma \end{bmatrix}. \tag{6.9}$$

Again, $\mathcal{F}_{II} = \operatorname{diag}_{i=1}^{N}(\mathcal{F}_{II}^{(i)})$ is a block-diagonal matrix containing the local matrices $\mathcal{F}_{II}^{(i)}$ from the nonoverlapping subdomains and can thus be solved block by block and in parallel:

$$\phi = \begin{bmatrix} \phi_I \\ \phi_\Gamma \end{bmatrix} = \begin{bmatrix} -\mathcal{F}_{II}^{-1}\mathcal{F}_{I\Gamma}\phi_\Gamma \\ \phi_\Gamma \end{bmatrix}. \tag{6.10}$$

The interface values of the coarse basis

$$\phi_\Gamma = \begin{bmatrix} \Phi_{\Gamma,u_0} & 0 \\ 0 & \Phi_{\Gamma,p_0} \end{bmatrix} \tag{6.11}$$

are decomposed into velocity- (u_0) and pressure-based (p_0) basis functions.

In particular, the columns of Φ_{Γ,u_0} and Φ_{Γ,p_0} are the restrictions of the nullspaces of the operators F and B^T to the interface components Γ_j, $j = 1, ..., M$; cf. section 6.1. Typically, the nullspace of the operator B^T consists of all pressure functions that are constant on Ω; cf. section 5.1.6. Therefore, the columns of Φ_{Γ,p_0} are chosen to be the restrictions of the constant function 1 to the faces, edges, and vertices.

Note that, in contrast to the Lagrangian coarse basis functions of eq. (6.6), where the basis functions do not couple the velocity and pressure degrees of freedom, the off-diagonal blocks in the block representation

$$\phi = \begin{bmatrix} \Phi_{u,u_0} & \Phi_{u,p_0} \\ \Phi_{p,u_0} & \Phi_{p,p_0} \end{bmatrix} \tag{6.12}$$

are, in general, not zero for the discrete saddle point harmonic coarse spaces; cf. fig. 6.4. This is beneficial for the scalability of the method if pressure normalization is enforced by condition eq. (5.9): if the blocks Φ_{u,p_0} and Φ_{p,u_0} are omitted, numerical scalability deteriorates.

If, on the other hand, the pressure is fixed through a boundary condition, the off-diagonal blocks have to be omitted instead to obtain good numerical scalability. Then, ϕ reads

$$\phi = \begin{bmatrix} \Phi_{u,u_0} & 0 \\ 0 & \Phi_{p,p_0} \end{bmatrix}. \tag{6.13}$$

This also reduces the computational cost of the Galerkin product in eq. (6.8).

Construction of ϕ_Γ for our model problems: For Stokes and Navier–Stokes problems in two dimensions, each interface node is set to

$$r_{u,1} := \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad r_{u,2} := \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad r_{p,1} := \begin{bmatrix} 1 \end{bmatrix}. \tag{6.14}$$

In three dimensions, each interface node is set to

$$r_{u,1} := \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad r_{u,2} := \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad r_{u,3} := \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad r_{p,1} := \begin{bmatrix} 1 \end{bmatrix}. \tag{6.15}$$

Additionally, we add one basis function for the Lagrange multiplier, as will be explained in section 6.3.4 for the LDC Stokes and LDC Navier–Stokes problems; cf. eq. (6.18).

For the two-dimensional mixed linear elasticity problems the coarse basis functions are defined by eq. (6.14) and further by the linearized rotation

$$r_{u,3} := \begin{bmatrix} -(x_2 - \hat{x}_2) \\ x_1 - \hat{x}_1 \end{bmatrix}.$$

In the three-dimensional case, the coarse basis functions are defined by eq. (6.15) and by the linearized rotations

$$r_{u,4} := \begin{bmatrix} x_2 - \hat{x}_2 \\ -(x_1 - \hat{x}_1) \\ 0 \end{bmatrix}, \quad r_{u,5} := \begin{bmatrix} -(x_3 - \hat{x}_3) \\ 0 \\ x_1 - \hat{x}_1 \end{bmatrix}, \quad r_{u,6} := \begin{bmatrix} 0 \\ x_3 - \hat{x}_3 \\ -(x_2 - \hat{x}_2) \end{bmatrix},$$

with the origin of the rotation at $\hat{\mathbf{x}} \in \Omega$. For straight edges in three dimensions, only two linearized rotations are linearly independent. Therefore, we only use two rotations for each of these edges. Furthermore, all rotations are omitted for vertices, and we add a basis function for the Lagrange multiplier.

6.3.4 Treating the Pressure of Fluid Flow Problems

The following considerations for pressure solutions with zero mean value were published in [79]. As already mentioned in section 5.1.6, there are several ways

N	4	9	16	25	36
$u_D p_D$, $\delta = 1h$	23	39	62	83	101
$u_D p_N$, $\delta = 1h$	18	28	44	61	79
$u_D p_D$, $\delta = 2h$	16	26	37	52	64
$u_D p_N$, $\delta = 2h$	12	24	35	44	57

Table 6.1: LDC Stokes problem in two dimensions using $H/h = 8$ and Taylor–Hood elements; GMRES iteration counts for both versions of local problems for a one-level Schwarz preconditioner without coarse level. Stopping criterion is a residual reduction of 10^{-6}. $*_D$ and $*_N$ indicate Dirichlet and Neumann boundary of the local problems, respectively, for the velocity u or the pressure p. Taken from [79].

N	16	64	196
System $\mathcal{F}$ using local projections	54	57	58
System $\mathcal{F}$ not using local projections	87	196	515
System $\overline{\mathcal{F}}$ with global Lagrange multiplier	56	60	61

Table 6.2: Iteration counts for the system $\mathcal{F}$ with and without local projections as well as for the system $\overline{\mathcal{F}}$. The latter is the system with global Lagrange multiplier. Two-dimensional LDC Stokes problem using $H/h = 50$, $\delta = 6h$, and Taylor–Hood elements. Iteration counts for the monolithic preconditioner with GDSW coarse space. GMRES is stopped when a reduction of 10^{-6} of the unpreconditioned relative residual is reached. Taken from [79].

to make the pressure unique. In particular, imposing Dirichlet boundary conditions for the pressure or restricting the pressure to the space $L_0^2(\Omega)$ are possible ways. As pointed out in section 6.3.1, the local overlapping problems possess homogeneous Dirichlet boundary conditions for the pressure. Additionally, we impose zero mean value by projecting the local pressure onto the subspace $L_0^2(\Omega_i)$, $i = 1, ..., N$.

Another way of imposing zero mean value locally is to introduce local Lagrange multipliers, resulting in local matrices

$$\overline{\mathcal{F}}_i = \begin{bmatrix} F_i & B_i^T & 0 \\ B_i & -C_i & a_i^T \\ 0 & a_i & 0 \end{bmatrix}, \; i = 1, ..., N, \tag{6.16}$$

where the vector a_i arises from the finite element discretization of the integral $\int_{\Omega_i'} p_h \, d\mathbf{x}$. In this case, we omit the projections $\overline{\mathcal{P}}_i$ in eq. (6.5) and eq. (6.7).

On the other hand, the matrix $\overline{\mathcal{F}}_i$ remains nonsingular even if, instead of Dirichlet boundary conditions, Neumann boundary conditions are imposed for the local pressure. Comparing both approaches separately for a monolithic one-level Schwarz preconditioner, we can observe that imposing Neumann boundary conditions performs slightly better; cf. table 6.1. However, this approach is not suitable for an algebraic implementation since, in general, we do not have access to the local Neumann matrices. Therefore, we use the variant with Dirichlet boundary conditions.

Furthermore, for the construction of the monolithic Schwarz preconditioners described in section 6.3, the local projections $\overline{P}_i$ have to be built. In practice, we do not compute the local projection matrices (6.4) explicitly, but just implement the application of $\overline{P}_i$, $i = 1, ..., N$, to a vector, requiring access to the local vectors a_i; they can be extracted from the global vector a, which arises from the discretization of eq. (5.9). If a is not available, geometric information is necessary for the construction of a_i.

For the saddle point problem with Lagrange multiplier of eq. (5.10), we can define an algebraic version of the monolithic GDSW preconditioner. Therefore, we omit the local projections $\overline{\mathcal{P}}_i$ in eq. (6.7). Then, the resulting two-level preconditioner is symmetric for a Stokes problem. In addition, we modify the definition of the restriction operators $\mathcal{R}_i$ such that the Lagrange multiplier is also added to the local problems:

$$\mathcal{R}_i = \begin{bmatrix} R_{i,u} & 0 & 0 \\ 0 & R_{i,p} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{6.17}$$

Consequently, the local overlapping matrices are of the form (6.16).

The construction of the GDSW coarse space is also slightly modified. Since the Lagrange multiplier is shared by all subdomains, we treat the Lagrange multiplier as a vertex of the domain decomposition, which is therefore part of the interface Γ. We add one coarse basis function corresponding to the Lagrange multiplier and modify the definition of ϕ_Γ accordingly:

$$\phi_\Gamma = \begin{bmatrix} \Phi_{\Gamma,u_0} & 0 & 0 \\ 0 & \Phi_{\Gamma,p_0} & 0 \\ 0 & 0 & \Phi_{\Gamma,\lambda_0} \end{bmatrix} = \begin{bmatrix} \Phi_{\Gamma,u_0} & 0 & 0 \\ 0 & \Phi_{\Gamma,p_0} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.18}$$

This monolithic GDSW preconditioner can be built in a purely algebraic fashion from system (5.10). It reads

$$\hat{\mathcal{B}}_{\text{GDSW}}^{-1} = \phi \mathcal{F}_0^{-1} \phi^T + \sum_{i=1}^{N} \mathcal{R}_i^T \mathcal{F}_i^{-1} \mathcal{R}_i,$$

where the coarse matrix is $\mathcal{F}_0 = \phi^T \overline{\mathcal{F}} \phi$ with ϕ_Γ defined in (6.18) and extended saddle point harmonically, using $\overline{\mathcal{F}}$ of eq. (5.10), to the interior degrees of freedom; cf. eq. (6.9). Note that the coarse matrix becomes nonsingular by adding the Lagrange multiplier. Furthermore, the local martices are constructed with the restrictions of eq. (6.17), $\mathcal{F}_i = \mathcal{R}_i \overline{\mathcal{F}} \mathcal{R}_i^T$, $i = 1, ..., N$.

We observe that both variants, i.e., using local projections and a global Lagrange multiplier, perform equally well; cf. table 6.2. Furthermore, without restricting the local pressure to $L_0^2(\Omega)$, the numerical scalability of the two-level monolithic Schwarz preconditioner with GDSW coarse space deteriorates.

6.3.5 GDSW Implementation Based on Trilinos

In this section, we will describe the different stages in the construction of a parallel two-level overlapping Schwarz GDSW preconditioner in Trilinos with the package `FROSch` [83]. It was previously published [79]. For simplicity, we begin with the setup of GDSW for the system matrix A of an elliptic problem. Then, we will highlight the necessary modifications for a Stokes or Navier–Stokes saddle point problem. In particular, special considerations must be made for a system with Lagrangian multiplier for the enforcement of pressure solutions with zero mean value. In this section and in Chapters 7 and 8, we highlight all `software packages` and specific `methods` with this `font`.

The GDSW setup of preconditioners in `FROSch` is partitioned into an `initialize` phase and a `compute` phase. The `initialize` phase sets up some basic structures of the preconditioner, i.e., it

- identifies the overlapping subdomains, and
- identifies the interface for the construction of Φ and constructs the coarse map and Φ_Γ.

The `compute` phase implements

- the construction of the matrix Φ that contains the coarse basis,
- the computation of the coarse matrix A_0,
- the extraction of local overlapping matrices A_i, $i = 1, ..., N$, from the global stiffness matrix A,
- the factorization of the local matrices A_i, $i = 1, ..., N$, and the coarse matrix A_0.

In the application phase, the two levels of the serial level additive Schwarz preconditioner are first applied separately and then summed up. In [85], exact local solvers are used, i.e., the local overlapping matrices A_i, $i = 1, ..., N$, are extracted from the global stiffness matrix and direct solvers are used for the solution of the local problems. The coarse basis functions are constructed as described in section 6.1. To optimize parallel performance we drop values from Φ which are smaller than 10^{-8}.

The current GDSW implementation in `FROSch` was restructured compared to the implementation [85]. In particular, it is now partitioned into a class for the first level, `AlgebraicOverlappingOperator`, and a class for the coarse level, `GDSWCoarseOperator`. Each of the classes contains both setup and application of the correspondig Schwarz operator. Both levels are coupled in an additive way using a `SumOperator`. For more details on the new structure of the implementation, we refer to [83]. Some results of this thesis were computed with an older GDSW version based on [79, 85]. This version was based on the `Epetra` linear algebra package of `Trilinos`. Instead, the newest version of `FROSch` is based on a linear algebra package called `Xpetra`, which allows to use `Epetra` and thus can be seen as a wrapper class. A more detailed discussion of the underlying software packages as well as more implementation details can be found in Chapter 7. However, we want to highlight some implementation details of the monolithic Schwarz preconditioners in the following sections. For our extension of the GDSW implementation to saddle point problems, we introduced the class `BlockMat` that implements block operators for `Epetra` objects. This class is used for both the implementation of the saddle point problem itself as well as the implementation of the monolithic preconditioners.

6.3.5.1 A Class for Block Matrices

We note that the following description is based on the GDSW version which exclusively used `Epetra`. To handle the block structure of saddle point problems, we implemented the `BlockMat` class. It is derived from the class `Epetra_Operator`, which defines an abstract interface for arbitrary operators in `Epetra`. To keep the implementation of the `BlockMat` class as general as possible, we allow for various types of blocks, e.g., `Epetra_Operator` or `Epetra_MultiVector` objects. Objects derived from `Epetra_Operator` are, e.g., `Epetra_CrsMatrix` objects, preconditioners like `Ifpack`, `ML`, and the GDSW implementation in [85], or even `BlockMat` objects.

Therefore, the class `BlockMat` could also be used to define block preconditioners, such as block-diagonal or block-triangular preconditioners, with `Ifpack`, `ML`, or GDSW preconditioners as blocks; cf. [78]. We use the `BlockMat` class throughout the implementation of the monolithic preconditioners with Lagrangian multipliers to maintain the block structure of all occurring matrices.

6.3.5.2 Setup of the First Level

As for elliptic problems, in the construction of the monolithic Schwarz preconditioner for saddle point problems, we extract the local overlapping matrices $\overline{\mathcal{F}}_i$ from the global matrix $\overline{\mathcal{F}}$.

In our algebraic implementation, we set up the index sets of the overlapping subdomains by adding layers of elements recursively based on the graph of the matrix $\mathcal{F}$. In each recursive step, the nonzero pattern of the subdomain matrices has to be gathered on the corresponding *Message Passing Interface* (MPI) [35] ranks, whereas the values of the matrix entries can be neglected. Even if a Lagrange multiplier is introduced to ensure zero mean value of the pressure, we only consider the submatrix $\mathcal{F}$ of $\overline{\mathcal{F}}$, because the Lagrange multiplier couples all pressure degrees of freedom.

Then, the local overlapping submatrices $\overline{\mathcal{F}}_i$, $i = 1, ..., N$, are communicated and extracted from $\overline{\mathcal{F}}$ using an `Epetra_Export` object based on the aforementioned local index sets of overlapping subdomains. Then, the matrices are stored in serial `Epetra_CrsMatrix` objects and factorized using direct solvers in serial mode. The setup is performed by passing the global `BlockMat` $\overline{\mathcal{F}}$ and a user-specified size of overlap. In addition to that, we specify the rows and columns of the block matrix which are excluded from the identification of the overlapping subdomains, i.e., the rows and columns corresponding to the Lagrange multiplier.

6.3.5.3 Setup of the Coarse Level

Communication	Avg. apply 1st lvl.	Avg. apply 2nd lvl.
Standard	1.42 s	0.59 s
Modified	1.28 s	0.12 s

Table 6.3: LDC Stokes problem in two dimensions with 4 096 subdomains using $H/h = 160$, $\delta = 16h$, and Taylor–Hood elements. Timings for critical parts w.r.t. communication time of the two-level preconditioner with GDSW coarse problem. Standard communication uses `Epetra_Import` and `Epetra_Export` objects only. Modified communication uses additional communication; cf. section 6.3.5.3. 'Avg. apply 1st lvl.' and 'Avg. apply 2nd lvl.' are times averaged over the number of iterations. Taken from [79].

As described in section 6.3.4, we add one coarse basis function with $\Phi_{\Gamma,\lambda_0} = 1$, that corresponds to the Lagrange multiplier, to the coarse space. The velocity and pressure variables are added, specifying the dimension of the domain, number of degrees of freedom per node, and the ordering of the degrees of freedom in the system. The ordering options are "`node-wise`", "`dimension-wise`", and "`user -defined`".

For the computation of the discrete harmonic extensions, we extract the matrices $\overline{\mathcal{F}}_{II}^{(i)}$ and $\overline{\mathcal{F}}_{I\Gamma}^{(i)}$, $i = 1, ..., N$, from $\overline{\mathcal{F}}$; since we assume that $\overline{\mathcal{F}}$ is distributed according to the nonoverlapping subdomains, no communication is needed for this step. Now, we factorize each $\overline{\mathcal{F}}_{II}^{(i)}$ in serial mode and solve $-\overline{\mathcal{F}}_{II}^{(i)}\phi_I^{(i)} = \overline{\mathcal{F}}_{I\Gamma}^{(i)}\phi_\Gamma^{(i)}$ column by column for $\phi_I^{(i)}$; cf. eq. (6.9). Then, the global `Blockmat` ϕ is assembled. Furthermore, the coarse matrix is computed as $\overline{\mathcal{F}}_0 = \phi^T\overline{\mathcal{F}}\phi$ and the resulting globally distributed coarse matrix is reduced to a smaller number of processes using multiple communication steps and a linear coarse map, cf. [85], or the package `Zoltan2` in combination with `ParMETIS` [100]. We used three communication steps in all numerical experiments with Lagrangian multiplier; in general, this led to the best total performance. Finally, if a direct coarse solver is used, the coarse matrix is factorized in serial or parallel mode.

6.3.5.4 Application of the Preconditioner

Our global `Epetra_Map` of the solution is uniquely distributed. Therefore, the Lagrange multiplier λ is assigned to only one MPI rank although it belongs to each overlapping subdomain. Also, the coarse basis function corresponding to the Lagrange multiplier couples all subdomains. Consequently, the handling of the Lagrange multiplier in the monolithic preconditioners requires all-to-one and one-to-all communication. As already mentioned in [85], such communication patterns are, in general, not handled well by `Epetra_Export` and `Epetra_Import` objects. One way to overcome this issue is to introduce multiple communication steps. Here, we use `MPI_Broadcast` and `MPI_Reduce` for the communication related to the Lagrange multiplier and `Epetra_Export` and `Epetra_Import` objects for the remaining d.o.f.. In table 6.3, we report the speedup due to separate communication of the Lagrange multiplier. We save 10% and 80% time in each application of the first and second level, respectively, for the LDC Stokes problem in two dimensions on 4 096 MPI ranks.

Besides this, the application of the monolithic GDSW preconditioner was handled as for elliptic problems; cf. [85].

6.3.6 Monolithic Reduced Dimension GDSW Preconditioners

So far, we mainly discussed the (standard) GDSW preconditioner. In order to reduce the dimension of our GDSW coarse spaces, we follow [54, 79] and introduce monolithic RGDSW coarse spaces. More precisely, we combine the construction described in section 6.3.3 with a different choice of interface components and interface values. For the parallel implementation of monolithic RGDSW coarse spaces, we extend our implementation of monolithic GDSW coarse spaces of section 6.3.3 and combine it with the parallel implementation of RGDSW coarse spaces for elliptic problems in `FROSch`; cf. [89]. We refer to this article for details on the parallel implementation. The derivation of the following monolithic RGDSW preconditioners for incompressible fluid flow problems was published in [80]. Only *Option 1* and *Option 2.2* of the RGDSW variants proposed in [54] are considered; Option 1 is algebraic and Option 2.2 additionally requires the coordinates of the finite element nodes. We will focus on the construction of the velocity basis functions Φ_{Γ_u}; the construction the pressure basis functions Φ_{Γ_p} is then performed analogously.

The index set $\mathcal{S}_{c_u}$ is the set of all subdomains which share the velocity interface component (i.e., vertex, edge, or face) c_u. In particular, velocity and pressure components are distinguished to allow for nonequal order discretizations or staggered grids. Moreover, a hierarchy of all interface components is defined: we call a component $c_{u,i}$ *ancestor* of $c_{u,j}$ if $\mathcal{S}_{c_{u,j}} \subset \mathcal{S}_{c_{u,i}}$; conversely, we call $c_{u,i}$ *offspring* of $c_{u,j}$ if $\mathcal{S}_{c_{u,j}} \supset \mathcal{S}_{c_{u,i}}$. We classify $c_{u,j}$ as *coarse component* if it has no ancestors. Its corresponding basis functions will be part of the RGDSW coarse space.

Let $\tilde{\gamma}_{u,i}$, $i = 1, ..., \tilde{M}_u$, be the coarse components of the RGDSW coarse space and

$$\tilde{\Gamma}_{u,i} := \bigcup_{\mathcal{S}_{c_u} \subset \mathcal{S}_{\tilde{\gamma}_{u,i}}} c_u$$

the union of the coarse component $\tilde{\gamma}_{u,i}$ and its respective offspring; the $\tilde{\Gamma}_{u,i}$, $i = 1, ..., \tilde{M}_u$, define an overlapping decomposition of the interface Γ_u. Furthermore, let $R_{\tilde{\Gamma}_{u,i}}$ be the restriction of all velocity d.o.f. from $\tilde{\Gamma}_u$ to $\tilde{\Gamma}_{u,i}$, similar to the GDSW coarse space, and let $S_{\tilde{\Gamma}_{u,i}} \in \mathbb{R}^{|\tilde{\Gamma}_u| \times |\tilde{\Gamma}_u|}$ be a suitable diagonal scaling matrix, such that we obtain an interface partition of unity

$$\sum_{i=1}^{\tilde{M}_u} S_{\tilde{\Gamma}_{u,i}} R_{\tilde{\Gamma}_{u,i}}^T R_{\tilde{\Gamma}_{u,i}} \underline{1} = \underline{1}_{\tilde{\Gamma}_u},$$

where $\underline{1}_{\tilde{\Gamma}_u} \in \mathbb{R}^{|\tilde{\Gamma}_u|}$ is the vector of ones on the interface. We obtain different reduced dimension coarse spaces, depending on the choice of the scaling matrices $S_{\tilde{\Gamma}_{u,i}}$, $i = 1, ..., \tilde{M}_u$. Next, we define

$$\tilde{R}_{\tilde{\Gamma}_{u,i}} := S_{\tilde{\Gamma}_{u,i}} R_{\tilde{\Gamma}_{u,i}}, \quad i = 1, ..., \tilde{M}_u.$$

The interface values of the velocity basis functions can then be written in the same form as for the classical GDSW coarse spaces

$$\Phi_{\tilde{\Gamma}_u} = \left[\; \tilde{R}^T_{\tilde{\Gamma}_{u,i}} \Phi_{\tilde{\Gamma}_{u,1}} \quad \ldots \quad \tilde{R}^T_{\tilde{\Gamma}_{u,\tilde{M}_u}} \Phi_{\tilde{\Gamma}_{u,\tilde{M}_u}} \; \right];$$

cf. (6.2). Analogous to the standard GDSW coarse spaces, the columns of $\Phi_{\tilde{\Gamma}_{u,i}}$ form a basis of the restriction of the nullspace Z_u to the $\tilde{\Gamma}_{u,i}$, such that the columns of $\Phi_{\tilde{\Gamma}_u}$ span the nullspace Z_u. The scaling matrices $S_{\tilde{\Gamma}^h_{u,i}}$ for variants of the RGDSW coarse space denoted as Option 1 and Option 2.2 in [54] are constructed as follows. In Option 1,

$$s_{\tilde{\Gamma}_{u,i}} = \begin{cases} 1/\left|\mathcal{C}_{c_u}\right| & \text{if } c_{u,i} \in \mathcal{C}_{c_u}, \\ 0 & \text{otherwise,} \end{cases}$$

with $\mathcal{C}_{c_u}$ being the set of all velocity ancestors of the interface component $c_{u,i}$. The corresponding scaling matrices read

$$S_{\tilde{\Gamma}_{u,i}} = \operatorname{diag}\left(s_{\tilde{\Gamma}_{u,i}}\right), \quad i = 1, ..., \tilde{M}_u.$$

Using basis functions based on an inverse distance weighting approach results in another option to define the scaling matrices; cf. [54]. In particular, the values of the scaling vectors are chosen as

$$s_{\tilde{\Gamma}_{u,i}} = \begin{cases} \frac{1/d_i(c_u)}{\sum\limits_{c_{u,j} \in \mathcal{C}_{c_u}} 1/d_j(c_u)} & \text{if } c_{u,i} \in \mathcal{C}_{u_n}, \\ 0 & \text{otherwise} \end{cases}$$

and $d_i(c_u)$ is the distance from the component c_u to the coarse component $c_{u,i}$. This construction is denoted as Option 2.2 in [54]. To compute the distance between different interface components the construction depends on additional geometric information. Option 1 is therefore more algebraic. The coarse pressure basis functions Φ_{Γ_p} are constructed correspondingly. We obtain the monolithic interface values analogously to (6.11) and extend them to the interior; cf. eq. (6.10). Compared to the standard GDSW coarse spaces, the reduced dimension coarse spaces have an advantage, since the dimension of the coarse problems are significantly smaller. In [89], it has been shown that RGDSW coarse problems can be smaller by more than 85 % for elliptic problems in three dimensions and structured domain decompositions; this typically results in a much improved parallel scalability. For results on the improved parallel scalability for incompressible fluid flow problems due to the use of RGDSW coarse spaces, see section 8.2.1.

6.3.7 Sequential and Parallel Computation of the Levels

The following discussion of parallel additive levels and a multiplicative sequential coupling of the levels was first presented in [80].

The levels are often computed in a sequential way when constructing two-level additive Schwarz preconditioners; cf. [79, 84, 85]. The coarse problem is typically solved on a small subset of ranks, and most of the cores are idle in the mean time. To circumvent this issue we will resort to the following two approaches: a multiplicative but sequential coupling of the levels and an additive coupling combined with parallel computation of the levels. The solver performance is improved with both methods; cf. section 8.2.3. A similar approach for the parallel computation of levels was already used in [8] for an implementation of a multilevel BDDC method.

Multiplicative Coupling of the Levels: In general, a better solver convergence is achieved through a multiplicative coupling of the levels. In particular, we use the hybrid preconditioner

$$\begin{aligned}\hat{\mathcal{B}}_{\text{hybrid}}^{-1} &= (\mathcal{I} - \mathcal{P}_0)\hat{\mathcal{B}}_{\text{AS}}^{-1}(\mathcal{I} - \mathcal{P}_0)^T + \phi\mathcal{A}_0^{-1}\phi^T, \text{with} \\ \mathcal{P}_0 &= \phi\mathcal{A}_0^{-1}\phi^T\mathcal{A};\end{aligned}$$

cf. [139]. If the projected Krylov method is started with a suitable initial vector $x_0 = \phi\mathcal{A}_0^{-1}\phi^T b$, the application of the hybrid preconditioner $\hat{\mathcal{B}}_{\text{hybrid}}^{-1}$ requires only one additional application of the system matrix $\mathcal{A}$ compared to the two-level additive preconditioner $\hat{\mathcal{B}}_M^{-1}$. Due to the multiplicative coupling of the levels, they have to be applied sequentially.

Parallel Computation of the Levels: When the levels are coupled additively, a significant amount of work for the construction and the application of the levels can be performed in parallel which is why the MPI ranks are split among the levels. For a fixed total number of MPI ranks, the number of subdomains is decreased and thus the size of the overlapping subdomains is increased slightly. Additionally, we compute the coarse basis functions ϕ and the RAP product (triple matrix product) of the coarse matrix $\mathcal{A}_0$ on MPI ranks assigned to the first level. In contrast, the factorizations and forward-backward solves of the local overlapping and the coarse problems are computed in parallel. We refer to section 8.2.3 for results on the speedup of the above described coupling strategies compared to the sequential additive coupling.

6.4 Fully Algebraic Construction of GDSW and RGDSW Coarse Spaces

The derivation of the fully algebraic methods described in this section have been previously published in [81]. We will focus on the algebraic construction of preconditioners for elasticity problems, where the nullspace not only consists of the three translations but also of the (linearized) rotations. The coarse basis functions of a three dimensional elasticity problem are defined by the three translations

$$r_1 := \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad r_2 := \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad r_3 := \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

and by the linearized rotations

$$r_4 := \begin{bmatrix} x_2 - \hat{x}_2 \\ -(x_1 - \hat{x}_1) \\ 0 \end{bmatrix}, \quad r_5 := \begin{bmatrix} -(x_3 - \hat{x}_3) \\ 0 \\ x_1 - \hat{x}_1 \end{bmatrix}, \quad r_6 := \begin{bmatrix} 0 \\ x_3 - \hat{x}_3 \\ -(x_2 - \hat{x}_2) \end{bmatrix},$$

with the origin of the rotation at $\hat{\mathbf{x}} \in \Omega$. By analogy with the construction of a GDSW preconditioner for a mixed linear elasticity, cf. section 6.3.3, for straight edges in three dimensions, only two linearized rotations are linearly independent. Therefore, we only use two rotations for each of these edges. Furthermore, all rotations are omitted for vertices.

As previously described, the construction of GDSW and RGDSW coarse spaces for various problems requires both the domain decomposition interface and the

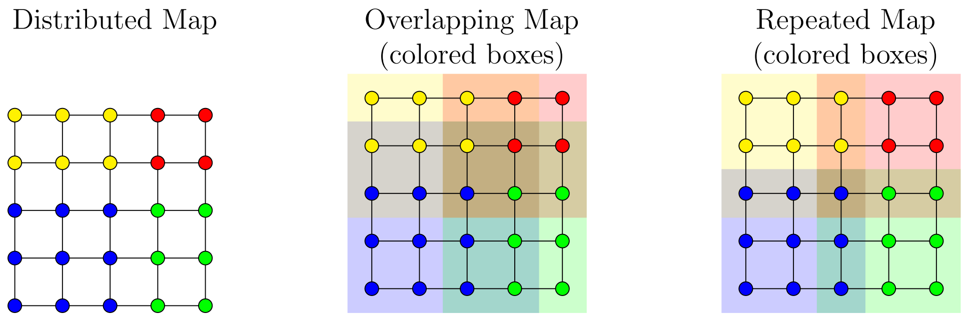


Figure 6.5: Sketch of the approximation of the nonoverlapping subdomains and the interface, respectively: uniquely distributed map (left); extension of the uniquely distributed map by one layer of elements resulting in an overlapping map where the overlap contains the interface (center); by selection, using the lower subdomain ID, a map approximating the nonoverlapping subdomains is constructed (right). Taken from [81, 84].

nullspace of the operator, i.e., all rigid body motions. In particular, the linearized rotations can only be computed if coordinate values are available. We will describe a method to approximate the nonoverlapping subdomains, resulting in an approximate interface, if no geometric information is provided; cf. [84]. In particular, we consider the case when the system matrix is uniquely distributed, such that the interface cannot be identified. To discuss the performance of the fully algebraic approach, we will compare it to the classical GDSW type coarse spaces using all necessary information. These results are presented in section 8.5, where we consider the solution of large, parallel distributed stationary and dynamic elasticity problems with a moderate Poisson ratio; i.e., we do not consider the almost incompressible limit case. Here, we describe how we construct the coarse space if this information is not available.

Algebraic approximation of the interface: If the distribution of the system matrix is unique, the interface cannot be recovered. Therefore, we will carry out the following process to approximate the nonoverlapping subdomains and hence the interface. Starting from the unique distribution, we first add one layer of elements to each subdomain. The overlap of the resulting domain decomposition now contains the interface but also other finite element nodes. In order to reduce the number of unnecessary nodes, we compare the subdomain ID of the original unique decomposition and the decomposition with one layer of overlap and remove nodes from the overlapping subdomains, if the subdomain ID is lower compared to the original decomposition; this process is sketched in [84] and fig. 6.5.

Incomplete nullspace: The rigid body modes are the translations and rotations of the elastic body. The translations are constant functions which can be constructed without any geometric information. Since we are not able to compute the rotations from the fully assembled matrix and without coordinates of the finite element nodes, we just omit them in the fully algebraic coarse space; see also [85]. For the results in section 8.5, only the number of iterations is negatively affected by omitting rotations from the coarse space, but the time-to-solution actually benefits from the smaller coarse space. Note that, from theory, the rotational basis functions are necessary for numerical scalability. Therefore, we expect that there are types of problems for which the full coarse space performs better.

6.5 Preconditioners for Fluid-Structure Interaction Problems

In this section we will present preconditioners for the geometry explicit and implicit FSI problems, presented in section 5.3.1. We will begin with an overview of the *FaCSI* preconditioner [43] and further discuss some variants of it. The name FaCSI originates from the *Factorzied* (Fa) FSI block system, the *static Condensation* (C) and the use of the *SIMPLE* (SI) preconditioner for the fluid problem. In general, variants of the FaCSI preconditioner can be obtained by using different preconditioners for the fluid subproblem. Instead of a SIMPLE preconditioner we will use our monolithic preconditioner, which was presented in section 6.3. Similarly, we can use any other preconditioner which is an approximation to the inverse of the Navier–Stokes problem matrix. Based on the previously presented monolithic overlapping Schwarz preconditioners for fluid flow problems, the construction of monolithic preconditioners for FSI problems will be presented in section 6.5.2.

6.5.1 FaCSI Block Preconditioner

We will follow the presentation in [43] and will start by reordering the blocks of our fully coupled GI FSI system, which was linearized using Newton's method. Therefore, we want to approximate the FSI Jacobian matrix with a FaCSI preconditioner, which also includes shape derivatives. We reorder the system of (5.28) in a way that the first block is the solid problem S, followed by the geometry problem G. The third block is the fluid problem $\mathcal{F}$, and the final block consists of our Lagrangian multiplier for the coupling of stresses on the interface. However, we will consider our 2×2 fluid subproblem in combination with the coupling blocks and will use this 3×3 block subsystem. Furthermore, we drop all explicit dependencies on the last solutions. In the following, $\mathcal{D}$ are the shape derivatives. The reordered Jacobian matrix of the FSI system reads

$$\mathcal{J} = \left(\begin{array}{c:c:cc} S & 0 & 0 & C_4 \\ \hdashline C_5 & G & 0 & 0 \\ \hdashline 0 & \mathcal{D} & \mathcal{F} & C_3 \\ C_2 & 0 & C_1 & 0 \end{array}\right).$$

The FaCSI preconditioner is based on an incomplete block factorization. Therefore, the C_3 block in the upper part is neglected in the construction and the resulting block preconditioner is an upper triangular block matrix:

$$\mathcal{B}_{\text{FaCSI}} = \begin{pmatrix} S & 0 & 0 & 0 \\ C_5 & G & 0 & 0 \\ 0 & \mathcal{D} & \mathcal{F} & C_3 \\ C_2 & 0 & C_1 & 0 \end{pmatrix} \tag{6.19}$$

Next, we can decouple the different physical subproblems in eq. (6.19) and obtain

$$\mathcal{B}_{\text{FaCSI}} = \begin{pmatrix} S & 0 & 0 & 0 \\ 0 & I_G & 0 & 0 \\ 0 & 0 & I_{\mathcal{F}} & 0 \\ 0 & 0 & 0 & I_\Gamma \end{pmatrix} \begin{pmatrix} I_S & 0 & 0 & 0 \\ C_5 & G & 0 & 0 \\ 0 & 0 & I_{\mathcal{F}} & 0 \\ 0 & 0 & 0 & I_\Gamma \end{pmatrix} \begin{pmatrix} I_S & 0 & 0 & 0 \\ 0 & I_G & 0 & 0 \\ 0 & \mathcal{D} & \mathcal{F} & C_3 \\ C_2 & 0 & C_1 & 0 \end{pmatrix} = \mathcal{B}_S \mathcal{B}_G \mathcal{B}_{\mathcal{F}}.$$

An approximate inverse to $\mathcal{B}_S$ can now be computed, since it is already block-diagonal. In general, we will use the previously presented two-level overlapping

Schwarz preconditioners for this approximation. However, the other two matrices $\mathcal{B}_G$ and $\mathcal{B}_\mathcal{F}$ can be further factorized.

$$\mathcal{B}_G = \begin{pmatrix} I_S & 0 & 0 & 0 \\ C_5 & I_G & 0 & 0 \\ 0 & 0 & I_\mathcal{F} & 0 \\ 0 & 0 & 0 & I_\Gamma \end{pmatrix} \begin{pmatrix} I_S & 0 & 0 & 0 \\ 0 & G & 0 & 0 \\ 0 & 0 & I_\mathcal{F} & 0 \\ 0 & 0 & 0 & I_\Gamma \end{pmatrix} = \mathcal{B}_{G_1}\mathcal{B}_{G_2}$$

$$\mathcal{B}_\mathcal{F} = \begin{pmatrix} I_S & 0 & 0 & 0 \\ 0 & I_G & 0 & 0 \\ 0 & \mathcal{D} & I_\mathcal{F} & 0 \\ 0 & 0 & 0 & I_\Gamma \end{pmatrix} \begin{pmatrix} I_S & 0 & 0 & 0 \\ 0 & I_G & 0 & 0 \\ 0 & 0 & I_\mathcal{F} & 0 \\ C_2 & 0 & 0 & I_\Gamma \end{pmatrix} \begin{pmatrix} I_S & 0 & 0 & 0 \\ 0 & I_G & 0 & 0 \\ 0 & 0 & \mathcal{F} & C_3 \\ 0 & 0 & C_1 & 0 \end{pmatrix} = \mathcal{B}_{\mathcal{F}_1}\mathcal{B}_{\mathcal{F}_2}\mathcal{B}_{\mathcal{F}_3}$$

We can now approximate the inverse of the block-diagonal matrix $\mathcal{B}_{G_2}$ similar to $\mathcal{B}_S$. Furthermore, we can exactly invert $\mathcal{B}_{G_1}$, $\mathcal{B}_{\mathcal{F}_1}$, and $\mathcal{B}_{\mathcal{F}_2}$. Now, only the 3 × 3 subproblem

$$\begin{pmatrix} \mathcal{F}_k & C_3 \\ C_1 & 0 \end{pmatrix} \begin{pmatrix} x_k \\ \lambda_k \end{pmatrix} = \begin{pmatrix} r_{\mathcal{F},k} \\ r_{\lambda,k} \end{pmatrix} \tag{6.20}$$

of $\mathcal{B}_{\mathcal{F}_3}$ must be approximated. In general, we now consider the k-th nonlinear iteration. Here, $r_{\mathcal{F},k}$ and $r_{\lambda,k}$ are residual vectors arising in the iterative solution process, and x_k and λ_k are the solution vectors which we want to compute. Moreover, the coupling matrix C_1 in eq. (6.20) restricts all fluid d.o.f's., i.e. velocity and pressure, to the fluid velocity interface. In particular, we separate interior variables I and interface variables Γ of x_k. We use the same notation for interior and interface variables as in the derivation of our domain decomposition methods. However, we highlight that we distinguish between physical parts of our domains, i.e., an interface of the fluid and solid domain and variables in the interior of our fluid domain. Furthermore, we note that the pressure is not restricted to the interface and only velocities are coupled. Rewriting eq. (6.20) as the linear system

$$\mathcal{F}_k x_k + C_3\lambda_k = r_{\mathcal{F},k} \tag{6.21}$$

$$C_1 x_k = r_{\lambda,k} \tag{6.22}$$

and using the separation into interior and interface d.o.f. gives us the solution $x_{k,\Gamma}$ from eq. (6.22). The solution $x_{k,\Gamma}$ is now inserted into eq. (6.21) and the elimination of λ_k follows:

$$\lambda_k = C_1 C_3 \lambda_k = C_1 (r_{\mathcal{F},k} - \mathcal{F}_k x_k).$$

Finally, we obtain the condensed system for the fluid subproblem

$$\mathcal{F}_{k,II} x_{k,I} = r_{k,I} - \mathcal{F}_{k,I\Gamma} x_{k,\Gamma}. \tag{6.23}$$

More details on the derivation of the FaCSI preconditioner can be found in [43]. Equation (6.23) can then be approximated with a SIMPLE preconditioner. However, any other block or monolithic preconditioner can be used as well. We will compare the performance of FaCSI using a SIMPLE preconditioner with FaCSI using the monolithic (R)GDSW preconditioners in section 8.6. For the subproblems in the SIMPLE preconditioner we will again use our two-level overlapping Schwarz preconditioners with GDSW type coarse spaces. We finish this section with a pseudo-code to highlight the different steps in the application of $\mathcal{B}_{\text{FaCSI}}^{-1}$; cf. [43]. Therefore, we consider the residual $r = (r_{d_s}^T, r_{d_f}^T, r_u^T, r_p^T, r_\lambda^T)^T$ and solve $\mathcal{B}_{\text{FaCSI}} w = r$. Furthermore, we approximate the inverses of S, G, and $\mathcal{F}_{II}$ with suitable preconditioners $\hat{S}^{-1}$, $\hat{G}^{-1}$, and $\hat{\mathcal{F}}_{II}^{-1}$, respectively, and denote the corresponding FaCSI preconditioner as $\hat{\mathcal{B}}_{\text{FaCSI}}^{-1}$. As we have seen in the derivation, the application of a FaCSI preconditioner can be divided into three separated parts: $\hat{\mathcal{B}}_S^{-1}$, $\hat{\mathcal{B}}_G^{-1}$, and $\hat{\mathcal{B}}_{\mathcal{F}}^{-1}$. Moreover, we do not use an explicitly condensed fluid problem but rather fix all fluid velocity interface d.o.f. in the preconditioner for the fluid problem. Instead of using an approximate inverse $\hat{\mathcal{F}}_{II}^{-1}$ of $\mathcal{F}_{II}$, we use the auxiliary matrix $\overline{\mathcal{F}}$ which is initialized as a copy of $\mathcal{F}$. Then, in addition to the Dirichlet boundary conditions of the original problem, we introduce a homogeneous Dirichlet condition for all fluid interface d.o.f. in $\overline{\mathcal{F}}$. In the following, we can use the approximate inverse $\hat{\overline{\mathcal{F}}}^{-1}$ of the auxiliary matrix $\overline{\mathcal{F}}$, which makes the implementation more compact. All of the above steps are summarized in algorithm 3.

The shape derivatives $\mathcal{D}$ may be zero due to a different linearization strategy. Moreover, if the underlying problem matrix $\mathcal{J}$ arises from a geometry explicit

Algorithm 3 Application of $\hat{\mathcal{B}}_{\text{FaCSI}}^{-1}$

```
In: $\hat{S}^{-1}$, $\hat{G}^{-1}$, $\hat{\overline{\overline{\mathcal{F}}}}^{-1}$, Jacobian FSI matrix $\mathcal{J}$, residual $r$
Out: Solution $w$ of $\hat{\mathcal{B}}_{\text{FaCSI}} w = r$
1. Solid part: $w_{d_s} = \hat{S}^{-1} r_{d_s}$
2. Geometry part: $w_{d_f} = \hat{G}^{-1}(r_{d_f} - C_4 w_{d_s})$
3. Fluid part:
  $z_{\mathcal{F}} = r_{\mathcal{F}} - \mathcal{D} w_{d_f}, z_\lambda = r_\lambda - C_2 w_{d_s}$, with $z_{\mathcal{F}} = (z_u^T, z_p^T)^T$
  Condense fluid: $z_u = z_u - C_3 C_1 z_u$, $z_u = z_u + C_3 z_\lambda$
  Apply auxiliary fluid preconditioner: $w_{\mathcal{F}} = \hat{\overline{\overline{\mathcal{F}}}}^{-1} z_{\mathcal{F}}$, with $w_{\mathcal{F}} = (w_u, w_p)^T$
  $w_\lambda = C_1(z_u - F w_u - B^T w_p)$
```

coupling, we simply skip the geometry and shape derivative parts in the above algorithm, since the geometry problem is not a part of $\mathcal{J}$ in this case.

6.5.2 One-level Monolithic Overlapping Schwarz Preconditioners for FSI

In this section, we present an extension of the monolithic overlapping Schwarz preconditioners for fluid flow problems to FSI problems. A similar monolithic approach was already used in [148]. We skip the definitions of the different (sub)spaces and only highlight the construction of the first level restriction operators.

The monolithic restriction operators for the overlapping subdomains of the GI FSI problem have the form

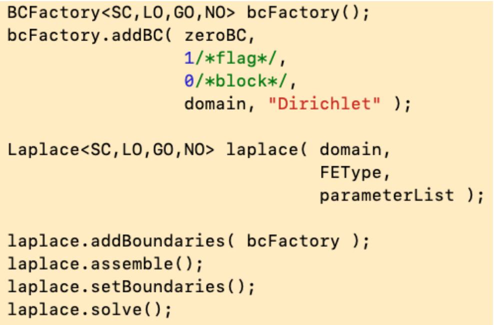
$$\mathcal{R}_{GI,i} := \begin{bmatrix} R_{u,i} & 0 & 0 & 0 & 0 \\ 0 & R_{p,i} & 0 & 0 & 0 \\ 0 & 0 & R_{\Gamma,i} & 0 & 0 \\ 0 & 0 & 0 & R_{s,i} & 0 \\ 0 & 0 & 0 & 0 & R_{g,i} \end{bmatrix},$$

and for the GE FSI problem they take the form

$$\mathcal{R}_{GE,i} := \begin{bmatrix} R_{u,i} & 0 & 0 & 0 \\ 0 & R_{p,i} & 0 & 0 \\ 0 & 0 & R_{\Gamma,i} & 0 \\ 0 & 0 & 0 & R_{s,i} \end{bmatrix},$$

$i = 1, ..., N$. Here, we combine the restriction operators of the velocities $R_{u,i}$, the pressure $R_{p,i}$, the interface $R_{\Gamma,i}$, the solid displacements $R_{s,i}$ and, in the GI case, the mesh displacements $R_{g,i}$. We highlight that the overlapping subdomains Ω_i', $i = 1, ..., N$, which correspond to the monolithic restriction operators, must be seen as monolithic subdomains. In the construction of monolithic preconditioners for fluid flow problems, we always used subdomains for the velocities and the pressure which occupied the same space w.r.t the geometry. In general, this is different for the monolithic one-level preconditioner of the FSI problem. Only the interface is shared by the fluid and solid domains. Therefore, special care has to be taken when constructing subdomains of FSI problems. We will highlight this issue in section 8.6. The next logical step is the construction of a monolithic two-level preconditioner with (R)GDSW coarse spaces. Although, our general implementation of the monolithic (R)GDSW preconditioners allows for an easy adaptation to the presented FSI problems this is a topic for future investigations.

7 FEDDLib



```
BCFactory<SC,LO,GO,NO> bcFactory();
bcFactory.addBC( zeroBC,
                 1/*flag*/,
                 0/*block*/,
                 domain, "Dirichlet" );

Laplace<SC,LO,GO,NO> laplace( domain,
                              FEType,
                              parameterList );

laplace.addBoundaries( bcFactory );
laplace.assemble();
laplace.setBoundaries();
laplace.solve();
```

The FEDDLib (Finite Element and Domain Decomposition Library) is an object-oriented C++ library which was developed for the parallel solution of partial differential equations. In general, the parallelization of the code is achieved with the MPI [35]. An MPI-parallel code can run the same program distributed among many processes. The MPI nomenclature defines these processes as *ranks*. MPI is a standard for programs which run on distributed memory systems. There, each rank has a dedicated memory and MPI manages the communication between the different ranks. In contrast, *OpenMP* [40] is an application programming interface for shared-memory systems, where each process has access to the same memory. These two parallelization techniques are not exclusive and can be used in combination.

In the following, we will focus on the MPI parallelization of the FEDDLib. Furthermore, we will use the terms *ranks*, *processes*, and *cores* synonymously. In general, this is not the case, as a single core can spawn multiple ranks or threads. However, we will always use one MPI rank per core.

We start with a short introduction of the C++ library Trilinos [90] in section 7.1. Trilinos consists of over 50 subpackages and several of these subpackages are used within the FEDDLib. In section 7.2, we give an overview of the general structure of the FEDDLib and the coupling with Trilinos. In particular, we present the wrapper for the Trilinos linear algebra classes. Section 7.3 explains the discretization methods of the FEDDLib. This is followed by section 7.4, where we discuss how specific PDEs are defined and used within the abstract FEDDLib framework. In section 7.5, we will give an overview of solution strate-

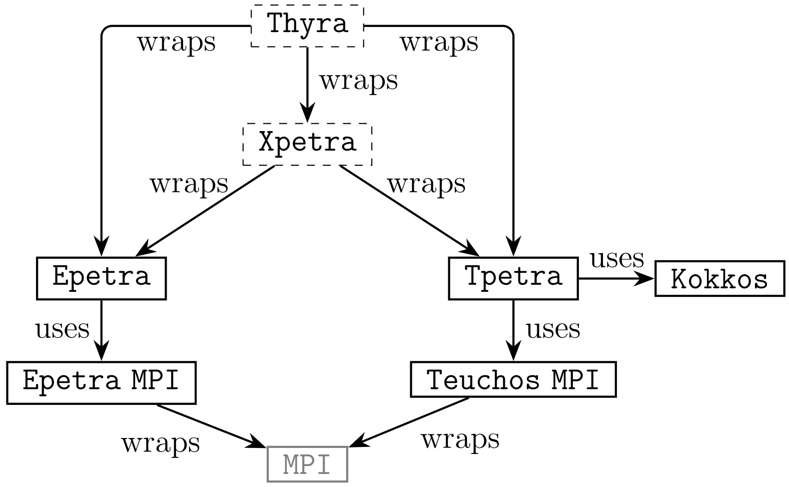


Figure 7.1: Trilinos parallel linear algebra overview. Dashed items are pure wrapper packages. Note that the faded `MPI` library is not part of Trilinos.

gies for the discretized systems. The last section 7.6 explains the post-processing and export tools.

7.1 Trilinos

The main building blocks of `Trilinos` are the two parallel linear algebra packages `Epetra` and `Tpetra`. `Epetra` is the older linear algebra package and it manages the parallel computation of all common operations between scalars, vectors, and matrices. The newer package `Tpetra` basically covers the same operations, but additionally supports the use of templates and the use of the package `Kokkos`; cf. [55]. Within the last years, it became popular to use *graphics processing units* (GPUs) for computations which were previously exclusively carried out with *central processing units* (CPUs). In general, code for CPUs cannot be used directly for GPUs. Therefore, existing code must be ported or rewritten for GPUs. Here, `Kokkos` provides a framework for advanced multicore architectures. It maximizes the amount of reusable user code without modifications and minimizes the amount of architecture specific knowledge. Performance of applications depends heavily on device specific memory access. This becomes more evident if we look at CPU and GPU vector access. CPU vector units require stride-one for the best performance. In constrast, GPU vector units require coalesced access, which allows for multiple memory access in a single transaction with thread parallelization; cf. [55]. These different access patterns are managed by `Kokkos`. Furthermore, `Kokkos` manages thread-parallel work and is orthogonal to MPI, which results in an additional layer of parallelization. With `Kokkos`, `Tpetra` can use OpenMP, *POSIX Threads* (Pthreads), or *Nvidia's CUDA model* for GPUs to allow for shared-memory programming. The package `Xpetra` is the third algebra package in `Trilinos`. It provides wrapper classes for a common access to the `Epetra` and the `Tpetra` linear algebra; cf. fig. 7.1. Moreover, in `Trilinos` a second linear algebra wrapper is used. The package `Thyra` provides some of the same wrapping mechanisms as `Xpetra`. `Thyra` wraps all of the three previously presented linear algebra packages `Epetra`, `Tpetra`, and `Xpetra`. The main difference between `Thyra` and `Xpetra` is that `Thyra` is more of an interface class for the interaction of different packages and it does not wrap all of the methods of `Epetra` and `Tpetra`. In contrast, `Xpetra` wraps almost all of the `Epetra` and `Tpetra` methods. In general, `Trilinos` packages must convert a given `Thyra` object internally to the underlying `Epetra`, `Tpetra`, or `Xpetra` object in order to be able to use the full range of methods. Another important package which is extensively used in the `FEDDLib` is `Teuchos`. It provides, among other things, wrappers to MPI, smart pointers, and parameterlist management tools. The smart pointers of the class `Teuchos::RCP` (Reference Count Pointers) are similar to the `Boost` smart pointers and the smart pointers of the `C++` standard. With the introduction of `C++11`, the `C++` standard library provides smart pointers which originate from the `Boost` smart pointers. In general, smart pointers use reference counts, which means that the underlying object of the pointer(s) is only freed if the last existing smart pointer is deleted. Furthermore, the allocated memory of the underlying object must not be freed by the user, but is instead cleared by the smart pointer which automatically calls the deconstructor of the underlying object. Other `Trilinos` packages are introduced in the following sections. There, we will discuss the use of the core `Trilinos` solver packages `Amesos`, `Amesos2`, and `Belos`. Additionally, the packages `NOX`, `Teko`, and `FROSch` are introduced, which provide solution strategies for nonlinear problems, block-preconditioners for Stokes and Navier–Stokes saddle point problems, and a framework for Schwarz preconditioners, respectively.

7.2 General Structure

As mentioned in the previous section, Xpetra provides a common interface to the Trilinos linear algebra packages. In the FEDDLib, we wrap the map, multi–vector, and matrix classes of Xpetra with the FEDD::Map, FEDD::MultiVector, and FEDD::Matrix classes, respectively. The combination of all these classes is denoted as the FEDD::LinearAlgebra (FEDD::LA) . Within the FEDDLib these classes are almost always used. Therefore, it is possible to use the FEDDLib with other linear algebra subroutines, e.g., PETSc [9], if the necessary methods for FEDD::Map, FEDD::MultiVector, and FEDD::Matrix are implemented. An exception to the use of these FEDDLib linear algebra classes is made when the solvers are called. There, we need to work with the underlying Thyra objects, which can be easily extracted. The Thyra objects are then passed to the abstract Thyra interface and the relevant underlying solvers, e.g., Belos GMRES, the nonlinear solvers of NOX, or the overlapping Schwarz preconditioners of FROSch are constructed with the information provided by a parameterlist. We will continue the discussion of these solvers in section 7.5.

The use of the matrix and multi-vector classes is straightforward. Here, a multi-vector is a vector with one or more columns which are equally distributed. In contrast to the sparse matrix format, a multi-vector is dense. The map classes of Trilinos and FEDDLib are used in the construction of vectors and matrices. A map determines the parallel distribution of indices across MPI ranks and it maps local indices to global indices and viceversa. In general, a map provides a distribution of matrix and vector rows. With a distribution of matrix rows, all columns can be accessed by each rank. For the assembly of finite element matrices and in the context of domain decomposition methods, we have to distinguish between two types of maps. The first type is a map which uses a unique distribution of global indices and we call a map of this type a *unique map*. The second type of map uses a nonunique distribution of global indices and we call it *repeated map*. Similar to the FEDD::Map, FEDD::MultiVector, and FEDD::Matrix classes, the FEDDLib implements corresponding classes for block systems. The classes FEDD::BlockMap, FEDD::BlockMultiVector, and FEDD::BlockMatrix manage the use of maps, multi-vectors, and matrices for block systems based on the FEDDLib classes for nonblock systems. Although Xpetra also provides block matrix classes, it is advantageous to implement block classes directly in the FEDDLib, since they do not need to be reimplemented if the underlying linear algebra is changed. Furthermore, we need to highlight that the FEDDLib mainly consists of template classes. In general, the same template parameters are used as in Xpetra. Each FEDD::Map uses a template for the local indices (Local Ordinal, LO), the global indices (Global Ordinal, GO), and the node type (Node Ordinal, NO) which specifies the node–level programming model. In addition, FEDD::MultiVector and FEDD::Matrix objects use a fourth template, the scalar type (SC). We use the following default template parameters:

```
LO = int; GO = long long;
SC = double; NO = KokkosClassic::DefaultNode;
```

The node type is a default type and it depends on the Trilinos build. If Trilinos was built without a threading library, then it defaults to KokkosClassic::SerialNode. For a Trilinos build with the threading library Pthreads it defaults to KokkosClassic::TPINode.

7.3 Meshes and Finite Elements

The starting point of an FEM implementation is a discretization of the computational domain Ω. The class FEDD::Mesh provides node and element lists which define an approximate discretization of Ω. Furthermore, we distinguish between structured and unstructured meshes which are implemented in the classes FEDD::MeshStructured and FEDD::MeshUnstructured, respectively. These classes are derived from FEDD::Mesh and thus can be used without explicit knowledge of the underlying type in the FEM. Structured meshes are only used for simple geometries, e.g., a square, cube, or backward facing step domain in two or three dimensions, which can be entirely built in parallel. In contrast, unstructured meshes are used for more complex domains which are usually defined by external files. We use the *MEDIT Inria* mesh format (.mesh) [71] for these external meshes. A .mesh-file is loaded globally and then partitioned with METIS [99] in a second step. In a third step, the mesh is distributed with the previously computed METIS partition among all relevant ranks. In section 6.3.7, we introduced a strategy for parallel coarse solves for our two-level overlapping Schwarz preconditioners. There, we used a set of ranks for the subdomains of the first level and the problem matrix and a second set of ranks dedicated to the coarse problem. The first set is chosen in accordance with the above METIS partition and the second set of ranks is left untouched until the computation of the coarse problem is needed. Furthermore, in a multi-physics problem with multiple computational domains, such as an FSI problem, we can define two different types of partitions. The first type uses a partition of all meshes across all relevant ranks. In contrast, the second type uses an individual partition for a set of ranks for each computational domain. In general, considering an FSI problem with a fluid and a solid domain, we partition the fluid domain among ranks 0 to M and the solid domain among ranks $M+1$ to $N-1$ for N total ranks. Here, the fractions for fluid and solid can be defined by the user or, more conveniently, based on the number of elements of the fluid and solid mesh. Furthermore, we highlight that the METIS partition is computed for the dual graph of the underlying mesh, which means that the mesh elements and not the nodes are partitioned. Consequently, this partition defines a repeated distribution of nodes which corresponds to the nonoverlapping subdomains of our domain decomposition methods. This repeated distribution of nodes also defines our first FEDD::Map. In a next step, this repeated map is reduced to a unique map, which then defines the distribution of nodes of the problem matrix. Up to this point, we only considered node maps, but we also need to handle maps for vector fields. This is done with the class FEDD::Domain which implements build and access methods for the corresponding d.o.f. maps and access methods for the underlying node maps.

Now, we have all necessary objects to proceed with the FEDDLib finite element assembly routines. However, we first need to take a closer look at the construction and use of matrices and vectors in Trilinos. Both Trilinos linear algebra packages, Epetra and Tpetra, and consequently the FEDDLib use the same set of maps for the description of MPI distributed objects. We emphasize the construction of matrix objects here. In general, a distributed Xpetra or FEDDLib matrix is defined by four maps: a domain map, a range map, a row map, and a column map. We initialize a parallel distributed matrix object with a distributed unique row map as follows:

```
MatrixPtr_Type A = rcp(new Matrix_Type(domain->getMapUnique()));
```

The matrix object of type Matrix_Type is passed to a Teuchos smart pointer and stored in A. We often use custom type definitions such as Matrix_Type or MatrixPtr_Type which reduces the amount of code and makes the code more readable. The above type definitions are simply the `FEDD::Matrix` and a smart pointer to an `FEDD::Matrix` with default template parameters:

```
typedef FEDD::Matrix<LO,GO,SC,NO> Matrix_Type;
typedef Teuchos::RCP<Matrix_Type> MatrixPtr_Type;
```

The second map which describes the distribution of a matrix is the column map. In general, this map is not created by the user, but built by the matrix object in the final phase of the matrix construction. After an empty matrix is constructed, we set new matrix values for each global row by passing an array of values and an array of global column indices to the matrix:

```
A->insertGlobalValues( row, colIndices, values );
```

However, at this point the matrix cannot be used for general linear algebra routines, such as matrix-vector or matrix-matrix multiplications, since all index sets are global, but are only locally available. The final step of the matrix construction is a transfer from global to local indices, which is performed with the method:

```
A->fillComplete( );
```

The matrix pattern is now fixed and we are not allowed to set new entries, only existing values can be changed. Furthermore, a matrix column map is now constructed by the matrix object. The column map is, in general, repeated and consists of all column indices that exist on a given rank. We highlight the structure of row and column maps with the following example. Let $A \in \mathbb{R}^{4\times 4}$ be distributed among two ranks. The row map is provided by the user and the column map is generated with a call of `fillComplete()`:

$$A = \left(\begin{array}{cccc} 3 & 2 & 0 & 4 \\ 0 & 0 & 1 & 0 \\ \hdashline 2 & 0 & 1 & 3 \\ 0 & 0 & 4 & 2 \end{array}\right) \qquad \begin{array}{rl} \text{row map rank 0:} & (0,1) \\ \text{rank 1:} & (2,3) \\ \text{column map rank 0:} & (0,1,2,3) \\ \text{rank 1:} & (0,2,3) \end{array}$$

The parallel distribution of the matrix is now complete, but two additional maps are needed for the use of parallel linear algebra routines. These maps are the `domainMap` and the `rangeMap`, and they must be consistent with the other participating linear algebra objects. With a call of `fillComplete()`, the domain map and range map are automatically set to the distributed row map of A. Other maps can be provided with the method `fillComplete(domainMap,rangeMap)`. Furthermore, this method must be used if nonsquare matrices are constructed.

We provide an example for these maps for a generic matrix A $\in \mathbb{R}^{n\times m}$ multiplied with a (multi-)vector x $\in \mathbb{R}^{m}$ and the resulting (multi-)vector y $\in \mathbb{R}^{m}$.

```
// objects for matrix-vector product y=Ax
MatrixPtr_Type A = rcp(new Matrix_Type(mapA));
MVPtr_Type x = rcp(new MV_Type(mapX));
MVPtr_Type y = rcp(new MV_Type(mapY));
... // setting values of A and x
A->fillComplete(mapX,mapY);
```

Although mapY and mapA (row map of A) possess the same global indices, they can be distributed differently. However, in practice we mostly use mapY = mapA. All FEDDLib finite element routines use an element-based assembly, which means that a node or d.o.f. is used by one or more elements. Therefore, a node or d.o.f. might also be used by elements of different ranks. Each newly computed value of an element is set to the global matrix and saved separately w.r.t. the ranks. Only a call of fillComplete() aggregates these values by summation for a unique row map. Additionally, node or d.o.f. contributions computed by ranks which do not own this node or d.o.f. with respect to the unique map are communicated. For blocks systems, we assemble each matrix separately and place the resulting matrix in the corresponding position in a FEDD::BlockMatrix object. Furthermore, the FEDDLib provides an interface to assembly routines, which are generated with AceGen and AceFEM. AceGen and AceFEM are subpackages of the mathematical computation program Mathematica [95]. Element assembly routines can be written down as symbolic expressions in AceFEM. Subsequently, these expressions are evaluated and exported as C code with AceGen. This code is then added to the FEDDLib code and can be used with the existing interface. The combination of AceFEM and AceGen uses automatic differentiation techniques; cf., e.g., [111]. These techniques are well suited for the discretization and linearization of nonlinear problems; e.g, the Jacobian matrix of a nonlinear elasticity problem with a complex material law can be easily computed.

We proceed with the setup of specific problems in the following section. There, we illustrate the assembly processes of a Stokes problem and general nonliner time-dependent problems. However, we first explain the general structure of the abstract interface for discretized PDEs.

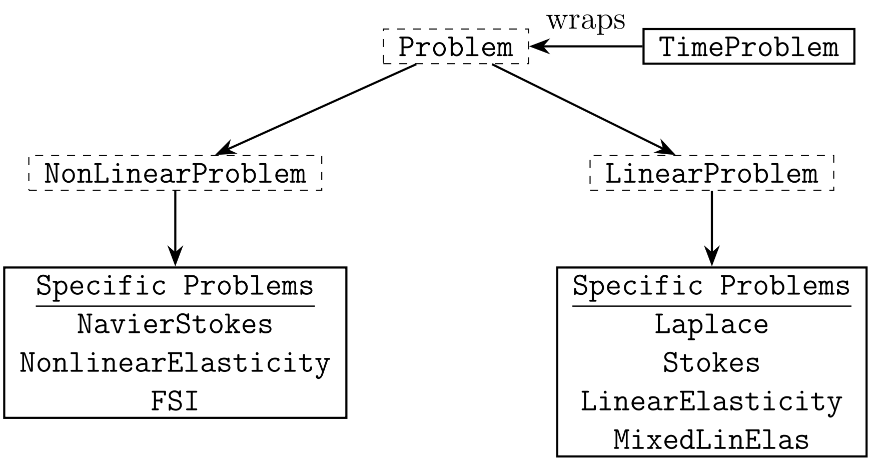


Figure 7.2: Structure of problem classes in the `FEDDLib`. Dashed boxes represent abstract classes with virtual functions. Each specific problem is a separate class.

7.4 Setup of Specific Problems

We use several different classes for the full description of a problem. In the first case, we assume to have a single time-independent PDE which is then discretized with the finite element method provided by the `FEDDLib`. This assembly process can be called for every `FEDD::Problem` with the pure virtual function `assemble()`. A concrete or specific implementation of this method must be provided by the specific problem which is considered. A general overview of abstract and concrete problem classes is depicted in fig. 7.2. After the assembly we have an `FEDD::Matrix` system and right-hand side vector object `FEDD::MultiVector`. Furthermore, we have a `FEDD::MultiVector` for the unknown solution. Actually, we always use the corresponding block linear algebra classes, even if we only consider a 1×1 system. A code example for the assembly and block structure of a Stokes problem with P2–P1 finite elements is given below.

```
/* domV is the velocity domain (e.g, P2 finite elements)
   domP is the pressure domain (e.g, P1 finite elements)
   domVP is a vector of both domains
    fe is a finte element class object */
// create 2x2 block matrix
BlockMatrixPtr_Type F = rcp(new BlockMatrix_Type(2));
MatrixPtr_Type A
  = rcp(new Matrix_Type(domV->getVecFieldMapUnique()));
MatrixPtr_Type BT
  = rcp(new Matrix_Type(domV->getVecFieldMapUnique()));
```

```
MatrixPtr_Type B
  = rcp(new Matrix_Type(domP->getMapUnique()));

fe->assembleLaplace(A,domV);
fe->assembleDiv(BT,B,domVP);

F->addBlock(A,0,0);
F->addBlock(BT,0,1);
F->addBlock(B,1,0);

this->initializeVectors(); /* solution and rhs objects are created after the
   full system matrix is known */
```

The three linear algebra objects for the solution, right-hand side, and problem matrix are all member variables of the abstract `FEDD::Problem` class. We further need to provide access to preconditioners and solvers. This is also handled by the abstract `FEDD::Problem` class, since we only need the matrix for the preconditioner, the matrix, right-hand side, and the solution for the solver. We will pass additional information to the preconditioner, which is done through parameterlists; cf. section 7.5, for the construction of preconditioner and solver objects. With these objects, we can represent and solve problems with a simple structure: linear and time-independent problems.

In case of a nonlinear problem, we need some additional methods and member variables. The abstract virtual class `FEDD::NonLinProblem` is derived from `FEDD::Problem`. Therefore, the `FEDD::BlockMultiVector` right-hand side of the `FEDD::Problem` still holds information of assembled source terms as well as Dirichlet boundary data. Similarly, the matrix object and solution vector of `FEDD::Problem` can be reused. However, the matrix object now corresponds to the linearized problem and we solve for an additional update vector, which is then added to the last solution; cf. section 5.4. Furthermore, we need an additional member variable for the nonlinear residual vector of the nonlinear problem. The virtual function `assemble()` is still used and must be implemented by a concrete nonlinear problem to assemble all constant matrices. W.r.t. specific nonlinear problems the additional pure virtual function `reAssemble()` must be implemented for each concrete nonlinear problem and it must manage the assembly of nonconstant matrices, which change from one nonlinear iteration to the next.

In a last step, we extend linear and nonlinear problems to time-dependent problems. We first highlight that, in many cases, a time-dependent problem

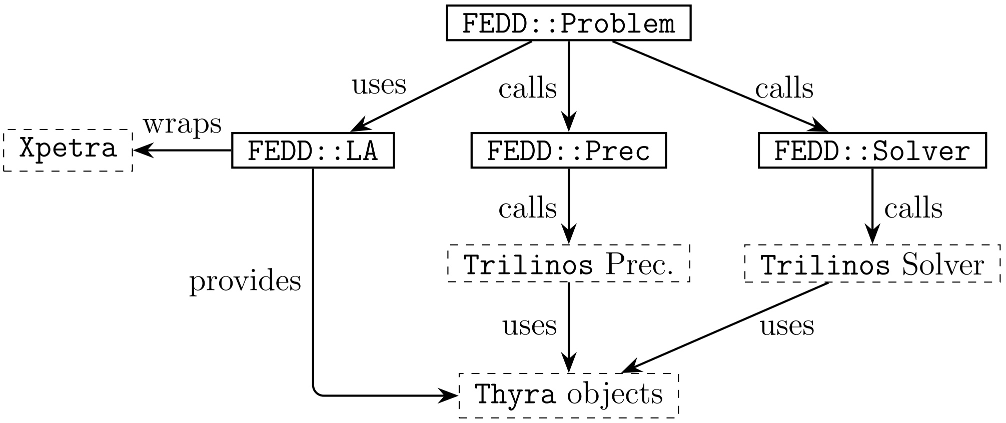


Figure 7.3: Interplay of `FEDDLib` and `Trilinos`. All dashed items are Trilinos objects/classes. The term "uses" should be red synonymously to an extensive use of many methods and functions. The term "calls" should be red synonymously to the use of a single method or function, e.g., `buildPreconditioner()`.

is obtained by adding an appropriate temporal-derivative $\partial u/\partial t$ to the time-independent problem; cf. section 2.4, for the difference between the steady and time-dependent Navier–Stokes equations, or even simpler, the difference between a Poisson problem and the heat equation. We can use the same concrete implementations of time-independent problems for the corresponding time-dependent problems. The class `FEDD::TimeProblem` manages the transition from the steady to the unsteady problem. It assembles corresponding mass matrices and combines them with the matrices of the underlying problem. The combination is based on the desired time-stepping scheme. The `FEDDLib` provides single-step Runge–Kutta methods, cf. section 5.2.1, with Butcher tables of up to four stages. Two of these tables represent the implicit Euler and the Crank–Nicolson method. Furthermore, the multi-step BDF2 method of section 5.2.2 can be used. For hyperbolic second-order problems, we can use the family of Newmark–β methods; cf. section 5.2.4.

Amesos
Direct solvers (Epetra based)
KLU Umfpack (interface) MUMPS (interface) Pardiso (interface) . . .

Amesos2
Direct solvers (Epetra & Tpetra based)
KLU2 Umfpack (interface) MUMPS (interface) Pardiso (interface) . . .

Belos
Iterative solvers (own algebra wrappers)
GMRES Flexible GMRES PCG . . .

Table 7.1: Trilinos direct and iterative solver packages with specific solvers.

Teko
Block preconditioner (Epetra and Tpetra)
LSC SIMPLE(C) PCD (not via Thyra)

NOX
Nonlinear solver (Epetra and Tpetra)
Inexact Newton methods Adaptive forcing terms Globalization techniques Steepest descent Nonlinear CG . . .

Table 7.2: Trilinos block-preconditioner and nonlinear solver package.

7.5 Solvers and Preconditioners

The FEDDLib mainly depends on the Trilinos linear algebra packages. However, several other important Trilinos packages are used in the solution phase. In tables 7.1 and 7.2, an overview of the Trilinos solver packages is given. All of these packages are called via Thyra and only need some additional code to generate the needed Thyra (block) objects. However, to use the package NOX to full efficiency, each specific problem of the FEDDLib must provide additional methods. Moreover, the package Teko for the Navier–Stokes block preconditioners needs some more advanced setup than the direct and iterative solvers of Amesos, Amesos2, and Belos. We discuss the necessary implementations later in this section. A visualization of the simplified connections between FEDDLib and Trilinos is depicted in fig. 7.3. For a description of the solvers and preconditioners, we start with the most outer loop, i.e., the loop over all time steps. The class FEDD::DAESolverInTime calls the assembly methods of all FEDD::

TimeProblem submatrices and manages the combination to these submatrices and right-hand sides for the desired timestepping scheme. After assembly and combination, the FEDD::DAESolverInTime calls the solve() method of FEDD ::TimeProblem and updates the timestepping information. Then, solve() of FEDD::TimeProblem simply calls the method solve() of the underlying FEDD ::Problem. Depending on the type of the underlying problem, i.e, linear or nonlinear, either the linear problem is solved or the desired nonlinear solver is called; cf. section 5.4, for a description of fixed point iterations and Newton's method. The FEDD::NonLinSolver class provides a Thyra interface to the package NOX for the solution of the specific nonlinear problem. In order to use NOX, the abstract class FEDD::NonLinProblem provides methods for the construction of Thyra vector spaces. Furthermore, the concrete FEDD::NonLinProblem must implement appropriate assembly routines for problem matrices, right-hand sides, and conversions to corresponding Thyra objects. Therefore, we can simply call the reAssemble() method, which was presented in the previous section, and use the conversion methods to Thyra objects which are provided by the FEDD::Matrix, FEDD::MultiVector, and corresponding block classes. For block classes, we provide a conversion to two different types of Thyra objects. On the one hand, a FEDD::BlockMatrix can be converted to a Thyra::LinearOpBase with the method getThyraLinOp(), which represents a single monolithic problem matrix. On the other hand, we can get a Thyra::BlockedLinearOpBase object with a call of getThyraLinBlockOp(), which represents the block structure of the problem. Furthermore, NOX calls the desired linear solver, which is specified in a parameterlist. For the efficient iterative solution the arising sparse linear systems, we additionally need to provide a construction method for a preconditioner object for NOX. The setupPreconditioner() method is used for all linear problems which are solved with an iterative method and will be explained in detail at the end of this section.

For the iterative solution of linear systems in NOX or if the underlying problem is linear, we use the Trilinos package Belos. In particular, we use GMRES or flexible GMRES and the desired solver and solver settings are specified with a parameterlist. If NOX is not used, we create the linear solver in the FEDDLib. This is done in the class FEDD::LinearSolver, which manages the setup of all relevant Thyra objects. Here, the central Thyra solver base object is Thyra::

LinearOpWithSolveBase, which has access to all other Thyra objects, such as the problem matrix, the right-hand side, the solution vector, and the preconditioner. The following code shows the setup of a Thyra linear solver with a preconditioner followed by a solve.

```
/* problem is a FEDD::Problem
   solverBuilder is Stratimikos::DefaultLinearSolverBuilder
*/
// initialize solver object
solverBuilder->setParameterList(pListSolver);
RCP<LinearOpWithSolveFactoryBase> lowsFactory
  = solverBuilder->createLinearSolveStrategy("");
RCP<Thyra::LinearOpWithSolveBase> solver
  = lowsFactory->createOp();

// provide Thyra objects
RCP<Thyra::LinearOpBase> thyraMatrix
  = problem->getSystem()->getThyraLinOp();
RCP<Thyra::MultiVectorBase> thyraX
  = problem->getSolution()->getThyraMultiVector();
RCP<Thyra::MultiVectorBase> thyraB
  = problem->getRhs()->getThyraMultiVector();
RCP<Thyra::PreconditionerBase> thyraPrec
  = problem->getPreconditioner()->getThyraPrec();

// pass previously computed preconditioner to solver
Thyra::initializePreconditionedOp( thyraMatrix,
                                   thyraPrec,
                                   solver);
// solve Ax=b
Thyra::solve(solver, thyraB, thyraX);
```

Note that the preconditioner was already computed prior to the solve in the above example. Since the class FEDD::LinearSolver provides a general interface to Thyra, it is also possible to use other solvers by changing the type in the parameterlist; e.g., direct solvers of the Trilinos package Amesos2.

The class FEDD::Preconditioner manages the setup of preconditioner objects and aggregates all necessary information for the setup. Again, we use the corresponding Thyra interfaces of the preconditioner objects. In general, we construct a Thyra::PreconditionerBase object and pass it to the Thyra solver base.

The computation of a preconditioner is performed with the method Thyra ::initializePrec(...) and the preconditioner and corresponding options are specified with a parameterlist:

```
// initialize preconditioner object
solverBuilder->setParameterList(pListPrec);
RCP<Thyra::PreconditionerFactoryBase> precFactory
  = solverBuilder->createPreconditioningStrategy("");
RCP<Thyra::PreconditionerBase> thyraPrec = precFactory->createPrec();

// compute preconditioner
Thyra::initializePrec(precFactory, thyraMatrix, thyraPrec);
```

Prior to the initialization of the preconditioner, we gather data to improve the performance of our FROSch preconditioners. In theory, we can construct many different types of preconditioner objects in the class FEDD::Preconditioner with different settings of the parameterlist; e.g., multigrid preconditioners of the package MueLu and inexact factorizations of the package Ifpack2. However, we highlight that the class FEDD::Preconditioner is designed for the construction of FROSch preconditioner objects. In general, FEDD::Preconditioner sets up all necessary information for a TwoLevelBlockPreconditioner of FROSch. Therefore, we loop over all blocks of the problem and separately extract information for each block. Furthermore, this information is wrapped with a Teuchos::RCP object and written to a Teuchos parameterlist; i.e., we essentially pass the memory location and not the whole data to the parameterlist and the preconditioner. For a TwoLevelBlockPreconditioner, we need the number of d.o.f. per node for each block. Moreover, for the best performance, we use the repeated maps of our domain decomposition. In Chapter 8, we will also present and discuss results for two-level Schwarz preconditioners with GDSW type coarse spaces, which are constructed from unique maps of a domain decomposition; cf. section 6.4.

```
/* domainVec is the array of domains */
ArrayRCP<FROSch::DofOrdering> dofOrderings(numBlocks);
ArrayRCP<unsigned> dofsPerNodeVector(numBlocks);
for (int i = 0; i < numBlocks; i++) {
  RCP<Xpetra::Map<LO,GO,NO> > mapXpetra;
  dofsPerNodeVector[i] = domainVec[i]->getNumDofs();
  // Ordering type, FROSch::DimensionWise or FROSch::NodeWise
  dofOrderings[i] = domainVec[i]->getDofOrdering();
  if (dofsPerNodeVector[i] > 1){ // block of a vector field
    // we extract the underlying Xpetra::Map from the FEDD::Map
    mapXpetra = domainVec[i]->getMapVecFieldRepeated()->getXpetraMap();
    repeatedMaps[i] = mapXpetra;
  }
  else{ // block of a scalar
    mapXpetra = domainVec[i]->getMapRepeated()->getXpetraMap();
    repeatedMaps[i] = mapXpetra;
  }
}
pListPrec->sublist("FROSch").set("Repeated Map Vector",repeatedMaps);
pListPrec->sublist("FROSch").set("DofOrdering Vector",dofOrderings);
pListPrec->sublist("FROSch").set("DofsPerNode Vector",dofsPerNodeVector);
```

If geometric information is needed, e.g., for an Option 2.2 RDGSW coarse space or a coarse space which consists not only of translations, but also of linearized rotations, the coordinates are also passed to `FROSch` via the parameterlist. With the above setup, we can automatically construct a monolithic preconditioner based on the input problem. The `FEDDLib` also provides setup methods of block preconditioners for saddle point problems and a block preconditioning approach for different FSI variants. For the construction of block preconditioners for Stokes and Navier–Stokes problems, we use the `Trilinos` package `Teko`. In particular, we can construct SIMPLE(C) and LSC preconditioners via the `Thyra` interface of `Teko`. As presented in sections 6.2.2 and 6.2.3, we need to approximate inverses of 1×1 blocks for SIMPLE(C) and LSC. This is done with the general setup for monolithic `FROSch` preconditioners, which was explained above. As a consequence, we can use other preconditioners for the approximation of these inverses, as in the general monolithic case. In section 6.5.1, we described the FaCSI preconditioner for FSI; the setup is similar to the `Teko` preconditioner setup. The class `FEDD::PrecOpFaCSI` is derived of a `Thrya::PreconditionerOperator` and implements all necessary application routines. Moreover, all inexact inverses which are needed for the FaCSI preconditioner are constructed separately for each block; i.e., for the fluid, the solid, and the geometry block. Again, we can choose between a monolithic approach and a SIMPLE(C) or LSC block preconditioner for the fluid block. We now have presented all necessary steps to solve complex multi-physics problems with an emphasis on the realization in the parallel, object-oriented C++ software library FEDDLib. In Chapter 8, we are going to present numerical results which were computed with the FEDDLib and FROSch.

7.6 Post-Processing

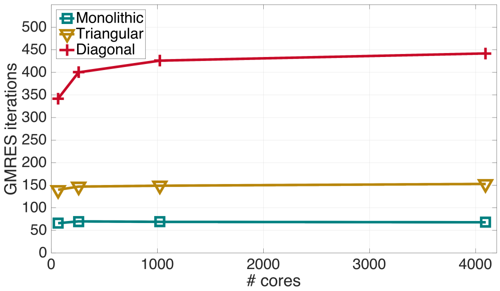
For the post-processing, the FEDDLib wraps Trilinos export methods for HDF5 files. HDF5 [138] is a software package and data format which provides parallel read and write methods for external data. In general, the FEDDLib writes approximate solutions of the discretized PDE problems together with the underlying mesh data to a HDF5 file. Additionally, the FEDDLib creates XDMF files. XDMF uses XML to store meta information of the underlying data, which is stored in separate HDF5 files. This meta data can then be used by visualization tools to view the HDF5 data. For the actual visualizations, we use ParaView [1, 7], and the XDMF files are created accordingly. To generate the XDMF and HDF5 files for a previously computed solution, only a few lines of code are needed:

```
// Export the velocity of a Stokes problem
Teuchos::RCP<ExporterParaView<SC,LO,GO,NO> > exporter =
    Teuchos::rcp(new ExporterParaView<SC,LO,GO,NO>());

exporter->setup( "fluid_velocity" /*name of HDF and XDMF files*/,
                 domainVelocity /* fluid velocity domain*/,
                 "P2" /*velocity discretization*/ );

 MVPtr_Type velocity = stokes.getSolution()->getBlock(0);
 exporter->addVariable(velocity , "v", "Vector");
 exporter->save(0.0); // save for time 0.0
 exporter->closeExporter();
```

8 Numerical Results



In this chapter, we will present results for the model problems presented in the previous chapters. In particular, we will focus on the discussion of results for the Stokes and Navier–Stokes problem, solid problems, and the FSI problems. Before we start with a discussion of the results, we specify the hardware which was utilized in the computation of our parallel results. All parallel results were computed on the magnitUDE supercomputer at University Duisburg-Essen, Germany. A normal node on magnitUDE has 64 GB of RAM and 24 cores (Intel Xeon E5-2650v4 12C 2.2 GHz), interconnected with Intel Omni-Path switches. Intel compiler version 17.0.1 and Intel `MKL` 2017 were used.

Moreover, we use the direct solver `MUMPS` 5.1.1 [2, 3] through the `Amesos` and `Amesos2` interfaces for all direct solves. The local problems are solved in serial mode, whereas in case of coarse solves, the coarse problem is solved in serial or parallel mode. We always consider one subdomain, and thus one local problem, per MPI rank. In particular, we use one MPI rank per core, if not stated otherwise. In order to reduce the computational costs of the RAP product for the coarse problem matrix, we drop entries that are smaller than 10^{-8} in ϕ after the computation of extensions to the interior d.o.f.. Furthermore, if not stated otherwise, the number of MPI ranks for the exact and inexact coarse solution phases is determined by the formula

$$0.5(1 + \min\{\mathrm{NumProcs}, \max\{\mathrm{NumRows}/10\,000,\ \mathrm{NNZ}/100\,000\}\}), \qquad (8.1)$$

where NumRows and NNZ are the numbers of rows and nonzeros of the coarse problem, respectively; cf. [85]. For the coarse matrices of the model problems in this thesis, this formula tends to be dominated by the number of nonzeros.

Two types of unstructured meshes are used during the numerical results. In the first case, we construct unstructured meshes from structured meshes by moving the interior nodes; cf. fig. 8.1. This type of mesh is only used to determine the parallel performance of our preconditioners for unstructured meshes. To partition the unstructured meshes in parallel, we use `ParMETIS` 4.0.3 [100]. The second type of unstructured meshes arises in the discretization of complex geometries. In particular, we load the mesh from a `.mesh`-file, cf. section 7.3, and partition it with `METIS` 5.1.0 [99]. We highlight that the pre-ordering method used in `MUMPS` to approximate the minimum fill-in of the LU decomposition is crucial for saddle point problems. We observed good scalability when `METIS` and `ParMETIS` were chosen for the pre-ordering. With an automatic selection of the pre-ordering algorithm we encounter problems w.r.t. to weak scalability; e.g., it took twice as much time to compute an LU decomposition of a subdomain of a problem with 4 096 total subdomains than it took for a corresponding subdomain of a problem with 1 000 total subdomains, although the subdomain size H/h was kept constant. For the sake of clarity, we characterize all meshes and decompositions by uniform parameters H and h throughout this chapter. Furthermore, N will generally denote the number of subdomains.

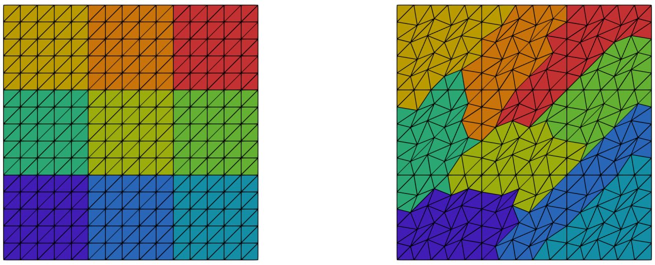


Figure 8.1: Structured (left) and unstructured (right) mesh and decomposition, $H/h = 5$. Taken from [79].

8.1 GDSW for Incompressible Fluid Flow Problems

The results and discussion of this section were already published in [79]. In the following numerical results, we report first level, second level, and total times. The first level time is the sum of construction and application time for the first level of the preconditioner, where the application is performed in every outer GMRES or FGMRES iteration. The total time is the sum of both levels.

The two- and three-dimensional Navier–Stokes problems are solved with Newton and Picard iteration, respectively, We could not observe a good convergence with Newton's method for the three-dimensional problem. Both methods are started with zero initial guess and the stopping criterion is $\|r_{nl}^{(k)}\|/\|r_{nl}^{(0)}\| \leq 10^{-8}$, with $r_{nl}^{(k)}$ being the k-th nonlinear residual. However, we highlight that the first linearization of the advective term is not computed with zero initial guess, but with a vector that satisfies the boundary conditions. Results in later sections are computed with zero initial guess for the linearization of the advective term, i.e., all advective forces are zero and only a Stokes system is solved in the first nonlinear iteration, which improves the convergence of the nonlinear solver. When using a direct solver for the coarse problem, we employ GMRES [128] for the solution of the linear systems. We use the stopping criterion $\|r^{(k)}\| \leq \varepsilon \|r^{(0)}\|$, where $r^{(k)}$ is the k-th unpreconditioned residual and $\varepsilon = 10^{-6}$ is the tolerance for the Stokes problems. For Navier–Stokes problems, we use the tolerance $\varepsilon = 10^{-4}$ for the linear systems. Let us note that in order to reduce computational cost, the recurrence relation in GMRES is evaluated until the stopping criterion is satisfied, and only after that the true residual vector $r^{(k)} = b - \mathcal{F}x^{(k)}$ is computed; this is a standard stopping criterion in the `Trilinos` package `Belos`. In case of inexact coarse solves, we employ FGMRES [129] for the outer iterations; cf. section 8.1.1.3. For both Krylov methods, we use the implementations in the Trilinos package `Belos`. We highlight that all parallel results of this section were computed with an older implementation of `FROSch`, which exclusively used `Epetra`. Many improvements to the overall efficiency have been made in newer versions of the code. For all parallel results of this section, the standard GDSW preconditioner is used.

δ	$1h$							$2h$						
N	4	9	16	25	36	49	64	4	9	16	25	36	49	64
#its. $\hat{\mathcal{B}}_{\mathrm{GDSW}}^{-1}$	25	33	35	37	38	39	40	21	27	29	32	32	32	33
#its. $\hat{\mathcal{B}}_{\mathrm{P2-P1}}^{-1}$	21	25	27	28	28	29	29	18	29	21	22	22	22	22

Table 8.1: Iteration counts for a LDC Stokes problem in two dimensions, varying number of subdomains N and overlap δ with $H/h = 8$ and Taylor–Hood elements. Stopping criterion $\|e^{(k)}\| \leq 10^{-6}$, $e^{(k)} = x^{(k)} - x^*$ with reference solution x^* computed with a direct solver. Taken from [79].

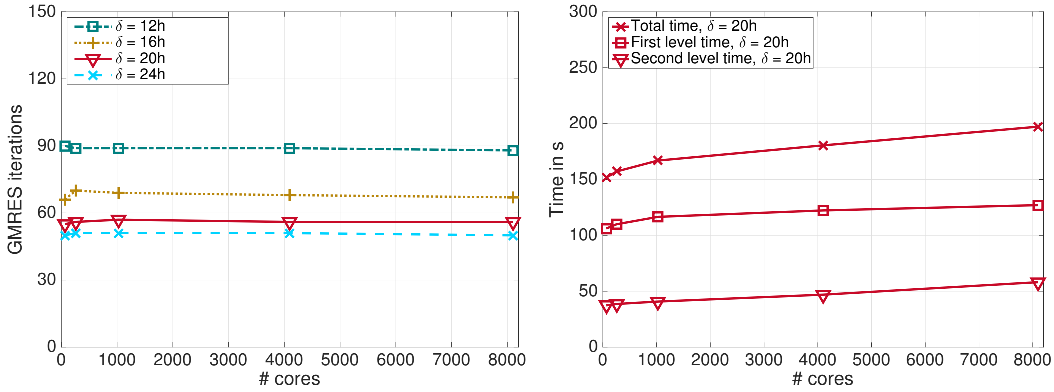


Figure 8.2: Monolithic GDSW preconditioner applied to the two-dimensional LDC Stokes problem with structured mesh and decomposition, $H/h = 160$, P2–P1 finite elements, serial coarse solve. Iteration counts for different levels of overlaps (left). First level, second level, and total time for $\delta = 20h$ (right). Taken from [79].

8.1.1 Numerical Results for Stokes Problems

First, we present a comparison of monolithic Schwarz preconditioners with GDSW and standard Lagrangian coarse spaces; cf. sections 6.3.1 and 6.3.3. Further, parallel scalability results for the monolithic GDSW preconditioner are presented for serial and parallel direct coarse solves and inexact coarse solves.

8.1.1.1 Comparison of Lagrangian and GDSW Coarse Spaces

Iteration counts for the solution of the LDC Stokes problem in two dimensions using monolithic Schwarz preconditioners with GDSW and Lagrangian coarse spaces are given in table 8.1; the results were computed with Matlab [117] on

	# cores	64	256	1 024	4 096	8 100
$\delta = 12h$	Time	174.4 s	183.8 s	187.7 s	206.6 s	224.8 s
	Effic.	87%	82%	81%	73%	67%
$\delta = 16h$	Time	154.7 s	166.4 s	172.9 s	185.5 s	202.6 s
	Effic.	98%	91%	88%	82%	75%
$\delta = 20h$	Time	151.7 s	157.1 s	166.9 s	180.5 s	197.2 s
	Effic.	**100%**	97%	91%	86%	77%
$\delta = 24h$	Time	156.6 s	160.9 s	169.7 s	185.1 s	199.4 s
	Effic.	97%	94%	89%	82%	76%

Table 8.2: Weak scalability for the monolithic preconditioner applied to the two-dimensional LDC Stokes problem, $H/h = 160$, P2–P1 finite elements, coarse problem on one core. Baseline for the efficiency is the fastest time on 64 cores with overlap $\delta = 20h$. Taken from [79].

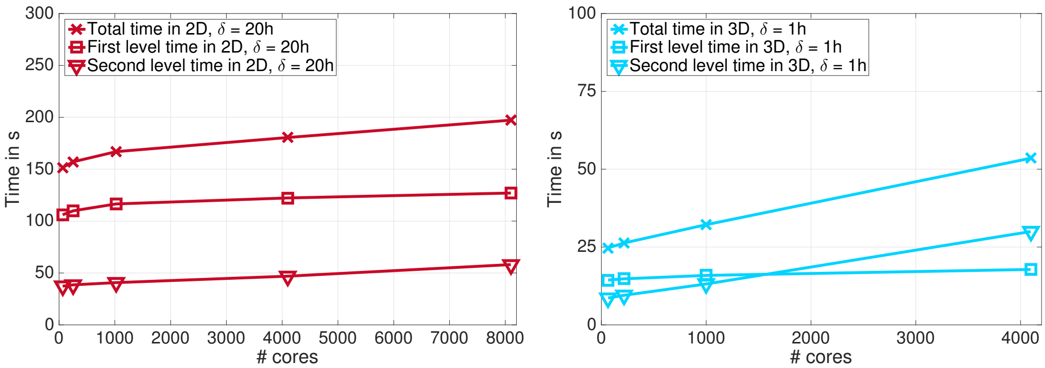


Figure 8.3: First level, second level, and total time for two- and three-dimensional LDC Stokes problems with structured meshes and decompositions using Taylor–Hood elements, $2D$ $H/h = 160$, $3D$ $H/h = 10$. Taken from [79].

structured meshes and domain decompositions. Here, GMRES terminates if the error $e^{(k)} = x^{(k)} - x^*$ of the current iterate $x^{(k)}$ to the reference solution x^* of the linear system satisfies $\|e^{(k)}\| \leq 10^{-6}$; as reference solution we use the one obtained by a direct solver. The results are qualitatively similar to those reported in [103].

We observe that the performance of Lagrangian coarse spaces is slightly better for structured domain decompositions. In contrast, the use of Lagrangian coarse spaces for unstructured decompositions requires additional coarse triangulations and is therefore unfeasible in the context of an algebraic implementation.

overlap	#cores	64	216	1 000	4 096	64	216	1 000	4 096
	NumProc	1	1	1	1	2	9	55	255
$\delta = 1h$	Time	23.5 s	26.4 s	33.4 s	78.4 s	24.7 s	26.3 s	32.2 s	53.5 s
	Effic.	**100%**	89%	70%	30%	95%	89%	73%	44%
$\delta = 2h$	Time	34.7 s	36.9 s	44.3 s	87.9 s	33.1 s	35.9 s	40.7 s	62.5 s
	Effic.	67%	64%	53%	27%	71%	65%	58%	38%

Table 8.3: Weak scalability for the three-dimensional LDC Stokes problem with a structured mesh and decomposition, $H/h = 10$, and Taylor–Hood elements. Different settings for the coarse problem: `MUMPS` used with one MPI rank (serial) and varying number of MPI ranks (parallel) for the coarse problem; 'NumProc' denotes the number of MPI ranks used for the computation of the coarse problem determined by (8.1). Baseline for the efficiency is the fastest time on 64 cores with overlap $\delta = 1h$ and `MUMPS` in serial mode. Taken from [79].

	#cores	64	216	512	1 000
Structured	NumRows	1 117	4 461	11 453	23 437
	NumNonZeros	165 929	830 409	2 352 361	5 086 985
	Vertices	55	251	687	1 459
	Edges	216	900	2 352	4 860
	Faces	288	1 080	2 688	5 400
Unstructured	NumRows	3 432	15 210	40 757	85 214
	NumNonZeros	1 466 640	8 729 840	26 303 223	58 576 054
	Vertices	389	1 932	5 421	11 587
	Edges	677	2 995	7 887	16 468
	Faces	558	2 254	5 809	11 883

Table 8.4: Number of rows and nonzero entries of the GDSW coarse matrix for the LDC Stokes problem in three dimensions. Interface components of structured and unstructured decompositions. Taken from [79].

Hence, we will focus on the performance of our parallel implementation of the new monolithic GDSW preconditioners for the remainder of this section.

8.1.1.2 Parallel Scalability Using an Exact Coarse Solver

In table 8.2 and fig. 8.2, we compare the weak scalability of our monolithic GDSW preconditioner for different levels of overlap δ for the two-dimensional LDC Stokes problem with structured subdomains of constant size $H/h = 160$. We observe

	# cores	64	256	1 024	4 096
	# its.	80	83	82	81
$\delta = 15h$	Time	33.0 s	36.9 s	37.3 s	41.5 s
	Effic.	94%	84%	83%	75%
	# its.	66	69	70	68
$\delta = 20h$	Time	31.8 s	33.5 s	35.4 s	41.4 s
	Effic.	97%	93%	87%	75%
	# its.	56	60	62	60
$\delta = 25h$	Time	31.0 s	33.3 s	35.3 s	38.9 s
	Effic.	**100%**	93%	88%	80%

Table 8.5: Iteration counts and weak scalability for two-dimensional channel Stokes problem, $H/h = 200$, and stabilized P1–P1 elements. Baseline for the efficiency is the fastest time on 64 cores with overlap $\delta = 25h$. Taken from [79].

	# cores	108	500	2 048	4 000
	# its.	44	48	48	49
$\delta = 1h$	Time	24.5 s	27.5 s	32.3 s	38.3 s
	Effic.	**100%**	89%	76%	64%
	# its.	37	47	47	48
$\delta = 2h$	Time	27.9 s	33.1 s	36.3 s	42.1 s
	Effic.	88%	74%	67%	58%

Table 8.6: Iteration counts and weak scalability for three-dimensional channel Stokes problem, $H/h = 20$, and stabilized P1–P1 elements. Baseline for the efficiency is the fastest time on 64 cores with overlap $\delta = 1h$. Taken from [79].

very good parallel scalability for every choice of δ, the best efficiency of 77% from 64 to 8 100 MPI ranks is obtained with $\delta = 20h$. Corresponding detailed timers are depicted in fig. 8.3 (left) for up to 8 000 MPI ranks. The time for the first level stays almost constant, whereas a slight increase of the second level time can be observed due to the increasing size of the coarse problem.

In fig. 8.3 (right), the corresponding times for the three-dimensional LDC Stokes problem with subdomain size $H/h = 10$ are presented. Again, the time for the first level stays constant, whereas a more significant increase of the second level time makes the weak scalability worse compared to the two-dimension case. In particular, the time for the second level exceeds that of the first level when

Coarse solve	partition	structured				unstructured			
	#cores	64	216	1 000	4 096	64	216	512	1 000
Exact	Time	22.8 s	24.7 s	28.1 s	49.1 s	95.2 s	118.9 s	135.1 s	191.1 s
	Effic.	**100%**	92%	81%	46%	98%	78%	69%	49%
	GMRES its.	40	40	38	36	51	52	62	63
$\varepsilon_c = 10^{-1}$	Time	24.3 s	26.4 s	33.0 s	49.4 s	95.0 s	125.9 s	140.1 s	143.7 s
	Effic.	94%	86%	69%	46%	98%	74%	66%	65%
	FGMRES its.	48	56	75	101	62	68	79	97
$\varepsilon_c = 10^{-2}$	Time	23.0 s	25.9 s	28.5 s	39.4 s	93.0 s	120.3 s	134.6 s	149.3 s
	Effic.	99%	88%	80%	58%	**100%**	77%	69%	62%
	FGMRES its.	43	53	52	56	54	58	68	79

Table 8.7: Weak scalability for the three-dimensional LDC Stokes problem, $\delta = 1h$, and Taylor–Hood elements. The coarse problem is solved exactly with `MUMPS` or with GMRES up to a tolerance ε_c. Baselines for the efficiencies are the fastest times on 64 cores for structured and unstructured meshes and decompositions.
Left: structured mesh and decomposition, $H/h = 10$, using 24 MPI ranks per node.
Right: unstructured mesh and decomposition, $H/h = 13$, using 12 MPI ranks per node. Taken from [79].

more than 1 000 processor cores are used; see fig. 8.3 (right). This is a typical behavior of two-level methods due to an increased size of the coarse problem and the fact that a direct solver is applied to solve it. A remedy to improve the parallel efficiency is either to reduce the size of the coarse problem, which will be presented in section 8.2.1, or to introduce a third level [88]. The Three-level approach was successfully applied to a problem matrix resulting from an elliptic PDE in [88]. As can be observed from table 8.3, the parallel scalability can already be improved slightly by using `MUMPS` in parallel mode as the coarse solver; the number of processors used is determined by the formula (8.1). Consequently, we will always use `MUMPS` in parallel mode for the following results with direct coarse solves. Similar results are obtained for the stabilized P1–P1 discretization. In tables 8.5 and 8.6, we present weak scaling results for the two- and three-dimensional channel Stokes problem, respectively.

As can be observed in table 8.4 for the three-dimensional LDC Stokes problem, the dimension of the coarse problem becomes very large for an increasing

Reynolds number	Re = 20				Re = 200			
# cores	64	256	1 024	4 096	64	256	1 024	4 096
Time	78.6 s	78.0 s	86.4 s	94.8 s	87.4 s	84.5 s	98.3 s	104.0 s
Effic.	100%	**102%**	90%	84%	100%	**103%**	89%	84%
Avg. GMRES its.	78.8	77.3	74.5	71.5	101.3	93.3	107.8	99.0
Newton its.	3	3	2	2	4	4	4	4

Table 8.8: Weak scalability for the two-dimensional LDC Navier–Stokes problem using $H/h = 130$, $\delta = 13h$, and Taylor–Hood elements. Baseline for the efficiencies is the time on 64 cores for each Reynolds number. Times are averages over the number of Newton iterations. Taken from [79].

Reynolds number	Re = 20				Re = 200			
# cores	64	216	1 000	4 096	64	216	1 000	4 096
Time	26.0 s	27.8 s	35.0 s	53.7 s	28.3 s	32.0 s	36.7 s	59.0 s
Effic.	**100%**	93%	74%	48%	**100%**	88%	77%	48%
Avg. GMRES its.	61.8	63.8	63.0	62.3	77.4	86.9	89.9	87.1
Newton/Picard its.	4	4	3	3	18	14	11	7

Table 8.9: Weak scalability for the three-dimensional LDC Navier–Stokes problem using $H/h = 10$, $\delta = 1h$, and Taylor–Hood elements. Baseline for the efficiencies is the time on 64 cores for each Reynolds number. Times are averages over the number of Picard iterations. Taken from [79].

number of subdomains. In particular for unstructured domain decompositions, the dimension of the coarse problem and the connectivity of the coarse matrix increase significantly; the dimension can be almost four times and the number of nonzeros more than ten times as large as for the structured case.

Therefore, the factorization of the coarse problems with `MUMPS` becomes very costly. In the next section 8.1.1.3, we investigate inexact coarse solves for GDSW coarse problems, where the coarse problems are only solved up to a tolerance ε_c. A similar approach was used in [93] for two-dimensional, stabilized Stokes and Navier–Stokes problems.

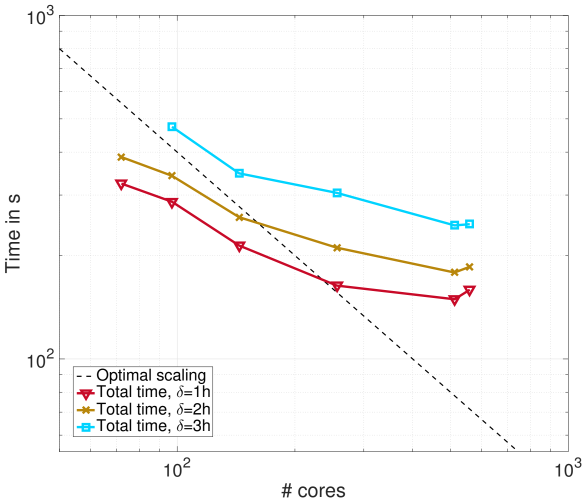


Figure 8.4: Strong scalability for the monolithic GDSW preconditioner applied to the Navier–Stokes benchmark problem with unstructured decompositions and Taylor–Hood elements. Taken from [79].

8.1.1.3 Parallel Scalability Using an Inexact Coarse Solver

Instead of solving the coarse problem with a direct method, we solve $\mathcal{F}_0$ iteratively with GMRES, and we use FGMRES for the outer Krylov iterations. In FGMRES, $\hat{\mathcal{B}}$ is used as a right preconditioner and the k–th residual is $r^{(k)} = b - \mathcal{F}\hat{\mathcal{B}}^{-1}(\hat{\mathcal{B}}x^{(k)})$. The coarse problem is solved with GMRES and $\|r_c^{(k)}\| \leq \varepsilon_c \|r_c^{(0)}\|$, with coarse residual $r_c^{(k)}$ is used as the stopping criterion.

In table 8.7, we compare inexact coarse solves using $\varepsilon_c = 10^{-1}, 10^{-2}$ with exact coarse solves for structured and unstructured decompositions. Here, we use right preconditioned GMRES as the iterative Krylov solver in the case of exact coarse solves, such that the residual and the stopping criterion are the same as for FGMRES. We observe that using an inexact coarse solver can be beneficial for both, structured and unstructured decompositions. For structured decompositions, inexact coarse solves with $\varepsilon_c = 10^{-2}$ start to be more efficient when using more than 1 000 subdomains, whereas for unstructured decomposition, it is already more efficient for more than 512 cores to use inexact solves.

8.1.2 Numerical Results for Navier–Stokes Problems

The scalability results for the LDC Navier–Stokes problems with Reynolds number Re = 1 are comparable to the results for the LDC Stokes problems presented in section 8.1.1 and are therefore not reported. In this section, we use Re = 20 or Re = 200. Inexact coarse solves, as presented in section 8.1.1.3 for the LDC Stokes problem, are not as beneficial for the LDC Navier–Stokes problem. The number of coarse GMRES iterations to reach an acceptable tolerance ε_c is too high and thus an additional preconditioner for the coarse problem would be required; cf., e.g., [93]. Therefore, we only consider exact coarse solvers and left preconditioning.

In table 8.8 and table 8.9, we present weak scalability results for two- and three-dimensional LDC Navier–Stokes problems, respectively. All weak scalability times are averaged over the number of nonlinear Newton or Picard iterations. For Re = 20 and a structured decomposition, we observe a slight reduction in iterations counts for finer meshes, as also observed for the LDC Stokes problem; see, e.g., table 8.7. Furthermore, the number of Newton or Picard iterations may be reduced when the problem is solved on finer meshes. For two-dimensional problems, we observe 84% parallel efficiency from 64 to 4 096 processor cores, whereas the efficiency deteriorates to 48% in three dimensions. Similar to the results of the LDC Stokes, we observed that an overlap of roughly 10 % is most efficient.

Finally, we present results for the flow around a cylinder benchmark with circular cross-section and Re = 20; see fig. 5.3 for the velocity field of the solution. We obtain good values for the drag and lift coefficients, $c_D = 6.178$, $c_L = 0.0095$, for a problem with 2.9 million d.o.f. (2.8 million velocity, 127 k pressure). The domain was partitioned using `METIS`. Both coefficients are computed with the volume integrals given in [96]. Strong scalability results for the monolithic GDSW preconditioner with $\delta = 1h, 2h, 3h$ are presented in fig. 8.4. The good strong scalability deteriorates from 256 to 512 cores, since the additional time to construct and solve the larger coarse problem exhausts the other time savings.

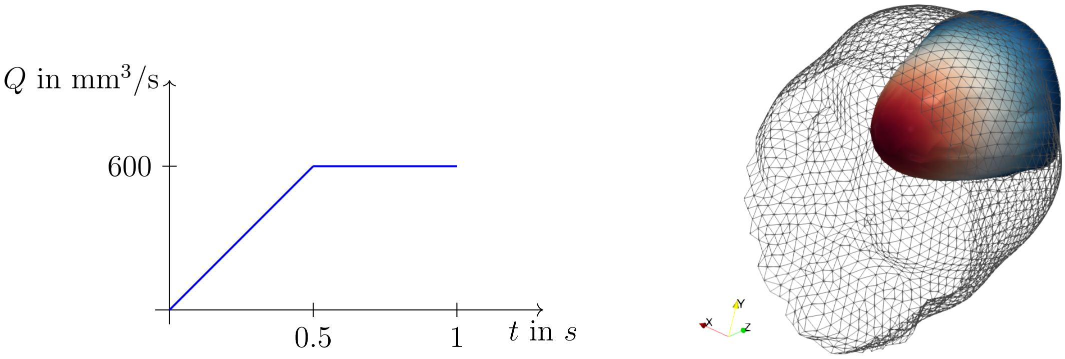


Figure 8.5: Inflow rate for the time-dependent Navier–Stokes problem in a coronary artery (left); see figs. 8.6 and 8.7 for the corresponding mesh and flow field. Magnitude of the solution to a three-dimensional Laplacian problem restricted to the inflow boundary (right). Taken from [80].

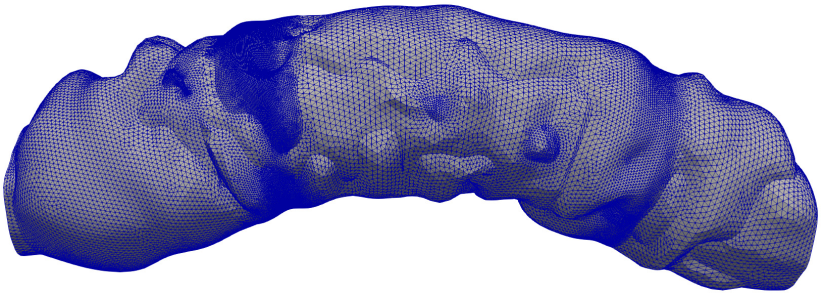


Figure 8.6: Coronary artery volume mesh with 1 032 k tetrahedral elements. Resulting Navier–Stokes systems discretized with P2–P1 elements consists of 4.6 million d.o.f.. Taken from [80].

8.2 Reduced Dimension GDSW Preconditioners for Incompressible Fluid Flow Problems

In this section, we present numerical results of our parallel implementation of the monolithic RGDSW preconditioners. These results have previously been published in [80] and the discussion is along the lines of this article. If not stated otherwise, the coarse problems are solved in parallel mode. To determine the number of MPI ranks, the default setting of `FROSch` is used; cf. eq. (8.1).

Here, we solve the nonlinear steady-state Navier–Stokes problems using Newton's method with zero initial guess. The stopping criterion is $\|r_{nl}^{(k)}\|/\|r_{nl}^{(0)}\| \leq \varepsilon_{\mathrm{nl}}$, with $r_{nl}^{(k)}$ being the k-th nonlinear residual. In the solution of the problems we apply right preconditioned GMRES with the relative stopping criterion $\|r^{(k)}\| \leq \varepsilon\|r^{(0)}\|$, where $\varepsilon = 10^{-6}$ and $\varepsilon = 10^{-4}$ are the tolerances for the Stokes and steady-state Navier–Stokes problems, respectively, and $r^{(k)} = b - \mathcal{F}\hat{\mathcal{B}}^{-1}(\hat{\mathcal{B}}x^{(k)})$ is the k-th residual. Right preconditioned GMRES is always used from this point on, if not stated otherwise.

In this section, we consider the BFS Stokes and BFS Navier–Stokes problems, where the largest problem possesses more than 400 million d.o.f.. Additionally, a time-dependent fluid flow for a realistic arterial geometry is simulated.

The geometry of a straight coronary artery was bent to generate the mesh depicted in fig. 8.6. The original straight coronary artery was used for simulations of stress distributions in the walls of patient-specific atherosclerotic arteries in [10, 26]. We prescribe a parabolic inflow profile with increasing flow rate for 0.5 s; cf. fig. 8.5. After the flow rate of $Q_{\mathrm{steady}} = 600\,\mathrm{mm}^3/\mathrm{s}$ is reached, it is kept constant for further 0.5 s. We apply no-slip and do-nothing boundary conditions at the wall and the outlet of the arterial geometry, respectively. Furthermore, we use a time step length of $\Delta t = 0.01\,\mathrm{s}$, the BDF2 time discretization scheme, and the kinematic viscosity $\nu = 3.0\,\mathrm{mm}^2/\mathrm{s}$. The length of the artery is 12 mm and the inflow diameter is approx. 2 mm. The linearized systems are solved up to a tolerance $\varepsilon = 10^{-6}$. In the following numerical results, we report combined setup times of the first and second level since a distinction is not straightforward for a parallel computation of the levels. Moreover, the identification of the interface is omitted from our setup times.

8.2.1 Comparison of Monolithic GDSW and RGDSW Coarse Spaces

Table 8.10 depicts the comparison of the performance of different (R)GDSW coarse spaces for the monolithic Schwarz preconditioners for the BFS Stokes problem using structured meshes and domain decompositions in three dimensions. In particular, we consider the GDSW coarse space of section 6.3.3 as well as Option 1 and Option 2.2 of the RGDSW coarse space as described in section 6.3.6.

Prec.	#cores	243	1 125	4 608	11 979
GDSW	#its.	84	116	160	202
	setup	14.7 s	19.7 s	36.6 s	73.9 s
	solve	10.7 s	20.9 s	37.5 s	220.0 s
	total	**25.4 s**	40.6 s	74.1 s	293.9 s
RGDSW	#its.	128	117	111	110
	setup	12.6 s	13.7 s	16.2 s	24.0 s
Option 1	solve	14.8 s	15.1 s	21.3 s	36.2 s
	total	27.4 s	**28.8 s**	37.5 s	**60.2 s**
RGDSW	#its.	135	131	121	123
	setup	12.2 s	12.8 s	15.8 s	22.7 s
Option 2.2	solve	15.7 s	16.9 s	17.9 s	39.4 s
	total	27.9 s	29.7 s	**33.7 s**	62.2 s

Table 8.10: Weak scalability results for different coarse spaces: standard, reduced Option 1 & 2.2 applied to the three-dimensional P2–P1 BFS Stokes problem, $H/h = 10$ and $\delta = 1h$. Taken from [80].

A significant reduction of the coarse space dimension is obtained when using the RGDSW coarse spaces. For the largest problem with 11 979 subdomains, the dimension of the coarse problem for the RGDSW coarse space is 40 530 (30 390 velocity and 10 140 pressure basis functions), while it is 305 157 (228 852 velocity and 76 305 pressure basis functions) for the standard GDSW coarse space. Compared to the standard GDSW coarse space, the setup of the reduced dimension Option 1 and Option 2.2 coarse spaces is more than twice as fast for the largest BFS Stokes problem. Surprisingly, for the largest problem, iterations counts for the standard GDSW variant are also higher than for the reduced dimension variants. In the case of elliptic problems, this is typically the opposite; cf. [54, 89]. In comparison to the standard GDSW coarse space, the time-to-solution for the reduced dimension coarse spaces is lower by more than 50 % on 4 608 cores. It is also important to note that Option 1 of the RGDSW coarse space performs better than Option 2.2 with respect to iteration counts; also the compute time is generally lower except for the case of 4 608 cores. This is also different compared to elliptic problems; cf. [54, 89]. In fact, it is beneficial since Option 1 is more algebraic than Option 2.2.

		$\delta = 1h$			$\delta = 2h$		
First level	#cores	243	1 125	4 608	243	1 125	4 608
AS	#its.	272	515	862	180	348	595
	setup	8.7 s	9.1 s	10.3 s	16.6 s	17.6 s	22.5 s
	solve	30.2 s	68.1 s	106.5 s	34.2 s	70.5 s	153.7 s
	total	38.9 s	87.1 s	116.8 s	50.8 s	86.4 s	176.2 s
RAS	#its.	242	460	785	185	366	649
	setup	8.9 s	9.6 s	9.9 s	17.1 s	22.0 s	20.7 s
	solve	26.3 s	52.1 s	89.4 s	34.6 s	72.4 s	163.9 s
	total	35.2 s	61.7 s	99.3 s	51.1 s	94.4 s	184.6 s
SAS	#its.	222	433	740	168	336	591
	setup	8.7 s	9.1 s	10.4 s	16.7 s	17.5 s	19.1 s
	solve	24.6 s	50.2 s	88.1 s	32.0 s	69.3 s	146.9 s
	total	**33.3 s**	**59.3 s**	**98.5 s**	**48.7 s**	**86.8 s**	**166.0 s**

Table 8.11: Comparison of the different monolithic one-level Schwarz preconditioners with $H/h = 10$ applied to the BFS Stokes problem with P2–P1 finite elements: AS, RAS, and SAS; cf. section 6.3.2. Taken from [80].

8.2.2 Restricted and Scaled First Level Variants

Table 8.11 presents weak scalability results for the three first level variants AS, RAS, and SAS with overlap $\delta = 1h, 2h$; cf. section 6.3.2 for definitions of the different variants.

An overlap of $1h$ yields the best performance for all three different approaches. Compared to the standard (AS) and the restricted (RAS) approach, the iteration counts are always lower for the scaled variant (SAS). Therefore, although we save some communication in RAS, SAS performs best for all configurations in this comparison. Moreover, iteration counts for RAS are even higher than for AS for a wider overlap $\delta = 2h$. We will therefore use SAS with overlap $\delta = 1h$ as our default first level from this point on.

8.2.3 Parallel Coupling Strategies for the Levels

The the parallel coupling strategies for the first and the second level, which were introduced in section 6.3.7, will be applied to further improve the performance of the simulations. We present parallel scalability results comparing sequential

	#cores	243	1 125	4 608	11 979
Coupling	#its.	120	114	105	108
sequential additive	setup	18.6 s	18.8 s	21.4 s	29.4 s
	solve	17.6 s	19.2 s	20.5 s	27.6 s
	total	36.2 s	38.0 s	41.9 s	57.3 s
parallel additive (+1 core)	setup	17.7 s	17.9 s	19.8 s	27.9 s
	solve	17.1 s	19.0 s	17.6 s	21.0 s
	total	34.8 s	36.9 s	37.4 s	**48.9 s**
	#its.	89	90	84	91
multiplicative	setup	17.6 s	18.1 s	19.1 s	29.6 s
	solve	14.7 s	15.8 s	16.9 s	23.5 s
	total	**32.3 s**	**33.9 s**	**36.0 s**	53.1 s

Table 8.12: Weak scalability results for monolithic preconditioners with SAS first level applied to the three-dimensional BFS Stokes problem with P2–P1 discretization; $H/h = 11$, $\delta = 1h$, and RGDSW Option 1. We always use one core for the solution of the coarse problem; therefore, for the parallel additive coupling, we allocate one additional core for the solution of the coarse problem. Taken from [80].

additive, parallel additive, and multiplicative coupling in table 8.12. There, we use one core for the solution of the coarse problems, and we allocate one additional core for the solution of the coarse problem in the parallel approach to obtain the same domain decompositions for all three approaches. The hybrid version of the two-level preconditioner performs better than the sequential additive version, since iteration counts are lower. Specifically, we save more than 7 % in total computing time on 11 979 cores. However, more than 14 % computing time can be saved by using the parallel additive coupling. The performance of the parallel additive coupling with a varying number of cores for the solution of the coarse problem is depicted in table 8.13. The increase of the number of cores from 1 to 10 yields a further speedup by more than 10 %. We anticipate an increasing advantage of the parallel additive approach if a larger configuration with more cores is used.

	#cores	243	1 125	4 608	11 979
Coupling	#its.	120	114	105	108
parallel additive (+1 core)	setup	17.7 s	17.9 s	19.8 s	27.9 s
	solve	17.1 s	19.0 s	17.6 s	21.0 s
	total	**34.8 s**	**36.9 s**	**37.4 s**	48.9 s
parallel additive (+5 cores)	setup	17.6 s	18.5 s	20.0 s	25.3 s
	solve	18.8 s	19.0 s	20.1 s	20.8 s
	total	36.4 s	37.5 s	40.1 s	46.1 s
parallel additive (+10 cores)	setup	17.3 s	18.6 s	18.1 s	22.8 s
	solve	18.7 s	18.9 s	19.5 s	21.0 s
	total	36.0 s	37.5 s	37.6 s	**43.8 s**

Table 8.13: Weak scalability results for monolithic preconditioners with SAS first level and parallel additive coupling applied to the three-dimensional BFS Stokes problem; $H/h = 11$, $\delta = 1h$, and RGDSW Option 1. We allocate additional cores for the solution of the coarse problem (in brackets). Taken from [80].

8.2.4 Recycling Strategies

The information of previous Newton iterations or time steps can be reused to save computing time. All index sets corresponding to the overlapping subdomains and the interface components, are typically constant over all iterations and the reuse is straightforward. The nonzero pattern typically stays the same, while the entries of the system matrix change during Newton and time iterations, which is why the symbolic factorizations of the local matrices and the global coarse matrix could be reused. To save compute time we dropped small matrix entries in ϕ before the computation of the coarse RAP product. Therefore, the nonzero pattern of the coarse matrix is changed and not constant. However, we save more time in the computation of the coarse RAP product by dropping small matrix entries in ϕ than by reusing the symbolic factorizations. The reuse of the symbolic factorizations of the local overlapping matrices $\mathcal{F}_i$ and interior subdomain matrices $\mathcal{F}_{II}^{(i)}$ used in the saddle point extensions is denoted as *Symbolic Factorization* (SF) recycling strategy. The symbolic factorizations require between 15 % and 20 % of the total factorization time for overlapping subdomain matrices $\mathcal{F}_i$ for a Navier–Stokes problem with $H/h = 8$ and $\delta = 1h$. The numeric factorization and the symbolic factorization require approximately 3.6 s and 0.7 s, respectively.

Recyling strategy	#cores	243	1 125	4 608
-	#its.	155.25 (4)	158.3 (3)	149.0 (3)
	setup	28.4 s	23.7 s	30.1 s
	solve	40.5 s	37.7 s	42.2 s
	total	68.9 s	61.4 s	72.3 s
SF	#its.	155.25 (4)	158.3 (3)	149.0 (3)
	setup	24.1 s	20.3 s	25.6 s
	solve	40.7 s	35.0 s	42.1 s
	total	64.8 s	55.3 s	67.7 s
SF + CB	#its.	157.0 (4)	159 (3)	151.0 (3)
	setup	18.7 s	16.7 s	21.8 s
	solve	40.7 s	35.1 s	42.4 s
	total	**59.4 s**	**51.8 s**	**64.2 s**
SF + CB + CM	#its.	165 (4)	175.3 (3)	170.3 (3)
	setup	18.0 s	15.4 s	19.4 s
	solve	42.8 s	38.0 s	46.3 s
	total	60.8 s	53.4 s	65.7 s

Table 8.14: Weak scalability results for monolithic preconditioners with coarse space recycling applied to the BFS Navier–Stokes problem; $\varepsilon_{\text{nl}} = 10^{-6}$, $H/h = 8$, $\delta = 1h$, $\nu = 0.01$, $Re = 200$, and RGDSW Option 1. The numbers in parentheses denote the number of Newton iterations. SF, CB, and CM denote the reuse of the symbolic factorizations for the matrices $\mathcal{A}_i$ and $\mathcal{A}_{II}^{(i)}$, of the coarse basis ϕ, and of the coarse matrix $\mathcal{A}_0$, respectively. The times for the solution of a Stokes problem for the initial guess are included. Taken from [80].

The effect is similar for interior subdomain matrices $\mathcal{F}_{II}^{(i)}$. We reuse the symbolic factorizations all following results in this section. Furthermore, the reuse of the numeric factorizations of the matrices $\mathcal{F}_i$ is not a viable approach as it yields significantly worse iteration counts. For the coarse level, we propose two recycling strategies: the *Coarse Basis* (CB) recycling strategy according to which the the coarse basis ϕ is reused while the coarse RAP product (6.8) is recomputed, and the *Coarse Matrix* (CM) recycling strategy according to which the coarse matrix $\mathcal{F}_0$ is reused, which saves the computation time of the coarse RAP product as well as the time of the coarse factorization.

A comparison of a complete recomputing of the preconditioner and three different combinations of the recycling strategies is presented in table 8.14 for a steady-state Navier–Stokes problem. Clearly, the SF approach should always be preferred to the recomputation of the whole preconditioner. Furthermore, for larger numbers of subdomains, the combination SF+CB is most efficient, while the scalability deteriorates for the combination SF+CB+CM. This is due to the fact that a recycled basis can still represent the nullspace of the operator, whereas a recycled coarse problem might be a bad approximation of the current linearized problem. Specifically, we reach 69 % efficiency from 243 to 4 608 cores with basis recycling. In the next subsection we present further comparisons of the proposed recycling strategies for time-dependent problems, where we observe a substantial increase in efficiency.

8.2.5 Speedup for a Time-Dependent Navier–Stokes Problem

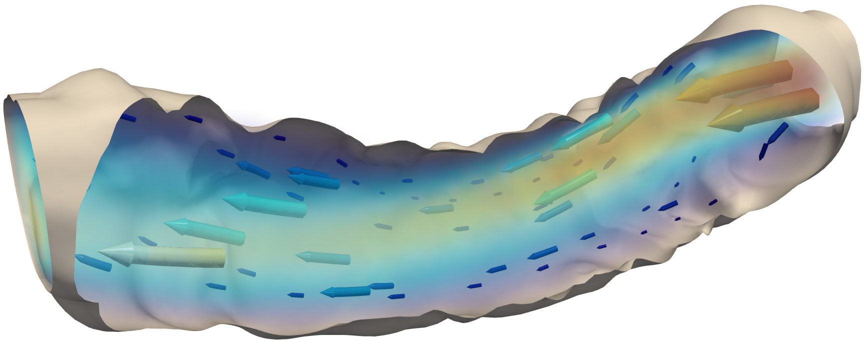


Figure 8.7: Solution of the time-dependent Navier–Stokes problem at time 1.0 s for the coronary artery; cf. section 5.1.2. Taken from [80].

Due to the added mass matrix, time-dependent problems are much better conditioned than their steady-state counterparts for small time steps. Certain problems do not even require a coarse space for numerical scalability; see, e.g., [29], for the special case of a symmetric parabolic problem in two dimensions. In general, for the time-dependent Navier–Stokes problem studied in this section, we prefer to use an RGDSW coarse space due to the significantly lower iteration counts. For the following time-dependent Navier–Stokes problem, we consider the realistic coronary artery geometry; cf. figs. 8.6 and 8.7. We require on average 82.8 iterations per time step for the one level preconditioner, and only 38.5 iterations

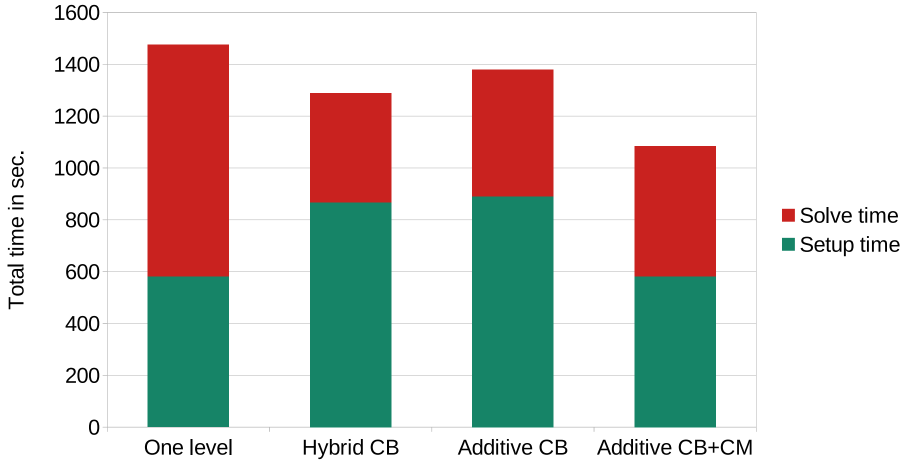


Figure 8.8: Timings for the simulation of the coronary artery with the time-dependent Navier–Stokes problem with 4.6 million d.o.f. solved on 240 cores. SAS for the first level with $\delta = 1h$. Hybrid and additive two-level preconditioners with different recycling strategies. Taken from [80].

for the additive two-level method with coarse basis and coarse matrix recycling. With the recycling methods presented in section 8.2.4, the additional time for the setup of the second level is neglectable.

Figure 8.8 shows the comparision of a one-level SAS preconditioner with two-level hybrid and additive SAS preconditioners. The coarse basis recycling (SF+CB) and the full recycling (SF+CB+CM) for the additive preconditioner and basis recycling for the hybrid preconditioner are compared. Full recycling (SF+CB+CM) for the hybrid preconditioner is not presented, since we did not observe good convergence. This is due to the larger effect of the coarse operator when coupled in a multiplicative way. For the additive two-level preconditioner with full recycling, Only 7.7 s are spent for the construction of second level, for the additive two-level preconditioner with full recycling. 82.0 s of 480.5 s total computing time are spent during the application of the coarse level. We observed no advantage w.r.t total computing time when a full reset of the recycled operators, i.e., recomputing the coarse basis functions ϕ and the coarse matrix, after a certain number of time steps, was used. In fig. 8.9, we present strong scaling results for the realistic artery geometry. All simulations with this realistic artery were carried out with LifeV [69], which is based on the `Epetra` linear algebra.

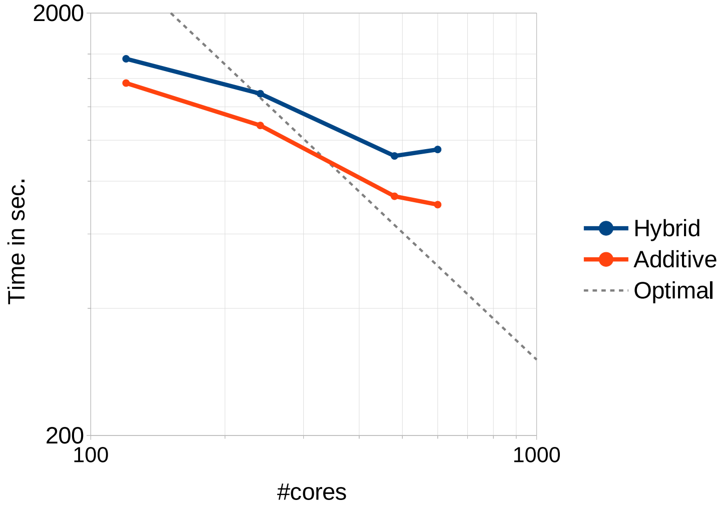


Figure 8.9: Strong scaling results for time-dependent Navier–Stokes problem with 4.6 million d.o.f. for the realistic coronary artery. SAS for the first level with $\delta = 1h$. Hybrid two-level preconditioner with coarse basis recycling and additive two-level preconditioner with coarse basis and coarse matrix recycling. Simulation to final time of 1.0 s with Δt =0.01 s; cf. section 5.1.2, for the description of the model problem. Taken from [80].

In particular, we used the time-dependent Navier–Stokes implementation with extrapolation of the convection, BDF2 for the temporal discretization, and P2–P1 mixed finite elements for the spatial discretization. A problem with 4.6 million d.o.f. using a two-level additive RGDSW preconditioner with full recycling (SF+CB+CM) and a two-level hybrid RGDSW preconditioner with basis recycling (SF+CB) is solved. Both preconditioners achieve similarly good scaling results from 120 to 480 cores. As opposed to the additive preconditioner, the speedup of the hybrid preconditioner stagnates for more than 480 cores. Therefore, we prefer the additive preconditioner for this configuration due to the lower total computing time (between 5 % and 25 %).

By combining the RGDSW Option 1 coarse space, the scaled first level (SAS), and the multiplicative coupling of the levels, we achieve a reduction of the time-to-solution by 60 % compared to the previous results obtained with an additive two-level GDSW coarse space for a BFS Stokes problem solved on 4 608 cores; cf. fig. 8.10 for timings. For time-dependent and nonlinear problems, we can

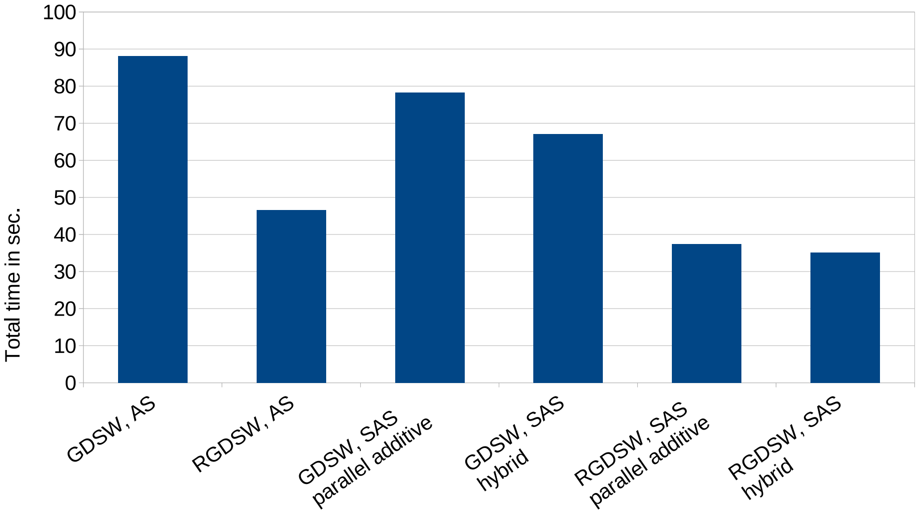


Figure 8.10: Total time for the three-dimensional BFS Stokes problem with P2–P1 finite elements, $H/h = 11$, and $\delta = 1h$ on 4 608 cores. Improved preconditioner versions use SAS for the first level; cf. section 6.3.2. The improved (R)GDSW preconditioners with additive coupling between the levels use parallel coarse solves with 10 dedicated MPI ranks for the coarse problem; cf. section 6.3.7. Taken from [80].

further recycle the symbolic factorizations, the coarse basis, coarse basis, and the coarse matrix. For the best configuration of the GDSW coarse spaces a reduction of 75 % of total time was achieved, solving the first 10 time steps of the coronary artery problem compared to the previous implementation using the GDSW preconditioner. Similarly, a reduction of 85 % was achieved with the best configuration for the RGDSW coarse spaces; cf. fig. 8.11.

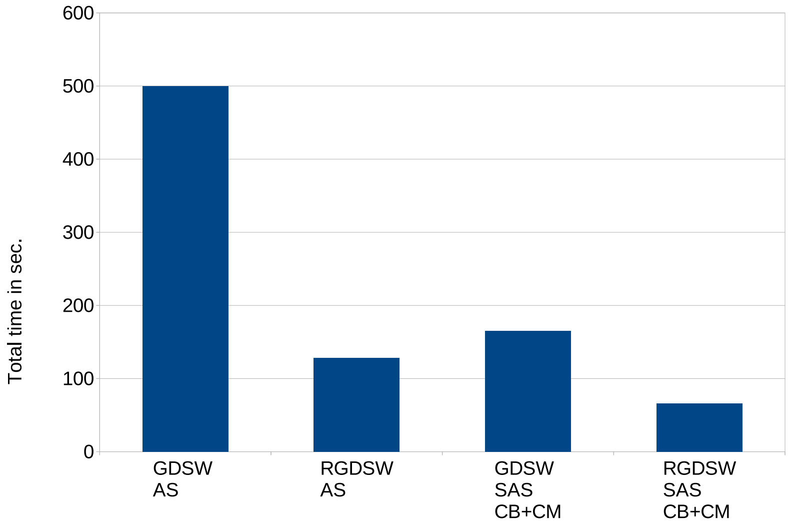


Figure 8.11: Speedup for the time–dependent Navier–Stokes problem on 240 cores. Simulation of 0.1 s of the ramp phase, $\delta = 1h$. Improved preconditioner versions use SAS for the first level and full recycling; cf. section 6.3.2 and section 8.2.4, respectively. Taken from [80].

8.3 Comparison of Block and Monolithic Preconditioners for Incompressible Fluid Flow Problems

8.3.1 Standard Block Preconditioners

Similar to section 8.1.1, we begin with a comparison of block and monolithic preconditioners for two types of coarse spaces, i.e., for GDSW and standard Lagrangian coarse spaces. Here, we use the standard block-diagonal and -triangular preconditioners with a pressure mass matrix approximation of the Schur complement; cf. section 6.2.1. All parallel results in this section are based on the older `FROSch` implementation, which exclusively used `Epetra`.

Iterations counts for GDSW and Lagrangian coarse spaces are given in table 8.15. The LDC Stokes problem in two dimensions is solved using a Matlab implementation on a structured mesh and decomposition. For both types of coarse spaces, results for the iteration of the block-diagonal, -triangular, and monolithic preconditioners are presented. GMRES terminates if the error $e^{(k)} = x^{(k)} - x^*$ of

		GDSW coarse space			Lagrangian coarse space		
δ	N	$\hat{\mathcal{B}}_M^{-1}$	$\hat{\mathcal{B}}_T^{-1}$	$\hat{\mathcal{B}}_D^{-1}$	$\hat{\mathcal{B}}_M^{-1}$	$\hat{\mathcal{B}}_T^{-1}$	$\hat{\mathcal{B}}_D^{-1}$
$1h$	4	25	62	120	21	61	122
	9	33	83	165	25	80	154
	16	35	97	186	27	88	166
	25	37	102	197	28	89	170
	36	38	104	207	28	90	173
	49	39	106	213	29	90	173
	64	40	108	219	29	90	173
$2h$	4	21	71	138	18	73	142
	9	27	92	166	20	90	170
	16	29	98	177	21	93	174
	25	31	101	188	22	95	177
	36	32	103	196	22	95	179
	49	32	105	205	22	95	180
	64	33	107	211	22	96	180

Table 8.15: Iteration counts for the LDC Stokes problem in two dimensions discretized with P2–P1 finite elements, varying number of subdomains N, overlap δ, and fixed $H/h = 8$. Stopping criterion $\|e^{(k)}\| \leq 10^{-6}$, $e^{(k)} = x^{(k)} - x^*$ with solution x^* obtained by a direct solver. Preconditioners are monolithic (M), block-triangular (T), and block-diagonal (D) two-level Schwarz methods with GDSW or Lagrangian coarse spaces and standard a first level (AS).

the current iterate $x^{(k)}$ to the solution x^*, which is computed with a direct solver, satisfies $\|e^{(k)}\| \leq 10^{-6}$. The results of table 8.15 are qualitatively similar to those obtained in [105] for the Lagrangian coarse space. The block-diagonal preconditioners need twice as many iterations as the block-triangular preconditioners. Surprisingly, iteration counts are lower for the block-triangular GDSW preconditioner as well as for the block-triangular and block-diagonal preconditioners with Lagrangian coarse space if the smaller overlap $\delta = 1h$ is used, instead of $\delta = 2h$. A similar behavior was observed in [105] for Lagrangian coarse spaces. As expected, the monolithic preconditioners provide the lowest iterations counts.

Weak scalability results for the three different preconditioners with GDSW coarse spaces are given in table 8.16. In particular, only a one-level approximation of the pressure mass matrix is used. We observe that the monolithic preconditioner is the most efficient one for the two-dimensional LDC Stokes problem.

Preconditioner	#cores	64	256	1 024	4 096
Monolithic	#its.	66	70	69	68
	Total time	154.7 s	170.0 s	175.8 s	188.7 s
	Effic.	**100**%	91%	88%	82%
Triangular	#its.	137	143	146	150
	Total time	309.4 s	329.1 s	359.8 s	396.7 s
	Effic.	49%	44%	43%	37%
Diagonal	#its.	342	400	426	442
	Total time	736.7 s	859.4 s	966.9 s	1105.0 s
	Effic.	21%	18%	16%	14%

Table 8.16: Weak scalability for block and monolithic two-level GDSW preconditioners with a standard first level (AS) applied to the two-dimensional LDC Stokes problem, $H/h = 160$, $\delta = 16h$, coarse problem on one core. Baseline for the efficiency is the fastest time on 64 cores with the monolithic preconditioner.

This is due to the much lower number of iterations compared to the slightly higher setup time. Compared to the block-diagonal preconditioner, the block-triangular preconditioner only needs one additional matrix-vector product and one additional dot product. Apart from this, the setup time of both block preconditioners is the same. Thus, the block-triangular preconditioner should be preferred to the block-diagonal preconditioner due to the lower number of iterations.

8.3.2 SIMPLE, LSC, and Monolithic Preconditioners

In this section, we present parallel results for SIMPLE and LSC block preconditioners and compare them to the monolithic preconditioner with RGDSW coarse space. In particular, we will discuss results for different (R)GDSW variants as approximations for the block inverses in SIMPLE and LSC. The `Trilinos` package `Teko` is used for the general setup and application of SIMPLE and LSC. The underlying block approximations are constructed with `FROSch`. All RGDSW preconditioners use the more algebraic Option 1, where no geometric information is needed and 5 ranks are used for all coarse solvers. For problems with low Reynold's, the pressure mass matrix approach can compete with SIMPLE and LSC. However, the performance of a mass matrix approach quickly deteriorates if the Reynolds number is increased. Therefore, we use SIMPLE and LSC in the solution of a Navier–Stokes problem with a Reynolds number of $Re = 200$. In particular, we will discuss results for the BFS Navier–Stokes problem in three dimensions. The solution of nonlinear problems is carried out with an inexact Newton method and adaptive forcing term of type 2; cf. section 5.4. In particular, the following parameters for the forcing term were used: maximum forcing term $\eta_{\text{max}} = 10^{-3}$, minimum forcing term $\eta_{\text{min}} = 10^{-8}$, and initial forcing term $\eta_{\text{init}} = 10^{-3}$. Moreover, the relative tolerance for Newton's method is $\varepsilon_{\text{nl}} = 10^{-8}$.

Results of different coarse space configurations for SIMPLE are depicted in table 8.17. The same coarse space configurations are used for the LSC preconditioner and the results are given in table 8.18. For approximations of subproblems, we use GDSW and RGDSW with and without coarse basis recycling. Similar to the result of recycling strategies for the monolithic RGDSW preconditioner reported in table 8.14, a coarse matrix recycling (CM) does not yield a further decrease in total computation time compared to the coarse basis recycling (CB) approach. Therefore, these results are not reported here. Additionally, all preconditioners reuse symbolic factorizations; cf. section 8.2.4. Basis recycling is always beneficial for the steady BFS Navier–Stokes problem since it reduces the total computation time compared to the corresponding preconditioners without recycling. A comparison of the different (R)GDSW and recycling combinations for SIMPLE and LSC shows us that SIMPLE requires less total computation time than LSC. This is mainly due to the lower number of iterations. We need to highlight that there are more efficient versions of LSC which might perform better for the BFS Navier–Stokes problem. It is possible to incorporate boundary information to the commutator which improves the performance [63, 64].
Furthermore, it is important to note that a one-level SAS preconditioner for the approximation of the Schur complement in LSC is not a good choice. The setup is only slightly faster, but the increase in iterations counts is too large. However, using only a one-level SAS approach for the Schur complement approximation in SIMPLE is beneficial. Suprisingly, a one-level preconditioner for the Schur complement is even better than a two-level preconditioner w.r.t. iteration counts. The most efficient combination is a SIMPLE block preconditioner with the following block approximations: for the fluid block F, a two-level RGDSW preconditioner

		SIMPLE		
Prec.'s for blocks	N	243	1125	4608
F: GDSW S: one-lvl SAS	avg. #its.	132.4(5)	142.0(5)	149.4(5)
	setup	53.4 s	70.6 s	124.6 s
	solve	61.1 s	75.9 s	126.1 s
	total	114.5 s	146.5 s	250.7 s
F: GDSW CB S: one-lvl SAS	avg. #its.	135.4(5)	144.6(5)	153.6(5)
	setup	36.0 s	45.3 s	99.9 s
	solve	60.7 s	74.4 s	127.9 s
	total	**96.7** s	119.7 s	227.8 s
F: RGDSW S: one-lvl SAS	avg. #its.	152.2(6)	159.6(5)	168.2(5)
	setup	50.5 s	44.5 s	53.4 s
	solve	80.6 s	77.3 s	98.4 s
	total	131.1 s	121.8 s	151.8 s
F: RGDSW CB S: one-lvl SAS	avg. #its.	155.7(6)	161.2(5)	171.4(5)
	setup	39.4 s	35.4 s	37.9 s
	solve	84.4 s	77.2 s	91.7 s
	total	123.8 s	**112.6** s	**129.6** s
F: GDSW S: GDSW	avg. #its.	150.6(5)	183.2(5)	207.4(5)
	setup	54.9 s	78.7 s	159.0 s
	solve	72.3 s	103.0 s	206.2 s
	total	127.2 s	181.7 s	365.2 s
F: GDSW CB S: GDSW CB	avg. #its.	151.8(5)	185.8(5)	212.8 (5)
	setup	37.2 s	53.7 s	159.0 s
	solve	70.7 s	102.7 s	208.7 s
	total	107.9 s	156.4 s	367.7 s
F: RGDSW S: RGDSW	avg. #its.	156.8(5)	194.4(5)	218.4(5)
	setup	43.2 s	45.2 s	55.9 s
	solve	71.2 s	96.1 s	133.7 s
	total	114.4 s	141.3 s	189.6 s
F: RGDSW CB S: RGDSW CB	avg. #its.	157.4(5)	197(5)	221.8(5)
	setup	32.0 s	34.6 s	42.7 s
	solve	70.6 s	96.6 s	134.7 s
	total	102.6 s	131.2 s	177.4 s

Table 8.17: Results for SIMPLE preconditioner for the three-dimensional steady BFS Navier–Stokes problem. P2–P1, $H/h = 9$, and overlap $\delta = 1h$. All first level preconditioners use SAS. Number of Newton iterations in parenthesis.

with coarse basis recycling (CB) is used, and a one-level SAS preconditioner for the Schur complement S_{SIMPLE} is employed.

		LSC		
Prec.'s for blocks	N	243	1125	4608
F: GDSW S: one-lvl SAS	avg. #its.	261.8(6)	448.2(5)	930.0(5)
	setup	64.6 s	62.7 s	120.7 s
	solve	158.4 s	258.1 s	778.9 s
	total	223.0 s	320.8 s	899.6 s
F: GDSW CB S: one-lvl SAS	avg. #its.	259(6)	462.6(5)	922.8(5)
	setup	42.0 s	45.5 s	97.8 s
	solve	154.1 s	261.6 s	758.0 s
	total	196.1 s	307.1 s	855.8 s
F: RGDSW S: one-lvl SAS	avg. #its.	291.3(6)	492.3(6)	951.8(5)
	setup	49.5 s	53.7 s	49.5 s
	solve	170.2 s	315.2 s	562.4 s
	total	219.7 s	368.9 s	611.9 s
F: RGDSW CB S: one-lvl SAS	avg. #its.	291.0(6)	492.2(6)	947.8(5)
	setup	37.2 s	39.9 s	38.2 s
	solve	168.9 s	310.6 s	555.0 s
	total	206.1 s	350.5 s	593.2 s
F: GDSW S: GDSW	avg. #its.	194.7(6)	259.2(5)	377.2(5)
	setup	64.1 s	70.0 s	170.0 s
	solve	124.9 s	165.3 s	421.7 s
	total	189.0 s	235.3 s	591.7 s
F: GDSW CB S: GDSW CB	avg. #its.	196.8(6)	261(5)	384.6(5)
	setup	42.4 s	52.0 s	147.4 s
	solve	123.9 s	164.1 s	425.4 s
	total	166.1 s	216.1 s	572.8 s
F: RGDSW S: RGDSW	avg. #its.	201.2(6)	279.8(5)	406.0(5)
	setup	50.4 s	45.4 s	52.8 s
	solve	122.6 s	155.9 s	260.2 s
	total	173.0 s	201.3 s	**313.0** s
F: RGDSW CB S: RGDSW CB	avg. #its.	199.7(6)	280(5)	409.2(5)
	setup	37.8 s	34.5 s	43.7 s
	solve	120.4 s	154.1 s	278.0 s
	total	**158.2** s	**188.6** s	321.7 s

Table 8.18: Results for LSC preconditioner for the three-dimensional steady BFS Navier–Stokes problem. P2–P1, $H/h = 9$, and overlap $\delta = 1h$. All first level preconditioners use SAS. Number of Newton iterations in parenthesis.

In table 8.19, we compare the most efficient SIMPLE preconditioner with the monolithic RGDSW CB preconditioner. This monolithic preconditioner required, in general, the lowest total computation time of all monolithic variants for the

Disc.	Prec.	N	243	1 125	4 608
P1–P1	Monolithic RGDSW CB	avg. #its.	48.6(5)	51.2(5)	51.8(5)
		setup	49.7 s	52.8 s	61.7 s
		solve	32.7 s	36.7 s	41.7 s
		total	**82.4** s	**89.5** s	**103.4** s
	SIMPLE RGDSW CB	avg. #its.	166.8(5)	196.6(5)	199.2(5)
		setup	37.1 s	39.2 s	43.4 s
		solve	99.4 s	126.4 s	140.4 s
		total	136.5 s	165.6 s	183.7 s
P2–P1	Monolithic RGDSW CB	avg. #its.	108.2(5)	103.6(5)	92.0(5)
		setup	39.6 s	43.0 s	49.0 s
		solve	52.0 s	54.9 s	54.7 s
		total	**91.6** s	**97.9** s	**103.7** s
	SIMPLE RGDSW CB	avg. #its.	155.7(6)	161.2(5)	171.4(5)
		setup	39.4 s	35.4 s	37.9 s
		solve	84.4 s	77.2 s	91.7 s
		total	123.8 s	112.6 s	129.6 s
Q2–P1disc	Monolithic GDSW CB	avg. #its.	21.5(5)	25.6(4)	28.4(5)
		setup	41.9 s	51.4 s	103.6 s
		solve	8.4 s	10.5 s	51.4 s
		total	**50.3** s	**61.9** s	155.0 s
	SIMPLE RGDSW CB	avg. #its.	144.4(5)	164.4(5)	156(5)
		setup	30.3 s	32.5 s	35.8 s
		solve	49.4 s	61.5 s	66.1 s
		total	79.7 s	94.0 s	**101.9** s

Table 8.19: Comparison of the monolithic preconditioner and SIMPLE for the three-dimensional steady BFS Navier–Stokes problem. P1–P1 with $H/h = 20$, P2–P1 with $H/h = 9$, and Q2–P1disc with $H/h = 8$. The overlap is $\delta = 1h$. All first level preconditioners use SAS. Number of Newton iterations in parenthesis.

given steady BFS Navier–Stokes problem; cf. table 8.14. A history of the adaptive type 2 forcing terms for the monolithic preconditioner applied to the BFS Navier–Stokes problem with a P2–P1 discretization is depicted in table 8.20.

Furthermore, three different finite element discretizations are used for the comparison in table 8.19. For the stabilized P1–P1 elements, the monolithic preconditioner provides the lowest total computation time. Nearly half the time is needed for the setup and solution of problems on 1 125 and 4 608 MPI ranks compared to SIMPLE. Furthermore, the monolithic preconditioner is around 20% faster than

Newton step	1	2	3	4	5
Forcing term η_k	1.0e-3	1.8e-05	1.0e-3	1.0e-3	1.0e-3

Table 8.20: History of forcing terms for the P2–P1 problem of table 8.19 preconditioned with the monolithic variant and 243 subdomains.

SIMPLE for the P2–P1 discretizations. It is not possible to employ the monolithic RGDSW preconditioner for the Q2–P1disc discretization. There, a singular coarse matrix arises after the RAP product, which is a topic for future research. As a remedy we employ a monolithic GDSW CB preconditioner. Even though the setup of GDSW is much more expensive, the performance is competitive. This is due to the low iteration counts of the monolithic GDSW preconditioner for the Q2–P1disc elements. For a problem with 4 608 subdomains it takes SIMPLE 1/3 less time to solve it compared to the monolithic GDSW preconditioner. Since the coarse space of GDSW is, in general, too large to be competitive for a three-dimensional problem with several thousand of ranks and subdomains, we have constructed a smaller coarse space. This GDSW variant is denoted *GDSW–Star*. We follow the idea of RGDSW and use vertex-based basis functions. In contrast to RGDSW, these functions only prescribe values for adjacent edges and not for adjacent faces. The values for edges are determined by an inverse multiplicity scaling in analogy to RGDSW Option 1. To obtain a partition of unity, we further add all face basis functions to the coarse space. Therefore, the GDSW–Star coarse space can be seen as a combination of GDSW and RGDSW. Results for the GDSW–Star preconditioner with coarse basis recycling are given in table 8.21. For smaller problems with 243 and 1 125 subdomains this coarse space is slower than GDSW but faster than SIMPLE. This relation is reversed for 4 608 ranks, where SIMPLE is 20 % faster.

8.4 Results for Almost Incompressible Linear Elasticity Problems

In this section, we consider the three-dimensional mixed linear elasticity cube and beam problems; cf. section 5.1.4. All parallel results in this section are based on the older `FROSch` implementation, which exclusively used `Epetra`. In table 8.22, we compare the weak scalability using a direct coarse solver and the full coarse

Prec.	N	243	1 125	4 608
Monolithic GDSW–Star CB	avg. #its.	88.3(6)	91.0(5)	98.4(5)
	setup	35.3 s	35.0 s	51.4 s
	solve	36.5 s	41.9 s	71.9 s
	total	71.8 s	76.9 s	123.3 s

Table 8.21: GDSW–Star for the three-dimensional steady BFS Navier–Stokes problem. Q2–P1disc with $H/h = 8$. The overlap is $\delta = 1h$. All first level preconditioners use SAS. Number of Newton iterations in parenthesis.

space with inexact coarse solves and neglecting the linearized rotations from the coarse space. For most results of this section, we only use 12 MPI ranks on a node with 24 cores due to the higher memory demands for GDSW and a mixed linear elasticity problem. For inexact coarse solves, we employ GMRES with the tolerance ε_c. Therefore, FGMRES is used for the global problem; cf. section 8.1. If rotations are omitted from the the coarse space, inexact coarse solves are most efficient for 4 096 cores. Only results for a coarse solve tolerance of $\varepsilon_c = 10^{-1}$ are shown, as for lower tolerances, the total time becomes worse both with and without rotations. For inexact coarse solves, we can observe that the number of iterations is similar for coarse spaces with and without rotations. In the case of 1 000 and 4 096 MPI ranks, we even need less iterations without rotations.

Coarse solve	coarse space	with rotations				without rotations			
	#cores	64	216	1 000	4 096	64	216	1 000	4 096
Exact	Time	35.7s	38.1s	44.5s	93.4s	35.8s	37.4s	42.8s	98.7s
	Effic.	**100%**	94%	80%	38%	100%	96%	83%	36%
	GMRES its.	50	50	50	50	69	70	71	72
$\varepsilon_c = 10^{-1}$	Time	40.4s	48.7s	61.1s	104.6s	38.2s	45.2s	58.4s	90.9s
	Effic.	87%	73%	58%	34%	94%	79%	61%	39%
	FGMRES its.	90	122	157	226	93	125	139	203

Table 8.22: Weak scalability with and without rotations for MLE cube problem in three dimensions with structured mesh and decomposition, $H/h = 11$, $\delta = 1h$, $\nu = 0.49999$; 12 MPI ranks per node. The coarse problem is solved with `MUMPS` or GMRES up to a tolerance ε_c. Baseline for the efficiency is the fastest time on 64 cores with rotations and exact coarse solves.

This can be explained with the high coarse tolerance ε_c, as the coarse GMRES problem is often solved in a single iteration. Furthermore, we only report results for an overlap $\delta = 1h$, as larger overlaps decreased the overall efficiency. However, as discussed in section 8.1.1.2, the parallel efficiency can be improved for larger numbers of cores by speeding up the computations on the coarse level. Let us note that the coarse problem in the case of elasticity using rotations is even larger than in the case of Stokes equations.

For an elliptic linear elasticity problem with $\nu = 0.3$ and exact coarse solves, it was reported in [85] that it is more efficient to set up the GDSW coarse problem without rotations.

Here, for $\nu = 0.49999$, the coarse space with rotations and exact coarse solves is slightly more efficient than the coarse space without rotations and exact coarse solves. In total, using an inexact coarse solver, the coarse space without rotations, and an overlap of $\delta = 1h$, is the most efficient approach for a large number of MPI ranks. Moreover, from the implementation and usability point of view, neglecting the rotations in the coarse space should be preferred since coordinates of mesh nodes or basis vectors of the nullspace are needed as input to set up the coarse space with rotations. For an algebraic setup of our preconditioner, the approach without rotations should thus be preferred. In table 8.23, iteration counts both with and without rotations for an unstructured decomposition and ν approaching 0.5 are presented. We observe very good numerical scalability for both coarse spaces. However, the numerical scalability depicted in table 8.24 for the MLE beam problem deteriorates for higher Poisson ratios and number of subdomains for both types of coarse spaces. The main difference between the MLE cube and the MLE beam problem are the boundary conditions. While the

coarse space	with rotations				without rotations			
ν \ N	0.3	0.49	0.4999	0.49999	0.3	0.49	0.4999	0.49999
64	36	43	45	45	41	52	56	56
216	45	51	51	51	54	68	69	69
512	50	55	55	55	63	79	78	78
1 000	53	60	59	67	68	86	82	82

Table 8.23: Iteration counts for the three-dimensional MLE cube problem with an unstructured mesh and decomposition, $H/h = 11$, $\delta = 2h$.

coarse space	with rotations					without rotations				
N \ ν	0.3	0.4	0.49	0.499	0.4999	0.3	0.4	0.49	0.499	0.4999
64	43	44	55	66	70	85	90	110	124	128
216	51	52	70	105	115	107	114	145	190	208
512	57	59	81	131	152	130	135	173	249	287
1 000	61	63	88	147	182	140	146	192	297	>300

Table 8.24: Iteration counts for the three-dimensional MLE beam problem, $H/h = 11$, $\delta = 2h$, unstructured decomposition. Maximum number of iterations is 300.

MLE cube problem possesses only homogeneous Dirichlet boundary conditions for the displacements, the MLE beam problem has only a small portion of Dirichlet boundary. The beam is clamped at the bottom and the rest of the beam can move freely. We have observed a similar behavior for Stokes and Navier–Stokes problems with an outflow boundary. There, a monolithic two-level overlapping Schwarz GDSW preconditioner was used. The monolithic RGDSW coarse spaces led to an improved scalability for fluid flow problems with a Neumann boundary, compared to the monolithic GDSW coarse space; cf. table 8.10. The scalability of the RGDSW coarse spaces for mixed linear elasticity problems could be evaluated in future numerical tests.

8.5 Results of Fully Algebraic GDSW and RGDSW Preconditioners for Nonlinear Elasticity Problems

In our numerical simulations of the nonlinear elasticity Cube and Foam problems, we will employ the recycling strategies presented in section 8.2.4. Here, we always reuse the symbolic factorizations from the previous Newton iteration and/or time step. Moreover, the coarse basis functions from previous iterations are reused for the stationary problem. For the dynamic problem we additionally reuse the coarse matrix. Furthermore, a SAS first level operator with overlap $\delta = 1h$ is employed for all problems in this section. The following discussion and results have been published in [81]. In this section, we compare the GDSW and RDSW preconditioners with exact interface maps and a full coarse space, GDSW and RGDSW preconditioners with an exact interface map but without rotational basis functions, and the fully algebraic variant with approximated interface and without rotational basis functions; for the sake of brevity, we denote the three variants as “rotations”, “no rotations”, and “algebraic”, respectively. As discussed in section 5.1.3, we consider a stationary elasticity problem with homogeneous shear modulus of $\mu = 5 \cdot 10^3$ and a dynamic elasticity problem with two material phases; cf. fig. 8.12 (left) for a graphical representation of the coefficient distribution of the shear modulus. For both cases, we choose $\nu = 0.4$.

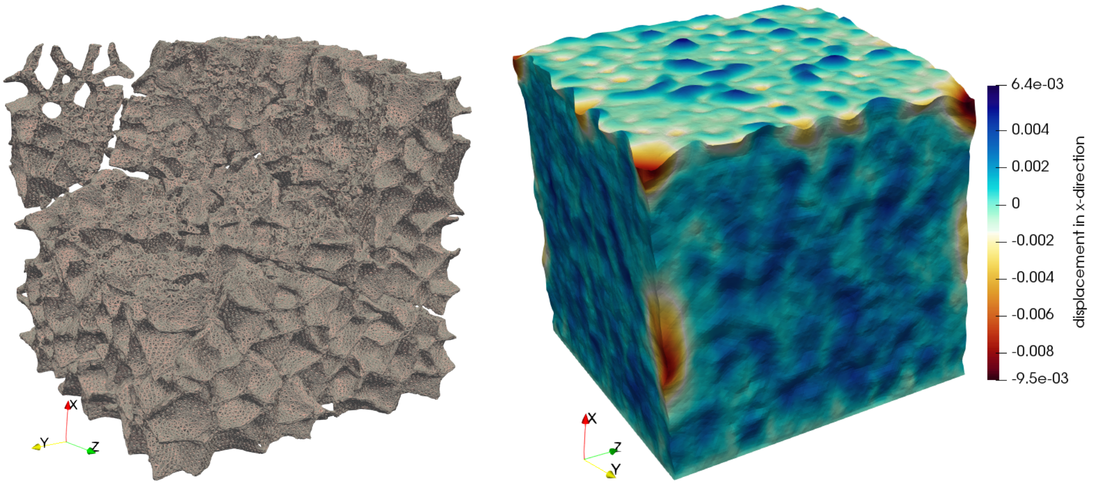


Figure 8.12: Left: Slice through elements with high coefficient ($\mu_{\text{high}} = 10^3$) displayed as a wireframe. Low coefficient is $\mu_{\text{low}} = 1$; cf. [82], for a detailed discussion of the foam geometry used for an heterogeneous Poisson problem. Right: Solution of dynamic Foam problem at $T = 10^{-2}$ for $\Delta t = 10^{-3}$ with a warp filter and a scaling factor of 5. Taken from [81]

For the stationary homogeneous model problem, we use structured meshes and structured decompositions into cubic subdomains, whereas for the dynamic problem, we use a fixed unstructured tetrahedral mesh with roughly 3.3 million elements and 588 k nodes. We use the inexact Newton method of section 5.4 with a type 2 forcing term until a relative residual of $\varepsilon_{\text{nl}} = 10^{-8}$ is achieved. The initial forcing term is $\eta_{\text{init}} = 10^{-3}$ and the maximum and minimum forcing terms are $\eta_{\max} = 10^{-2}$ and $\eta_{\min} = 10^{-8}$, respectively. Furthermore, backtracking line search is used for the globalization of Newton's method. In particular, the step length is chosen as $\frac{1}{2^l}$ with $l = 0, 1, ...$ until the Armijo condition is satisfied. All linearized problems are solved with right-preconditioned GMRES with the corresponding GDSW and RGDSW preconditioners and the tolerance for the relative residual error is the forcing term η.

Prec.	Type	#cores	64	512	4 096
GDSW	rot.	avg. #its.	17.8	19.0	19.0
		setup	35.1 s	45.3 s	167.1 s
		solve	7.4 s	9.7 s	26.1 s
		total	42.5 s	55.0 s	183.2 s
	no rot.	avg. #its.	27.3	32	35.5
		setup	29.3 s	32.9 s	70.8 s
		solve	10.6 s	13.8 s	23.3 s
		total	39.9 s	46.7 s	94.1 s
	algebraic	avg. #its.	32.8	38.5	39.0
		setup	39.5 s	41.6 s	84.3 s
		solve	13.4 s	17.2 s	27.3 s
		total	52.9 s	58.8 s	111.6 s
RGDSW	rot.	avg. #its.	20.5	22.5	22.5
		setup	28.8 s	30.9 s	42.0 s
		solve	8.2 s	9.5 s	11.7 s
		total	37.0 s	40.4 s	53.7 s
	no rot.	avg. #its.	33.0	37.3	39.5
		setup	25.2 s	26.5 s	30.1 s
		solve	12.4 s	14.7 s	18.0 s
		total	27.6 s	41.2 s	48.1 s
	algebraic	avg. #its.	40.0	42.0	43.0
		setup	27.2 s	28.7 s	32.9 s
		solve	15.5 s	16.8 s	19.6 s
		total	42.7 s	45.5 s	52.5 s

Table 8.25: Stationary Cube problem, discretization P1 ($H/h = 21$), iteration counts are averages over all Newton iterations. All problems were solved in 4 Newton iterations. Taken from [81].

In Tables table 8.25 and table 8.26, weak scaling results for the stationary model problem with piecewise linear and piecewise quadratic elements are depicted. Although iteration counts are slightly higher for the RGDSW coarse spaces compared to the respective GDSW coarse spaces, the total computation

Prec.	Type	#cores	64	512	4 096
GDSW	rot.	avg. #its.	16.3	17.3	19.3
		setup	40.1 s	55.0 s	223.3 s
		solve	5.9 s	8.5 s	24.4 s
		total	46.0 s	63.5 s	247.7 s
	no rot.	avg. #its.	24.5	29.3	32.3
		setup	32.5 s	38.4 s	102.2 s
		solve	8.4 s	11.8 s	20.0 s
		total	40.9 s	50.2 s	122.2 s
	algebraic	avg. #its.	57.5	74.8	78.0
		setup	42.0 s	46.0 s	124.8 s
		solve	20.5 s	29.9 s	50.5 s
		total	62.5 s	75.9 s	175.3 s
RGDSW	rot.	avg. #its.	18.8	21.3	19.8
		setup	27.8 s	31.1 s	41.3 s
		solve	6.4 s	8.0 s	8.9 s
		total	34.2 s	39.1 s	50.2 s
	no rot.	avg. #its.	29.0	32.8	35.5
		setup	26.2 s	27.3 s	31.1 s
		solve	9.4 s	11.8 s	14.3 s
		total	35.6 s	39.1 s	45.4 s
	algebraic	avg. #its.	60.7	78.5	83.0
		setup	27.9 s	28.7 s	34.1 s
		solve	19.9 s	27.9 s	33.1 s
		total	47.8 s	56.6 s	67.2 s

Table 8.26: Stationary Cube problem, discretization P2 ($H/h = 9$), iteration counts are averages over all Newton iterations. All problems were solved in 4 Newton iterations. Taken from [81].

time is much smaller for RGDSW due to the lower dimension of the coarse problem. This effect is even stronger for larger numbers of subdomains and cores; cf. table 8.27. Furthermore, we observe competitive iteration counts and computing times when using the fully algebraic coarse spaces. In addition to that, the approximation strategy for the interface seems to perform better for piecewise linear than for piecewise quadratic elements.

In Figure 8.13, we present strong scaling results from 48 to 720 cores for the dynamic model problem. The reported times are the total times for our preconditioners, i.e., the sum of the times needed for their construction and their

	#cores	64	512	4 096
GDSW	rotations	1 593	16 149	144 045
	no rotations	837	8 589	77 085
	algebraic P1 disc.	1 395	11 355	84 762
	algebraic P2 disc.	1 554	11 466	84 708
RGDSW	rotations	162	2 058	20 250
	no rotations	81	1 029	10 125
	algebraic P1 disc.	93	1 065	10 218
	algebraic P2 disc.	93	1 038	10 134

Table 8.27: Comparison of coarse matrix sizes for a structured domain decomposition and the approximated subdomain maps for a P1 ($H/h = 21$) and P2 ($H/h = 9$) discretizaion. Taken from [81].

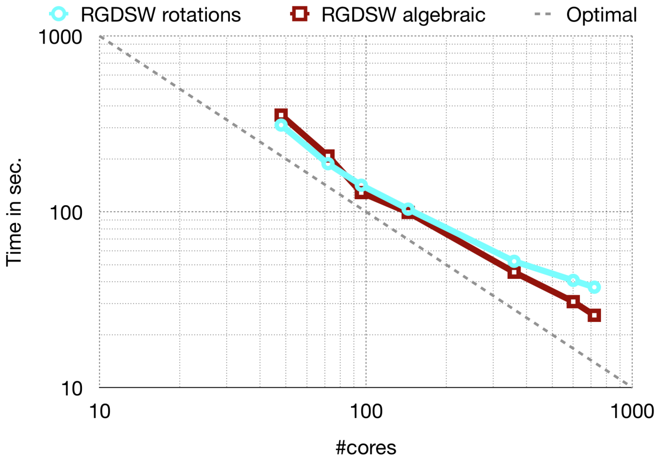


Figure 8.13: Strong scaling for dynamic problem up to time $T = 2 \cdot 10^{-2}$ for the foam geometry. Taken from [81].

applications in GMRES. We solve the problem with $\Delta t = 10^{-3}$ up to a final time $T = 2 \cdot 10^{-2}$ using the RGDSW rotations coarse space and using the RGDSW algebraic coarse space both with matrix recycling. Here, we observe very good strong scalability results for both variants even though the model problem has coefficient jumps. Again, the fully algebraic variant is competitive.

8.6 Results for Fluid-Structure Interaction Problems

In this final section of numerical results, we combine the previously presented preconditioners to efficiently solve FSI problems. We present strong scaling results for the FSI benchmark problem which are computed with the geometry explicit approach. The comparison of the SIMPLE block preconditioner for the fluid and the monolithic approach is continued. In particular, we solve the FSI benchmark problem with GMRES and the FaCSI preconditioner; cf. section 6.5.1. In FaCSI, we apply the monolithic RGDSW preconditioner with CB and CB+CM recycling as well as the SIMPLE preconditioner with the following block approximations to the fluid subproblem: RGDSW preconditioner with CB and CB+CM recycling for the F block of the fluid problem and a one-level Schwarz approximation for the approximate Schur complement. This was also the best performing combination of SIMPLE block preconditioners for a BFS Navier–Stokes problem; cf. section 8.3.2. Moreover, the solid and geometry problems are solved using a one-level Schwarz preconditioner. All preconditioners use SAS in the first level with overlap $\delta = 1h$.

The results were computed on an unstructured tetrahedral mesh with 34 352 P1 fluid nodes and 7 155 P1 solid nodes. Furthermore, we used P2–P1 mixed finite elements for the spatial discretization of the Navier–Stokes equations in ALE form. Consequently, we use P2 elements for the solid problem to guarantee matching nodes on the fluid-solid interface. For the solid problem, we consider the nonlinear elasticity equation with the St. Venant–Kirchhoff material law. As the geometry problem, a scaled harmonic extension problem with a P2 discretization is used. For the description of the benchmark and material parameters of the subproblems, we refer to section 5.1.7. Newmark and BDF2 are used as time stepping schemes for solid and fluid, respectively, and the time step length is $\Delta t = 0.001\,\mathrm{s}$. The results in table 8.28 are taken from a simulation of the first 100 time steps, since there was no significant increase in iteration counts during later time steps. The geometry explicit FSI system is linearized with Newton's method. We employ an inexact Newton method with type 2 forcing term: $\eta_{\max} = 10^{-3}$, $\eta_{\min} = 10^{-8}$, and $\eta_{\text{init}} = 10^{-3}$ until a relative residual norm smaller than $\varepsilon_{\text{nl}} = 10^{-8}$ is achieved. Moreover, the line search backtracking

# cores	Prec.	avg. #its.	setup	solve	total
240	Monolithic CB	51.5	743.6 s	552.6 s	1 296.2 s
	Monolithic CB+CM	48.7	462.8 s	522.8 s	**985.6** s
	SIMPLE CB	59.9	569.5 s	578.0 s	1 147.5 s
	SIMPLE CB+CM	65.0	413.3 s	622.8 s	1 036.1 s
360	Monolithic CB	51.7	699.0 s	515.6 s	1214.6 s
	Monolithic CB+CM	48.4	326.1 s	494.4 s	**820.5** s
	SIMPLE CB	63.0	466.2 s	499.9 s	966.1 s
	SIMPLE CB+CM	68.3	304.1 s	541.5 s	845.6 s
480	Monolithic CB	52.4	631.1 s	564.3 s	1 195.4 s
	Monolithic CB+CM	49.3	306.2 s	531.2 s	**837.4** s
	SIMPLE CB	67.1	475.1 s	660.6 s	1 135.7 s
	SIMPLE CB+CM	72.8	275.6 s	721.9 s	997.5 s

Table 8.28: Results for SIMPLE and monolithic preconditioners in FaCSI for the three-dimensional FSI benchmark. P2–P1 elements for the fluid, overlap $\delta = 1h$, one-level Schwarz preconditioner for solid and geometry parts. All first levels are SAS preconditioners. St. Venant–Kirchhoff material law. Simulation of 100 time steps. An average of 3.1 Newton iterations was needed for all preconditioners.

globalization with step length $\frac{1}{2^l}$ for $l = 0, 1, \ldots$ is used.
After 3.5 s of simulation time, we obtain a deflection of $9.1164{\cdot}10^{-4}$ m in x-direction and $2.0292{\cdot}10^{-5}$ m in y-direction of the reference point $(0.45,\ 0.15,\ 0.15) \in \hat{\Omega}_s$, i.e., a point in the upper left part of the solid domain. The values reported in [125] for a corresponding steady-state benchmark are much lower: $5.95{\cdot}10^{-5}$ m in x-direction. In [6] an incompressible Mooney–Rivlin material has been used and the resulting deflections after 4.6 s are closer to our results: $4.558{\cdot}10^{-3}$ m in x-direction and $3.871{\cdot}10^{-4}$ m in y-direction.

The usage of monolithic RGDSW and SIMPLE preconditioners for the fluid block provides good results. Similar to the standalone time-dependent Navier–Stokes problem, we obtain lower total computing times if coarse matrix recycling is used. Surprisingly, CB+CM recycling for the monolithic preconditioner yields a lower amount of average GMRES iterations compared to CB recycling. The monolithic preconditioner with CB+CM grants the fastest computation times. We highlight that the timings in table 8.28 also include setup and application of the one-level SAS preconditioner of the solid problem as well as the one-level SAS preconditioner of the geometry problem.

Realistic artery: We also consider a realistic artery which was constructed from a surface mesh of arteries with an aneurysm [91]. The aneurysm and branching sections were cut off and a solid geometry was constructed around the remaining part of the vessel resulting in an artery and arterial wall with a total diameter of 2.5 mm. In fig. 8.14, the tetrahedral meshes of the fluid and the solid domain are shown. The lumen diameter of the artery is approximately 2 mm, which corresponds to the size of a middle cerebral artery; cf. [119]. The fluid domain consists of 21 277 P1 nodes and 99 969 elements, while the solid domain is discretized with 23 889 P1 nodes and 84 734 elements. Furthermore, the resulting GE FSI matrix problem possesses 1 028 422 d.o.f.. Again, Newmark and BDF2 are used as time stepping schemes for solid and fluid, respectively, and the time step length is $\Delta t = 0.001\,\mathrm{s}$. The geometry problem is fixed with homogeneous Dirichlet boundary conditions on $\partial\Omega_{f,\mathrm{in}}$ and $\partial\Omega_{f,\mathrm{out}}$. In particular, the scaled harmonic extension problem uses the scaling $\alpha = 1000$ and the distance $dist = 0.1$; cf. section 5.3. We model the physical properties of blood with the following fluid parameters: density $\rho_f = 10^{-6}\,\frac{\mathrm{kg}}{\mathrm{mm}^3}$ and dynamic viscosity $\mu_f = 3{\cdot}10^{-6}\,\frac{\mathrm{kg}}{\mathrm{mm}\cdot\mathrm{s}}$. Blood can be modeled as a Newtonian fluid in larger arteries like the middle cerebral artery. To model the arterial wall we use the St. Venant–Kirchhoff material model and the following paramters: density $\rho_s = 1.3{\cdot}10^{-6}\,\frac{\mathrm{kg}}{\mathrm{mm}^3}$, Young's modulus $E = 910\,\frac{\mathrm{kg}}{\mathrm{mm}\cdot\mathrm{s}^2}$, and Poisson's ratio $\nu_s = 0.49$. The structure is clamped at both ends and a parabolic inflow profile in z-direction is prescribed on the left; all figures of the realistic artery are positioned such that the blood flows from left to right. The maximum inflow velocity is linearly increased from 0 at $t = 0$ to $u_{\max} = 600\,\frac{\mathrm{mm}}{\mathrm{s}}$ at $t = 0.5\,\mathrm{s}$. This peak velocity occurs in middle cerebral arteries; cf. [110]. At the outflow, a do-nothing boundary condition is used. For the realistic artery problem, this outflow condition was sufficient. However, to prevent unphysical oscillations due to wave reflections at the outflow boundary, resistance or resistive boundary conditions can be used; cf. [143, 144].

We use the same line search backtracking globalization as for the FSI benchmark. Moreover, the following parameters for the inexact Newton method with type 2 forcing term are used: $\eta_{\max} = 10^{-2}$, $\eta_{\min} = 10^{-6}$, and $\eta_{\mathrm{init}} = 10^{-3}$ until a relative residual norm smaller than $\varepsilon_{\mathrm{nl}} = 10^{-8}$ is achieved. The solution at $t = 0.5\,\mathrm{s}$ is shown in fig. 8.15.

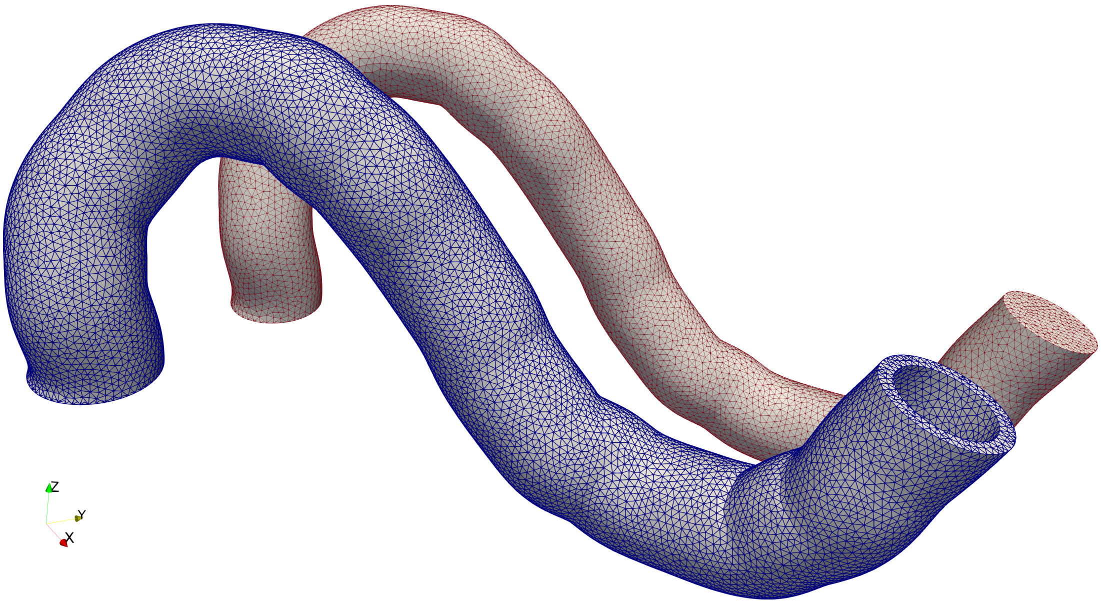


Figure 8.14: Solid domain (front) with highlighted mesh in blue and fluid domain (back) with highlighted mesh in red of the realistic FSI problem.

Figure 8.16 shows a comparison of the St. Venant–Kirchhoff material and a Neo–Hookean material. The deformation of the artery with the Neo–Hooke material is slightly larger, but no significant difference can be observed and the overall deformation of the artery is small. With these material models a realistic behavior can only be described for small deformations. For larger deformations, more complex material models which include fiber-orientation should be used; cf. [10, 11, 26]. Furthermore, in fig. 8.17, the number of average linear iterations per Newton iteration for the St. Venant–Kirchhoff material and the Neo–Hooke material are compared. A similar number of total linear iterations is needed for both problems. For the Neo–Hookean material, the average number of linear iterations per time step and Newton iteration is 69.4. Similarly, the St. Venant–Kirchhoff material requires on average 71.0 linear iterations. Roughly the same number of total Newton iterations is needed for the Neo–Hookean and the St. Venant–Kirchhoff material. On average, the FSI problems are solved in 3.2 Newton iterations.

The use of a fully monolithic one-level SAS preconditioner, cf. section 6.5.2,

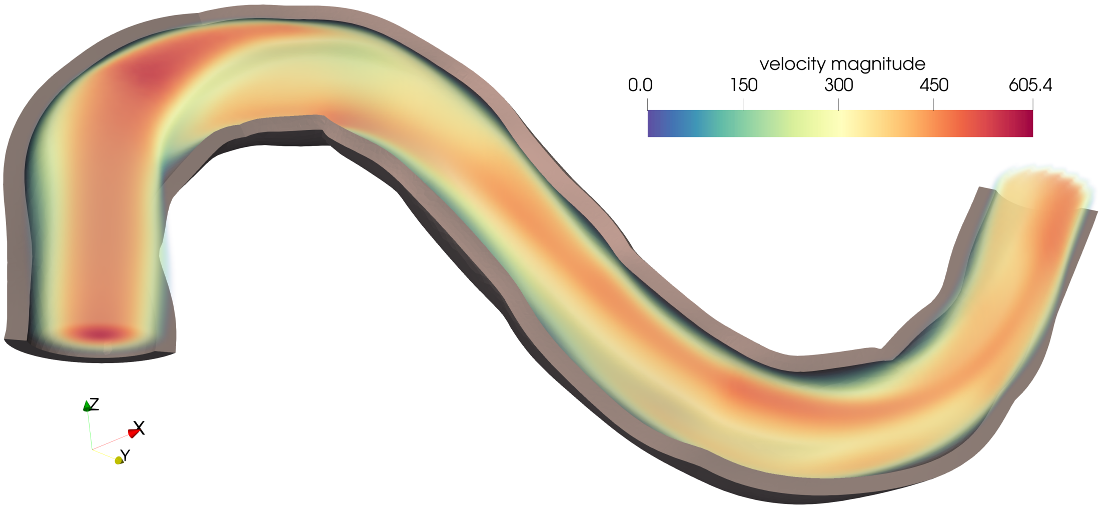


Figure 8.15: Velocity magnitude at $t = 0.5\,\mathrm{s}$ and clipped deformed artery wall with scaling factor 200.

for the simulation of the realistic artery yields no adequate results. Iterations counts are 10 times higher than for the FaCSI preconditioner. The partitions of the fluid and solid domains are independent, which can result in geometrically independent fluid subdomains and solid subdomains without a shared fluid-solid interface. The resulting monolithic preconditioner cannot account for the correct coupling and only aggregates fluid and solid subdomains based on the rank. Figure 8.18 shows independent decompositions of the fluid and the solid domain into 24 subdomains. These partitions are likely the cause of the bad performance of a fully monolithic preconditioner. As a remedy, we would need to combine both meshes and partition the resulting mesh. This approach is currently not implemented and a topic for future investigation.

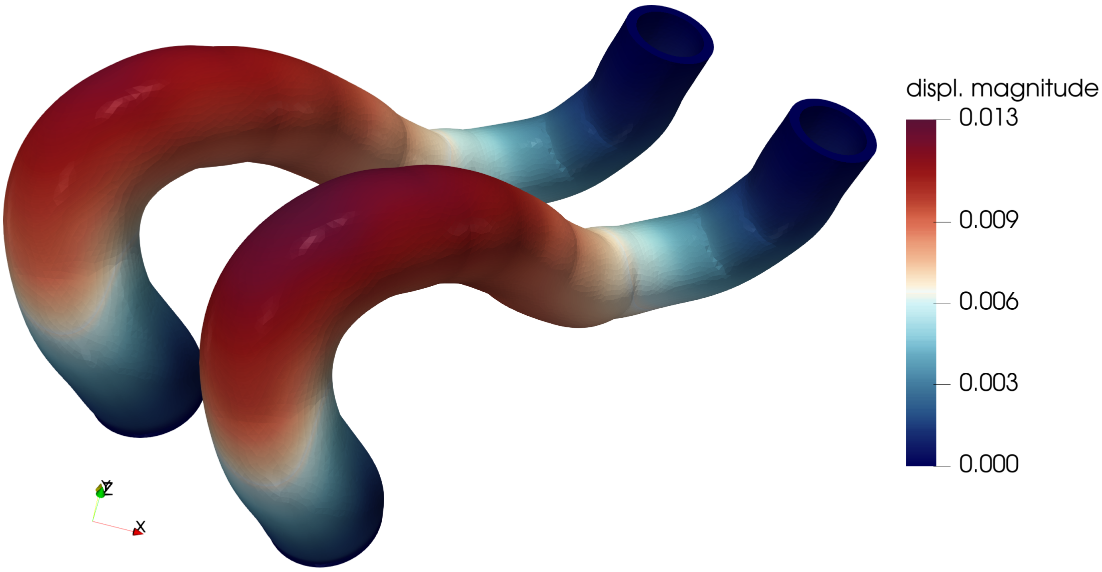


Figure 8.16: Displacements of artery wall with Neo–Hooke material (front) and St. Venant–Kirchhoff material (back) at $t = 0.5\,\mathrm{s}$ with scaling factor 200.

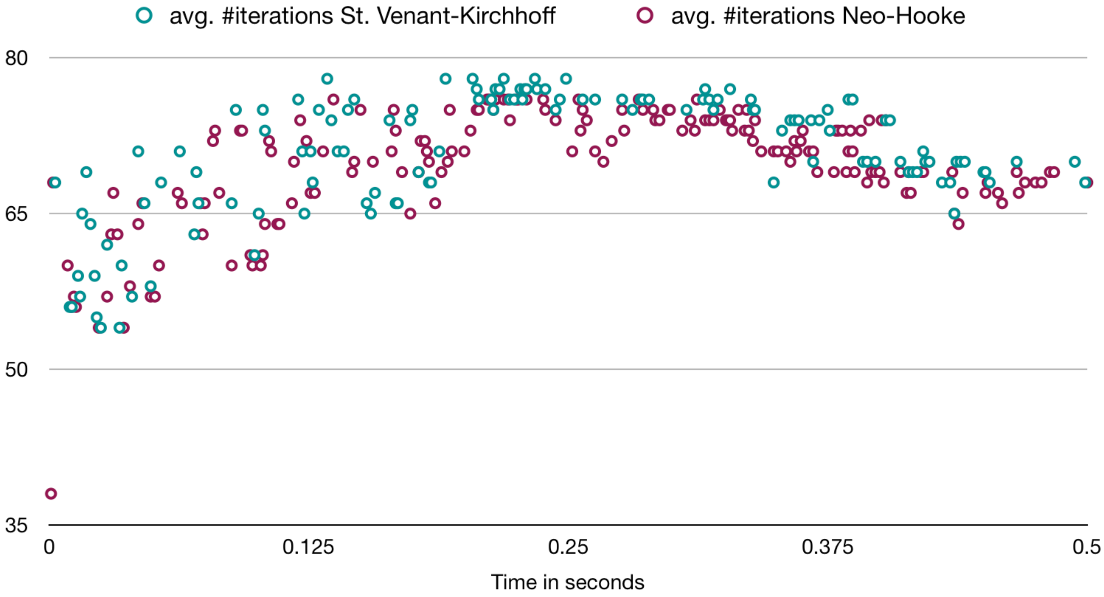


Figure 8.17: Average number of linear iterations per Newton iteration for each time step of the St. Venant–Kirchhoff material and the Neo–Hooke material.

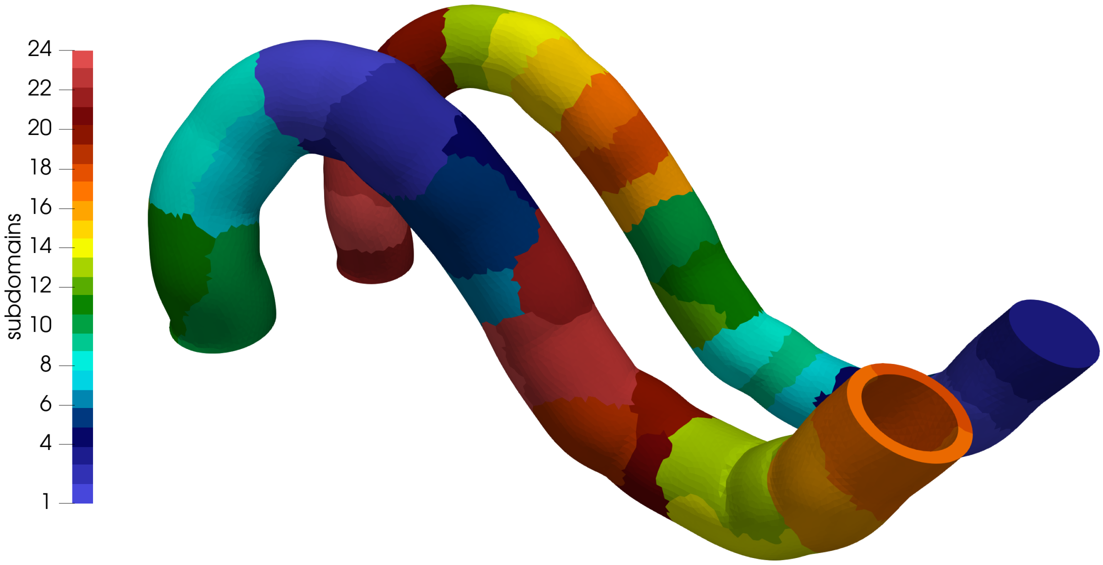


Figure 8.18: Solid domain (front) and fluid domain (back), decompositions into 24 subdomains. Same color of subdomain corresponds to same MPI rank.

9 Conclusion

The efficient solution of incompressible fluid flow problems with monolithic parallel preconditioners on HPC systems with several thousand cores has been the main focus of this thesis. For the simulations of these fluid flow problems and additional fluid-structure interaction problems, the parallel object-oriented C++ software library FEDDLib has been developed. Furthermore, the Trilinos package FROSch for parallel Schwarz preconditioners has been extended to saddle point problems and general monolithic systems. In particular, the essentially algebraic monolithic GDSW and RGDSW preconditioners were developed for Stokes and Navier–Stokes fluid problems. These preconditioners can be constructed likewise for structured and unstructured spatial finite element discretizations. To reduce the time-to-solution, different recycling approaches for steady-state and time-dependent Navier–Stokes problems have been considered. In particular, the coarse basis recycling for steady-state problems resulted in significant reductions in computing time. Similarly, the coarse basis and additional coarse matrix recycling provided substantial reductions in time-to-solution for time-dependent Navier–Stokes problems. Moreover, advanced combination techniques such as the multiplicative coupling and the parallel additive coupling of the first and coarse level operators provided further efficiency gains. Specifically, the simultaneous computation of the first and second level of the additive two-level RGDSW preconditioner has been most efficient. Weak scalability results for up to 11 979 cores showed an efficiency of over 80 % for a three-dimensional Stokes problem. The largest possible configuration on the employed supercomputer allowed for problems with over 400 million degrees of freedom.

A comparison of the novel monolithic preconditioners with block-diagonal and block-triangular preconditioners for incompressible fluid flow problems resulted in faster convergence rates and lower total computing times. As a proof of concept, the most efficient monolithic RGDSW preconditioner for time-dependent Navier–

Stokes problems as well as two-level RGDSW preconditioners for solid problems were combined to solve an FSI problem of a realistic artery. Extended simulations of FSI problems with a focus on parallel two- and multi-level overlapping Schwarz methods are a topic for future investigations.

Bibliography

[1] James Ahrens, Berk Geveci, and Charles Law, *Paraview: An end-user tool for large data visualization*, The visualization handbook **717** (2005).

[2] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications **23** (2001), no. 1, 15–41.

[3] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, *Hybrid scheduling for the parallel solution of linear systems*, Parallel Computing **32** (2006), no. 2, 136–156.

[4] Donald G. Anderson, *Iterative procedures for nonlinear integral equations*, J. Assoc. Comput. Mach. **12** (1965), 547–560.

[5] Kenneth Joseph Arrow, Leonid Hurwicz, Hirofumi Uzawa, and Hollis Burnley Chenery, *Studies in linear and non-linear programming* (1958).

[6] Eugenio Aulisa, Simone Bnà, and Giorgio Bornia, *A monolithic ALE Newton-Krylov solver with multigrid-Richardson-Schwarz preconditioning for incompressible fluid-structure interaction*, Comput. & Fluids **174** (2018), 213–228.

[7] Utkarsh Ayachit, *The paraview guide: a parallel visualization application*, Kitware, Inc., 2015.

[8] S. Badia, A. Martín, and J. Principe, *Multilevel balancing domain decomposition at extreme scales*, SIAM Journal on Scientific Computing **38** (2016), no. 1, C22–C52.

[9] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Dmitry Karpeyev, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang, *PETSc Web page*, 2019.

[10] D. Balzani, D. Böse, D. Brands, R. Erbel, A. Klawonn, O. Rheinbach, and J. Schröder, *Parallel simulation of patient-specific atherosclerotic arteries for the enhancement of intravascular ultrasound diagnostics*, Engrg. Comput. **29** (2012), no. 8, 888–906.

[11] Daniel Balzani, Simone Deparis, Simon Fausten, Davide Forti, Alexander Heinlein, Axel Klawonn, Alfio Quarteroni, Oliver Rheinbach, and Jörg Schröder, *Numerical modeling of fluid-structure interaction in arteries with anisotropic polyconvex hyperelastic and anisotropic viscoelastic material models at finite strains*, Int. J. Numer. Methods Biomed. Eng. (2015).

[12] Randolph E. Bank, Bruno D. Welfert, and Harry Yserentant, *A class of iterative methods for solving saddle point problems*, Numer. Math. **56** (1990), 645 –666.

[13] Andrew T. Barker and Xiao-Chuan Cai, *Scalable parallel methods for monolithic coupling in fluid-structure interaction with application to blood flow modeling*, J. Comput. Phys. **229** (2010), no. 3, 642–659.

[14] ———, *Two-level Newton and hybrid Schwarz preconditioners for fluid-structure interaction*, SIAM J. Sci. Comput. **32** (2010), no. 4, 2395–2417.

[15] Ted Belytschko and Thomas J. R. Hughes (eds.), *Computational methods for transient analysis*, Mechanics and Mathematical Methods. A Series of Handbooks. Subseries: Computational Methods in Mechanics, vol. 1, Elsevier Scientific Publishing Co., Amsterdam, 1983.

[16] Michele Benzi, Gene H. Golub, and Jörg Liesen, *Numerical solution of saddle point problems*, Acta Numer. **14** (2005), 1–137.

[17] Michele Benzi and Xue-Ping Guo, *A dimensional split preconditioner for Stokes and linearized Navier-Stokes equations*, Appl. Numer. Math. **61** (2011), no. 1, 66–76.

[18] Michele Benzi, Michael Ng, Qiang Niu, and Zhen Wang, *A relaxed dimensional factorization preconditioner for the incompressible Navier-Stokes equations*, J. Comput. Phys. **230** (2011), no. 16, 6185–6202.

[19] Daniele Boffi, Franco Brezzi, and Michel Fortin, *Mixed finite element methods and applications*, Springer Series in Computational Mathematics, vol. 44, Springer, Heidelberg, 2013.

[20] ———, *Mixed finite element methods and applications*, Springer Series in Computational Mathematics, vol. 44, Springer, Heidelberg, 2013.

[21] D. Braess and C. Blömer, *A multigrid method for a parameter dependent problem in solid mechanics*, Numer. Math. **57** (1990), no. 8, 747–761.

[22] Dietrich Braess, *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*, Springer-Verlag, 2013.

[23] James H. Bramble and Joseph E. Pasciak, *A domain decomposition technique for Stokes problems*, Appl. Numer. Math. **6** (1990), no. 4, 251–261.

[24] James H. Bramble, Joseph E. Pasciak, and Apostol T. Vassilev, *Analysis of the inexact Uzawa algorithm for saddle point problems*, SIAM J. Numer. Anal. **34** (1997), no. 3, 1072–1092.

[25] Susanne C. Brenner, *Multigrid methods for parameter dependent problems*, RAIRO Modél. Math. Anal. Numér. **30** (1996), no. 3, 265–297.

[26] S. Brinkhues, A. Klawonn, O. Rheinbach, and J. Schröder, *Augmented Lagrange methods for quasi-incompressible materials - Applications to soft biological tissue*, International Journal for Numerical Methods in Biomedical Engineering **29** (2013), no. 3, 332–350.

[27] Alexander N. Brooks and Thomas J. R. Hughes, *Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations*, Comput. Methods Appl. Mech. Engrg. **32** (1982), no. 1-3, 199–259. FENOMECH ”81, Part I (Stuttgart, 1981).

[28] J. C. Butcher, *Implicit Runge-Kutta processes*, Math. Comp. **18** (1964), 50–64.

[29] Xiao-Chuan Cai, *Some domain decomposition algorithms for nonselfadjoint elliptic and parabolic partial differential equations*, Ph.D. Thesis, 1989. Tech. Rep. 461, Department of Computer Science, Courant Institute.

[30] Xiao-Chuan Cai and David E. Keyes, *Nonlinearly preconditioned inexact Newton algorithms*, SIAM J. Sci. Comput. **24** (2002), no. 1, 183–200.

[31] Xiao-Chuan Cai, David E. Keyes, and Leszek Marcinkowski, *Non-linear additive Schwarz preconditioners and application in computational fluid dynamics*, Internat. J. Numer. Methods Fluids **40** (2002), no. 12, 1463–1470.

[32] Xiao-Chuan Cai and Marcus Sarkis, *A restricted additive Schwarz preconditioner for general sparse linear systems*, SIAM Journal on Scientific Computing **21** (1999), 239–247.

[33] Deniz Cevik, *Eine parallele Implementierung monolithischer Fluid-Struktrur-Interaktion in ALE-Formulierung*, Master's Thesis, Universität zu Köln, 2020.

[34] Philippe G. Ciarlet, *Mathematical Elasticity Volume I: Three–Dimensional Elasticity*, North-Holland, 1988.

[35] Lyndon Clarke, Ian Glendinning, and Rolf Hempel, *The MPI message passing interface standard*, Programming environments for massively parallel distributed systems, 1994, pp. 213–218.

[36] Jean-Michel Cros, *A preconditioner for the Schur complement domain decomposition method*, Domain decomposition methods in science and engineering, 2003, pp. 373–380. Proceedings of the 14th International Conference on Domain Decomposition Methods in Science and Engineering.

[37] P. Crosetto, S. Deparis, G. Fourestey, and A. Quarteroni, *Parallel algorithms for fluid-structure interaction problems in haemodynamics*, SIAM J. Sci. Comput. **33** (2011), no. 4, 1598–1622.

[38] Paolo Crosetto, *Fluid-structure interaction problems in hemodynamics parallel solvers, preconditioners, and applications*, EPFL, MATHICSE, Lausanne, 2011.

[39] Eric C. Cyr, John N. Shadid, and Raymond S. Tuminaro, *Stabilization and scalable block preconditioning for the Navier-Stokes equations*, J. Comput. Phys. **231** (2012), no. 2, 345–363.

[40] Leonardo Dagum and Ramesh Menon, *OpenMP: an industry standard API for shared-memory programming*, IEEE computational science and engineering **5** (1998), no. 1, 46–55.

[41] Ron S. Dembo, Stanley C. Eisenstat, and Trond Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal. **19** (1982), no. 2, 400–408.

[42] Simone Deparis, Davide Forti, Gwenol Grandperrin, and Alfio Quarteroni, *Facsi: A block parallel preconditioner for fluid–structure interaction in hemodynamics*, Journal of Computational Physics **327** (2016), 700–718.

[43] ______, *FaCSI: a block parallel preconditioner for fluid-structure interaction in hemodynamics*, J. Comput. Phys. **327** (2016), 700–718.

[44] Simone Deparis, Gwenol Grandperrin, and Alfio Quarteroni, *Parallel preconditioners for the unsteady Navier-Stokes equations and applications to hemodynamics simulations*, Comput. & Fluids **92** (2014), 253–273.

[45] Clark Dohrmann, Axel Klawonn, and Olof B. Widlund, *A family of energy minimizing coarse spaces for overlapping Schwarz preconditioners*, Domain decomposition methods in science and engineering, 2008, pp. 247–254.

[46] Clark R. Dohrmann, *Some domain decomposition algorithms for mixed formulations of elasticity and incompressible fluids*, Workshop on Adaptive Finite Elements and Domain Decomposition Methods.

[47] ———, *A preconditioner for substructuring based on constrained energy minimization*, SIAM J. Sci. Comput. **25** (2003), no. 1, 246–258.

[48] Clark R. Dohrmann and Pavel B. Bochev, *A stabilized finite element method for the Stokes problem based on polynomial pressure projections*, Internat. J. Numer. Methods Fluids **46** (2004), no. 2, 183–201.

[49] Clark R. Dohrmann, Axel Klawonn, and Olof B. Widlund, *Domain decomposition for less regular subdomains: overlapping Schwarz in two dimensions*, SIAM J. Numer. Anal. **46** (2008), no. 4, 2153–2168.

[50] Clark R. Dohrmann and Olof B. Widlund, *An overlapping schwarz algorithm for almost incompressible elasticity*, SIAM J. Numer. Anal. **47** (2009), no. 4, 2897–2923.

[51] ———, *Hybrid domain decomposition algorithms for compressible and almost incompressible elasticity*, Internat. J. Numer. Meth. Engng **82** (2010), no. 2, 157–183.

[52] ———, *An alternative coarse space for irregular subdomains and an overlapping Schwarz algorithm for scalar elliptic problems in the plane*, SIAM J. Numer. Anal. **50** (2012), no. 5, 2522–2537.

[53] ———, *Lower dimensional coarse spaces for domain decomposition*, Domain decomposition methods in science and engineering XXI, 2014, pp. 527–535.

[54] ———, *On the design of small coarse spaces for domain decomposition algorithms*, SIAM J. Sci. Comput. **39** (2017), no. 4, A1466–A1488.

[55] H Carter Edwards, Christian R Trott, and Daniel Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing **74** (2014), no. 12, 3202–3216.

[56] Evridiki Efstathiou and Martin J. Gander, *Why Restricted Additive Schwarz converges faster than Additive Schwarz*, BIT Numerical Mathematics **43** (2003), no. 5, 945–959.

[57] Stanley C. Eisenstat and Homer F. Walker, *Globally convergent inexact Newton methods*, SIAM J. Optim. **4** (1994), no. 2, 393–422.

[58] ———, *Choosing the forcing terms in an inexact Newton method*, 1996, pp. 16–32. Special issue on iterative methods in numerical linear algebra (Breckenridge, CO, 1994).

[59] Howard Elman, V. E. Howle, John Shadid, Robert Shuttleworth, and Ray Tuminaro, *A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier-Stokes equations*, J. Comput. Phys. **227** (2008), no. 3, 1790–1808.

[60] Howard Elman, Victoria E. Howle, John Shadid, Robert Shuttleworth, and Ray Tuminaro, *Block preconditioners based on approximate commutators*, SIAM J. Sci. Comput. **27** (2006), no. 5, 1651–1668.

[61] Howard Elman and David Silvester, *Fast nonsymmetric iterations and preconditioning for Navier-Stokes equations*, SIAM J. Sci. Comput. **17** (1996), no. 1, 33–46. Special issue on iterative methods in numerical linear algebra (Breckenridge, CO, 1994).

[62] Howard C. Elman and Gene H. Golub, *Inexact and preconditioned Uzawa algorithms for saddle point problems*, SIAM J. Numer. Anal. **31** (1994), no. 6, 1645–1661.

[63] Howard C. Elman, David J. Silvester, and Andrew J. Wathen, *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*, Second, Numerical Mathematics and Scientific Computation, Oxford University Press, Oxford, 2014.

[64] Howard C. Elman and Ray S. Tuminaro, *Boundary conditions in approximate commutator preconditioners for the Navier-Stokes equations*, Electron. Trans. Numer. Anal. **35** (2009), 257–280.

[65] Charbel Farhat, *A Lagrange multiplier based on divide and conquer finite element algorithm*, J. Comput. System Engrg **2** (1991), 149–156.

[66] Charbel Farhat, Michel Lesoinne, Patrick LeTallec, Kendall Pierson, and Daniel Rixen, *FETI-DP: a dual–primal unified FETI methodpart I: A faster alternative to the two-level FETI method*, International journal for numerical methods in engineering **50** (2001), no. 7, 1523–1544.

[67] Charbel Farhat and Francois-Xavier Roux, *A method of Finite Element Tearing and Interconnecting and its parallel solution algorithm*, Int. J. Numer. Meth. Engrg. **32** (1991), 1205–1227.

[68] Miguel Ángel Fernández and Marwan Moubachir, *A newton method using exact jacobians for solving fluid–structure coupling*, Computers & Structures **83** (2005), no. 2-3, 127–142.

[69] L. Formaggia, M. Fernandez, A. Gauthier, J.F. Gerbeau, C. Prud'homme, and A. Veneziani, *The LifeV Project.*

[70] Luca Formaggia, Alfio Quarteroni, and Alessandro Veneziani (eds.), *Cardiovascular mathematics*, MS&A. Modeling, Simulation and Applications, vol. 1, Springer-Verlag Italia, Milan, 2009. Modeling and simulation of the circulatory system.

[71] Pascal Frey, *MEDIT : An interactive Mesh visualization Software*, Technical Report RT-0253, INRIA, 2001.

[72] M. W. Gee, U. Küttler, and W. A. Wall, *Truly monolithic algebraic multigrid for fluid-structure interaction*, Internat. J. Numer. Methods Engrg. **85** (2011), no. 8, 987–1016.

[73] Vivette Girault and Pierre-Arnaud Raviart, *Finite element methods for Navier-Stokes equations*, Springer-Verlag, New York, 1986.

[74] P. M. Gresho, *Some current CFD issues relevant to the incompressible Navier-Stokes equations*, Comput. Methods Appl. Mech. Engrg. **87** (1991), no. 2-3, 201–252.

[75] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving ordinary differential equations. I*, Second, Springer Series in Computational Mathematics, vol. 8, Springer-Verlag, Berlin, 1993. Nonstiff problems.

[76] E. Hairer and G. Wanner, *Solving ordinary differential equations. II*, Springer Series in Computational Mathematics, vol. 14, Springer-Verlag, Berlin, 2010. Stiff and differential-algebraic problems, Second revised edition, paperback.

[77] A. Heinlein, *Parallel overlapping schwarz preconditioners and multiscale discretizations with applications to fluid-structure interaction and highly heterogeneous problems*, PhD thesis, 2016.

[78] Alexander Heinlein, Christian Hochmuth, and Axel Klawonn, *GDSW preconditioners and block-preconditioners for saddle point problems.* In preparation.

[79] ______, *Monolithic overlapping Schwarz domain decomposition methods with GDSW coarse spaces for incompressible fluid flow problems*, SIAM J. Sci. Comput. **41** (2019), no. 4, C291–C316.

[80] ______, *Reduced dimension GDSW coarse spaces for monolithic Schwarz domain decomposition methods for incompressible fluid flow problems*, International Journal for Numerical Methods in Engineering **121** (2020), no. 6, 1101–1119.

[81] ———, *Fully algebraic two-level overlapping Schwarz preconditioners for elasticity problems*, accepted for publication to the proceedings of the ENUMATH 2019 conference, Springer LNCSE, April 2020.

[82] Alexander Heinlein, Axel Klawonn, Jascha Knepper, and Oliver Rheinbach, *Adaptive GDSW coarse spaces for overlapping Schwarz methods in three dimensions*, SIAM Journal on Scientific Computing **41** (2019), no. 5, A3045–A3072.

[83] Alexander Heinlein, Axel Klawonn, Sivasankaran Rajamanickam, and Oliver Rheinbach, *FROSch – a parallel implementation of the GDSW domain decomposition preconditioner in Trilinos*. In preparation.

[84] ———, *FROSch: A Fast and Robust Overlapping Schwarz Domain Decomposition Preconditioner Based on Xpetra in Trilinos*, Universität zu Köln, 2018.

[85] Alexander Heinlein, Axel Klawonn, and Oliver Rheinbach, *A parallel implementation of a two-level overlapping Schwarz method with energy-minimizing coarse space based on Trilinos*, SIAM J. Sci. Comput. **38** (2016), no. 6, C713–C747.

[86] ———, *Parallel two-level overlapping Schwarz methods in fluid-structure interaction*, Numerical mathematics and advanced applications—ENUMATH 2015, 2016, pp. 521–530.

[87] ———, *Parallel overlapping Schwarz with an energy-minimizing coarse space*, Domain decomposition methods in science and engineering XXIII, 2017, pp. 353–360.

[88] Alexander Heinlein, Axel Klawonn, Oliver Rheinbach, and Friederike Röver, *A three-level extension of the GDSW overlapping Schwarz preconditioner in two dimensions*, March 2018. Submitted for publication to Lecture Notes in Computational Science and Engineering.

[89] Alexander Heinlein, Axel Klawonn, Oliver Rheinbach, and Olof B Widlund, *Improving the parallel performance of overlapping Schwarz methods by using a smaller energy minimizing coarse space*, 2017, pp. 383–392.

[90] Michael A Heroux, Roscoe A Bartlett, Vicki E Howle, Robert J Hoekstra, Jonathan J Hu, Tamara G Kolda, Richard B Lehoucq, Kevin R Long, Roger P Pawlowski, Eric T Phipps, Andrew G Salinger, Heidi K Thornquist, Ray S Tuminaro, James M Willenbring, Alan Williams, and Kendall S Stanley, *An overview of the Trilinos project*, ACM Trans. Math. Softw. **31** (2005), no. 3, 397–423.

[91] AneuriskWeb http://ecm2.mathcs.emory.edu/aneuriskweb, Emory University, Department of Math&CS, 2012.

[92] Feng-Nan Hwang and Xiao-Chuan Cai, *A parallel nonlinear additive Schwarz preconditioned inexact newton algorithm for incompressible Navier–Stokes equations*, Journal of Computational Physics **204** (2005), no. 2, 666–691.

[93] ———, *Parallel fully coupled Schwarz preconditioners for saddle point problems*, Electron. Trans. Numer. Anal. **22** (2006), 146–162.

[94] ———, *A class of parallel two-level nonlinear Schwarz preconditioned inexact newton algorithms*, Computer methods in applied mechanics and engineering **196** (2007), no. 8, 1603–1611.

[95] Wolfram Research, Inc., *Mathematica, Version 11.3*. Champaign, IL, 2018.

[96] Volker John, *Higher order finite element methods and multigrid solvers in a benchmark problem for the 3D Navier-Stokes equations*, International Journal for Numerical Methods in Fluids **40** (2002), no. 6, 775–798.

[97] ———, *Finite element methods for incompressible flow problems*, Springer Series in Computational Mathematics, vol. 51, Springer, Cham, 2016.

[98] Volker John and Joachim Rang, *Adaptive time step control for the incompressible Navier-Stokes equations*, Comput. Methods Appl. Mech. Engrg. **199** (2010), no. 9-12, 514–524.

[99] George Karypis and Vipin Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on scientific Computing **20** (1999), no. 1, 359–392.

[100] George Karypis, Kirk Schloegel, and Vipin Kumar, *ParMETIS - Parallel graph partitioning and sparse matrix ordering. Version 4.0.3*, University of Minnesota, Department of Computer Science and Engineering, 2011.

[101] David Kay, Daniel Loghin, and Andrew Wathen, *A preconditioner for the steady-state Navier-Stokes equations*, SIAM J. Sci. Comput. **24** (2002), no. 1, 237–256.

[102] Axel Klawonn, *Block-triangular preconditioners for saddle point problems with a penalty term*, SIAM Journal on Scientific Computing **19** (1998), no. 1, 172–184.

[103] ———, *An optimal preconditioner for a class of saddle point problems with a penalty term*, SIAM J. Sci. Comput. **19** (1998), no. 2, 540–552.

[104] Axel Klawonn and Luca Pavarino, *Overlapping Schwarz methods for mixed linear elasticity and Stokes problems*, Comput. Methods Appl. Mech. Engrg. **165** (1998), no. 1-4, 233–245.

[105] ———, *A comparison of overlapping Schwarz methods and block preconditioners for saddle point problems*, Numer. Linear Algebra Appl. **7** (2000), no. 1, 1–25.

[106] Axel Klawonn and Oliver Rheinbach, *Inexact FETI-DP methods*, Internat. J. Numer. Methods Engrg. **69** (2007), no. 2, 284–307.

[107] ———, *Highly scalable parallel domain decomposition methods with an application to biomechanics*, ZAMM Z. Angew. Math. Mech. **90** (2010), no. 1, 5–32.

[108] Axel Klawonn and Gerhard Starke, *Block triangular preconditioners for nonsymmetric saddle point problems: field-of-values analysis*, Numer. Math. **81** (1999), no. 4, 577–594.

[109] Axel Klawonn, Olof B. Widlund, and Maksymilian Dryja, *Dual-primal FETI methods for three-dimensional elliptic problems with heterogeneous coefficients*, SIAM J. Numer. Anal. **40** (2002), no. 1, 159–179.

[110] W Andrew Kofke, Patric Brauer, Raymond Policare, Susan Penthany, David Barker, and Joseph Horton, *Middle cerebral artery blood flow velocity and stable xenon-enhanced computed tomographic blood flow during balloon test occlusion of the internal carotid artery*, Stroke **26** (1995), no. 9, 1603–1606.

[111] Joze Korelc and Peter Wriggers, *Automation of finite element methods*, Springer, 2016.

[112] Martin Kronbichler, Timo Heister, and Wolfgang Bangerth, *High accuracy mantle convection simulation through modern numerical methods*, Geophysical Journal International **191** (2012), no. 1, 12–29.

[113] Li Luo, Wen-Shin Shiu, Rongliang Chen, and Xiao-Chuan Cai, *A nonlinear elimination preconditioned inexact Newton method for blood flow problems in human artery with stenosis*, J. Comput. Phys. **399** (2019), 108926, 20.

[114] JB Malone, JC Narramore, and LN Sankar, *Airfoil design method using the Navier–Stokes equations*, Journal of Aircraft **28** (1991), no. 3, 216–224.

[115] Jan Mandel and Clark R. Dohrmann, *Convergence of a balancing domain decomposition by constraints and energy minimization*, Numer. Linear Algebra Appl. **10** (2003), 639–659.

[116] Jan Mandel and Radek Tezaur, *On the convergence of a dual-primal substructuring method*, Numer. Math. **88** (2001), no. 3, 543–558.

[117] MATLAB, *version 8.6.0 (R2015b)*, The MathWorks Inc., Natick, Massachusetts, 2015.

[118] Fadl Moukalled, L Mangani, Marwan Darwish, et al., *The finite volume method in computational fluid dynamics*, Vol. 113, Springer, 2016.

[119] Daichi Nakagawa, Masaaki Shojima, Masanori Yoshino, Taichi Kin, Hideaki Imai, Seiji Nomura, Toki Saito, Hirofumi Nakatomi, Hiroshi Oyama, and Nobuhito Saito, *Wall-to-lumen ratio of intracranial arteries measured by indocyanine green angiography*, Asian journal of neurosurgery **11** (2016), no. 4, 361.

[120] Nathan M Newmark, *A method of computation for structural dynamics*, Journal of the engineering mechanics division **85** (1959), no. 3, 67–94.

[121] S.V. Patankar and D.B. Spalding, *A calculation procedure for heat, mass and momentum transfer in three dimensional parabolic flows*, International J. on Heat and Mass Transfer **15** (1972), 1787–1806.

[122] M. Pernice and M. D. Tocci, *A multigrid-preconditioned Newton-Krylov method for the incompressible Navier-Stokes equations*, SIAM J. Sci. Comput. **23** (2001), no. 2, 398–418.

[123] Alfio Quarteroni, Fausto Saleri, and Alessandro Veneziani, *Analysis of the Yosida method for the incompressible Navier-Stokes equations*, J. Math. Pures Appl. (9) **78** (1999), no. 5, 473–503.

[124] ______, *Factorization methods for the numerical approximation of Navier-Stokes equations*, Comput. Methods Appl. Mech. Engrg. **188** (2000), no. 1-3, 505–526.

[125] Thomas Richter, *Fluid-structure interactions: models, analysis and finite elements*, Vol. 118, Springer, 2017.

[126] EM Rønquist, *A domain decomposition solver for the incompressible Navier-Stokes equations*, 1994 Workshop on Spectral Element Methods, North Carolina State University.

[127] Torgeir Rusten and Ragnar Winther, *A preconditioned iterative method for saddlepoint problems*, SIAM J. Matrix Anal. Appl. **13** (1992), no. 3, 887–904.

[128] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comp. **7** (1986), 856–869.

[129] Youcef Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Comput. **14** (1993), no. 2, 461–469.

[130] M. Schäfer, S. Turek, F. Durst, E. Krause, and R. Rannacher, *Benchmark computations of laminar flow around a cylinder* (Ernst Heinrich Hirschel, ed.), Flow simulation with high-performance computers II. Notes on Numerical Fluid Mechanics, vol. 48, Vieweg+Teubner Verlag, 1996.

[131] David Silvester, Howard Elman, David Kay, and Andrew Wathen, *Efficient preconditioning of the linearized Navier-Stokes equations for incompressible flow*, 2001, pp. 261–279.

[132] ______, *Efficient preconditioning of the linearized Navier-Stokes equations for incompressible flow*, J. Comput. Appl. Math. **128** (2001), no. 1-2, 261–279.

[133] David Silvester and Andrew Wathen, *Fast iterative solution of stabilised Stokes systems. II. Using general block preconditioners*, SIAM J. Numer. Anal. **31** (1994), no. 5, 1352–1367.

[134] B.F. Smith, P.E. Bjørstad, and W. Gropp, *Domain decomposition: Parallel multilevel methods for elliptic partial differential equations*, Cambridge University Press, 1996.

[135] G. D. Smith, *Numerical solution of partial differential equations*, Third, Oxford Applied Mathematics and Computing Science Series, The Clarendon Press, Oxford University Press, New York, 1985. Finite difference methods.

[136] Gerhard Starke, *Field-of-values analysis of preconditioned iterative methods for nonsymmetric elliptic problems*, Numerische Mathematik **78** (1997Nov), no. 1, 103–117.

[137] Govindjee S. Taylor R. L., *FEAP - finite element analysis program - version 8.5 user manual* (2017).

[138] The HDF Group, *Hierarchical Data Format, version 5*, 1997-2020. http://www.hdfgroup.org/HDF5/.

[139] Andrea Toselli and Olof Widlund, *Domain decomposition methods: algorithms and theory*, Vol. 3, Springer, 2005.

[140] RS Tuminaro, CH Tong, JN Shadid, KD Devine, and DM Day, *On a multilevel preconditioning module for unstructured mesh Krylov solvers: two-level Schwarz*, International Journal for Numerical Methods in Biomedical Engineering **18** (2002), no. 6, 383–389.

[141] S. P. Vanka, *Block-implicit multigrid solution of Navier-Stokes equations in primitive variables*, J. Comput. Phys. **65** (1986), no. 1, 138–158.

[142] R. Verfürth, *A multilevel algorithm for mixed problems*, SIAM J. Numer. Anal. **21** (1984), no. 2, 264–271.

[143] Irene E. Vignon-Clementel, C. Alberto Figueroa, Kenneth E. Jansen, and Charles A. Taylor, *Outflow boundary conditions for three-dimensional finite element modeling of blood flow and pressure in arteries*, Comput. Methods Appl. Mech. Engrg. **195** (2006), no. 29-32, 3776–3796.

[144] Irene E Vignon-Clementel, CA Figueroa, KE Jansen, and CA Taylor, *Outflow boundary conditions for 3d simulations of non-periodic blood flow and pressure fields in deformable arteries*, Computer methods in biomechanics and biomedical engineering **13** (2010), no. 5, 625–640.

[145] Homer F. Walker and Peng Ni, *Anderson acceleration for fixed-point iterations*, SIAM J. Numer. Anal. **49** (2011), no. 4, 1715–1735.

[146] Andrew Wathen and David Silvester, *Fast iterative solution of stabilised Stokes systems. I. Using simple diagonal preconditioners*, SIAM J. Numer. Anal. **30** (1993), no. 3, 630–649.

[147] Gabriel Wittum, *Multi-grid methods for Stokes and Navier-Stokes equations*, Numerische Mathematik **54** (1989), no. 5, 543–563.

[148] Yuqi Wu and Xiao-Chuan Cai, *A parallel two-level method for simulating blood flows in branching arteries with the resistive boundary condition*, Comput. & Fluids **45** (2011), 92–102.

[149] ———, *A fully implicit domain decomposition based ALE framework for three-dimensional fluid-structure interaction with application in blood flow computation*, J. Comput. Phys. **258** (2014), 524–537.

[150] Attila Zsaki, Daniel Rixen, and Marius Paraschivoiu, *A substructure-based iterative inner solver coupled with Uzawa's algorithm for the Stokes problem*, International journal for numerical methods in fluids **43** (2003), no. 2, 215–230.

Erklärung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie - abgesehen von unten angegebenen Teilpublikationen - noch nicht veröffentlicht worden ist, sowie, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde.

Die Bestimmungen der Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Prof. Dr. Axel Klawonn betreut worden.

Teilpublikationen

- Alexander Heinlein, Christian Hochmuth and Axel Klawonn, *Monolithic Overlapping Schwarz Domain Decomposition Methods with GDSW Coarse Spaces for Incompressible Fluid Flow Problems*, SIAM J. Sci. Comput. 41 (2019), no. 4, C291-C316
- Alexander Heinlein, Christian Hochmuth and Axel Klawonn, *Reduced dimension GDSW coarse spaces for monolithic Schwarz domain decomposition methods for incompressible fluid flow problems*, International Journal for NumericalMethods in Engineering 121 (2020), no. 6, 1101-1119.
- Alexander Heinlein, Christian Hochmuth and Axel Klawonn, *Fully algebraic two-level overlapping Schwarz preconditioners for elasticity problems*, accepted for publication to the proceedings of the ENUMATH 2019 conference, Springer LNCSE, April 2020.

(Christian Hochmuth)