Technische Universität München

Munich School of Engineering

Bachelor's Thesis of Alexander Rusch

Lehrstuhl für Wissenschaftliches Rechnen, Fakultät für Informatik

Extending SU^2 to fluid-structure interaction via preCICE

Author: Alexander Rusch

Examiner: Prof. Dr. Hans-Joachim Bungartz

Supervisor: Dipl.-Math. Benjamin Uekermann

Date of submission: 29. April 2016

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst zu haben. Ich habe keine anderen Quellen und Hilfsmittel als die angegebenen verwendet.

Ort, Datum

Alexander Rusch

Abstract

Partitioned fluid-structure interaction (FSI) simulations involve a fluid solver, a solid solver and a tool, which manages the coupling of the former two. In this thesis, the computational fluid dynamics suite *Stanford University Unstructured* (SU^2) is linked with the multiphysics coupling library *Precise Code Interaction Coupling Environment* (preCICE). Therefore, a C++ adapter is developed, which is integrated into the source code of SU^2. Despite recently added, intrinsic fluid-structure interaction functionalities of SU^2, the coupling with preCICE is reasonable as it allows to flexibly choose any partner code, including well-validated, commercial solvers. The coupling approach is successfully tested with two- and three-dimensional, generic scenarios, as well as quantitatively validated with a well-known FSI benchmark problem. Finally, the adapter is used to simulate a single wire of a brush seal under turbulent flow conditions with up to 230 processes on the fluid domain. This exemplifies the suitability of the realized coupling for real-world applications of larger scale. Moreover, this thesis includes a description on how to practically integrate the developed adapter into SU^2 and subsequently, build the fluid solver with preCICE.

Zusammenfassung

Partitionierte Ansätze zur Simulation von Fluid-Struktur-Interaktion umfassen einen Strömungslöser, einen Strukturlöser und ein Programm, das die Kopplung der beiden Ersteren handhabt. In dieser Arbeit wird der numerische Strömungslöser *Stanford University Unstructured* (SU^2) mit der multiphysikalischen Kopplungsbibliothek *Precise Code Interaction Coupling Environment* (preCICE) verbunden. Dazu wird ein Adapter in C++ entwickelt, der in den Quellcode von SU^2 eingebettet wird. Obwohl SU^2 kürzlich um intrinsische Fähigkeiten zur Fluid-Struktur-Interaktion (FSI) erweitert wurde, ist die Kopplung mit preCICE sinnvoll, da hiermit die Flexibilität geboten wird, einen beliebigen Partnerlöser für die Simulation zu wählen. Dies beinhaltet auch gut validierte, kommerzielle Löser. Die Kopplung wird erfolgreich anhand von zwei- und dreidimensionalen, generischen Szenarios getestet und zudem mittels eines bekannten FSI Benchmark-Tests quantitativ validiert. Schließlich wird der Adapter verwendet, um einen einzelnen Draht einer Bürstendichtung unter turbulenten Strömungsbedingungen zu simulieren, wobei das Strömungsgebiet mit 230 Prozessen parallel berechnet wird. Exemplarisch wird damit die Eignung der entwickelten Kopplung für größere, praxisrelevante Anwendungsfälle nachgewiesen. Diese Arbeit beinhaltet außerdem eine Beschreibung zur Integration des Adapters in SU^2 und zur anschließenden Installation des Strömungslösers mit preCICE.

Contents

Acknowledgements

I want to thank my supervisor Benjamin Uekermann for his valuable, professional advice and his admirable effort to introduce me to preCICE, as well as the topics of fluid-structure interaction, software development and high-performance computing. His patience and descriptive explanations made my fast progress possible.

Moreover, I want to thank Hans-Joachim Bungartz, who not only gave me the opportunity to work at the Chair of Scientific Computing, but mentored me during my whole bachelor studies from the first semester forth and helped me, whenever I had questions regarding my career. Also, he granted me access to the computational resources, which were necessary to run the simulations.

Thanks go also out to the Munich School of Engineering and the "Verein der Förderer des Computational Engineering e.V." for funding my participation in the "International Symposium and Winter-School on Modeling, Adaptive Discretizations and Solvers for Fluid-Structure Interaction" in Linz, January 2016.

To my friends and my family, who unconditionally supported me during creation of this thesis: Thank you so much. This work would not have been possible without you.

Finally, for never letting me down, always believing in me and supporting my studies: Thank you, Mum and Dad.

Acronyms

ALE Arbitrary Lagrangean-Eulerian.

AME Added Mass Effect.

API Application Programming Interface.

CFD Computational Fluid Dynamics.

CGNS CFD General Notation System.

CPS Conventional Parallel Staggered.

CSM Computational Solid Mechanics.

CSS Conventional Serial Staggered.

FEM Finite Element Method.

FSI Fluid-Structure Interaction.

FVM Finite Volume Method.

HPC High-Performance Computing.

MPI Message Passing Interface.

NN Nearest-Neighbor.

NP Nearest-Projection.

NSE Navier-Stokes Equations.

PDE Partial Differential Equations.

preCICE Precise Code Interaction Coupling Environment.

RANS Reynolds-Averaged Navier Stokes.

RBF Radial Basis Functions.

SU^2 Stanford University Unstructured.

TCP/IP Transmission Control Protocol/Internet Protocol.

XML Extensible Markup Language.

List of Figures

1. Introduction

Airbags, parachutes and aircraft wings. Submarines, wind turbine blades and the human circulatory system. All of these have something in common: Their physical behavior is dominantly governed by the mutual influence of fluid and solid materials. E.g. an airbag inflates due to expansion of a gas. The gas exerts forces on the airbag membrane and causes it to deform. This deformation represents a change of the physical boundary conditions for the fluid and thus, influences the flow field in reverse. Due to the flow field changes, the fluid forces acting on the membrane may vary again. This two-way coupled physical nature is highly complex, but advances in computational science and engineering over the past decades made tackling these multiphysical problems by means of numerical simulations feasible. This allows to analyse technical systems with respect to fluid-structure interaction (FSI) and possibly improve them. It also helps to understand complex FSI processes in nature and can, for instance, even be used to estimate rupture risks in the cardiovascular system. However, efficient tools are needed to simulate real-world scenarios, as the complexity of FSI problems implies high computational effort. One way of solving them is to simulate fluid and solid domains with respective single-physics solvers and link these by a third component, which respects the interaction aspects. Such a *partitioned approach* is chosen in this thesis in order to connect the computational fluid dynamics (CFD) solver Stanford University Unstructured (SU^2) with the multiphysics coupling library Precise Code Interaction Coupling Environment (preCICE). For this purpose, a C++ coupling adapter is developed in this work, which is integrated into SU^2. This allows to subsequently choose any preferred structure solver to execute FSI simulations with. The modern and promising open-source fluid solver SU^2 is well-validated and developed with emphasis on large-scale simulations, which predestines it for coupling with preCICE. As this fluid solver becomes more and more established, the coupling performed in this thesis might be interesting and useful to a fairly wide community of researchers and engineers. Recently, SU^2 was extended to FSI simulations by a newly-implemented, intrinsic structural solver. However, coupling the fluid solver with preCICE still offers many sophisticated features that SU^2 does not provide itself, such as elaborate coupling algorithms, data-mapping methods for non-matching meshes and fully-parallel simulation runs. Moreover, the coupling approach of this work does not limit users to a single structural solver but preserves the flexibility to choose any coupling partner, including highly elaborate, commercial solvers. Finally, the coupling procedure is successfully tested with scenarios of different dimensionality (2D, 3D) and complexity, including the well-known, challenging FSI benchmark test *FSI3* proposed in [39].

Concerning mathematical notation in this thesis, I use both index notation (e.g. v_i for velocity) and direct tensor notation (e.g. $\boldsymbol{v}$ for velocity). In the latter case, tensors of first and second order are denoted by **bold symbols**. No tensors of higher order are included in this work. Furthermore, all coordinate systems are Cartesian and denoted either by (x, y, z) or (x_1, x_2, x_3). I mostly stick with typical conventions when explaining mathematical expressions in order to conform with corresponding literature as far as possible.

The thesis starts with an introduction to FSI problems with emphasis on physical aspects in Chapter 2. This provides the reader with necessary knowledge to understand the complexity of FSI. Since this thesis concentrates on numerical simulations, Chapter 3 focuses on how to computationally treat these problems efficiently and gives an introduction to numerical problems of partitioned FSI simulations. The coupling approach chosen in this work is categorized, its details are explained and alternative solution methods are stated. Chapter 4 familiarizes the reader with the two software suites, which are relevant for coupling, i.e. the coupling tool preCICE and the fluid solver SU^2. The development of the adapter itself, its embedment into SU^2 and strategies for implementing the coupling are extensively explained in Chapter 5. Successful validation of the adapter is presented in Chapter 6 and a first real-world application is simulated. Chapter 7 shortly summarizes the findings and outcome of this work and gives an outlook to future work on this topic. Eventually, two appendices give further practical information on how to integrate the adapter in SU^2 (Appendix A) and subsequently build the fluid solver with preCICE (Appendix B).

2. Mathematical and Physical Basics of Fluid-Structure Interaction Problems

In this part I give an introduction to the mathematical and physical properties of FSI problems. First of all, in Section 2.1, I briefly explain why the application of continuum mechanics is valid for this kind of problems. A motivation of different kinematic descriptions of motion related to computational meshes follows in Section 2.2. Finally, in Section 2.3, I define the fluid and structure domain, as well as their common interface.

2.1 Continuum Assumption

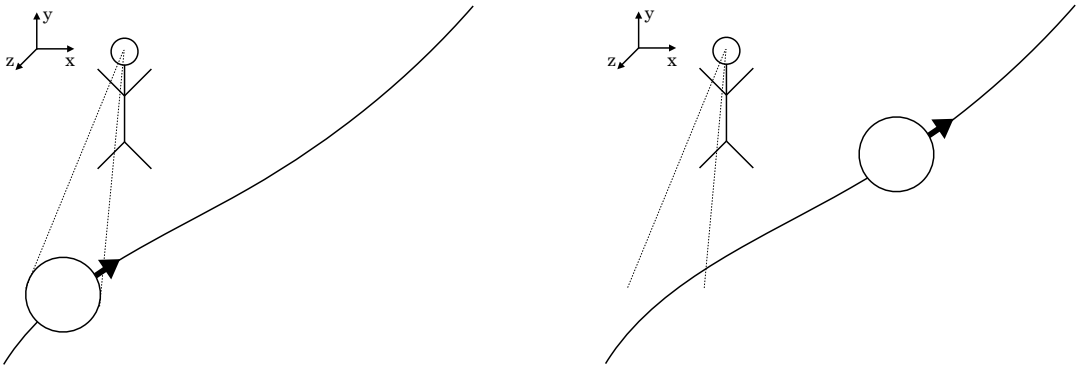
Fluids as well as solids are composed of molecules and atoms on a (sub-)microscopic level. However, describing a certain volume of solid or fluid material by explicitly characterizing every single molecule or even atom is way too extensive: For example, a volume of V = 1 cm^3 of liquid water with molar volume V_m = 18.0182 cm^3/mol ([24]) at a temperature of T = 25°C and an absolute pressure of p = 1 atm contains

$$\mathrm{n} = \frac{V}{V_m} N_A \approx 3.34 \cdot 10^{22} \tag{2.1}$$

molecules with $N_A = 6.022 \cdot 10^{23}\ mol^{-1}$ being the Avogadro constant ([29]). Thus a molecule-by-molecule description is not feasible, even for such relatively small volumes of material. Furthermore, it is also not necessary as the typical relevant time and spatial scales perceivable in everyday life exceed the microscopic ones by far. Rather, a so called *continuum approach* is used: Since properties (e.g. position, velocity, etc.) of neighboring atoms and molecules differ only slightly, it is sufficient to consider only averaged values of these quantities. In that sense, although in reality materials become discrete at some level of refinement, for the purpose of larger-scale descriptions like those typically used in CFD or computational solid mechanics (CSM), fluids and solids are considered as continuous materials. They can be continually subdivided (to an infinitesimally small extent), yet their properties do not loose their smoothness. Based on this assumption, in the following the term *(material) particle* is not used to denote molecules or atoms but rather an infinitesimally small volume of continuum material. Within the scope of this thesis the continuum assumption is valid as larger-scale motions are considered.

2.2 Description of Motion

A fluid in motion is considered and interaction with a solid occurs via deformations of the latter due to viscous and/or inviscid forces exerted on the structure by the fluid. The resulting change in shape of the structural material also implicates geometrical changes of the fluid domain. This yields different flow behavior in reverse. Thus, it is necessary to represent kinematic and dynamic processes formally. Therefore, some thoughts concerning the motion of the before mentioned continuum particles must be made. In classical continuum mechanics there are two different perspectives ([26]): The Eulerian description, which is discussed in Section 2.2.1, and the Lagrangian point of view, which I explain in Section 2.2.2. Those two perspectives can be combined to the arbitrary Lagrangean-Eulerian (ALE) method, described in Section 2.2.3.



(a) Eulerian observer and particle at time t_1 (b) Eulerian observer and particle at time $t_2 > t_1$

Figure 2.1: The Eulerian observer does not move and focuses on the same spatial point as time passes. The dashed lines indicate the monitored point and underline that the observer does not follow the particle. This situation is shown from a Eulerian point of view (implied by the Eulerian coordinate system in the upper left corner). The Eulerian coordinate system, from which the observer monitors the situation, originates at his head. It is not shown here for the sake of readability.

2.2.1 Eulerian Perspective

In a Eulerian perspective the change of quantities of interest (e.g. density, velocity, pressure) is observed at spatially fixed locations. In other words, a Eulerian observer does not vary the point of focus during different time steps. This is depicted intuitively in Figure 2.1. Located at a certain point in Euclidean space, the observer always focuses on the same location, no matter where particles may move. Thus, in Eulerian description, quantities can be expressed as functions of a fixed location as well as time. This may be denoted by

$$\Theta = \tilde{\Theta}(x, y, z, t), \tag{2.2}$$

where Θ is a quantity of interest and $\tilde{\Theta}$ denotes it in a Eulerian point of view. (x, y, z) represent a fixed position in Euclidean space and t refers to time. Clearly, different particles can occupy the spatial location, which the observer focuses on, at different instances of time. Therefore, in general no direct information regarding the change of quantities of a single particle is available when motion is described in a Eulerian perspective ([26]).

Eventually, a description of motion is needed not only for single particles and points in space, but rather computational domains and meshes being central aspects of FSI problems. A computational mesh can be interpreted as a number of observers distributed across the domain of interest and connected so as to form a grid with nodes. If particles of the underlying domain move, a purely Eulerian mesh does not change the positions of its nodes throughout the whole mesh at different instances of time. This is graphically shown in Figure 2.2. Since this behavior of the mesh is independent of large-scale movements of particles, it is the typical choice for CFD problems, where in general fluid particles move throughout the whole computational domain. However, this approach also has its drawbacks as the level of refinement of the mesh is crucial to the accuracy of computations because it defines to what extent small-scale changes can be observed. If a mesh is of a much coarser scale than the motion occurring in the underlying domain, the motion cannot be resolved ([10], [26]).

2.2.2 Lagrangian Point of View

On the contrary, a Lagrangian description implies that the observer focuses on a specific particle and follows it, regardless of the speed and distance it may travel. Therefore, provided that the particle moves, changes of the quantities of interest are observed at different spatial locations. The Lagrangian observer tracks a particle and moves with it, as illustrated in Figure 2.3.

The motion of the particle as well as all other quantities of interest, can therefore be described by *reference coordinates* (or *material coordinates*) in Euclidean space, (X, Y, Z), uniquely identifying the observed particle at a reference configuration. Often $t = 0$ is chosen as reference but in general any time instance can be used. Once the particle to be observed is specified, the Lagrangian observer only registers

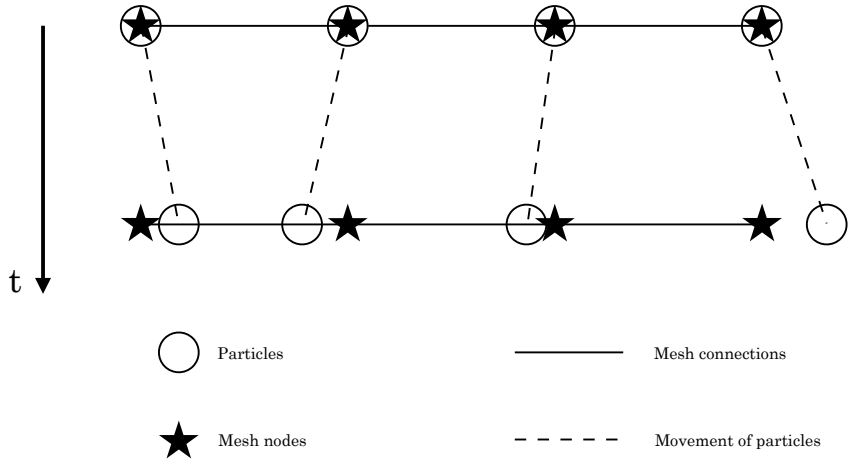
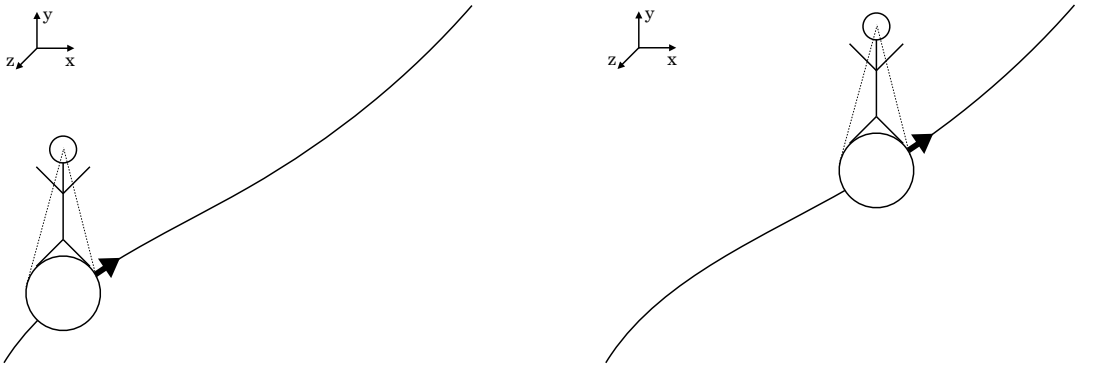


Figure 2.2: The Eulerian mesh nodes remain at the same spatial points as time passes, i.e. the Eulerian mesh does not move. However, particles may change their positions. t denotes the time axis. Figure adapted from [10].



(a) Lagrangian observer and particle at time t_1 (b) Lagrangian observer and particle at time $t_2 > t_1$

Figure 2.3: The Lagrangian observer "stands" on the particle and moves with it as time passes. The dashed lines indicate that the observer only focuses on this very particle. This situation is shown from a Eulerian point of view (implied by the Eulerian coordinate system in the upper left corner). The Lagrangian coordinate system, from which the observer monitors the situation, originates at his head (and moves with the observer). It is not shown here for the sake of readability.

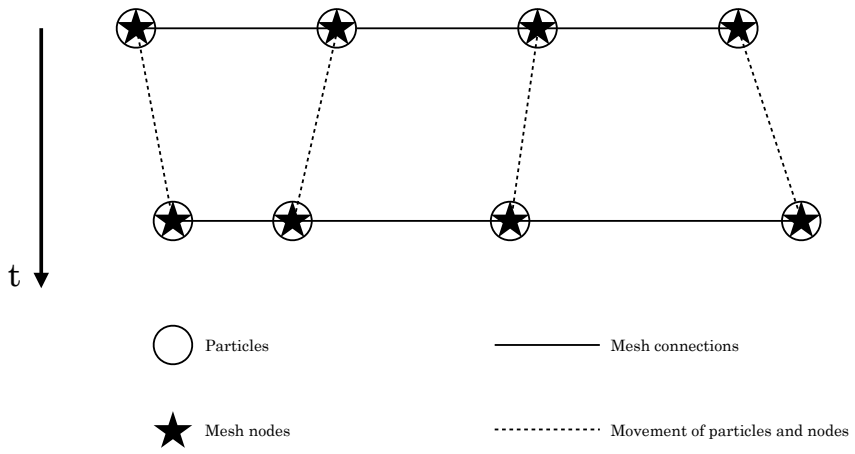


Figure 2.4: Particles may change their positions as time passes. The nodes of the Lagrangian mesh follow the respective particles in order to coincide with them. t denotes the time axis. Figure adapted from [10].

changes concerning this one particle as time passes. Thus, quantities of interest can be described as

$$\Theta = \hat{\Theta}(X, Y, Z, t). \tag{2.3}$$

In contrast to the Eulerian perspective (Equation 2.2) however, the obtained information is strictly limited to the specific observed particle (implied by the usage of the capital reference coordinate variables). Thus, no convective fluxes appear in a Lagrangian description. No general information about a specific, fixed Euclidean point in space is available ([26]).

Again, computational domains and meshes are considered: At a reference instance of time, usually at the beginning of a simulation, mesh nodes are attached to the underlying material particles. As time passes and particles move, the mesh nodes move with them causing the mesh to deform (except for cases in which all particles move smoothly with equal speed and distance). Figure 2.4 depicts such a situation. As it can be seen, the mesh nodes always coincide with their respective particles. A drawback of this Lagrangian technique is that large-scale and irregular motions lead to distortions of the computational mesh, which yields smaller accuracy in simulations as a consequence of the strictly enforced tracking. However, from this point of view, small-scale motions, which often occur in solids, can easily be observed without the need of using extremely fine meshes, which would be necessary in case a Eulerian perspective was used. This results in reduced computational effort. Therefore, in general, the Lagrangian description is the method of choice for CSM problems ([10], [26]).

Eulerian and Lagrangian descriptions are related. A mapping between them can be derived if the motion is known:

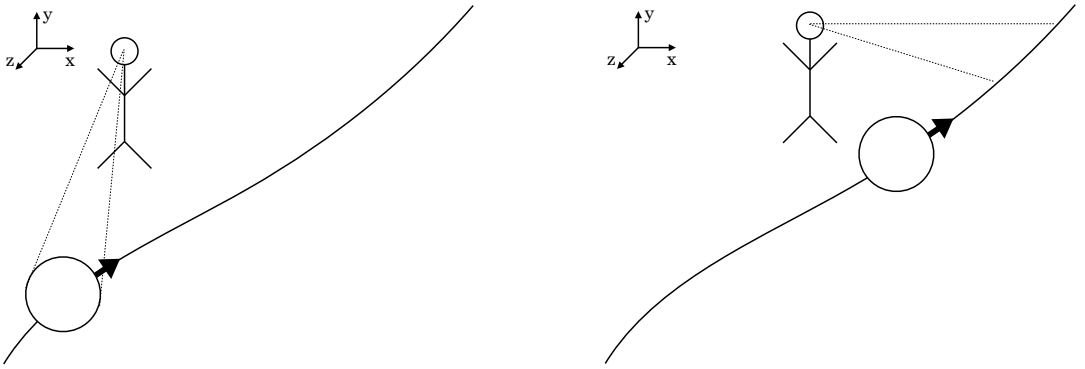
$$x_i = X_i + u_i(X_i, t) \quad \forall i = 1, 2, 3. \tag{2.4}$$

Equation 2.4 can be explained as follows: The Eulerian, spatial position $\boldsymbol{x}$ of a particle at time t is the position of this particle at its reference configuration $\boldsymbol{X}$ plus the displacements $\boldsymbol{u}$ that it traveled since the point of time of the reference state ([26]).

2.2.3 ALE Method

Finally, I explain the ALE approach, a combination of the Eulerian and Lagrangian perspective widely used for FSI problems. As the name implies, an ALE observer can arbitrarily decide whether to move the point of focus or not. Furthermore, the observer is in no way restricted to the movement of particles. Figure 2.5 depicts such a situation. The observer moves independently of the particle motion.

By analogy with the Eulerian and Lagrangian meshes before, an ALE mesh is considered as it can be seen from a Eulerian perspective in Figure 2.6. Mesh nodes can move almost arbitrarily regarding the motion of the underlying particles. The only restriction is, that node movements should not distort the mesh too much as this leads to inaccuracy. It is reasonable to allow the nodes to follow moving particles up to a certain extent, which is defined by mesh quality criteria. Since this approach does not allow to directly



(a) ALE observer and particle at time t_1 (b) ALE observer and particle at time $t_2 > t_1$

Figure 2.5: The ALE observer may start from the Eulerian configuration and then move independently of the particle motion. This situation is shown from a Eulerian point of view (implied by the Eulerian coordinate system in the upper left corner). The ALE coordinate system, from which the observer monitors the situation, originates at his head (and moves with the observer), which is not shown here for the sake of readability.

link mesh motion and material particle motion, a new unknown is introduced to such a problem, namely the relative movement between the ALE mesh and the material domain. This approach is especially interesting for FSI problems because it is an alternative description to the Eulerian frame for the fluid domain. As it is further explained in Section 2.3.3, fluid and solid material have to follow the moving interface between them for physical reasons. Since the solid domain is usually described in a Lagrangian view, there is no problem with keeping the solid mesh attached to the FSI interface. However, if a purely Eulerian approach was used for the fluid domain, movements of the interface would lead to gaps between the wet surface and the fluid mesh. Therefore, in ALE methods the fluid mesh nodes at the interface always move with it. This can be interpreted as Lagrangian fashion of the approach, as fluid mesh nodes follow the fluid particles sticking to the interface, while the rest of the fluid mesh is allowed to move in such way that mesh distortions are kept minimal in order to preserve computational accuracy. Since preserving mesh regularity refers more to a Eulerian approach, the choice of the name ALE becomes apparent ([32], [10]).

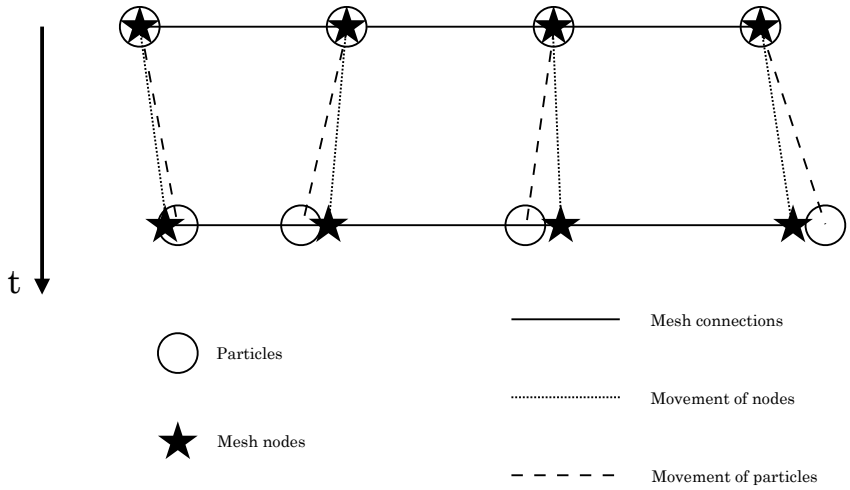


Figure 2.6: The particles may change their positions as time passes. The nodes of the ALE mesh neither have to stay at the same spatial points nor have to follow the particles necessarily. They can move independently of the particles. t denotes the time axis. Figure adapted from [10].

2.3 Domains and Interface

As the name *fluid-structure interaction* implies, this type of problems is determined by the fluid and solid domain, covered in Sections 2.3.1 and 2.3.2, respectively. Furthermore, their interaction is of importance, which underlines the necessity of suitable coupling conditions at the domain common interface. The interface is also referred to as *wet surface*. Its formal definition is stated in Section 2.3.3.

2.3.1 Fluid Domain

In the following, all of my considerations are limited to viscous Newtonian flows in the compressible regime as this kind of model is the only relevant one for this thesis. Nevertheless, I want to point out that throughout the FSI community also incompressible and inviscid flow regimes are commonly considered, depending on the type of physical problem.

The before mentioned kind of flow is described by the *Navier-Stokes equations (NSE)*, which I consider in the general three-dimensional case in a Eulerian description. They consist of the continuity equation (conservation of mass, Equation 2.5a), the momentum equation (conservation of momentum, Equation 2.5b) and the energy equation (conservation of energy, Equation 2.5c). The equations are shown in index notation. Repeated indices imply Einstein's summation convention. For a detailed explanation of this convention, I refer to [26]. The NSE are usually derived by applying Newton's Law to a fluid control volume and an elaborate derivation can be found in [14]. The equations are taken from [26] and [14].

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_j}(\rho u_j) = 0 \tag{2.5a}$$

$$\frac{\partial}{\partial t}(\rho v_i) + \frac{\partial}{\partial x_j}(\rho v_i v_j + p\delta_{ij} - \tau_{ji}) = 0 \quad \forall i = 1, 2, 3 \tag{2.5b}$$

$$\frac{\partial}{\partial t}(\rho e_0) + \frac{\partial}{\partial x_j}(\rho v_j e_0 + v_j p + q_j - v_i \tau_{ij}) = 0 \tag{2.5c}$$

ρ denotes density, t time, $\boldsymbol{x}$ the spatial dimensions, $\boldsymbol{v}$ flow velocities in all dimensions and p pressure. δ_{ij} is the Kronecker Delta ($\delta_{ij} = 1$, if $i = j$ and $\delta_{ij} = 0$ otherwise, for further explanations see [26]), $\boldsymbol{\tau}$ the viscous stress tensor, e_0 total energy (per unit mass) and $\boldsymbol{q}$ heat flux (via conduction). For a Newtonian fluid the viscous stress tensor is given by

$$\tau_{ij} = -\frac{2}{3}\mu \frac{\partial v_k}{\partial x_k}\delta_{ij} + 2\mu S_{ij} = -\frac{2}{3}\mu \frac{\partial v_k}{\partial x_k}\delta_{ij} + \mu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) \quad \forall i, j = 1, 2, 3. \tag{2.6}$$

With μ being the dynamic viscosity and $\boldsymbol{S}$ the rate of deformation tensor (symmetric part of the velocity gradient $\nabla \boldsymbol{v}$):

$$S_{ij} = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) \quad \forall i, j = 1, 2, 3. \tag{2.7}$$

In order to form a closed set of these partial differential equations (PDE), it is necessary to choose a conductive heat flux model (usually Fourier's Law), specify the caloric and thermodynamic equations of state and finally, choose appropriate initial and boundary conditions for the problem ([17], [14], [26]).

2.3.2 Solid Domain

As described in Section 2.2.2, in solid mechanics usually a Lagrangian point of view is used (as is here), because particles do not travel as far as they do in fluid dynamical problems. Also, the structural model explained in this section is limited to the *Saint Venant-Kirchhoff* model, which is very common since it is capable of handling large deformations often occurring in FSI problems ([18]). The model assumes that the solid material is homogenous, meaning that mechanical properties of a particle of the body do not depend on the location of the particle. In other words, these properties are the same throughout the whole solid domain. Moreover, isotropy is assumed, such that the direction in which a stress is applied to the solid does not matter, as the mechanical properties of the body are the same in all directions ([26], [45]).

The following explanation is a short version, since this thesis does not focus on the solid mechanical aspects of FSI problems. It is inspired by and partly taken from [18] and [26], where derivations are given to a more detailed level.

By analogy with the NSE (see Equations 2.5), the description of the solid arises from considering a control volume and applying Newton's Law to it. A typical equation of motion in the form *Mass* · *Acceleration* = *Forces* can be derived (again, the general three-dimensional case is considered):

$$\rho \frac{\partial^2 u_i}{\partial t^2} = \frac{\partial S_{ij}}{\partial X_j} + \rho f_i \quad \forall i = 1, 2, 3. \tag{2.8}$$

Here, ρ corresponds to the structure density, the second derivative of the displacements $\boldsymbol{u}$ with respect to time t to the acceleration of a material particle and $\boldsymbol{S}$ to the *second Piola-Kirchhoff stress tensor*. $\boldsymbol{X}$ denotes the material coordinates as mentioned before. In this case, also the volume force $\boldsymbol{f}$ is considered, because gravity can often not be ignored for solid materials. Again, a constitutive law must be taken into account, defining the relationship between stress and strain:

$$S_{ij} = \lambda E_{kk} \delta_{ij} + 2\mu E_{ij} \quad \forall i, j = 1, 2, 3, \tag{2.9}$$

with

$$E_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial X_j} + \frac{\partial u_j}{\partial X_i}\right) + \frac{1}{2}\frac{\partial u_k}{\partial X_i}\frac{\partial u_k}{\partial X_j} \quad \forall i, j = 1, 2, 3. \tag{2.10}$$

Note that $\boldsymbol{E}$ is the *Lagrangian (finite) strain tensor*. The latter summand in Equation 2.10 is non-linear and can be neglected for small deformations, leading to the *Lagrangian infinitesimal strain tensor*. However, since we deal with possibly large deformations, this relationship remains non-linear. δ_{ij} again refers to the Kronecker delta. λ and μ are material parameters named *Lamé constants*. They relate directly to *Young's modulus* E and *Poisson ratio* ν, which are of more practical use[1]. Their relationship is given as follows:

$$E = \frac{\mu(3\lambda + 2\mu)}{\lambda + \mu}, \tag{2.11a}$$

$$\nu = \frac{\lambda}{2(\lambda + \mu)}. \tag{2.11b}$$

Either (E, ν) or (λ, μ) are enough to fully characterize this material under specific assumptions:

- The solid is linearly elastic and isotropic.
- The strain tensor $\boldsymbol{E}$ is symmetric,
- as well as the stress tensor $\boldsymbol{S}$.
- Furthermore, a scalar, positive definite strain energy density function (not shown in this shortened explanation) relating stress and strain tensor via a potential formulation exists.

For more sophisticated explanations the interested reader may refer to [26].

Eventually, as for the fluid domain, appropriate initial and boundary conditions must be specified.

2.3.3 Interface and Interaction

Since FSI problems are centered on the interaction of the fluid and solid domain, their common interface is of vital importance. A schematic picture of a sample situation at the wet surface is shown in Figure 2.7. Note that all quantities related to the solid and fluid domain, as well as the interface are subscripted with S, F and FS, respectively. Also, in order to avoid simultaneous usage of sub- and superscripts, I switch from index to direct notation in this section. In order to couple both domains via the interface in a physically correct way, some conditions must be met. These conditions are commonly used for FSI problems. However, in this specific case they are taken from [18] and [20].

First of all, fluid and solid domain should neither overlap, nor separate from each other at the interface as there can be no space occupied by fluid and solid particles at the same time and "empty" space is

[1] Note that E (Young's modulus, scalar) and $\boldsymbol{E}$ (Lagrangian strain tensor) are not the same.

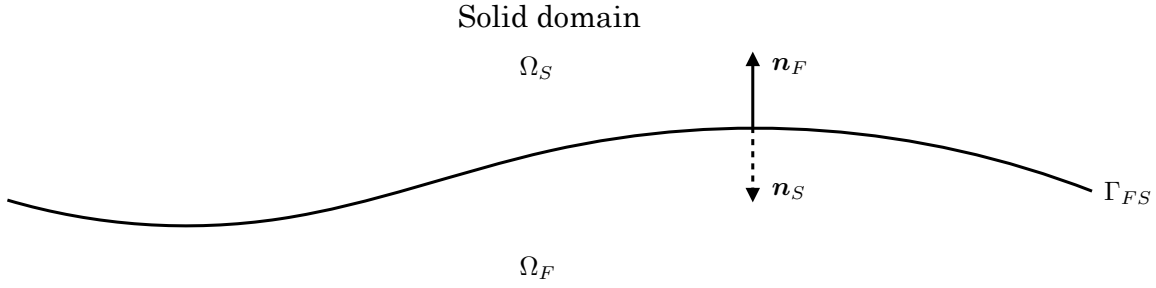


Figure 2.7: Fluid (Ω_F) and solid domain (Ω_S) meet at the interface Γ_{FS}. The outward normal vectors $\boldsymbol{n}_F$ and $\boldsymbol{n}_S$ point in opposite directions.

non-physical. Furthermore, for a viscous fluid the flow velocity at the domain boundary has to be equal to the boundary velocity itself, which is called *no-slip condition*. Together, this results in the kinematical requirement that the displacements of fluid and solid domain, as well as their respective velocities have to be equal at the wet surface (denoted by Γ_{FS}):

$$\boldsymbol{x}_F = \boldsymbol{u}_S \quad \text{on } \Gamma_{FS}, \tag{2.12a}$$

$$\boldsymbol{v}_F = \frac{\partial \boldsymbol{u}_S}{\partial t} \quad \text{on } \Gamma_{FS}. \tag{2.12b}$$

For inviscid fluids only velocity components normal to the wet surface have to be equal to the structural velocity as the fluid may slip freely in tangential direction at any boundary.

It is not sufficient to consider only kinematic constraints at the interface. In addition, an equilibrium of forces at the wet surface is needed such that it is not torn apart by resultant forces. Force vectors originate from the stresses at the interface and the outward normal vectors of fluid and solid domain, respectively. They have to be equal and opposite leading to the dynamic coupling condition:

$$\boldsymbol{\sigma}_F \cdot \boldsymbol{n}_F = -\boldsymbol{\sigma}_S \cdot \boldsymbol{n}_S \quad \text{on } \Gamma_{FS}. \tag{2.13}$$

$\boldsymbol{\sigma} \in \mathbb{R}^{3\times3}$ denotes the stress tensor and $\boldsymbol{n} \in \mathbb{R}^3$ the outward normal vector of the fluid and solid domain. Note that here viscous as well as inviscid stresses are included.

3. Computational Aspects of FSI Simulations

Subsequent to the basic physics of FSI problems drawn up in the last chapter, this section goes into detail on computational treatment, which also allows for FSI techniques to be categorized. Two fundamentally different classes of procedures for solving FSI simulations have been established, namely *monolithic* and *partitioned* approaches. They are discussed in Section 3.1. Since this thesis focuses on the latter kind, a characterization of weak and strong coupling in partitioned approaches is described in Section 3.2. Subsequently, in Section 3.3, a different way of categorizing FSI techniques is presented, which is based on *conforming* or *non-conforming mesh treatment* of respective solver strategies. The chapter is closed by Section 3.4, having a look at the added mass effect (AME) - a stability issue arising from problems with strong interaction in partitioned FSI simulations.

3.1 Monolithic and Partitioned Approaches

First of all, I want to point out that the terms *monolithic* and *partitioned* are interpreted differently throughout FSI literature. However, in this thesis, they are only used in the hereafter explained sense.

Again, let the name *fluid-structure interaction* serve as a motivation for this section: On the one hand, it implies a single (but coupled) physical problem while on the other hand, its clear multiphysical characteristic is emphasized. This is also reflected in the monolithic and partitioned approaches, respectively. However, monolithic and partitioned approaches should not be interpreted as solely oppositional. There exist solver strategies for which the boundaries between the two approaches blur.

Monolithically treating both fluid and solid domain implies that they are solved simultaneously. This means that one multiphysics solver deals with a single system of equations describing fluid, solid and their coupling. Figure 3.1 depicts this strategy schematically. Such monolithic solvers are designed specifically for the sole purpose of solving FSI problems. Therefore, a high level of specialization can be realized. Simultaneously treating both flow and structural equations often results in good numerical stability of the calculations. Furthermore, monolithic approaches solve the system of equations exactly, meaning there are no errors (other than those which are inherent to numerical techniques) introduced by this form of numerical treatment. However, development of such solvers from scratch requires a lot of coding work and is often cumbersome ([8], [38], [20], [18]).

In contrast, partitioned approaches make use of existing single-physics solvers. The FSI problem is split into a fluid and solid problem, which are both treated by their respective solvers separately, while a third software module, the coupling component, incorporates the interaction aspects. It communicates forces

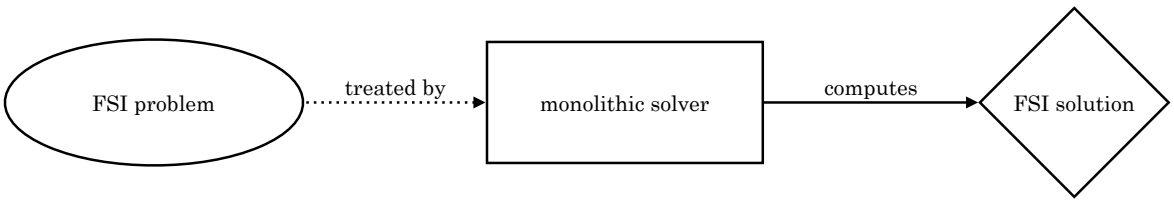


Figure 3.1: The FSI problem is undivided and solved by a single monolithic solver, yielding the multiphysics solution.

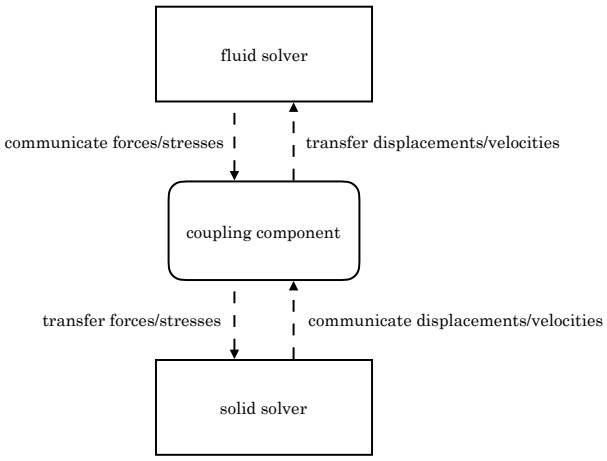


Figure 3.2: Exchange of displacements/velocities as well as forces/stresses is managed by the coupling component. All exchanged data is limited to the wet surface, i.e. these quantities are only communicated for the respective nodes at the FSI interface.

or stresses (dynamic data) calculated by the fluid solver at the wet surface to the solid component and exchanges displacements or velocities (kinematic data) computed by the solid solver at the interface to the fluid component in return ([18], [38]). A schematic sketch of this situation is shown in Figure 3.2. More detailed and practical explanations about the coupling component are given in Section 4.1. For now it is sufficient to know that the coupling component exchanges kinematic and dynamic data between the single-physics solvers in order to preserve the coupled nature of the overall problem. In the end, fluid and structural solutions together yield the FSI solution. By analogy with the monolithic approach, it is graphically depicted in Figure 3.3. A big advantage of this approach is that existing solvers for the fluid and solid problem can be reused, ranging from commercial to academic and open-source codes. Especially in the commercial sector, these are often highly elaborate solvers with decades of experience in their particular single-physics fields. They typically support very sophisticated solution techniques. Also, those solvers are usually well-validated and compared to monolithic procedures the programming efforts are lower for partitioned approaches, as only the coupling of the existing solvers has to be implemented rather than the solvers themselves. Nevertheless, these advanced solvers can only be of good use for FSI simulations if the coupling component is sufficiently precise ([8], [38], [20]).

3.2 Weakly and Strongly Coupled Partitioned Strategies

Partitioned strategies for solving FSI problems can be subclassed into *weakly* and *strongly coupled* approaches. They are also referred to as *explicit* and *implicit* methods in this thesis. Note that this nomenclature is in no way fully consistent throughout FSI literature. However, all usage of these terms in this thesis is limited to the sense of the explanations given in this section.

The distinction between explicit and implicit techniques is based upon the question, how often solutions for the fluid and solid subproblem are computed within one time step and also, how frequently the relevant kinematic and dynamic quantities are exchanged. For weakly coupled strategies solving is done a certain, fixed number of times (often only once) per time step and data may not even be communicated after each discrete time instance. In general, this is not sufficient to regain the monolithic ("exact") solution of the FSI problem as the coupling conditions are not enforced within each time step. Thus, no balance between fluid and structural domain with respect to energy, forces and displacements at the interface can be guaranteed ([8], [18], [38], [2]). However, this coupling strategy can still yield good results if the interaction between fluid and solid is rather weak (further explanations about the strength of the interaction follow in Section 3.4). E.g. in aeroelastic simulations, where small displacements of the structure appear within single time steps, the flow field is influenced by the structural displacements only to a little extent ([13], [8], [2]).

In contrast, strongly coupled strategies make use of *subiterations* possibly resulting in multiple computations of the separate solvers and exchanges of the interface coupling quantities (as a reminder, see Figure

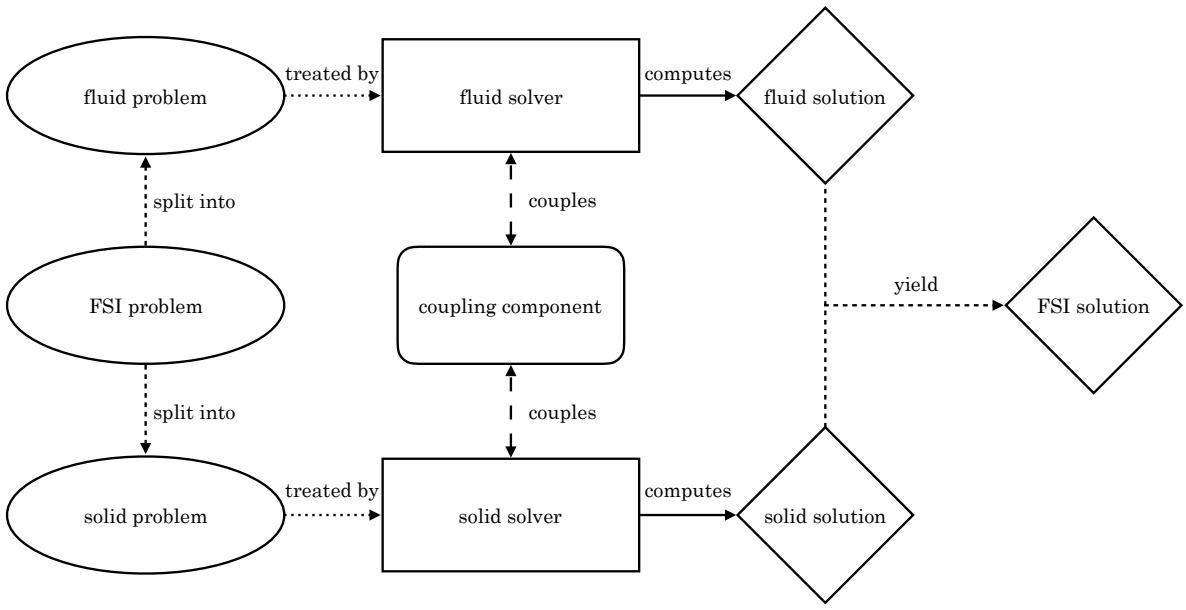


Figure 3.3: In a partitioned approach, the FSI problem is divided into a fluid and solid subproblem. These are treated by a fluid and structure solver, respectively, while a coupling component ensures the interaction of the domains (see also Figure 3.2 for a more precise explanation). Separate solutions are computed, which together yield the solution of the original problem. Note that the arrows yielding the FSI solution are just of conceptual manner and should not be interpreted as some sort of solution merging technique.

3.2) per physical time step. However, acceleration techniques are necessary to converge the underlying coupling equation system. The coupling conditions at the wet surface are enforced in each time step up to a convergence criterion. If the criterion is not met sufficiently, another subiteration within the same time instance is calculated. Therefore, the solution can approximate the monolithic solution to an arbitrary accuracy as the convergence criterion can be chosen as strict as needed. Such a method is in general applicable to both FSI problems, which can be solved by weakly coupled approaches and those, for which explicit procedures fail due to dominant interaction. However, strongly coupled algorithms are usually used in the latter case - when weak coupling reaches its limits - since the implicit approach requires more computational effort ([18], [38], [2]).

Figure 3.4 sums up the categorization of different coupling techniques mentioned in this and the previous section. They are ordered with respect to stability and programming effort.

3.3 Conforming and Non-Conforming Mesh Methods

In this section, FSI methods are classified by means of two different mesh treatment procedures: conforming or non-conforming techniques. The basic question is, whether fluid and solid mesh need to align with each other at the FSI interface or not. Unless stated otherwise, the explanations of this section are taken from [20]. Note that some aspects of conforming mesh methods are already included in the previous sections without explicitly mentioning so, in order to develop a better understanding of partitioned FSI simulations.

Conforming mesh methods usually consist of three major subtasks, namely computation in the fluid and solid domain, as well as interface and mesh movement. They require both fluid and structural meshes to conform to the wet surface, because the coupling conditions are applied via the interface as physical boundary conditions for the respective domains. This does not necessarily imply node-to-node matching of fluid and structure meshes at the interface. This must hold for all time instances, which means that both fluid and structural grids need to be moved in case deformations of the solid appear. This is a simple task for the solid mesh, since it is usually expressed in a Lagrangian fashion anyway. However, as a typical Eulerian fluid mesh would not follow the interface motion, the necessity of the ALE method as discussed in Section 2.2.3 becomes apparent. Also, mesh smoothing techniques need to be introduced in

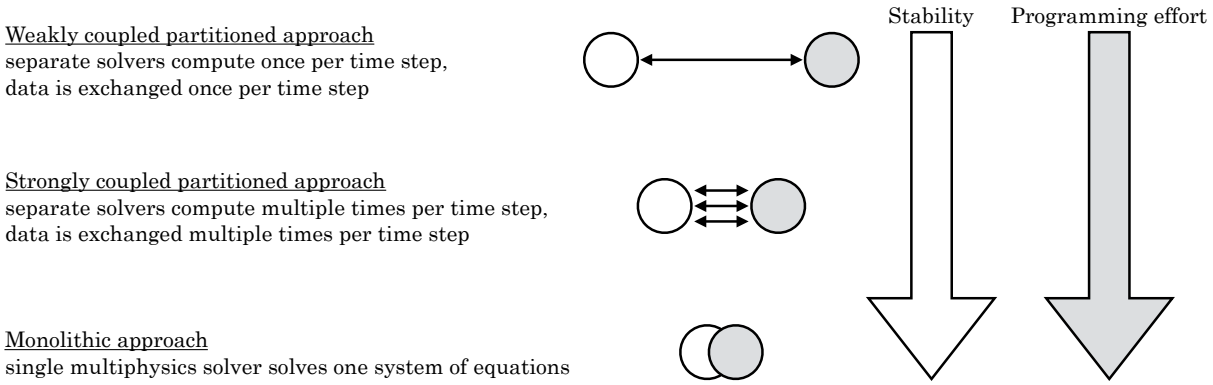


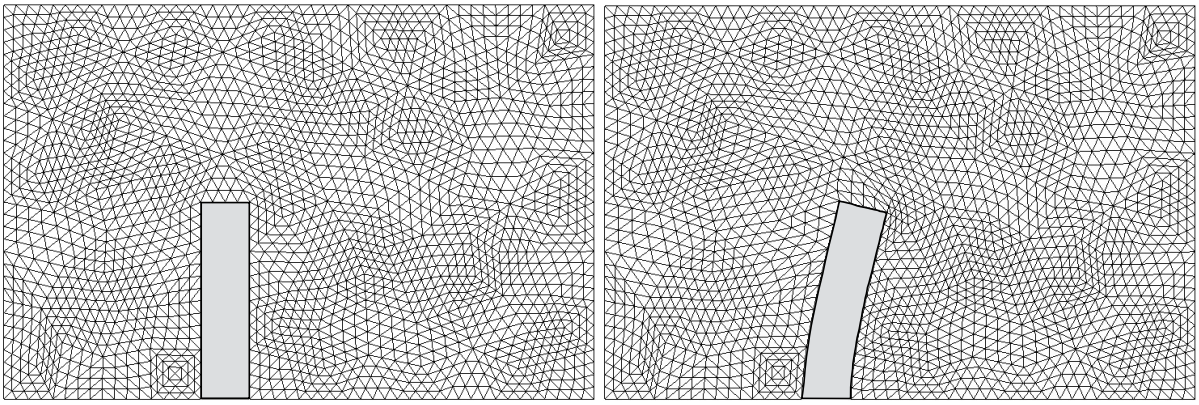
Figure 3.4: Summary of the different discussed coupling approaches. The circles represent the fluid and solid solver for the two partitioned approaches. Their overlap in the monolithic case implies that a single solver is used for both domains. The arrows between the solvers indicate single (weakly coupled partitioned approach) and multiple (strongly coupled partitioned approach) data exchanges per time step. Figure adapted from [38].

order to prevent quality losses of the fluid mesh in terms of distorted elements. These irregularities lead to accuracy loss in simulations. In Figure 3.5, a conforming mesh is shown at two different points in time. At the first instance (Figure 3.5a) the solid is undeformed and therefore, also the fluid mesh remains in its initial configuration. In contrast, at the second point in time (Figure 3.5b) the solid is deformed and the fluid mesh conforms to the displaced wet surface. Consequently, also mesh smoothing is applied.
There is a wide variety of such mesh updating procedures. Some common techniques compute the mesh movement by considering mesh edges as springs (*(torsional) spring analogy*), solving the *Laplace equation* or solving a *pseudo-structural* system of equations (see e.g. [18], [43], [20] and their respective references for further explanations of these techniques). Conforming mesh strategies are widely, but not exclusively used in partitioned FSI approaches. Furthermore, they typically also utilize the ALE method ([20]).

In contrast, in non-conforming mesh strategies all interface conditions are directly imposed as constraints on the flow and structural governing equations. Therefore, it is possible to use non-conforming meshes for fluid and solid domains as they remain geometrically independent from each other. Thus, also mesh smoothing techniques are obsolete [43]. Figure 3.6 depicts such a situation. By analogy with Figure 3.5, again the initial configuration (Figure 3.6a) and an instance when the solid is deformed (Figure 3.6b) are shown. It is clearly visible that the fluid mesh does not conform to the wet surface as all nodes stay at the same position regardless of the solid deformation.
This approach is mostly used in *immersed methods*. The considerations in this section are limited to them, as they are also very common for FSI simulations. Coupling is imposed via additional force-equivalent terms appearing in the model equations of the fluid, enforcing the kinematic and dynamic conditions. These FSI forces are computed from the structural model, which is dealt with separately together with tracking the position of the interface. The forces represent the effects of a boundary or body being immersed in the fluid domain (leading to *immersed boundary* and *immersed domain methods*). A purely Eulerian mesh can be applied to the whole computational domain for solving the fluid equations, since the force-equivalent terms are dynamically added in a spatially specific manner to those locations, which are currently occupied by the structure. After solving the fluid equations, forces exerted on the solid at the wet surface are computed and used as input for the structural solver, which still employs a Lagrangian mesh. Subsequently, the deformation of the solid material is calculated and the displacement of the FSI interface is fed back to the fluid model in form of updated force-equivalent terms ([31], [20], [43]).

3.4 Stability Issue: Added Mass Effect

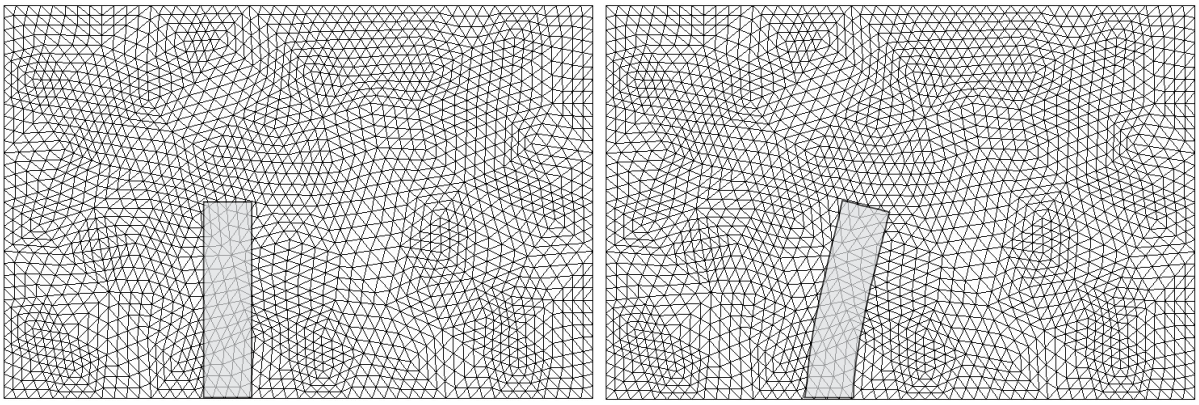
To conclude this chapter about computational aspects of FSI simulations, the AME is briefly described. Explanations of this effect can be found in a great variety throughout FSI literature, typically explicated for specific solver strategies or flow regimes (see e.g. [5], [42], [15], [2]). Therefore, in the scope of this thesis only a short phenomenological introduction to the concept of *added mass* and numerical problems arising from it is given. However, this suffices to focus on both weakly and strongly coupled



(a) Conforming fluid mesh before deformation of the structure.

(b) Conforming fluid mesh after deformation of the solid domain.

Figure 3.5: A conforming fluid mesh is shown in undeformed (Figure 3.5a) and deformed configuration (Figure 3.5b) from a Eulerian point of view. The solid is represented by the gray rectangle. Fluid nodes on the wet surface stick to the interface even after deformation. The mesh fully conforms to the geometry of the structural domain. The rest of the fluid mesh is smoothed in order to avoid highly distorted elements. The fluid mesh does not penetrate the structural domain at any time. The solid mesh is not shown for a better general view. Note the change in shape of elements, which are close to the displaced, upper corners of the solid.



(a) Non-conforming fluid mesh before deformation of the structure.

(b) Non-conforming fluid mesh after deformation of the solid domain.

Figure 3.6: A non-conforming fluid mesh is shown in undeformed (3.6a) and deformed configuration (3.6b) from a Eulerian point of view. The solid is represented by the gray, transparent rectangle. Fluid nodes throughout the mesh do not move upon deformation of the solid domain, also they are not aligned with the solid. Thus, the mesh is non-conforming. There is no need for mesh smoothing techniques. The fluid mesh underlies the structural domain at all times. The solid mesh is not shown for the sake of simplicity and readability.

partitioned approaches, which are practically relevant in this thesis. The AME is inherent to partitioned FSI approaches as the single-physics fields are not continuously coupled but interaction only occurs at a finite number of discrete time instances, when coupling quantities are exchanged.

As already mentioned in Section 2.3.3, there can be no gaps between structure and fluid. Also their respective particles cannot occupy the same spatial locations simultaneously. Thus, if the solid is moved, also fluid particles move. Changing the state of motion of the structural component consequently requires taking into account inertial effects not only of the solid itself, but also of the surrounding fluid, which artificially rests for the span of a single structure solver time step. In more descriptive words: Moving the solid also implies moving fluid particles close to the solid. Therefore, the structure behaves more inert due to artificially added mass ([42], [2]). Since inertia is dependent on mass and therefore density, the AME is also. More precisely, it is dependent on the ratio (M_A) of structural (ρ_S) und fluid density (ρ_F) ([42], [5]):

$$M_A = \frac{\rho_S}{\rho_F}. \tag{3.1}$$

This ratio is often used to describe how *strong* the interaction between solid and fluid is. For cases, in which the solid density is much higher than the fluid density ($M_A \gg 1$), this effect does not dominantly influence the FSI problem (*weak interaction*). But as fluid and structure densities approach each other ($M_A \approx 1$) or the fluid becomes even denser than the solid ($M_A < 1$), its consideration is crucial (*strong interaction*) and imposes stability limits on partitioned solution techniques ([5], [42], [2]). Note that the AME is not only governed by the density ratio of Equation 3.1 but also by geometric properties of the problem, the stiffness of the solid ([5]) and the speed of sound in the flow domain ([42]). Nonetheless, for the sake of simplicity and intuitiveness, explanations in this thesis are mostly limited to the density ratio.

In general, the AME is of bigger concern for incompressible flows than for compressible regimes. From a physical point of view, deformations of the structural domain can be interpreted as perturbations for the flow field. In compressible flows the speed of sound (speed at which perturbations propagate through the flow) is finitely large. Thus, the influence of a geometrical change of the fluid domain caused by deformations of the solid is locally limited during a certain period of time. In contrast, in incompressible flows the speed of sound is infinitely large, hence all perturbations propagate through the flow without time delay. Therefore, regardless of how much time has passed since a perturbation, the whole flow field is directly affected ([42], [5]).

In the following it is assumed that a weakly coupled algorithm allows computation of the fluid and solid solution only once per time step. In addition, coupling data is also exchanged once per time instance. In contrast, a strongly coupled solver does the same at least twice per time increment (for a reminder see Section 3.2 and compare Figure 3.4). As can be shown, in the compressible case a more dominant AME can be compensated for by reducing the time step size of strongly coupled, partitioned solution algorithms. This however, does not hold for the incompressible regime, where even in the limit of vanishing time step size strongly coupled, partitioned algorithms might fail ([42]). These observations are consistent with the above mentioned physical explanation.

First of all, considering compressible flows, indeed, the lack of repeated subiterations in weakly coupled partitioned techniques leads to a strict limit for the density ratio (of Equation 3.1) due to the fact that the interface conditions are not enforced and energy balance at the wet surface is generally not given. If that limit is exceeded the algorithm fails due to instability ([2]). Likewise, in such a case a strongly coupled partitioned algorithm converges slowly, resulting in possibly many necessary subiterations, which is computationally costly. Yet it does not become unstable, given that the time step size is chosen sufficiently small. Reducing the time step size to an arbitrarily small extent cannot stabilize a weakly coupled approach if the stability criterion on the density ratio is not met ([15]). Conversely, the convergence rate of strongly coupled algorithms increases by the same factor, by which the time step size decreases, meaning that in the limit of vanishing time step size the monolithic solution is obtained ([5], [42]).

In the incompressible case however, a strict stability limit exists for both weakly and strongly coupled algorithms. It is independent of the size of time increments[1]. Furthermore, in order for an implicit method to achieve the monolithic solution (assuming its convergence is given, i.e. the before mentioned stability limit is not exceeded) the number of subiterations must be increased as time step size decreases ([15], [42]).

[1] Recall that perturbation propagation is infinitely fast in incompressible flows.

4. Utilized Software Packages

Approaching the more practical aspects of this work, the used software packages are introduced. First of all, the coupling component preCICE is described in Section 4.1. Afterwards, I explain the fluid solver being coupled - SU^2 - in Section 4.2.

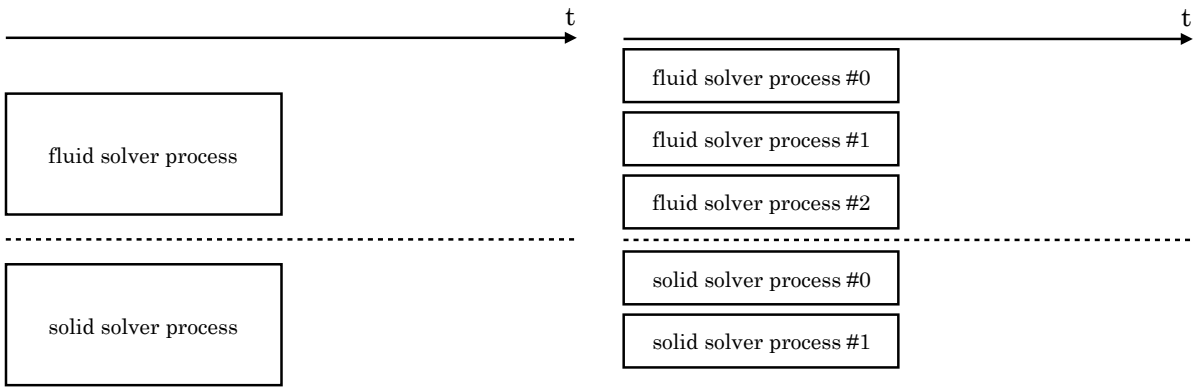
4.1 preCICE - Flexible Coupling of Existing Solvers for Multi-physics Simulations

Unless stated otherwise, the information concerning preCICE is taken from [18] and [4].

preCICE stands for Precise Code Interaction Coupling Environment[1]. It is an open-source coupling tool (see also Section 3.1 for general explanations concerning the coupling component) developed for partitioned multiphysics simulations such as FSI or *fluid-structure-acoustic interaction* mainly at the Chair of Scientific Computing (Technical University of Munich) and the Institute for Parallel and Distributed Systems (University of Stuttgart). The code can be accessed via Github[2]. Its goal is to enable a nearly plug-and-play approach for coupling existing solvers in a black-box fashion, meaning that only restricted solver information is available and coupling is limited to involving the interface nodes. Thus, no information regarding derivatives and element shape functions at the interface is available. This restriction of solver internal data is often the case for closed-source, commercial solvers. To preserve the plug-and-play character, code changes of the single-physics solvers should be as minimally invasive as possible. Therefore, a high-level application programming interface (API) is necessary, which preCICE offers in various programming languages, such as C, C++, Fortran90/95 and Fortran2003. preCICE takes care of all interaction-related activities such as different coupling strategies (Section 4.1.1), communication between the solvers (see Section 4.1.2), data mapping for non-matching meshes (Section 4.1.3), convergence acceleration of implicit, partitioned techniques (for a reminder see Section 3.2) and possibly time stepping. Configuration is done via an extensible markup language (XML) file (extension *.xml*), which is not discussed in this thesis. Instead, the interested reader is referred to [18]. The software package is optimized for massively parallel systems ([41]) due to minimizing computation time bottlenecks and enabling both *interfield* and *intrafield parallelism*. Interfield parallelism means that e.g. in the case of FSI simulations both fluid and structural solver can be executed simultaneously, while intrafield parallelism describes the possibility to run the single-physics solvers themselves on multiple processes. The distinction is schematically depicted in Figure 4.1. preCICE is able to combine the possibilities of parallelism flexibly such that, for instance, fluid and solid problems can be solved concurrently (interfield parallelism), using multiple fluid solver processes (intrafield parallelism), but only a single solid solver process. This flexibility is important when it comes to optimizing overall runtimes of simulations and efficient utilization of computing resources, since computational effort should be spread as equally as possible among the computing capacities. Typically, the solid domain is much less computationally costly compared to the fluid field, so for runtime optimization often a set of a few solid processes but a larger number of fluid processes is required.

Fluid and solid solvers do not have to use the same time step size. In order to communicate data between the single-physics solvers, however, it is necessary that the solvers align with each other at certain times. Thus, if the solvers advance with different time increments, preCICE might have to enforce time step sizes so that data communication at a common point in time is possible. The situation is depicted schematically in Figure 4.2.

[1] See `http://precice.org`, also a list of currently coupled solvers is given there.
[2] See `https://github.com/precice/precice`.



(a) Interfield parallelism: Fluid and solid solver run simultaneously as time passes. It is assumed that no intrafield parallelism is employed. Note that fluid and structural solvers do, in general, not have the same runtime. For the sake of simplicity it is implied here though.

(b) Intrafield parallelism: Both fluid and solid problems are solved by multiple solver processes, which run concurrently for the respective single-physics fields. Note that also interfield parallelism is shown here. For the sake of simplicity all processes have the same runtime.

Figure 4.1: Differentiation of inter- (4.1a) and intrafield parallelism (4.1b). This is only a schematic sketch and does not represent realistic solver execution times.

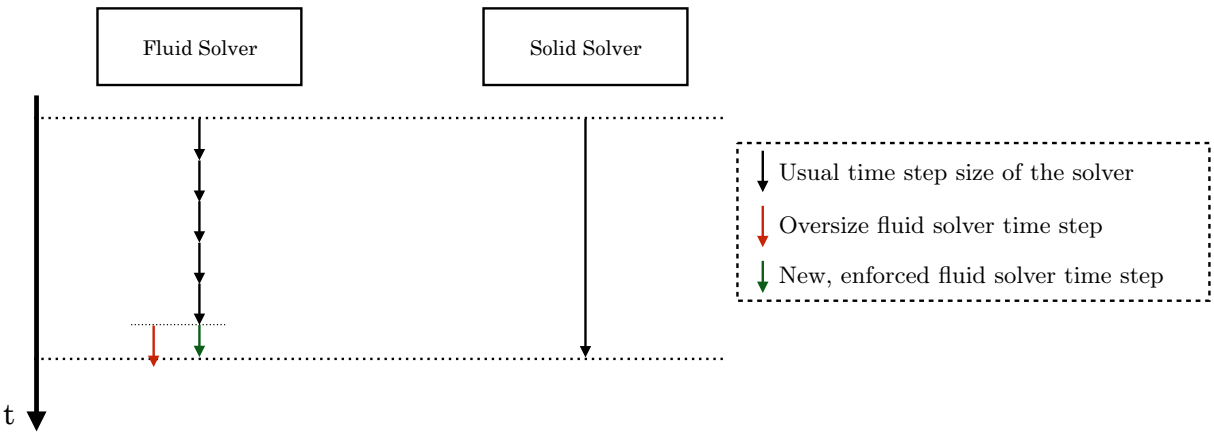


Figure 4.2: Time stepping control by preCICE: The fluid solver performs smaller time steps than the solid solver. At coupling instances (dotted, vertical lines), both solvers need to align. As the last time step of the fluid solver would exceed this instance, an adequate time increment is enforced by preCICE. The dashed box explains the used symbols and t denotes time.

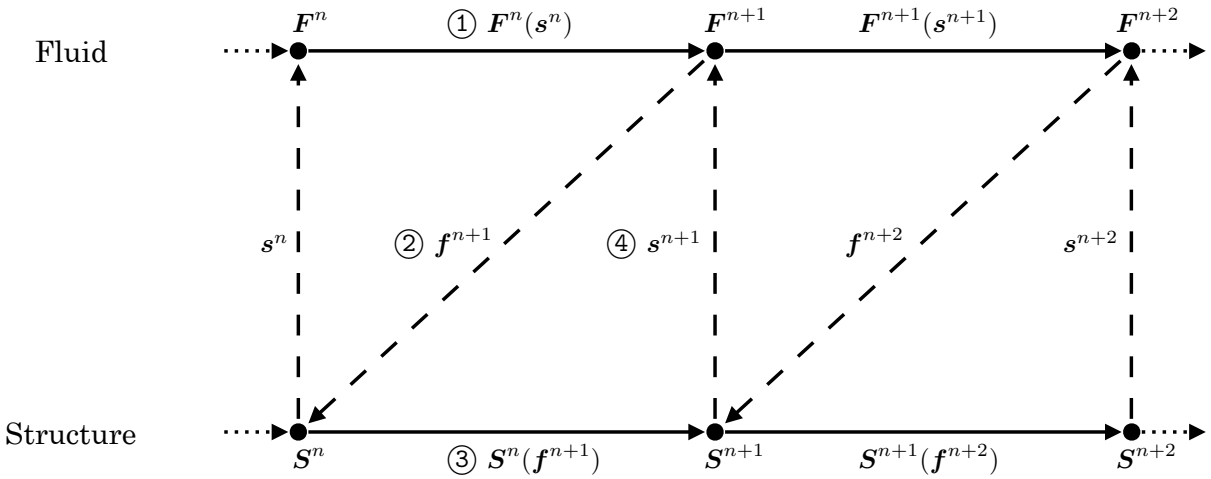


Figure 4.3: The CSS algorithm is shown for a complete computation cycle including: ① Explicitly obtaining the fluid solution of time instance $n+1$: $\boldsymbol{F}^n(\boldsymbol{s}^n)$ ② Communicating the dynamic data $\boldsymbol{f}^{n+1}$ to the structure solver ③ Implicit calculation of the solid solution of time step $n+1$: $\boldsymbol{S}^n(\boldsymbol{f}^{n+1})$ ④ Forwarding the kinematic data $\boldsymbol{s}^{n+1}$ to the fluid solver. Figure adapted from [18].

In the following, I give short explanations about the main tasks of preCICE without focusing on the actual code structure. The interested reader is referred to [18] and encouraged to have a look at the source code, where necessary.

4.1.1 Implemented Coupling Strategies

preCICE allows both for usage of explicit and implicit coupling techniques. See Section 3.2 to recapitulate the main differences of these approaches. Also, serial and parallel algorithms have been implemented, as well as elaborate procedures particularly suited for black-box coupling.

Concerning notation, for all algorithms explained in this section n denotes the current time step of the computation, $\boldsymbol{F}^n$ and $\boldsymbol{S}^n$ are operators representing the fluid and solid solver computation at time instance n, respectively. They yield the particular fluid and solid solutions $\boldsymbol{f}^{n+1}$ and $\boldsymbol{s}^{n+1}$ at time instance $n+1$.

Explicit, Serial Algorithm

A serial, weakly coupled algorithm implemented in preCICE is the conventional serial staggered (CSS) procedure ([18]). The algorithm is graphically depicted in Figure 4.3. The fluid solver uses the solid solution at the last time step to compute its current solution (explicit). In contrast, the solid solver needs the current flow solution to compute the structure solution at the same time instance (implicit). Because of this dependency, fluid and structural solvers are said to be executed in a *serial*, *staggered* way as they cannot run in parallel and their succession is strictly alternating ([12]).

Explicit, Parallel Algorithm

The conventional parallel staggered (CPS) algorithm is similar to the CSS procedure, but allows for parallel execution of both fluid and structure solvers and represents a parallel, explicit coupling algorithm implemented in preCICE ([18]). No implicit dependency as in the case of CSS is present. The succession of this algorithm is shown in Figure 4.4. Fluid and solid solver advance in parallel and exchange coupling data at the end of the time step. However, a drawback compared to the CSS algorithm is the loss of implicitly involving the two solvers, which yields a less stable procedure. Since fluid and solid solvers have to exchange coupling data after each iteration, the overall computation time of the scheme is dependent on the most time-consuming solver, which makes an efficient distribution of computational effort between flow and structure solvers crucial ([12], [18]).

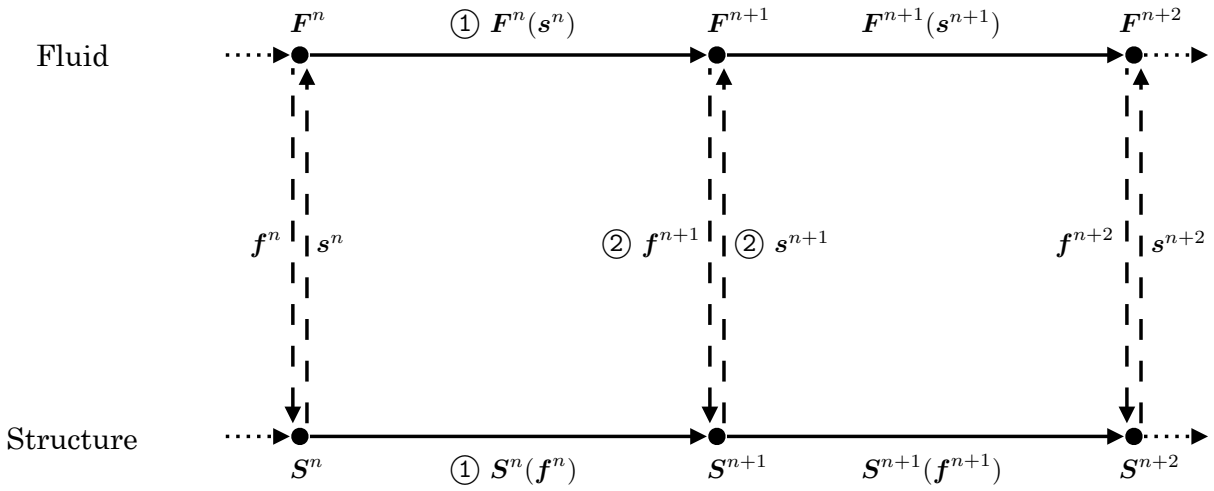


Figure 4.4: The CPS algorithm is shown for a complete computation cycle including: ① Explicitly obtaining the fluid and solid solution, respectively, of time instance $n+1$: $\boldsymbol{F}^n(\boldsymbol{s}^n)$, $\boldsymbol{S}^n(\boldsymbol{f}^n)$ ② Communicating the coupling data $\boldsymbol{f}^{n+1}$ and $\boldsymbol{s}^{n+1}$ to the particular other solver. Figure adapted from [18].

Implicit, Serial Strategies

In the following only one time step is discussed, unless stated otherwise. Therefore, all superscripts indicating the time step are neglected. However, subscripts are used to denote subiterations within one time instance.

Algorithm 1 Block Gauss-Seidel method with relaxation technique

1: $\boldsymbol{s}_0 = \boldsymbol{s}_p$
2: $k = 0$
3: **while** convergence criterion not met **do**
4: $\quad \boldsymbol{F}(\boldsymbol{s}_k) = \boldsymbol{f}_{k+1}$
5: $\quad \boldsymbol{S}(\boldsymbol{f}_{k+1}) = \tilde{\boldsymbol{s}}_{k+1}$
6: $\quad$ compute $\boldsymbol{s}_{k+1}$ by relaxation
7: $\quad k = k + 1$
8: **end while**

Several implicit algorithms are implemented in preCICE including a *Block Gauss-Seidel method* with either *constant* or *dynamic Aitken under-relaxation* (see Algorithm 1). The Block Gauss-Seidel method is basically an implicit extension of the CSS procedure employing a fixed-point iteration of the form:

$$\tilde{\boldsymbol{s}}_{k+1} = \boldsymbol{S} \circ \boldsymbol{F}(\boldsymbol{s}_k). \tag{4.1}$$

First of all, the fluid solver runs using the old structural solution. Afterwards, the updated fluid solution is used by the solid solver to compute the new structure solution. Note that $\tilde{\boldsymbol{s}}_{k+1}$ is used to indicate that the structural solution is solely obtained by the respective solvers without any modification like e.g. relaxation, while $\boldsymbol{s}_{k+1}$ indicates that such postprocessing has been applied to the solution. The fixed-point formulation allows to define a residual as shown in Equation 4.2, which can be used to obtain a scalar, absolute convergence criterion (see Equation 4.3), useful for close to zero values of the coupling quantities, when rounding errors become important:

$$\boldsymbol{r}_{k+1} = \boldsymbol{S} \circ \boldsymbol{F}(\boldsymbol{s}_k) - \boldsymbol{s}_k = \tilde{\boldsymbol{s}}_{k+1} - \boldsymbol{s}_k, \tag{4.2}$$

$$\|\boldsymbol{r}_{k+1}\| < \epsilon_{abs}. \tag{4.3}$$

$\|\cdot\|$ denotes the Euclidean norm. Also, a scalar, relative convergence criterion is defined in order to keep track of convergence between two subsequent subiterations:

$$\frac{\|\boldsymbol{r}_{k+1}\|}{\|\tilde{\boldsymbol{s}}_{k+1}\|} < \epsilon_{rel}, \tag{4.4}$$

where $0 < \epsilon_{rel,abs} < 1$. By analogy with the CSS algorithm, the solid solver depends implicitly on the updated solution of the fluid solver, thus not allowing for parallel execution of the single-physics solvers. Since the fluid solver needs a value of the structural solution as input, which is not available for the first iteration of a time step, a predictive value $\boldsymbol{s}_p$ (Line 1, Algorithm 1) is used. The convergence criterion mentioned in Line 3 can be understood as a combination of the absolute and relative limits defined in Equations 4.3 and 4.4.
Relaxation techniques as mentioned in Line 6 are used to stabilize the subiteration method. For the case of constant under-relaxation, the relaxed value of the kinematic coupling data is computed as a combination of the old, relaxed and the new, unrelaxed value:

$$\boldsymbol{s}_{k+1} = (1 - \omega)\boldsymbol{s}_k + \omega\tilde{\boldsymbol{s}}_{k+1}, \tag{4.5}$$

with $0 < \omega < 1$. Roughly speaking, values of ω close to 1 have little stabilizing effect, but result in fast convergence (assuming the method is stable and convergent), whereas for values close to 0 the stabilization is strong, yet convergence is slow and this leads to high computational effort as more subiteration steps are necessary. Consequently, the value should be chosen such that the subiteration process is stable and moreover, converges as fast as possible. Therefore, the choice of ω can be difficult in practice.
In contrast to constant under-relaxation, where the relaxation factor ω is fixed, dynamic Aitken relaxation allows for a new factor to be computed in each subiteration. Basically, the dynamic factor is computed from the last two subiteration solutions via linear extrapolation in order to find the solution of the fixed-point system with zero residual, analogous to a root-finding problem with the secant method. The algorithm for obtaining the dynamic Aitken factor is not further described here, as it exceeds the scope of this thesis. Refer to [18], [21] and [25] for details.

Procedures Predestinated for Black-Box Coupling

As coupling of black-box solvers is one of the major issues of preCICE, no Newton methods can be used for implicit coupling, since they require interface Jacobian information, which is typically not accessible for this kind of solvers. Therefore, algorithms which make use of approximations of the Jacobian are of importance. Two variants of these are implemented in preCICE, for serial and parallel usage. The serial algorithm is called IQN-ILS, which stands for Interface Quasi-Newton with Approximation of the Inverse of the Interface Jacobian Matrix by Least-Squares. In a sequential manner, fluid and structural solvers are executed, followed by the IQN-ILS algorithm, which modifies the structural solution such that the underlying fixed-point iteration converges. This solution is then fed back to the fluid solver and the computation circle starts over, as long as the fixed-point is not yet reached up to a specified convergence criterion.

The parallel procedure is named V-IQN, originating from a vectorial fixed-point formulation of the problem (displacements and forces are gathered in a single vector). Again the procedure is based on an Interface Quasi-Newton approach. The method allows for fluid and solid solver to be run simultaneously, before the V-IQN algorithm modifies the vector of displacements and forces such that the fixed-point iteration converges. If the convergence limit is not yet reached after a subiteration, these modified values are used as input for the next iteration of fluid and structure solvers, respectively.
I do not explain these elaborate methods further. Refer to [8] for the IQN-ILS method and to [40] and [27] for the V-IQN algorithm.

4.1.2 Communication Methods

Interfield parallelism is an important topic when it comes to high-performance computing (HPC) on massively parallel systems. But the execution time of the solver processes does not solely determine overall runtimes of simulations. A high level of parallelization also induces the necessity of efficient forms of communication between the distributed processes so that data transfer does not become a dominant bottleneck. preCICE offers three different means of communication based on files, sockets and message passing interface (MPI) ([18]). A fully parallel process-to-process communication approach (both via sockets and MPI) for preCICE is implemented in [37].

Files

File communication is a very basic form of communication. Solver processes write data to and read data from files, which are stored on the hard drive of a computer. Communication is limited to a one-to-one fashion, meaning that a single writer and a single reader communicate with each other. Each file is named uniquely in order to identify the writer and reader by their respective names. Furthermore, the file name contains a message counting number. In order to avoid writer and reader processing the same file simultaneously, the currently working process renames the file and hides it by that manner from the communication partner. Reading is implemented such that busy waiting is used to check for the correctly named file. This is a drawback of file communication as it blocks computational resources. Moreover, due to the inherent latency of the hard drive, the technique is not feasible for frequent data exchange. It is implemented in preCICE solely for testing purposes as it allows to easily trace back errors ([18]).

MPI

MPI ([16]) is typically used for parallelizing applications (intrafield parallelism) in a way that parallel processes run the same code, differing by the MPI rank/index[3]. However, in preCICE, MPI communication is used to exchange data between processes of different applications, namely the instances of the single-physics solvers. A big advantage of this communication method is that MPI is available on most scientific computers and offers qualities, which are relevant in the field of HPC, such as high data throughput and small latency. In preCICE, MPI communication can be set up either via a single communication space containing all executables (which are then subgrouped for the individual solvers) or via different spaces. This means of communication can be quite prone to incompatibilities of software with different implemented versions of MPI. Therefore, it may be necessary to adapt/change the underlying MPI versions of the respective single-physics solvers or of preCICE. Especially in the case of closed-source solvers, this adaption might not be possible and thus communication via MPI might have to be discarded. Communication is performed asynchronously (non-blocking), which is highly relevant for fully parallel process-to-process communication ([37], [18]).

Sockets

preCICE also supports communication via Transmission Control Protocol/Internet Protocol (TCP/IP) sockets. Although their usage is rather unconventional in HPC, they are used in preCICE as they are a very elaborate, well-known means of network communication and therefore, mostly bug-free. Furthermore, unlike in the case of MPI, different TCP/IP socket versions for different solvers to be coupled do usually not yield runtime incompatibilities. Again, the communication procedure is asynchronous (non-blocking) ([37]).

Gatzhammer shows in [18] that, indeed, MPI is the best-performing communication technique implemented in preCICE as it outperforms socket- and file-based communication especially for use cases of data exchange with higher numbers of nodes[4]. However, socket communication follows closely, such that both techniques are very well-suited for larger-scale simulations.

4.1.3 Data Mapping for Non-Matching Meshes

In FSI simulations, fluid and structure meshes do not necessarily coincide in a node-to-node manner at the wet surface. In fact, typically fluid and structural domains are spatially discretized to different levels of refinement. In most cases, fluid meshes are finer than solid grids, meaning that at the wet surface more fluid than structural nodes appear. This situation is sketched in Figure 4.5. Therefore, when coupling data needs to be exchanged at the wet surface, some mapping between fluid and solid nodes must be applied in order to be able to interpolate data between them. preCICE offers three different methods for this, namely *nearest-neighbor (NN)* and *nearest-projection (NP) mapping*, as well as an interpolation method based on *radial basis functions (RBF)* ([18]).

For all implemented procedures in preCICE, mapping can be applied in either *consistent* or *conservative* fashion, which must be chosen dependent on what quantities are exchanged at the wet surface (e.g. forces

[3] An identifier for each of the parallel processes.
[4] $4 \cdot 10^1$ to $4 \cdot 10^6$ nodes are tested.

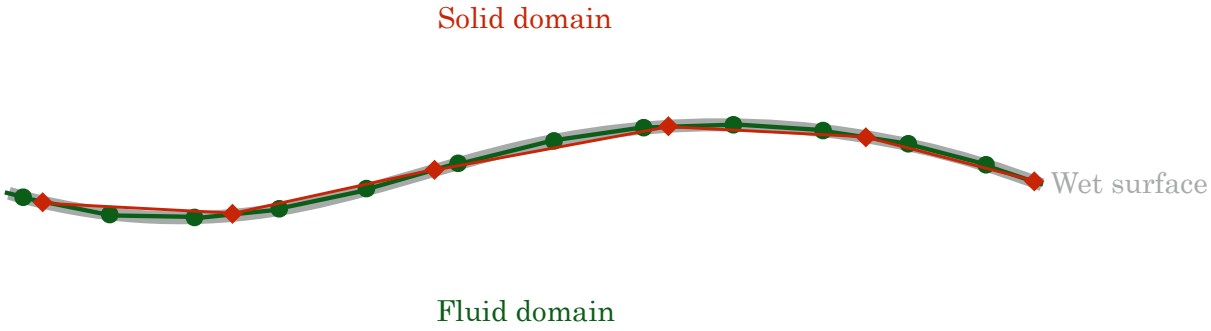
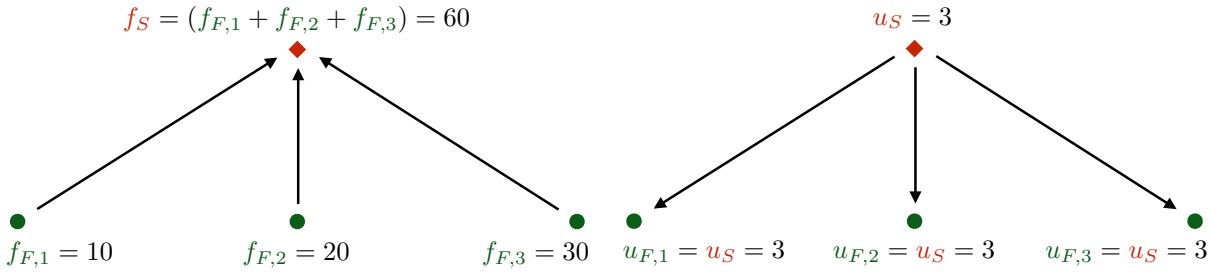


Figure 4.5: Fluid and structure domains are shown with their respective mesh discretizations at the wet surface in a two-dimensional case, i.e. the interface is a line. Nodes do not necessarily coincide and the fluid mesh is denser than the solid mesh. Mesh connections besides the edges at the wet surface are not shown for the sake of a clear view.

or stresses, displacements or velocities) ([18]). As is the case for the coupling described in this thesis (and most of the numerical testcases, which are shown), a fine fluid mesh and a coarse solid mesh meet at their common interface (as depicted in Figure 4.5). Forces need to be mapped from fluid to corresponding structural nodes and displacements in reverse from solid to fluid nodes. As the number of fluid nodes exceeds that of the structure, in general a single solid node is assigned to several fluid nodes[5]. If forces are mapped from these multiple fluid nodes to the solid node, all of the assigned fluid nodes contribute to the overall force value at the structural node in an additive manner. Such a mapping is called conservative, as the overall sum of the forces on both fluid and structure side at the wet surface remains constant, i.e. it is conserved. In contrast, when displacements are mapped from a single solid node to the fluid nodes, it is not useful to distribute the single displacement value among the fluid nodes such that the displacements of the fluid nodes sum up to the displacement of the solid node. Rather, all fluid nodes assigned to that single solid node experience the same displacement. Such a mapping is called consistent, as the mapped value is transferred exactly ([18], [7]). A schematic example of conservative and consistent interpolation is depicted in Figure 4.6 for the NN method. Conservative mapping of forces and consistent mapping of displacements is used in the coupling procedure performed in this thesis.



(a) Conservative mapping of forces from three fluid nodes to one solid node. f represents exemplary scalar force values without units. Fluid and solid values are indicated with subscripted F and S, respectively. Also, the fluid nodes are numbered via subscripts.

(b) Consistent mapping of displacements from one structure node to three fluid nodes. u represents an exemplary scalar displacement value without units. Fluid and solid values are indicated with subscripted F and S, respectively. Again, the fluid nodes are numbered.

Figure 4.6: Conservative mapping of forces (4.6a) and consistent mapping of displacements (4.6b) between a solid node and three assigned fluid nodes with the NN method. The spatial distribution of the nodes and the assignment are chosen arbitrarily as the whole setting is of generic character.

Nearest-Neighbor

NN mapping is the most basic method available in preCICE. Each node at the wet surface of a mesh searches for the corresponding closest neighboring node of the other mesh. In this context, "closest" is to be interpreted in the sense of the shortest Euclidean distance ([6]). Of course, this allows for multiple

[5]Note that for this generic example, NN mapping is assumed exemplarily.

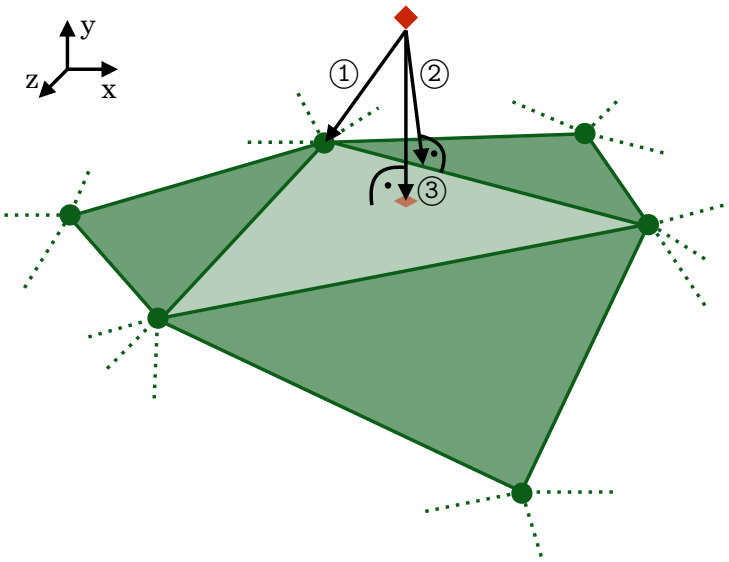


Figure 4.7: Determining the shortest distance with the NP method in a three-dimensional case. The fluid surface mesh is an unstructured, triangular mesh. Exemplary, the distances of a solid node to the fluid mesh are depicted by arrows: ① The distance to the nearest neighboring fluid node. ② The orthogonal distance to the nearest edge of the fluid mesh. ③ The orthogonal distance to the nearest surface element of the fluid mesh.

nodes of a fine mesh to be assigned to a single node of a coarse mesh as mentioned before. For this kind of mapping, preCICE needs no information regarding mesh connections and elements, the sole position of the nodes at the wet surface is sufficient ([18]).

Nearest-Projection

In the case of NP mapping, the shortest distance of a node of one mesh to the other mesh is detected. For a general three-dimensional case, when the interface between fluid and solid is a surface, a node's shortest distance to the other mesh may occur at either a node, an edge or a surface element of the partner mesh. Thus, for each node, preCICE computes the distance to the NN, the nearest edge and the nearest surface element ([18]). For a graphical representation of this situation, see Figure 4.7. Consequently, the shortest distance is chosen among those three, which determines whether one node (node is nearest), two nodes (edge is nearest) or multiple (≥ 3) nodes (surface element is nearest[6]) have to be taken into account for mapping. If the shortest distance occurs at an edge or a surface element, in general, not all nodes of the respective edge or surface element have the same influence on the assigned node of the partner mesh. Depending on how close these nodes are to the projected one, weights are calculated, which describe the differently strong influence. As for this method preCICE needs to know not only about the positions of all interface nodes, but also mesh connections in order to recognize edges and elements, at least one full mesh representation must be fed to preCICE during startup of a simulation ([18]).

Radial Basis Functions

Interpolation with RBF can be done with either *compactly* or *globally supported* functions. This means that the spatial influence of nodes, from which data is to be mapped, is either limited to a certain Euclidean range, the *support radius* r, or not. In the latter case, each node at the wet surface of one mesh influences each node at the interface of the other mesh. In contrast, with compactly supported RBF, only those nodes of the partnering mesh are influenced, which are located within a sphere around a node of the first mesh with radius equal to the support radius. In both cases the exact strength of the influence (and its dependency on distance between two nodes) is then determined by the RBF itself:

$$\phi(\|\boldsymbol{x}\|), \tag{4.6}$$

[6]Number of involved nodes is dependent on type of element, for a triangular surface element, it would be three.

where $\|\boldsymbol{x}\|$ denotes the Euclidean distance between a node of the first and the second mesh. Moreover, for compactly supported RBF holds: $\phi(\|\boldsymbol{x}\|) = 0$ for $\|\boldsymbol{x}\| > r$.
Generally, the wider the support of a basis function is, the better is the approximation. Yet the computation of the interpolation requires more effort as influences of a lot of nodes have to be taken into account. In reverse, a narrow support yields an interpolation system, which is easy to solve, but the approximation might suffer from it resulting in larger mapping errors. Choosing an adequate support radius is, therefore, a difficult task in practice. Moreover, the support radius should be kept fixed for all wet surface nodes of a mesh as varying from node to node might not lead to an interpolation solution ([1], [33]).

Several different RBF are available in preCICE[7]. Globally supported functions include a thin-plate spline, (inverse) multiquadrics, a volume spline and a Gaussian, while the following compactly supported RBF are implemented: A thin-plate spline of continuity C^2, as well as polynomials of continuity C^0 and C^6, respectively ([18]).

4.2 SU^2 - A Modular, Flexible CFD Solver

SU^2 stands for Stanford University Unstructured[8]. It is an open-source software suite initiated and mainly developed at the Aerospace Design Laboratory of Stanford University[9]. For this thesis the current release version of SU^2 is **4.1 "Cardinal"**. All explanations are therefore limited to this version.

Being motivated from an aerodynamical point of view, the core capabilities of the tool include CFD analysis and design-driven tasks like shape optimization[10] for single- and multiphysical problems based on a finite volume method (FVM). More fundamental, SU^2 solves problems governed by PDE on arbitrarily unstructured meshes. However, SU^2 uses a vertex-based FVM (rather than a cell-based approach), such that solution variables are determined and stored at the vertices (= nodes). The vertices need to be embedded in control volumes (= elements), which the numerical solution procedures are applied to. The original mesh does not embed nodes in the cells, but they are defined on cell edges. Thus, a dual-mesh is calculated from the primal-grid using a median-dual, vertex-based scheme. Cells of this dual-mesh are calculated by connection of the centroids, faces and edge-midpoints of all primal-grid elements, which share the respective node. The latter is now embedded in a dual cell. For further information about this procedure, see [30].
Both steady and unsteady problems can be solved by SU^2. The former is relevant for coupling with preCICE in order to handle FSI simulations. In each physical time step of SU^2 the time-dependent problem is solved by converging it to a locally steady solution. Therefore, SU^2 offers a dual-time stepping procedure. Pseudo-time steps are performed for a single physical time step until sufficient convergence is reached (or the specified maximum number of dual time steps). This method allows to reuse acceleration methods, which are well-established for steady flow problems. I refer to [30] for further insights into this method.

The highly modular character of the object-oriented C++ package allows for extension to fields of science and engineering other than aerodynamics ([30]). However, in this thesis, I focus on its CFD and dynamic mesh capabilities - central functions relevant for partitioned FSI simulations.
SU^2 is subdivided into several software modules:

- **SU2_CFD**: CFD solver (PDE solution module), main component
- SU2_DEF: Mesh deformation
- SU2_DOT: Gradient projection for optimization tasks
- SU2_GEO: Definition of the geometry of problems
- SU2_MSH: Mesh adaption
- SU2_PY: Python scripts for automated tasks
- SU2_SOL: Solution export and conversion

[7] For the mathematical definition of these functions see [18].
[8] See `http://su2.stanford.edu`.
[9] The code can be accessed via Github: `https://github.com/su2code/SU2`.
[10] E.g. of aircrafts, aircraft wings or airfoils for wind turbines.

If not explicitly stated otherwise, I only describe the functional extent of the SU2_CFD module, as all other modules are of no or minor importance for this work. In Section 4.2.1, I shortly state the mathematical models used in SU^2. In the following, in Section 4.2.2, input and output files, the basic code structure and an exemplary run of the CFD solver is described. Section 4.2.3 gives an insight into parallelization of the software suite and Section 4.2.4 closes the description of SU^2 by motivating the coupling of SU^2 with preCICE, taking into account recently added, intrinsic FSI capabilities of SU^2.

4.2.1 Mathematical Modeling

Analogous to the previous sections, I consider the mathematical modeling of SU^2 in the most general, three-dimensional case. Unless stated otherwise, the equations in the following are taken and adapted from [30] and [36]. The suite is capable of solving various systems of PDE on a domain $\Omega \subset \mathbb{R}^3$, as long as the problem can be stated in the hereafter formal sense:

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot \boldsymbol{F}^{\mathrm{c}}(\boldsymbol{U}) - \nabla \cdot \boldsymbol{F}^{\mathrm{v}}(\boldsymbol{U}) = \boldsymbol{Q}(\boldsymbol{U}) \quad \text{in } \Omega,\, t > 0. \tag{4.7}$$

$\boldsymbol{U}$ is the solution vector of unknowns, which needs to be determined. $\boldsymbol{F}^{\mathrm{c}}$ is a vector of convective fluxes and $\boldsymbol{F}^{\mathrm{v}}$ refers to viscous fluxes. Both flux vectors are functions of the unknown solution. $\boldsymbol{Q}$ denotes a generic source term, which can also depend on $\boldsymbol{U}$ ([36], [30]).

From this basic formulation, a wide variety of PDE-based problems other than classical flow scenarios (Euler, NSE, Reynolds-averaged Navier Stokes (RANS)) can be solved by adapting $\boldsymbol{U}$, $\boldsymbol{F}^{\mathrm{c}}$, $\boldsymbol{F}^{\mathrm{v}}$ and $\boldsymbol{Q}$ to the concrete situation. For instance, the software is used in wind turbine and solar collector simulation, for naval engineering purposes (free surface flows) and also in chemical engineering ([11]). However, in this thesis, the treatment of the NSE is of greatest relevance. Defining the solution vector as $\boldsymbol{U} = (\rho, \rho v_1, \rho v_2, \rho v_3, \rho E)^\intercal$, where ρ is the fluid density, $\boldsymbol{v} = (v_1, v_2, v_3)^\intercal \in \mathbb{R}^3$ is the flow velocity and E is the total energy per unit mass of the flow, allows for usage of the formulation stated in Equation 4.7. The entries of the solution vector are also known as *conservative variables* since they refer to mass, momentum and energy, which are conserved by the NSE (for a reminder, recapitulate Section 2.3.1). In this model the convective and viscous fluxes are given as

$$\boldsymbol{F}^{\mathrm{c}}(\boldsymbol{U}) = \begin{pmatrix} \rho \boldsymbol{v} \\ \rho \boldsymbol{v} \otimes \boldsymbol{v} + p\boldsymbol{I} \\ \rho \boldsymbol{v} H \end{pmatrix}, \qquad \boldsymbol{F}^{\mathrm{v}}(\boldsymbol{U}) = \begin{pmatrix} 0 \\ \boldsymbol{\tau} \\ \boldsymbol{\tau} \cdot \boldsymbol{v} + \mu C_p \nabla T \end{pmatrix}. \tag{4.8}$$

p denotes the static pressure and $\boldsymbol{I} \in \mathbb{R}^{3\times 3}$ the identity matrix. H is the fluid enthalpy, $\boldsymbol{\tau} \in \mathbb{R}^{3\times 3}$ corresponds to the viscous stress tensor and μ is the dynamic viscosity of the fluid. C_p denotes the specific heat at constant pressure and T refers to temperature. This formulation corresponds to a Eulerian description of the fluid domain. For coupling SU^2 with preCICE the ALE derivation is needed. The viscous flux vector undergoes no changes and also the source term remains the same. Only the convective flux vector needs to incorporate the velocity of the mesh nodes such that a relative velocity between mesh and fluid particles is gained:

$$\boldsymbol{F}^{\mathrm{c}}_{\mathrm{ALE}}(\boldsymbol{U}) = \begin{pmatrix} \rho(\boldsymbol{v} - \dot{\boldsymbol{u}}_{\mathrm{mesh}}) \\ \rho \boldsymbol{v} \otimes (\boldsymbol{v} - \dot{\boldsymbol{u}}_{\mathrm{mesh}}) + p\boldsymbol{I} \\ \rho(\boldsymbol{v} - \dot{\boldsymbol{u}}_{\mathrm{mesh}})E + p\boldsymbol{v} \end{pmatrix}. \tag{4.9}$$

$\boldsymbol{u}_{\mathrm{mesh}} \in \mathbb{R}^3$ corresponds to the displacements of the mesh nodes. Thus, $\frac{\partial \boldsymbol{u}_{\mathrm{mesh}}}{\partial t} = \dot{\boldsymbol{u}}_{\mathrm{mesh}}$ refers to the mesh velocities. Inserting Equation 4.9 and the viscous flux vector from Equation 4.8 into Equation 4.7 yields the final PDE system suitable for FSI coupling via preCICE.

Note that the above derivation corresponds to compressible flows (density is an unknown variable). Up to the current version (4.1 "Cardinal") of SU^2 only the compressible solver has an implementation of the ALE method, the incompressible solver can solely be used with non-moving meshes. Consequently, also the FSI capabilities of SU^2 after coupling with preCICE are limited to the compressible regime.

4.2.2 Software Structure

Input and Output Files

SU^2 needs only two different input files for a regular solver run: A configuration file and a mesh file. Although the CFD General Notation System (CGNS) format can be used as mesh input for SU^2, its support is not part of the standard installation process as described in Appendix B. An own native mesh format has been developed for SU^2, which is easily readable and adaptable. Such mesh files carry the extension *.su2* and are written in ASCII format ([30]). These SU^2 meshes can directly be created with the free software Gmsh ([19]), for instance. The file contains the following information ([30]):

- Dimensionality of the problem[11],
- total number of elements of the mesh,
- element connectivity information[12]: Including an identifier for the kind of each element (triangle, rectangle, tetrahedron, etc.), indices of the involved nodes and a consecutively running number, which serves as element index.
- Total number of mesh nodes,
- coordinates of all nodes with a consecutively running number used as node index.
- In addition, the total number of boundaries of the mesh (also referred to as *markers*) is given
- and for each boundary the name of the marker, the total number of involved elements and a list of these elements is stated. This list includes an identifier for the kind of each element (line element (2D), rectangle or triangle (3D)) and the indices of the involved nodes.

The configuration file with extension *.cfg* is a simple, text-based file, which contains all options for the solver run of SU^2. Each option starts with a unique name, followed by "=" and finally, the value of the option:

$$\text{optionName} = \text{optionValue}. \tag{4.10}$$

For instance, the boundary marker tags, which are described in the listing above, are used in the configuration file to prescribe boundary conditions. This might look like as shown in Equation 4.11:

$$\text{MARKER\_INLET} = (\ \text{inlet}\ ). \tag{4.11}$$

In this example, "inlet" needs to be defined as a boundary marker in the SU^2 mesh file.
Besides options, the configuration file may contain comments, which are marked with a "%" at the beginning of a line. All white spaces (tabs, spaces, set-offs) are ignored when the file is parsed. Thus, they can be used for formatting the file for the sake of readability. Also, format and name of the mesh input file are specified in the configuration file, so the configuration file name is the only parameter, which needs to be passed to SU^2 upon starting a solver run ([30]).

The configuration file also describes what kind of output files should be written and how frequently. Also the respective output formats are specified. Files for Tecplot[13], Paraview[14] and Fieldview[15] are supported. The output files relevant for a typical solver run are

- flow volume solution files (containing all solution variables throughout the whole mesh),
- convergence history files
- and the forces breakdown file (contains force coefficients for surfaces, which are marked in the configuration file by "MARKER_MONITORING").
- Furthermore, flow restart files (for restarting a simulation from a specified point)
- and surface flow solution files (containing solution variables at surfaces, which are specified in the configuration file by "MARKER_PLOTTING") can be created.

[11] SU^2 can handle two- and three-dimensional problems.
[12] This is necessary as SU^2 uses unstructured meshes and thus, no logical ordering of the mesh nodes can be assumed.
[13] Commercial, see `http://www.tecplot.com`.
[14] Open-source, see `http://www.paraview.org`.
[15] Commercial, see `http://www.ilight.com`.

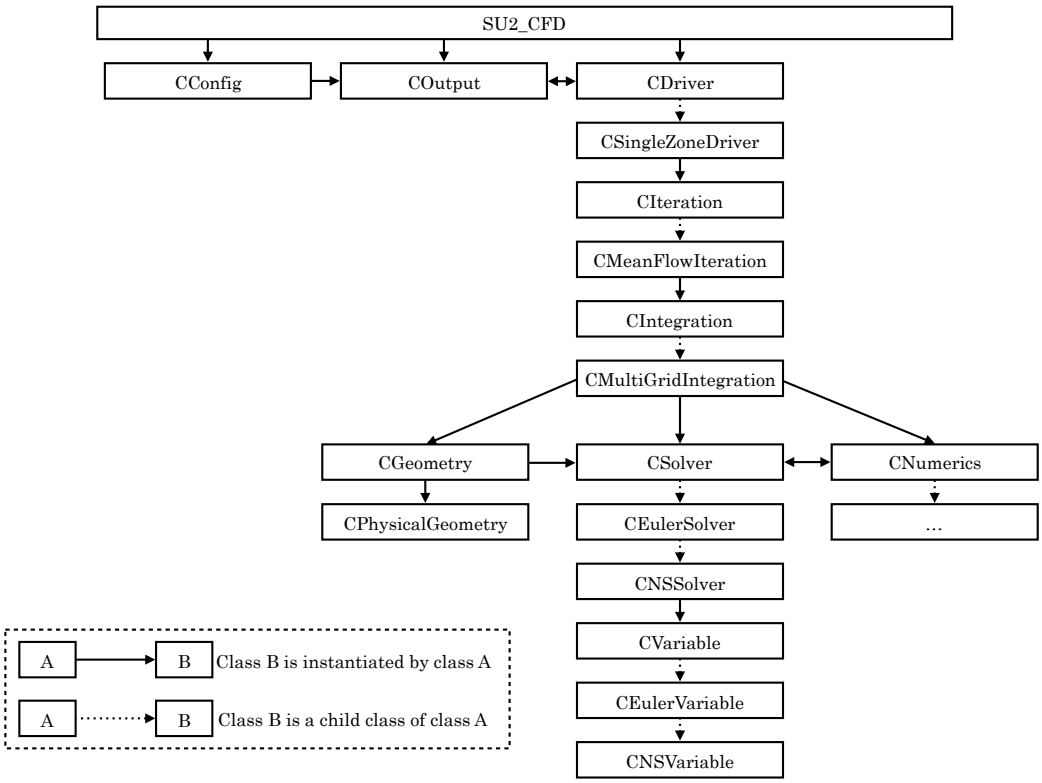


Figure 4.8: Class structure of SU^2 starting from *SU2_CFD* for a simulation solving the NSE. The high level of abstraction allows for flexible combination of different solvers for multiphysics simulations. The child classes of *CNumerics* are not specified as they depend on the numerical methods chosen by the user at configuration time. The dashed box explains the used relations. Figure adapted from [30] and [36]

Class Dependencies

In order to perform a coupling of SU^2 and preCICE as is explained in Chapter 5, some changes to the source code of SU^2 are necessary. However, to do so, the basic class structures and hierarchies of the fluid solver need to be understood. All CFD computations of SU^2 start with the file *SU2_CFD.cpp*, which contains the main method for a solver run. Within the main function several classes are instantiated. A graphical representation of these including their relations is given in Figure 4.8 for the case of a simulation based on the NSE. At the beginning of a run of SU2_CFD three classes are instantiated: *CConfig*, *COutput* and *CDriver*. In *CConfig* the configuration file is parsed and all therewith related information is saved. *COutput* handles all output corresponding tasks such as writing solution or restart files. *CDriver* is the class, which is in charge of the actual solving procedure. Due to various possible child classes, *CDriver* allows for usage and combination of several different solvers for the purpose of multiphysics simulations. In such a case, each solver works on its own, so called *zone*. Examples are combustion modeling, two-phase flows or magnetohydrodynamics simulations ([30]). However, in this thesis and for the example of solving a problem governed by the NSE, SU^2 is used as a single-physics solver, yielding only a single zone. Consequently, the child class *CSingleZoneDriver* is instantiated[16]. It basically just instantiates *CIteration* and handles its function calls, including pre-processing, running a single iteration and solution updating, once an iteration has converged. The concrete actions of these functions are implemented in the subclasses of *CIteration* and depend on the type of physical problem to be solved. In case of a NSE problem, the adequate child class is *CMeanFlowIteration*[16]. Pre-processing includes the important steps of setting initial conditions for an iteration and computing grid movements, if dynamic mesh capabilities are enabled. *CMeanFlowIteration* instantiates a child of the class *CIntegration*. If the multigrid functionalities of SU^2 are enabled, the child *CMultiGridIntegration* is chosen[17]. It connects the classes *CGeometry*, *CSolver* and *CNumerics*, which handle discretization and subsequent spatial and time integration of the problem. *CPhysicalGeometry*[16] converts the primal-grid structure into the new

[16] For an overview of all (other) respective child classes, see [36] and [30].
[17] Otherwise *CSingleGridIntegration*.

dual-mesh structure, which the computations are executed on. The kind of solver to use depends on the physical problem. For the NSE the final solver class is *CNSSolver*[16]. Here, the NSE are numerically solved, finally yielding a solution vector. This is done by calling several classes in *CNumerics*, which are determined by the numerical procedures chosen at configuration time[16]. These define the discretization methods, which, in case of the NSE as governing equations, are used to compute viscous and convective fluxes ([30], [36]).

Sequence of a Typical Solver Run

After explaining the structure of SU^2 from a static point of view, I now consider a typical solver run for a problem governed by the NSE, which is shown in Algorithm 2. Note that the run is reduced to its basic elements and does not contain arguments when calling functions, datatypes, declarations and initializations (except where needed for better understanding). At first, an iteration counter and

Algorithm 2 Typical SU^2 solver run in pseudo code, reduced to core functionalities

```
1: stopCalculation = false;
2: externalIteration = 0;
3: parseConfigurationFileAndStoreRespectiveInformation();
4: geometricalPreprocessing();
5: driverPreprocessing();
6: if dynamicMeshSimulation then
7:    setDynamicMeshStructure();
8: end if
9: while externalIteration < maxNumberOfExternalIteration do
10:   driverRun();
11:   updateConvergenceHistory();
12:   if convergence then
13:      stopCalculation = true;
14:   end if
15:   if solutionNeedsToBeOutput then
16:      writeOutputFiles();
17:   end if
18:   if stopCalculation then
19:      break;
20:   end if
21:   externalIteration++;
22: end while
23: return exitSuccess;
```

a flag for determining whether the simulation should be stopped are initialized. In the following, the configuration file is parsed and the chosen options are stored. In *geometricalPreprocessing()*, the dual mesh structure is computed. *driverPreprocessing()* instantiates all classes needed for the solver run and transfers respective options from the configuration to these instances. Subsequently, if the simulation deals with dynamic meshes, respective data structures such as for storing node displacements and mesh velocities are prepared. While the iteration counter is smaller than the maximum number of iterations (also prescribed in the configuration procedure), the solver keeps executing the following tasks: Run a single iteration (more precisely described in Algorithm 3) and update the convergence history afterwards. If convergence has been reached, set the flag for stopping the calculation to true. Depending on the options specified at configuration time, determine whether output files need to be written or not. Subsequently, the *stopCalculation* flag is checked. If it is set to true, the algorithm breaks out of the while-loop of the solver. If it evaluates to false, it proceeds with incrementing the iteration counter and starts again at the condition of the while-loop. After breaking out of the while-loop or if the maximum number of iterations is reached, SU^2 returns an *exitSuccess*, provided that the solver has not aborted during an iteration due to e.g. divergence or other problems. A single solver iteration (see Algorithm 3) starts with setting the initial conditions for the current iteration. If the simulation uses dynamic mesh capabilities, as next step the mesh deformation is computed from the change of coordinates of the nodes relative to the previous time step. A pseudo-structural problem is solved in SU^2 for this purpose. The stiffness

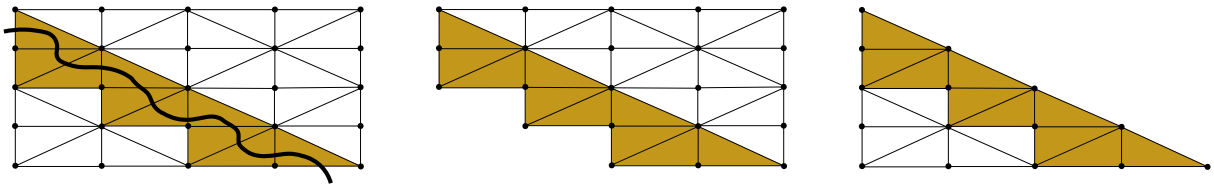
Algorithm 3 A single *driverRun* (see Algorithm 2) in pseudo code, reduced to core functionalities

```
1: setInitialConditions();
2: if dynamicMeshSimulation then
3:     computeMeshDeformation();
4: end if
5: storeOldSolution();
6: setTimeStep();
7: spacialIntegration();
8: timeIntegration();
9: computeNonDimensionalParameters();
10: monitorConvergence();
11: updateSolution();
```

of the mesh elements can be modeled to be proportional either to the inverse of the area/volume of an element or to its distance to a boundary or the stiffness can be set constant. In the following, the solution of the previous iteration is transferred from the equation system's solution vector to separate variables and the solution vector is reset. As SU^2 offers adaptive time stepping functionalities, in the next step the possibly adaptive time step is set. Afterwards, integration of the problem in space and time occurs utilizing the numerical discretization and solution techniques specified in the configuration file. This yields a new solution vector, from which non-dimensional parameters at surfaces of the problem are derived, such as coefficients of lift and drag forces or coefficients of torques. The final step of an iteration consists of updating the solution, which includes forwarding it to all different grid levels, assuming that the multigrid capabilities of SU^2 are used.

4.2.3 Parallelization

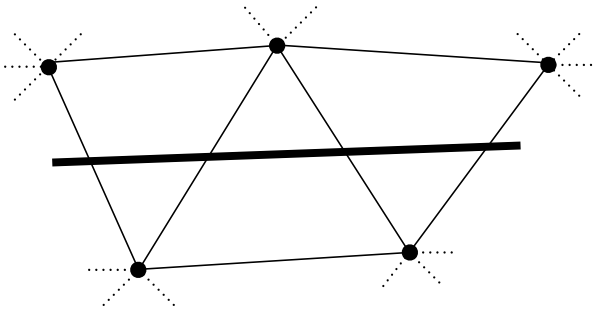
In order to run large-scale simulations in feasible time, SU^2 can be executed in an intrafield-parallel manner, meaning that the computational domain is split into parts, which separate processes work on. SU^2 uses the well-established partitioning software packages METIS and PARMETIS ([22], [23]) to deal with this domain decomposition task, while communication is managed via MPI. The mesh is divided using edge-cuts. Along the cut many elements remain fragmentary as nodes of one element may be assigned to different processes. It is not efficient to allow the different processes to communicate with each other whenever they need values of nodes that are not part of their own submesh. This would slow down computations considerably due to communication overhead. Instead, ghost points are initialized in SU^2 to sustain complete elements for all processes. As a consequence, overlapping, so called *halo cells* are created. For these, data is exchanged between the corresponding original and replicated nodes after each iteration ([30], [11]). Figure 4.9 depicts this situation for a sample, triangular mesh, which is divided into two submeshes. For the purpose of identifying which node (including the ghost nodes) is handled by



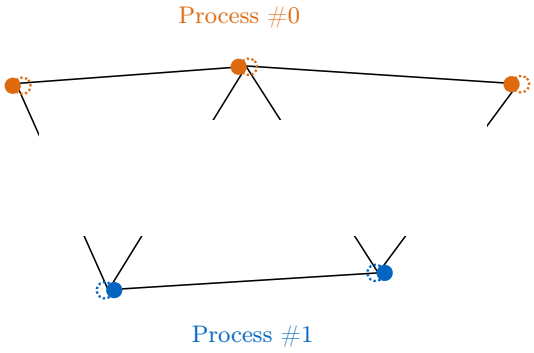
(a) Total mesh and cutting line. (b) Resulting submesh 1. (c) Resulting submesh 2.

Figure 4.9: Domain decomposition of a triangular mesh into two submeshes. The **thick, black line** in Figure 4.9a represents the edge cuts of the partitioning. The colored cells become halo cells due to duplication of their respective nodes. The two detached submeshes overlap at the halo cells and are shown in Figures 4.9b and 4.9c, respectively.

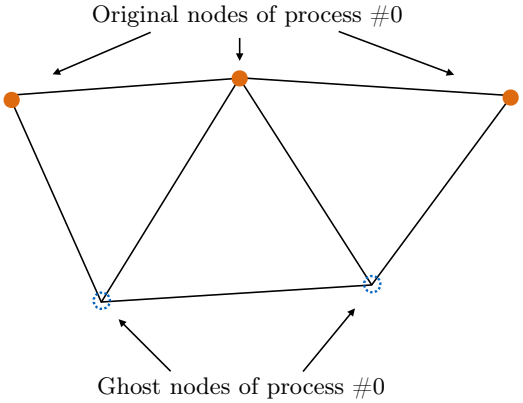
which process, so called *colors* are assigned to the nodes. Each color is a number which, equals the MPI rank of the respective process and is therefore unique among the processes ([30], [11]). The whole grid partitioning procedure including the coloring of nodes is depicted schematically for a two-dimensional, triangular mesh in Figure 4.10. It can be understood as a local, more detailed excerpt of Figure 4.9.



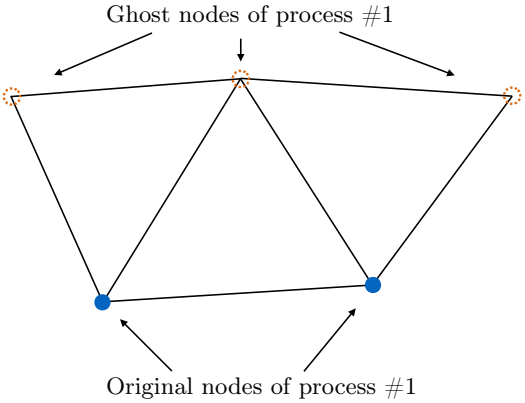
(a) The mesh is divided into two submeshes by cutting edges of triangular elements. The **thick line** illustrates the cut. Each "side" of the cutting line is handled by a different processor. In the figure only a small sector of a larger mesh is shown, which is indicated by the dashed lines.



(b) Two unconnected submeshes are derived. Fragmentary elements remain and ghost nodes (dashed circles) are introduced as duplicates of the original nodes. Their difference in position is only due to graphical reasons. The dashed lines indicating further elements are omitted for readability.



(c) Ghost and original nodes of process #0 are connected to recover the original elements. The dashed lines indicating further elements are omitted for readability.



(d) Ghost and original nodes of process #1 are connected to recover the original elements. The dashed lines indicating further elements are omitted for readability.

Figure 4.10: Schematic of domain decomposition in SU^2 among two processes. In Figure 4.10a the original mesh and the cutting procedure is shown. Figure 4.10b illustrates the introduction of ghost nodes for both processes and the respective coloring of all nodes. The final meshes including original and ghost nodes are shown in Figures 4.10c and 4.10d for processes #0 and #1, respectively.

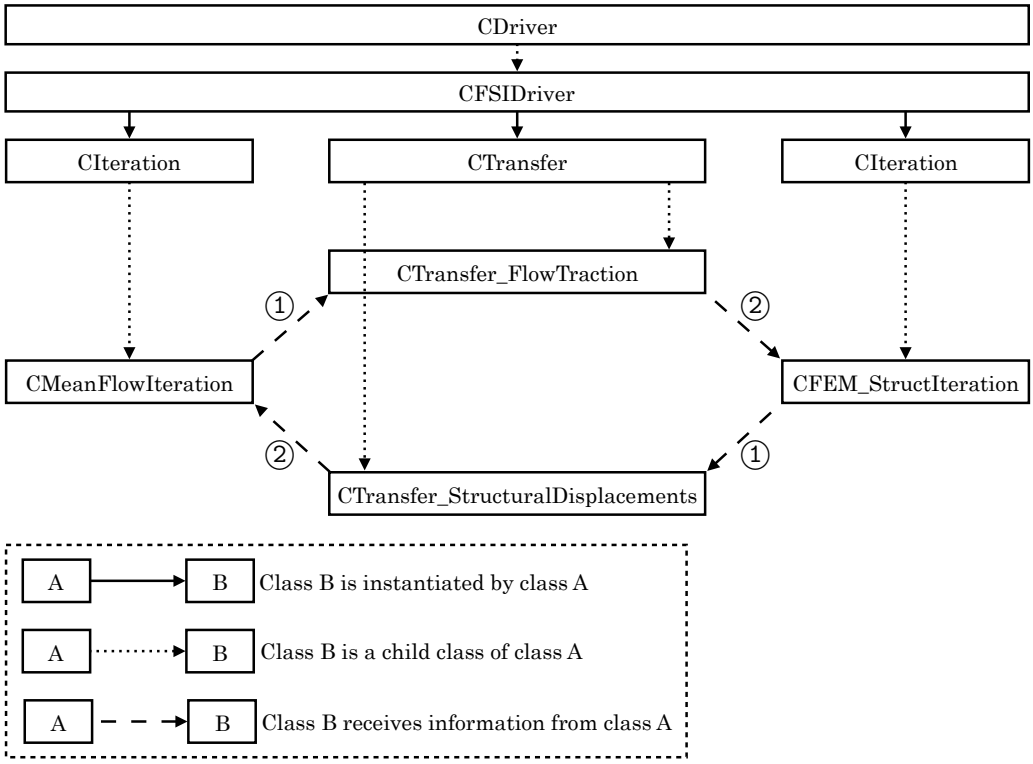


Figure 4.11: Excerpt of the class structure for (intrinsic) FSI simulations in SU^2 starting from *CDriver*. The single-physics solvers represented by *CMeanFlowIteration* and *CFEM_StructIteration* ① forward coupling data to the respective child classes of *CTransfer*, before they ② transfer it to the partner solver. The dashed box explains the used relations. Figure adapted from [36].

4.2.4 Intrinsic FSI Capabilities

Recently, SU^2 has been extended with a structural solver based on the finite element method (FEM)[18]. It is capable of handling both geometrical and material non-linearities. In a partitioned approach (for a reminder, see Section 3.1) the solid solver has been coupled with the original ALE-based flow solver for the purpose of treating FSI problems. However, as described in Section 4.2.1, the flow solver dealing with dynamic meshes is limited to compressible regimes, so are the FSI functionalities, consequently. All information regarding the intrinsic FSI capabilities of SU^2 is taken from [36].

Referring back to Figure 4.8, a new child class of *CDriver* is introduced, namely *CFSIDriver*, which takes care of combining flow and solid solver. Next to instantiating *CMeanFlowIteration* for the fluid domain and *CFEM_StructIteration* for the solid domain, the new class *CTransfer* is used for exchanging data between the single-physics solvers. Its child classes *CTransfer_FlowTraction* and *CTransfer_StructuralDisplacements* forward forces from fluid to solid and in reverse, displacements from structure to flow solver. Figure 4.11 depicts the newly added class dependencies. Both weakly and strongly coupled algorithms (for a reminder, see Section 3.2) are implemented, including the explicit CSS algorithm and an implicit Block Gauss-Seidel method with either constant or dynamic Aitken relaxation. Since these algorithms are also included in preCICE, no further explanation is given here. Instead, I refer back to Section 4.1.1. The intrinsic FSI implementation also contains strategies for non-matching meshes. Currently, NN (for a reminder, see Section 4.1.3) and *isoparametric* mapping methods are available, but extension to interpolation based on RBF is in progress. A consistent and conservative approach is chosen in SU^2, meaning that tractions are mapped conservatively, while displacements are transferred in a consistent fashion (compare Figure 4.6).

Despite the intrinsic FSI capabilities, it is reasonable to couple SU^2 with preCICE. First of all, via preCICE a wider range of coupled simulation problems can be addressed, including e.g. fluid-structure-acoustics interaction. Also, the user is given more flexibility when choosing a structural solver, ranging

[18] For now, the solver is limited to linear, first order elements (triangles and quadrilaterals in two-dimensional, tetra- and hexahedra in three-dimensional problems).

from elaborate, commercial solid solvers to open-source and academic codes that solely focus on the discipline of CSM. Moreover, although SU^2 includes coupling algorithms, which are also implemented in preCICE, none of these allow for interfield parallelism such that a fully-parallel execution of fluid and solid solver is not possible. This is especially of importance when it comes to efficient usage of computational resources and is therefore, highly relevant for HPC on massively parallel systems. SU^2 does not utilize coupling algorithms, which make use of the advantage that internal solver information is available, such as e.g. *Newton-Raphson methods*, due to their high computational cost. Consequently, preCICE's black-box approach faces no drawbacks compared to the intrinsic coupling of SU^2. Quite contrary, preCICE offers very elaborate acceleration schemes for implicit coupling, which outperform the constant and Aitken relaxation techniques implemented in SU^2. Finally, preCICE also offers a greater, more sophisticated selection of mapping procedures for non-matching meshes, possibly resulting in smaller interpolation errors. In summary, coupling for FSI simulations via preCICE offers more flexibility and possibly also more accurate results, while computational effort is spread effectively.

5. Description of the Coupling Adapter and its Integration

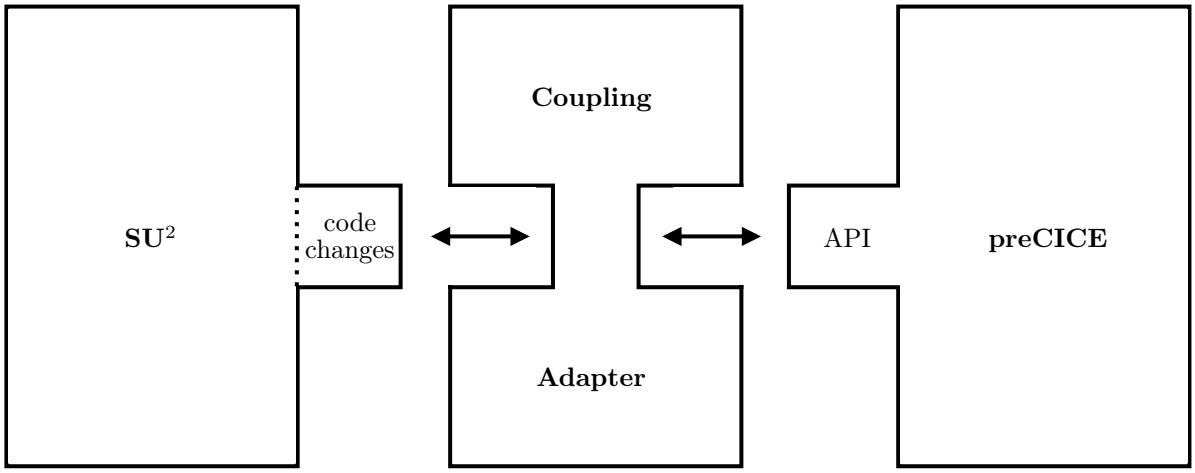


Figure 5.1: The source code of SU^2 is extended by minimally invasive code changes, which allow to use the adapter. The adapter makes use of the API of preCICE that is part of the coupling tool's source code.

In order to couple SU^2 with preCICE, a C++ adapter class named *Precice*[1] is developed in the scope of this work. A header file *precice.hpp* and a source file *precice.cpp* are the practical outcome. The *Precice* class encapsulates all coupling related activities and separates them from the original SU^2 source code. It makes use of the high-level API provided by preCICE. Since the adapter is integrated into the source code of SU^2, it is completely compiled with it (for a description on how to install SU^2 with preCICE, see Appendix B). This way, coupling is achieved with minimally invasive code changes in SU^2 and an adaption of the original code is, thus, possible with only small effort, basically reduced to copy-paste tasks. The adapter allows for usage of both explicit and implicit coupling strategies implemented in preCICE and fully conforms with intra- and interfield parallelism. Moreover, usage of the adapter is assimilated to the regular configuration process of SU^2, thus, it is embedded smoothly into the software suite. All options concerning the usage of preCICE (e.g. switching it on or off, specifying name and location of the preCICE configuration file, etc.) are set via the SU^2 configuration file. Consequently, no recompilation of SU^2 is necessary when the user decides to use/not to use preCICE. In addition, a single executable, SU2_CFD, is enough to account for single-physics simulations (without preCICE) as well as for FSI computations via preCICE. Figure 5.1 shows a schematic representation of the code coupling approach.

Concerning notation of code shown in this chapter (and in the corresponding referenced sections of the appendix), it is important to state that SU^2 uses several "containers" for storing information (technically, they are multiple pointers). E.g. a "config_container" is an instance of *CConfig* or a "geometry_container" refers to *CGeometry*. Furthermore, all shown code excerpts are reduced to the necessary information. Therefore, not all arguments of functions are stated but only the relevant ones. Also, ellipses (...) are

[1] Note that *Precice* and *preCICE* do not denote the same. The former refers to the adapter developed in this work, while the latter is used to describe the coupling tool.

used to denote further lines of code, which are not shown for the sake of simplicity.

This chapter is organized in the following top-down way: Starting from the most user-respective changes in SU^2, in Section 5.1, the newly added, coupling-related options included in the SU^2 configuration file are presented and their usage is explained. In addition, necessary code changes are stated. The chapter continues with a more technical, detailed description on how the coupling is embedded in SU^2, as in Section 5.2 adaptions to the main routine of SU2_CFD are described. Mostly, these modifications include calling several functions, which are incorporated in the adapter class *Precice*. However, the tasks hidden in these functions are not described until finally, the adapter itself is extensively explained with emphasis on both physical and computational details at the end of this chapter in Section 5.3. Referring back to Figure 5.1, Sections 5.1 and 5.2 correspond to "code changes" in SU^2, while Section 5.3 relates to the "Coupling Adapter".

5.1 Changes Concerning SU^2 Configuration

In order to fully control the usage of preCICE for FSI simulations within SU^2, new options in the configuration file of SU^2 are available (for a recapitulation of this file, see Section 4.2.2). They are listed in the following with their **default values**:

$$\text{PRECICE\_USAGE} = \textbf{NO}, \text{ YES} \tag{5.1a}$$

$$\text{PRECICE\_CONFIG\_FILENAME} = \textbf{precice.xml} \tag{5.1b}$$

$$\text{PRECICE\_VERBOSITYLEVEL\_HIGH} = \textbf{NO}, \text{ YES} \tag{5.1c}$$

$$\text{PRECICE\_WETSURFACE\_MARKER\_NAME} = \textbf{wetSurface} \tag{5.1d}$$

Most obvious, PRECICE_USAGE is a flag used for determining whether a simulation should be run with or without preCICE. Modifying the remaining three options is only reasonable if it is set to YES. PRECICE_CONFIG_FILENAME specifies the name of the configuration file of preCICE. Also, its path relative to the location of the SU^2 configuration file must be specified. In order to allow users to have more insight into the sequence of activities within the coupling adapter, the level of verbosity of the adapter can be chosen. If PRECICE_VERBOSITYLEVEL_HIGH is set to YES, several checkpointing information of the adapter is output to the console. Yet, too much console output can slow down simulations. Since this information is typically not relevant when running an FSI simulation productively, the verbosity level is chosen to be low by default. This feature of the coupling adapter is mainly included for tracing back runtime errors. Eventually, as explained in Section 4.2.2, physical boundaries are treated as markers in SU^2. Each boundary has a unique identifying name, which is specified in the SU^2 mesh file. The boundary marker name corresponding to the FSI interface of the fluid mesh must be passed to the coupling adapter, which is done via the option PRECICE_WETSURFACE_MARKER_NAME. A description on how to adapt SU^2 in order to use the new configuration options is given in Listings A.1, A.2 and A.3, Appendix A to a detailed level.

The dynamic mesh capabilities of SU^2 must be enabled, in order to use the implemented ALE formulation of the flow solver. This is done via:

$$\text{GRID\_MOVEMENT} = \text{YES} \tag{5.2}$$

Still, a specific kind of grid movement needs to be chosen. Intrinsically available are e.g. specifications for rigid motions or rotations of the mesh. Here, a new option is available, which is mandatory if SU^2 is used for FSI simulations via preCICE:

$$\text{GRID\_MOVEMENT\_KIND} = \text{PRECICE\_MOVEMENT} \tag{5.3}$$

A manual on how to add this new grid movement option to the configuration procedure of SU^2 is given in Listing A.4, Appendix A.

Yet, the implementation of PRECICE_MOVEMENT is missing. It defines the steps of which the mesh movement procedure consists. After displacements of the nodes at the wet surface are transferred to SU^2 via preCICE, the mesh needs to be deformed and smoothed (for a reminder, see Sections 2.2.3 and 3.3). Despite the sole mesh deformation, also grid velocities must be calculated in order to be able to

use the ALE method of SU^2. Finally, for cases in which the SU^2 multigrid capabilities are enabled, displacements and velocities of the mesh nodes need to be mapped to all grid levels. Intrinsic SU^2 mesh movement procedures cannot be reused as either they do not include the three necessary steps stated above (mesh deformation, grid velocity computation, forwarding information to multigrid levels) or they involve further computations, which are not necessary for FSI simulations and therefore, represent unnecessary computation overhead. The code defining the steps of PRECICE_MOVEMENT is given in Listing A.5, Appendix A.

5.2 Adaption of SU^2 Main Routine

After modifying files related to the configuration procedure of SU^2 in the previous section, it remains to adapt the SU2_CFD module in *SU2_CFD.cpp*, which relates to the solver run procedure itself. The goal is to add as little code as possible in the main solver routine of SU^2 such that the coupling adapter can be used. Detailed, corresponding code excerpts are stated in Appendix A.

One core criterion for integrating the adapter into SU^2 is that the solver executable should be able to run both single- and multiphysics simulations without recompilation. Only a single adapter-related variable needs to be initialized in the main routine of SU^2 regardless of whether preCICE is used for a simulation or not. The variable (called *precice_usage*) is a boolean flag corresponding to the newly added PRECICE_USAGE option of the SU^2 configuration file. This flag is the basis for conditionally triggering all coupling activities. If it is set to false, no more coupling variables are initialized in SU2_CFD and the single-physics solver runs according to the regular scheme previously shown in Algorithm 2, Section 4.2.2. The (only three) additional variables needed for coupling include an instance of the adapter class *Precice* and two time-stepping variables (namely *max_precice_dt* and *dt*).

preCICE needs to be able to shut down SU^2, in case the FSI simulation should be ended. Therefore, an adaption of the main solver while-loop is necessary. In case a simulation runs without preCICE, the usual condition of the while-loop is used, which checks that the number of solver iterations is smaller than a specified maximum. However, if a coupled FSI simulation is executed, the adapter additionally evaluates if preCICE signalizes a solver shut down.

Assuming that the solver loop is executed, a checkpointing procedure for implicit coupling strategies of preCICE starts. For strongly coupled algorithms, preCICE tries to find the fixed-point of the coupling equation system (as explained in Section 4.1.1). If the solution of a subiteration does not satisfy the preCICE convergence criteria, resetting the fluid solver to the start of the time step is necessary. Therefore, at the beginning of each solver iteration in SU^2, the current solver state is saved such that reloading it becomes possible. However, this is only done in the first subiteration of a new time step.

As mentioned in Section 4.1, preCICE might need to enforce time step sizes for the single-physics solvers. To allow for the same in SU^2, the minimally allowed time step size for an iteration needs to be determined before the solution procedure starts. Here, the variables *dt* and *max_precice_dt* come into play. While the former stores the current time increment of SU^2, the latter is the maximum prescribed by preCICE.

After SU^2 is done with executing a single solver iteration, preCICE is informed that a new flow solution is available. Therefore, the coupling tool might advance in time. This triggers preCICE to manage the data exchange between SU^2 and other coupled solvers, as well as to execute the coupling algorithm. In case of an implicit procedure, convergence acceleration techniques are also activated by this step.

In case a strongly coupled algorithm is chosen, preCICE needs to evaluate (by checking its convergence criteria) whether the old solver state of SU^2 needs to be reloaded, staying at the same physical time instance, or the simulation can proceed with the next time step. In the former case, SU^2 internal solver variables are reset to the last time instance. It is not reasonable to allow SU^2 to write output files if preCICE signalizes that the current time step has not sufficiently converged.

Finally, SU2_CFD handles the clean shut down procedure at the end of an FSI simulation. Communication channels are closed via the adapter and coupling-related memory is deallocated.

In Appendix A the extended solver procedure of SU2_CFD is depicted in Algorithm 4 by analogy with the original solver sequence shown in Algorithm 2.

5.3 Coupling Adapter

As shown in the previous section, most code changes in SU2_CFD consist of conditional clauses checking whether the adapter is used, followed by function calls on the adapter object. In this section I explain the adapter functions and how they relate to preCICE. No code excerpts are included in this section, as the files *precice.hpp* and *precice.cpp* will soon be included in the open-source preCICE repository. Thus, the interested reader is referred to the source code.

The adapter makes use of the high-level API provided by preCICE. Its main component is an interface with predefined functions that need to be integrated in the adapter. The corresponding class (in preCICE) is called *SolverInterface*. Simply calling functions of this class within SU^2 is not sufficient for coupling. Rather, the adapter also takes care of

- force calculation at the FSI interface,
- managing intrafield parallelization of SU^2 in the coupling process,
- converting data from SU^2 to preCICE specific representation and vice versa,
- setting up and triggering mesh deformation,
- as well as reading and writing iteration checkpoints.

All these functionalities are smoothly hidden within the adapter class. Directly integrating these tasks in SU^2 would imply highly invasive code changes in its main routines. Yet, the adapter class also contains some functions, which I refer to as *wrappers*. They consist of not much more but function calls on *SolverInterface*. The advantage of this technique is obvious: There is no need to instantiate an object of class *SolverInterface* directly in SU^2. Rather, only the adapter instantiates such an object and therefore hides it from the main solver routines. Table 5.1 gives an overview of functions[2] of the adapter class *Precice* and whether they are wrappers or not.

function name	wrapper function?
configure()	yes
initialize()	no
advance()	no
isCouplingOngoing()	yes
isActionRequired()	yes
getCowic()	no
getCoric()	no
saveOldState()	no
reloadOldState()	no
finalize()	yes

Table 5.1: Functions of the adapter class *Precice* and their characterization.

Some aspects of the adapter functions are already mentioned in Section 5.2. However, detailed explanations of what these functions do and their connection to preCICE is eventually given in the following.

Startup of a Coupled Simulation

The whole coupling process starts with the instantiation of the adapter within the main solver routine of SU^2. Upon creation of the adapter object, several information is passed to it, including MPI rank and size, as well as all geometry (*CGeometry*), solver (*CSolver*), configuration (*CConfig*) and grid movement (*CVolumetricMovement*) related data. Next to initializing data structures needed for coupling, the most important step is the instantiation of a *SolverInterface* object within the adapter, which represents the adapter's connection to preCICE (compare Figure 5.1).

The adapter object and its connection to preCICE are established, yet it still remains to configure preCICE from its configuration file. This happens when *configure()* is called on the adapter with the

[2]Constructor and destructor are neglected.

name and location of the configuration file as input argument. Since this function is a wrapper, internally the adapter calls the same function on *SolverInterface* and forwards name and location of the *.xml* file. Consequently, preCICE parses the configuration file and creates necessary data structures for coupling.

Subsequently, communication between SU^2 and its coupling partner, as well as preCICE internal meshing at the wet surface needs to be initiated, which happens upon calling *initialize()* on the adapter. It checks each node at the FSI interface and stores its coordinates in a data array, which is then forwarded to preCICE via the *SolverInterface* object. It is important to keep in mind that intrafield parallelism is possible in SU^2. The corresponding domain decomposition procedure (for a recapitulation, see Section 4.2.3) can lead to situations, in which a process does not work on the wet surface at all. This is taken into account in the adapter as follows: As mentioned in Section 5.1, the boundary marker name of the wet surface must be given to SU^2 during configuration. This name is now used to determine whether a process includes FSI interface nodes or not. Consequently, the respective process is marked by a boolean flag, *processWorkingOnWetSurface*. If it evaluates to false, the before mentioned coordinate transfer procedure is skipped. After receiving the node coordinate information, preCICE is prepared to create an interface mesh from it. Finally, *initialize()* is executed on the *SolverInterface* object, which triggers setting up communication between the coupling partners, creating the wet surface mesh and computing a possible restriction on the first time step of SU^2.

Coupling Step

The actual coupling activities start with the main solver loop of SU^2. As explained in Section 5.2, a solver iteration only occurs if preCICE signalizes that the coupling should not be stopped yet. Therefore, in the wrapper function *isCouplingOngoing()* a function of the same name is executed on the *SolverInterface* object and its boolean return value serves as the demanded signal. Subsequently, at the beginning of an iteration and in case an implicit coupling algorithm is chosen, preCICE needs to inform SU^2 whether the current iteration corresponds to the first one of a new time step. *isActionRequired()* is called on the adapter, which is again just a wrapper for the same function call on *SolverInterface*. The input argument, however, specifies whether the action refers to writing or reading an iteration checkpoint. For this purpose, the adapter includes two constant string member variables. One corresponds to writing and one to reading a checkpoint. They can be accessed by the respective getter-functions *getCowic()* and *getCoric()*. In this case, the former is chosen since the adapter might have to write an iteration checkpoint.
If so, *saveOldState()* is called on the adapter. The goal of saving the current ("old") solver state is to store all information, which is necessary to be able to rerun exact the same iteration, implying that an iteration of SU^2 with the same computational outcome is expected, if the input is not changed. This is important for implicit coupling algorithms of preCICE if convergence is not met and a time step needs to be restarted. Then, preCICE varies the input of SU^2 in terms of the transferred displacements, which might yield a better result (in terms of forces), meaning that the residuals of the coupled fixed-point system are reduced. The quantities, which need to be stored for this checkpointing strategy, include variables associated with the nodal coordinates of the fluid mesh, the grid movement and the solution of previous time steps.
After a solver iteration of SU^2 *advance()* is called on the adapter object. This function is the adapter's most extensive one as it includes computing forces at the wet surface and transferring them to preCICE. Moreover, it triggers the coupling algorithm in preCICE, as well as receiving and setting the nodal displacements at the FSI interface computed by the structural solver. There is no predefined function available in SU^2, which computes forces at certain mesh nodes. Therefore, I implemented this computation in the *advance()* function. First of all, the adapter again checks whether the respective MPI process works on the wet surface or not via *processWorkingOnWetSurface*. Computing and forwarding forces is only necessary for the nodes in immediate contact with the FSI interface. If a process includes wet surface nodes, the adapter determines the kind of flow regime of SU^2 (compressible or incompressible, viscous or inviscid flow). As mentioned in Section 4.2.1, SU^2 is currently not able to run incompressible simulations with ALE support. However, I already include the force computation for incompressible flows in the adapter, in case this capability will be added in future releases. In addition, the adapter computes a factor for redimensionalizing forces (as SU^2 features non-dimensionalized simulations as well). If the current simulation is dimensional, this factor evaluates to 1. In order to explain the force calculation, I assume the general, three-dimensional case of a simulation governed by the NSE. The overall force at a node acting on the FSI interface is then given by a viscous term arising from the viscous stress tensor

and an inviscid term determined by the (dynamic) pressure. The computation is done as follows:

$$f_i = -(p_{\text{total}} - p_{\text{static}})n_i A + \tau_{ij} n_j A \quad \forall i = 1, 2, 3, \quad \text{with} \tag{5.4a}$$

$$\tau_{ij} = \mu(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}) - \frac{2}{3}\mu \frac{\partial v_k}{\partial x_k}\delta_{ij} \quad \forall i, j = 1, 2, 3. \tag{5.4b}$$

$\boldsymbol{f}$ denotes the force vector, p pressure (total and static, respectively) and $\boldsymbol{\tau}$ the viscous stress tensor. A and $\boldsymbol{n}$ refer to area and unit outward normal vector of the dual mesh element associated with the node, for which the force is calculated. Note that the pressure term needs to be negated as by definition pressures point inward the fluid control volume, but the pressure force exerted on the solid (i.e. outward the fluid control volume) is required. The explanation of the viscous stress tensor in Equation 5.4b is identical to Equation 2.6. Again, the adapter needs to manage intrafield parallel execution of SU^2. As extensively explained in Section 4.2.3, ghost nodes are introduced in SU^2 in order to build halo-cells after domain decomposition. If decomposition occurs at the wet surface, interface nodes are replicated. Allowing each of the replicates and the original nodes to write forces to preCICE would yield unphysical computation results, as those nodes share the same mapping to the solid mesh and therefore, forces would accumulate at solid nodes. The adapter makes use of the colors assigned to the FSI interface nodes in SU^2. This is technically done by comparing the MPI rank (= color of the process) with the colors of the wet surface nodes. If they do not match, the process works on a duplicate and thus, is not allowed to write forces at such a node. The corresponding data array storing all FSI forces is eventually pushed to preCICE by calling *writeBlockVectorData()* on the *SolverInterface* instance.
The next step is executing *advance()* on the *SolverInterface* object, which uses the length of the current solver iteration of SU^2 as input and returns the prescribed maximum for the next iteration. Internally, preCICE now executes convergence acceleration techniques, if an implicit procedure is chosen and exchanges coupling data with the partner solvers.
In the following, the adapter needs to read the FSI interface nodal displacements calculated by the coupled structural solver. Thus, *readBlockVectorData()* is called on the *SolverInterface* object. The so obtained displacements are set as coordinate variations relative to the nodal positions of the last time step in SU^2. While writing forces must be restricted to the nodes originally belonging to a process, reading and setting displacements needs to be done also for the replicates.
Although it would be possible to trigger the mesh deformation procedure right away, I decided against this strategy as the current time step might have to be restarted and mesh deformation would be unnecessary computational overhead in such a case. Thus, it is triggered at the beginning of the next solver iteration as shown in Algorithm 3.
Now the counterpart of writing an iteration checkpoint comes into play (only for implicit algorithms). preCICE checks whether the fixed-point equation system converges sufficiently or not. In the latter case, upon calling the wrapper function *isActionRequired()* with input argument *getCoric()* on the adapter, the coupling tool signalizes that *reloadOldState()* needs to be executed. Consequently, the solver state prior to the current iteration is retrieved by resetting the respective variables in SU^2.

Clean Exit

The main solver loop of SU^2 is usually exited when *isCouplingOngoing()* in the while-loop condition evaluates to false, which means that preCICE tries to finish the FSI simulation. The last step of the coupling is initiated when the wrapper function *finalize()* is executed on the adapter object. This causes all communication channels related to the coupled simulation to be closed and used memory to be deallocated.

6. Selected Numerical Testcases

In order to validate the developed coupling adapter, several testcases were simulated, which ought to qualitatively and quantitatively confirm the physically correct implementation. Moreover, the capabilities of the adapter for larger-scale two- as well as three-dimensional simulations are shown in the following.

All simulations in this chapter were carried out on the MAC Cluster "CoolMAC" using the "Sandybridge" and "Bulldozer" partitions[1]. The coupled structural solver for all simulations was the module *SOLIDZ* (CSM simulations), which is part of the multiphysics simulation software suite *Alya* that is developed at the Barcelona Supercomputing Center ([44]). Meshes for the testcases were generated with the free software Gmsh ([19]).

Each section in this chapter starts with a short definition of the respective testcase, including its geometry, discretization, expectation of the physical behavior and a motivation for running the simulation. Subsequently, I state the physical and numerical settings used for the run. In the end, the obtained results are shown and discussed briefly.

Beginning with Section 6.1, a very simple two-dimensional testcase is described, which is altered to yield a three-dimensional scenario in Section 6.2. A quantitative comparison of simulation results is possible via running the well-known *FSI3* benchmark testcase ([39]) in Section 6.3. The chapter is closed by Section 6.4, simulating a slender cylinder. This example has a practical background and represents a real-world application.

As for all simulations material and physical parameters as well as solver configurations need to be defined, I forestall them here in order to avoid repeating myself in each section of this chapter. Also, this allows to compare simulations settings with each other more easily. Material and physical parameters are stated in Table 6.1. Fluids are modeled as ideal gases, solids as linearly elastic and isotropic.
Several solver configuration options remain the same for all simulations: Forces and the displacements relative to the last time step are chosen as coupling quantities. In case an implicit coupling algorithm is used, these are monitored by preCICE for convergence of the fixed-point system. In addition, an implicit coupling algorithm extrapolates the coupling quantities with a second-order scheme at the beginning of each time step. If IQN-ILS or V-IQN (recall Section 4.1.1) are chosen for coupling, preCICE takes into account up to 30 iterations of up to 10 previous time steps in order to solve the interface least-squares problem.
Concerning SU^2, time discretization is handled by the implicit Euler method and spatial discretization is dealt with by a second-order scheme in combination with a Roe approximate Riemann solver ([34]). The linear system, which finally yields the solution vector of the flow field, is solved by the FGMRES procedure ([35]). All other solver configurations are given in Table 6.2 or directly stated in the respective sections. Note that some FSI simulations are initialized from a fluid-only start solution, which is calculated for a time period specified with t_{start}. Also, some simulations make use of the dual time stepping technique of SU^2. The corresponding dual time convergence limit ϵ_{dual} relates to reducing the density residual of the fluid domain.
Fluid and solid calculations are started on different nodes of the cluster for all simulations.

[1] For more information on the cluster and its specifications, see `http://www.mac.tum.de/wiki/index.php/MAC_Cluster`.

			2D Flap	3D Flap	FSI3	Cylinder
fluid	density	ρ_F [kg/m^3]	$5.28 \cdot 10^{-5}$	$1.06 \cdot 10^{-4}$	10^3	1.185
	dynamic viscosity	μ [Ns/m^2]	$1.81 \cdot 10^{-5}$	$1.81 \cdot 10^{-5}$	1	$1.831 \cdot 10^{-5}$
	specific gas constant	R_S [J/kgK]	287.058	287.058	287.058	287.058
	specific heat ratio	κ [–]	1.4	1.4	1.4	1.4
solid	density	ρ_S [kg/m^3]	10^{-2}	10^2	10^3	$7.85 \cdot 10^3$
	Young's modulus	E [N/m^2]	10^3	10^6	$5.6 \cdot 10^6$	$5.58 \cdot 10^9$
	Poisson ratio	ν [–]	0.4	0.3	0.4	0.3
flow	Reynolds number	Re [–]	≈ 300	300	200	≈ 11326
	Reynolds length	l_{Re} [m]	1	1	0.1	0.005
	Mach number	Ma [–]	0.3	0.15	0.01	0.101
	free-stream temperature	T [K]	293.15	293.15	0.353	298.15
	free-stream velocity	v [m/s]	≈ 103	51.49	2	35

Table 6.1: Material and physical parameters of all simulation scenarios shown in this chapter.

			2D Flap	3D Flap	FSI3	Cylinder
general	start solution period	t_{start} [s]	1.5	–	2	10^{-2}
	simulation time	t [s]	1.5	7.8	6	0.1
	time step size	Δt [s]	10^{-2}	10^{-3}	10^{-3}	10^{-5}
SU^2	lin. solver conv. limit	ϵ_{lin}	10^{-6}	10^{-7}	10^{-7}	10^{-7}
	lin. solver max. iter.	$\text{maxIter}_{\text{lin}}$	5	50	50	20
	dual time method?		no	yes	yes	yes
	max. # dual steps	$\text{maxSteps}_{\text{dual}}$	–	20	20	20
	dual conv. limit	ϵ_{dual}	–	10^{-7}	10^{-7}	10^{-5}
preCICE	coupling algorithm		IQN-ILS	V-IQN	V-IQN	BGS (Aitken)
	interfield parallel?		no	yes	yes	no
	rel. conv. limit	ϵ_{rel}	10^{-4}	10^{-3}	10^{-4}	10^{-3}
	mapping method		RBF	NN	NN	NN

Table 6.2: Solver configurations for all simulation runs in this chapter.

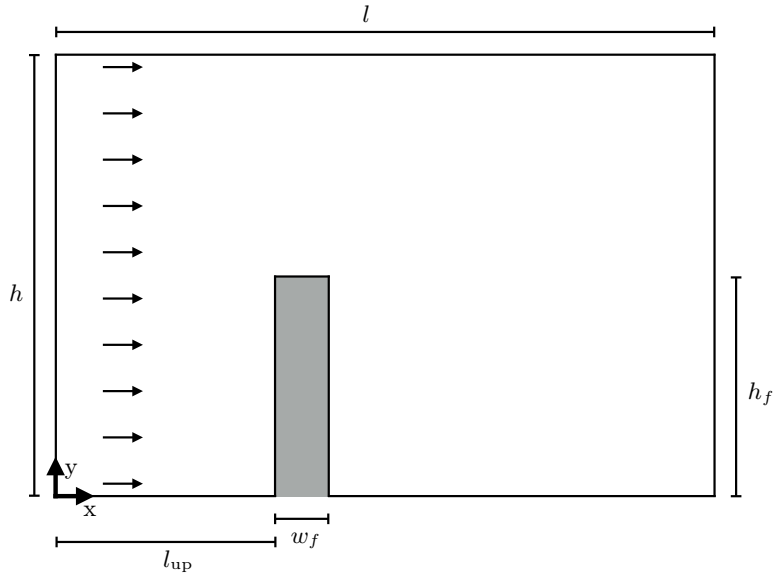


Figure 6.1: Geometry of the two-dimensional flap testcase. The solid material of the flap is colored gray. The inlet is depicted by arrows. Geometrical parameters are given in Table 6.3.

geometrical parameters		value [m]
channel length	l	12
channel height	h	8
channel length upstream	l_{up}	4
flap height	h_f	4
flap width	w_f	1

Table 6.3: Geometrical parameters of the two-dimensional flap testcase shown in Figure 6.1.

6.1 Qualitative Validation: 2D Flap

Case Definition

The first testcase is a two-dimensional simulation of a deformable flap which extends into a channel. At the lower end the flap is clamped to the channel bottom while the upper end can move freely. Figure 6.1 in combination with Table 6.3 defines the geometrical properties of the testcase. A compressible, laminar, viscous flow (governed by the NSE) with constant profile at the inlet is chosen. The inflow is marked by the accumulation of arrows in Figure 6.1. The opposite side of the channel is the outlet where no constraint on the flow is given. Upper and lower wall of the channel enforce no-slip conditions, as does the wet surface.
Fluid and solid domains are discretized by unstructured, triangular meshes, which are not adaptively refined in vicinity of the FSI interface. The initial configurations of the meshes are shown in Figure 6.2. Note that the meshes do, in general, not match at the wet surface although element sizes are almost the same. The fluid mesh is slightly coarser than the structural grid. Further quantitative descriptions of the meshes are given in Table 6.4.

discretization parameters		value
number of degrees of freedom, fluid mesh	$n_{\text{dof,F}}$	2081
number of degrees of freedom, solid mesh	$n_{\text{dof,S}}$	157
number of elements, fluid mesh	$n_{\text{elem,F}}$	3968
number of elements, solid mesh	$n_{\text{elem,S}}$	262
number of elements at wet surface, fluid mesh	$n_{\text{elem},\Gamma_{\text{F}}}$	36
number of elements at wet surface, solid mesh	$n_{\text{elem},\Gamma_{\text{S}}}$	45

Table 6.4: Discretization parameters of the two-dimensional flap testcase meshes shown in Figure 6.2.

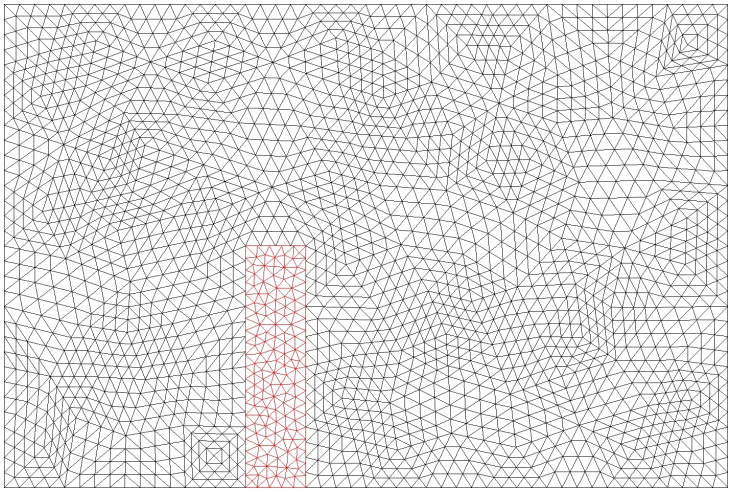


Figure 6.2: Triangular meshes of fluid and solid domain for the two-dimensional flap testcase. See Table 6.4 for quantitative descriptions of the meshes.

This simulation scenario serves solely as qualitative validation. The flap is expected to deform once the flow reaches it. Of course, the free end of the flap should move in positive x direction first due to increasing stagnation pressure upstream of the flap. Subsequently, depending on stiffness and density of the solid material, it is expected to spring back, before the fluid pushes it again in flow direction. For low stiffness values this oscillatory behavior should decline rather quickly as time passes, finally leading to a steady state in which the flap stays in its deformed configuration. Due to the fact that the flap's height is half of the channel's height, it represents a fairly large barrier for the fluid flow. Thus, the movement of the flap should considerably influence the flow field.

The results of this simulation can be used to check whether force computations of the adapter are feasible as the solid domain is expected to deform continuously without regions of dents, which would imply accumulation of forces. Moreover, correct displacement transfer from structural to flow solver and mesh deformation of the fluid domain should yield conformity of both meshes at the wet surface.

Simulation Settings

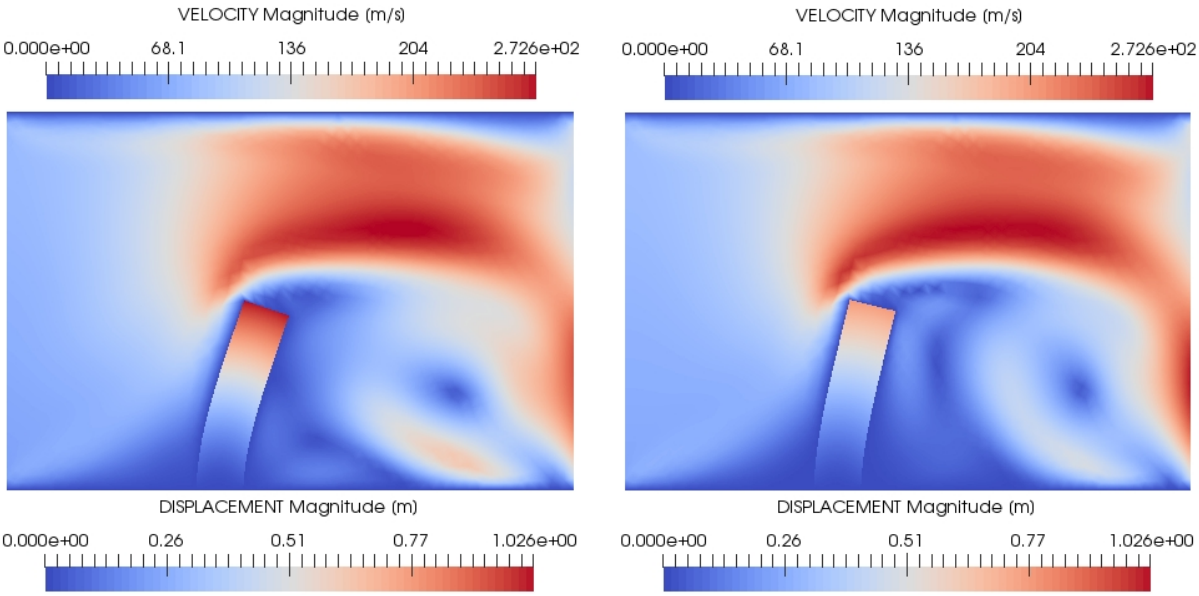
The materials (see Table 6.1) are chosen empirically such that a strong interaction between fluid and solid is present and rather large deformations of the solid domain occur. The AME is thus, a crucial issue in this simulation. The flow conditions are defined using the flap width w_f as Reynolds length (see Table 6.3).

Table 6.2 summarizes the solver configurations. The FSI simulation is started from a single-physics fluid solution allowing the flow field to fully develop. During this period the flap remains in its initial configuration and represents a non-moving, rigid obstacle in the flow channel.

The simulation is run in serial in both intra- and interfield manner as the low number of nodes and elements allows to do so. Computation time does not exceed a 2 minute range (for the 1.5 s of FSI simulation). In total, 324 FSI iterations are needed for the simulation, thus, on average 2.16 iterations are necessary for convergence of the FSI system per time step.

Results

As expected, once the FSI simulation is started from the initial flow solution, the flap is bent towards the outlet of the channel and starts oscillating back and forth until it finally reaches a steady deformation state. The deformation appears to be smooth on the flap surface, which indicates that no unphysically large forces are calculated by the adapter. Also, fluid and solid meshes conform perfectly at the FSI interface, so errors in displacement transfer and mesh deformation procedure can be excluded. Figure 6.3 shows the velocity magnitude of the flow and the displacement magnitude of the flap for two different points in time. The first corresponds to the state of maximum deflection of the flap, which occurs during



(a) Maximum deflection at $t = 0.17$ s.

(b) Steady state at $t = 1.5$ s.

Figure 6.3: Results for the two-dimensional flap testcase for two different time instances. Velocity magnitude of the flow and displacement magnitude of the flap are shown in the maximum deflection state (Figure 6.3a) and in the steady state at the end of the simulation (Figure 6.3b).

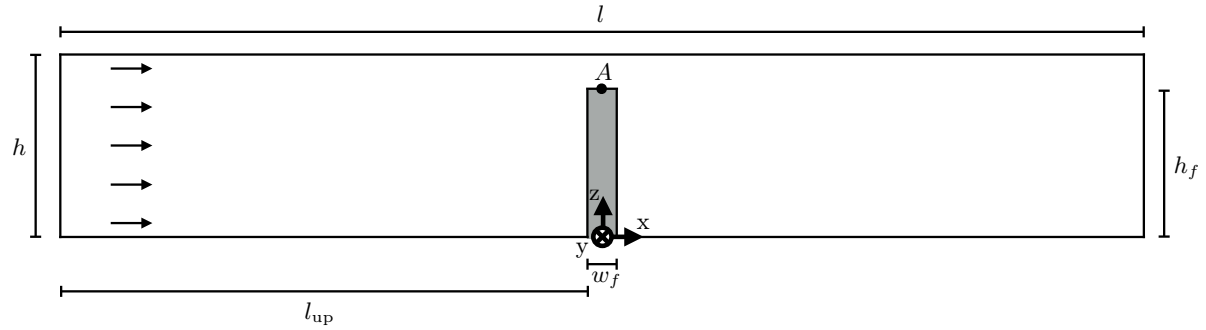
the first oscillation period. The second shows the situation of the steady deformation state at the end of the simulation.

The strong interaction between fluid and structure does not allow to run the same simulation with an explicit coupling algorithm. Attempting to do so fails immediately after the start of such simulations due to instabilities. Although about three orders of magnitude separate fluid and solid density, the low Young's modulus in combination with the rather large time step size yields great influence of the AME. Reducing the time step size to $\Delta t = 10^{-3}$ s is not enough to stabilize the explicit computation. An even smaller time step size should result in stability according to the explanations of Section 3.4, yet this is not tested as it is not the main purpose of this testcase to find a stability margin for weakly coupled algorithms.

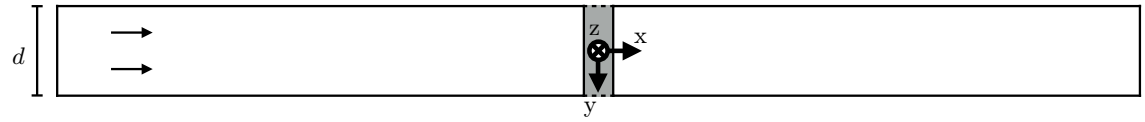
6.2 3D Capabilities: Extended Flap

Case Definition

The next testcase is basically a three-dimensional extension of the scenario shown in the previous section with slight variations in the geometry. Again, a deformable flap is clamped at one end to the channel bottom and can move freely at the other end. The main difference is the additional, constant depth of the problem expanding the simulation to three dimensions. The geometrical properties of the testcase are stated in Figure 6.4, where the setup is shown from two different perspectives. Table 6.5 assignes values to the geometrical parameters. This time, the flap narrows the channel even more than in the previous simulation. At the inlet of the channel, which is highlighted by the arrows in Figure 6.4, a constant inflow profile is prescribed. The flow is laminar, viscous and compressible, thus the NSE describe the fluid motion. The outlet is opposed to the inlet and prescribes no constraint on the flow field. Upper and lower wall in Figure 6.4a (with normals in z-direction) impose no-slip conditions, as well as the complete wet surface. The channel walls with normal vectors pointing in y-direction are chosen as symmetry boundaries. Consequently, the simulation corresponds to an infinitely deep flap.
Spatial discretization of fluid and structural domain is achieved by unstructured tetrahedral meshes. Close to the flap surface the fluid mesh is adaptively refined, while the solid mesh remains uniform throughout the structural domain. Unlike in the two-dimensional flap testcase, the fluid mesh is much finer than the solid grid at the wet surface which corresponds to a more realistic simulation situation when it comes to



(a) Geometry in the xz-plane.



(b) Geometry in the xy-plane.

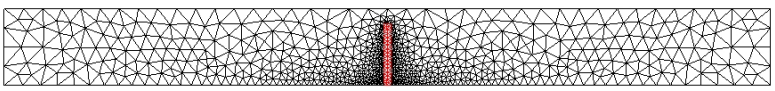
Figure 6.4: Geometry of the three-dimensional flap testcase shown from two different perspectives. The geometrical parameters are defined in Table 6.5.

geometrical parameters		value [m]
channel length	l	20
channel height	h	2
channel length upstream	l_{up}	9.9
flap height	h_f	1.6
flap width	w_f	0.2
channel depth	d	1
watchpoint (at t = 0)	A	(0, 0, 1.6)

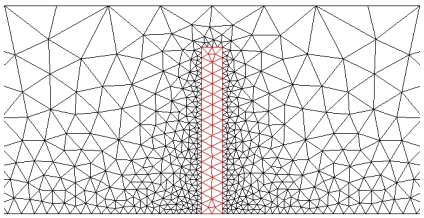
Table 6.5: Geometrical parameters of the three-dimensional flap testcase shown in Figure 6.4.

practical FSI applications. The initial configurations of both meshes are shown in Figures 6.5 and 6.6 from two different perspectives. Note that only the surface meshes are shown, as it is difficult to present the volumetric meshes in a feasible way. In Table 6.6, the mesh parameters are extensively described. In the middle of the flap's top a watchpoint is set (in preCICE), which allows to track the material particle associated with that point as the simulation is run. The movement of the flap can, therefore, be described in terms of the displacements of this watchpoint.

The simulation's main purpose is to check whether the adapter works for three-dimensional problems as well. The adapter is coded such that memory allocation, force calculation and data transfer are dependent on the dimensionality of the scenario. Therefore, no problems should arise switching from a two- to a three-dimensional simulation. Besides, this testcase is also a first glance at real-world applications since due to the third dimension the number of degrees of freedom and the amount of mesh elements becomes



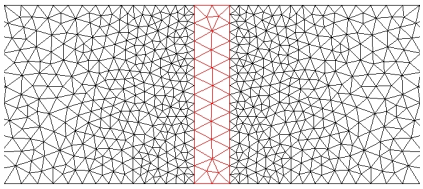
(a) Whole surface mesh shown in xz-plane.



(b) Close up of surface mesh at the flap shown in xz-plane.

Figure 6.5: Fluid and solid surface meshes of the three-dimensional flap testcase shown in the xz-plane with orientation according to Figure 6.4a. The total mesh is shown in Figure 6.5a while Figure 6.5b focuses on the zone nearby the flap.

(a) Whole surface mesh shown in xy-plane.



(b) Close up of surface mesh at the flap shown in xy-plane.

Figure 6.6: Fluid and solid surface meshes of the three-dimensional flap testcase shown in xy-plane with orientation according to Figure 6.4b. The total meshes are shown in Figure 6.6a, while Figure 6.6b focuses on the zone nearby the flap.

discretization parameters		value
number of degrees of freedom, fluid mesh	$n_{\text{dof,F}}$	6291
number of degrees of freedom, solid mesh	$n_{\text{dof,S}}$	689
number of elements, fluid mesh	$n_{\text{elem,F}}$	24161
number of elements, solid mesh	$n_{\text{elem,S}}$	2535
number of elements at wet surface, fluid mesh	$n_{\text{elem},\Gamma_{\text{F}}}$	3522
number of elements at wet surface, solid mesh	$n_{\text{elem},\Gamma_{\text{S}}}$	868

Table 6.6: Discretization parameters of the three-dimensional flap testcase meshes shown in Figures 6.5 and 6.6.

considerably larger, even for such simple simulation setups. By analogy with the two-dimensional testcase, the flap should deform as soon as the fluid flow reaches it. Again the tractions exerted on the structure by the fluid and the counteracting elastic forces are expected to initiate an oscillation of the deformable solid. Consequently, the watchpoint is expected to show oscillatory behavior in x- and z-directions, while due to the symmetry of the problem no notable movement in y-direction should be tracked.

Simulation Settings

The material parameters defined in Table 6.1 are less critical with respect to the AME, compared to the previous testcase. The used Reynolds length corresponds to the channel depth d as given in Table 6.5.

The flow is not initialized from a starting solution as in the previous testcase. Rather, the free-stream values of the flow are used for initialization. Table 6.2 contains all other solver options used for this simulation.

Due to the larger number of mesh nodes and elements, both inter- and intrafield parallelism are employed. The fluid domain is parallelized with 192 processes, while the solid domain is dealt with by five processes. Overall computation time for 7.8 s of physical simulation time is exactly 24 hours. Including all subiterations due to the strongly coupled algorithm an FSI iteration needs 7 s runtime (averaged value). Per time step an averaged number of 1.58 subiterations is needed for convergence of the fixed-point equation system.

Results

Since the simulation is started from the free-stream values, the flow field needs several time steps to develop. Due to the selected material parameters, interaction between the flap and the fluid is weaker than in the previous testcase. Nevertheless, the flap begins to oscillate as expected. However, the oscillation does not decline as fast as in the two-dimensional testcase as the solid material is much stiffer and denser now. This also yields smaller deformations of the flap. Again the deformation of the flap appears to be smooth, which indicates correct force calculation by the adapter. Fluid and solid meshes align at the FSI interface at all times. Thus, displacement transfer and the following mesh deformation work fine. The flap almost fully blocks the flow in the channel leading to a great pressure difference between the upstream and downstream side of the flap. This is depicted for the final time instance at $t = 7.8$ s in Figure 6.7. Note that again the channel is shown in the xz-plane. Streamline representations

of the flow field are given in Figure 6.8 for different time instances in order to illustrate the development of the fluid flow with respect to time. In the wake of the flap a recirculation zone is formed while at the throughput the fluid is vastly accelerated.

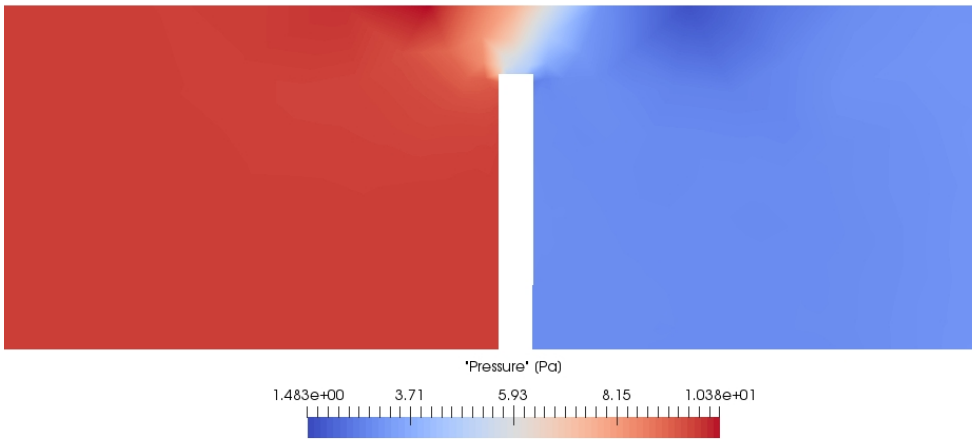
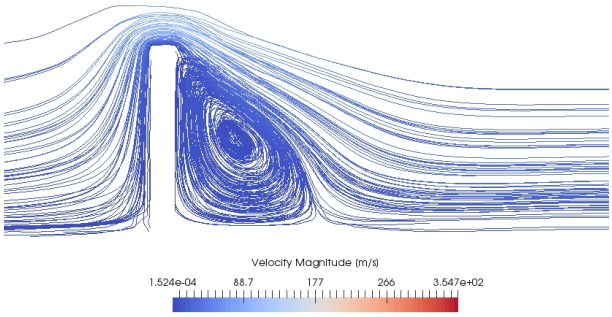
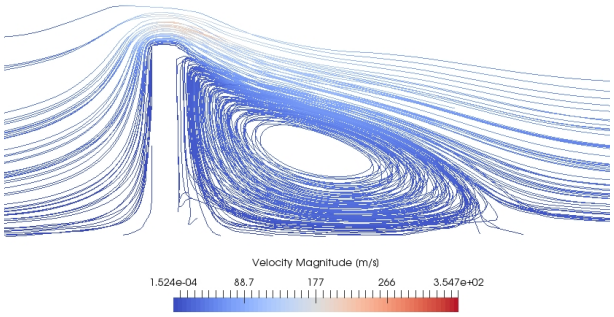


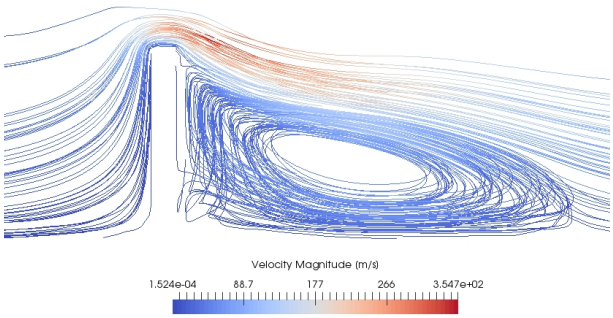
Figure 6.7: Pressure upstream (left) and downstream (right) of the flap shown in xz-plane at $t = 7.8$ s. The orientation is according to Figure 6.4a. Note that the flap displacements are very small, such that they are barely visible in this figure.



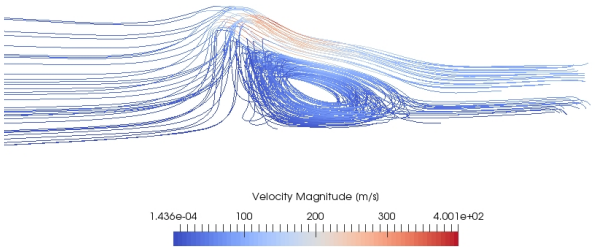
(a) $t = 0.05$ s.



(b) $t = 1$ s.



(c) $t = 7.8$ s, Perspective 1.



(d) $t = 7.8$ s. Perspective 2.

Figure 6.8: Streamlines of the velocity field in vicinity of the flap for three different time instances. The orientation is according to Figure 6.4a (xz-plane) for the first three figures. Figure 6.8d shows the same time instance as Figure 6.8c, yet the view is rotated around the z-axis to yield a more spatial perspective. Note the development of the recirculation zone behind the flap and the increasing flow speed at the gap between flap and channel walls as time passes.

In contrast to the previous testcase, the movement of the flap cannot be easily captured graphically because the deformations are much smaller. However, the watchpoint defined in Figure 6.4a and Table 6.5 allows a quantitative description of the flap oscillation. Figure 6.9 depicts the watchpoint movement in all three spatial directions. The influence of the high stiffness of the flap material is reflected in the oscillations in x- and z-direction, which show a relatively high frequency and seem to decay only very slowly. The transient phase lasts for about three seconds. After this time span the flap oscillates quite

steadily. Surprisingly, the oscillation is also present in the y-direction. However, the amplitudes are in the range of 10^{-8} m such that these displacements can be neglected. They may result from numerical errors such as the not perfectly symmetric mesh. The bias towards positive y-displacement values underlines this as physical oscillations would include both positive and negative displacement values in y-direction for such a symmetric setting. Table 6.7 characterizes the oscillations with respect to frequency, mean value and amplitude averaged over the time interval [3 s, 7.8 s]. Displacements in x- and z-direction oscillate with the same frequency since they are geometrically related. Looking closer to Figure 6.9c, the oscillation in y-direction is not as smooth as in the other two cases. Many smaller waves of higher frequency are superimposed to form the oscillation (which has the same frequency as the other two oscillations). This again indicates that the movement in y-direction originates from numerical errors.

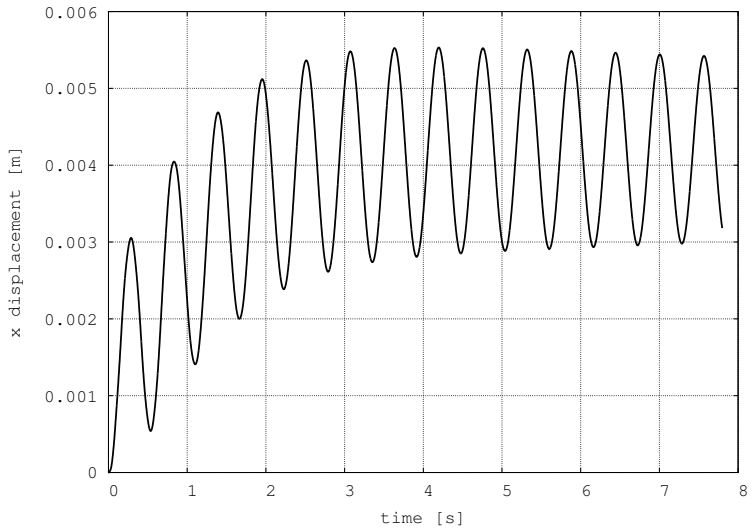
	mean $\pm$ amplitude [frequency]
u_x	$(4.18 \pm 1.30) \cdot 10^{-3}$ m [1.78 Hz]
u_y	$(2.45 \pm 1.75) \cdot 10^{-8}$ m [1.78 Hz]
u_z	$(-2.90 \pm 3.89) \cdot 10^{-6}$ m [1.78 Hz]

Table 6.7: Further description of the oscillations (shown in Figure 6.9) of the watchpoint A in terms of mean and amplitude value, as well as frequency of the displacements u in all three directions. The quantities are obtained by averaging over the time period [3 s, 7.8 s].

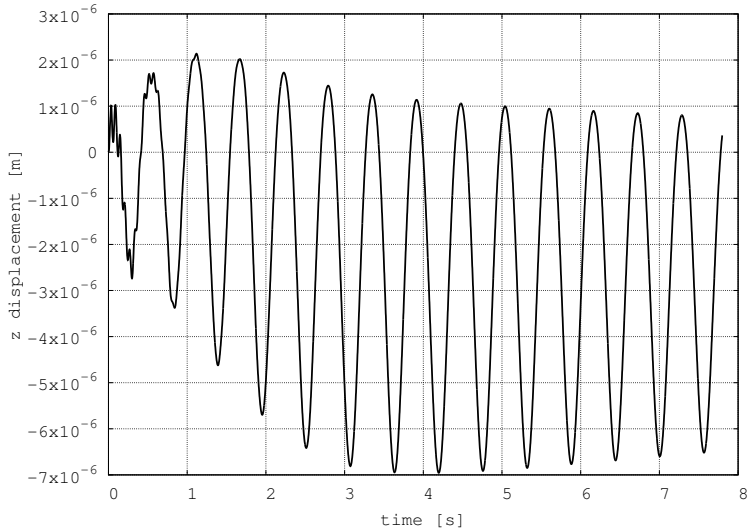
6.3 Quantitative Validation: FSI3 Benchmark

Case Definition

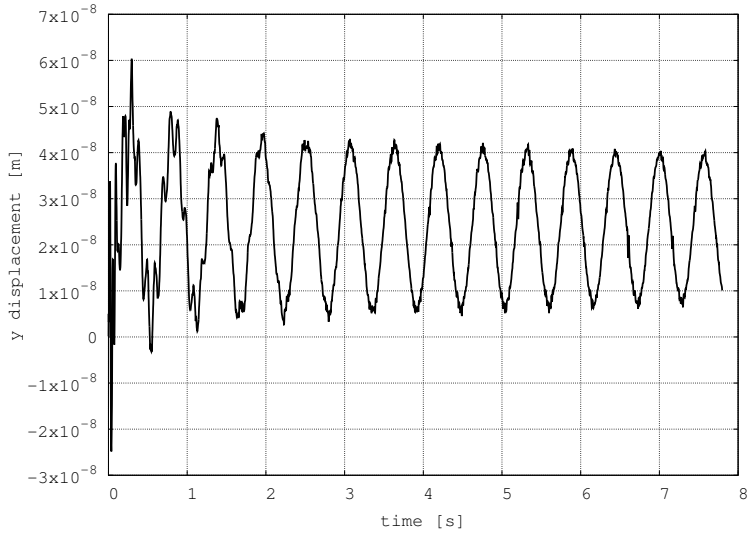
The testcases in Sections 6.1 and 6.2 show the technical and qualitative correct implementation of the adapter. However, no reference results are given for these two generic scenarios such that quantitative benchmarking of the adapter is still necessary. In this section, the widely used numerical FSI benchmark testcase *FSI3* proposed in [39] is used to rank the performance of simulations with the developed adapter. The testcase is two-dimensional and consists of a rigid cylinder (actually a circle in 2D) with a deformable cantilever attached downstream. The geometry of the problem is defined in Figure 6.10. Definitions of the corresponding parameters are given in Table 6.8. The inlet is again marked by arrows, its opposed side is the outlet. Upper and lower wall of the channel, as well as cylinder and cantilever prescribe no-slip conditions. The original benchmark proposes a parabolic inflow velocity profile. However, SU^2 only supports constant inflow profiles in its current version. Therefore, the channel length upstream the cylinder is extended with an offset. Due to the no-slip conditions of upper and lower wall a parabolic velocity profile develops naturally by analogy with the well-known plane Poiseuille flow (see e.g. [26], [3]). Because of the higher computational cost resulting from the additional channel offset, the channel length behind the cantilever is reduced in comparison to the original geometry. This should not significantly influence the physics of the testcase as the length is chosen mostly in order to visualize the flow conditions behind the test object. Vortices in the wake of the cylinder induce oscillations of the cantilever. This is also fostered by the slightly asymmetric position of the cylinder in the channel. The movement of the cantilever approaches a periodic oscillation after a transient phase. Two more variations of the benchmark scenario are given in [39]. However, differences are only due to material parameters as well as the inflow velocity (determined by different Reynolds numbers). The FSI3 setting is the most challenging as it includes the highest fluid velocity and the lowest density ratio $M_A = \frac{\rho_S}{\rho_F}$. Thus, the interaction between fluid and solid is very strong and the AME becomes a crucial issue. Moreover, the parameters of the scenario allow for large displacements of the cantilever. Originally, this test was designed for incompressible, laminar, viscous flow regimes. As mentioned in 4.2.1, due to current limitations of SU^2 no incompressible FSI simulations can be executed. Therefore, a low Mach number is chosen for the fluid flow in order to reduce compressible effects to a minimum.
Unstructured, triangular meshes are used to discretize fluid and solid domain. Around the cylinder with its attached flap the fluid mesh is adaptively refined, while the much coarser structural mesh is rather constant throughout its domain. The initial configurations of the meshes are shown in Figure 6.11. A numerical description of the grids is given in Table 6.9. In order to be able to describe the oscillations of the cantilever, a watchpoint A is set at the end of the deformable solid as defined in Figure 6.10 and Table 6.8.



(a) Displacement in x-direction.



(b) Displacement in z-direction.



(c) Displacement in y-direction.

Figure 6.9: Displacements of the watchpoint with respect to time for all three spatial directions. The oscillations are further characterized in Table 6.7.

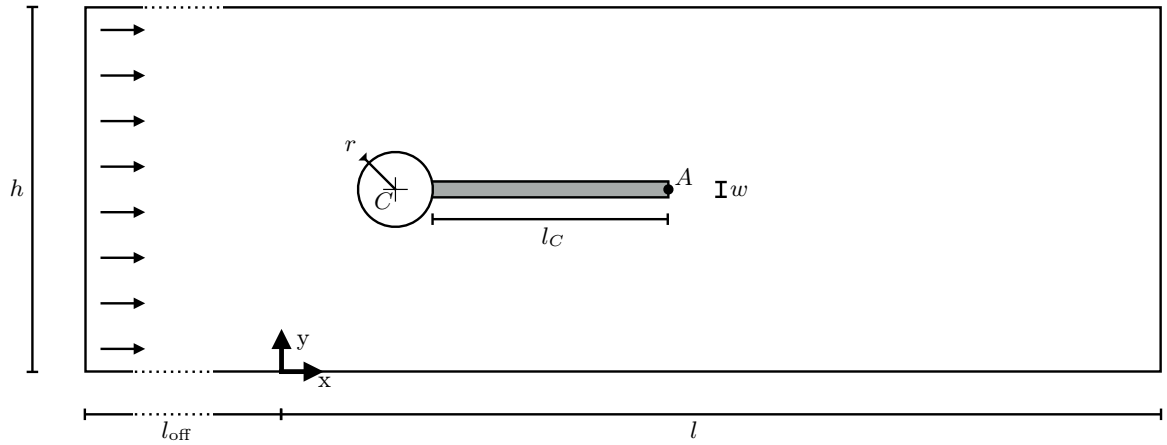


Figure 6.10: Geometry of the numerical FSI benchmark proposed in [39]. Geometrical parameters are given in Table 6.8. The cylinder is positioned slightly off-centered.

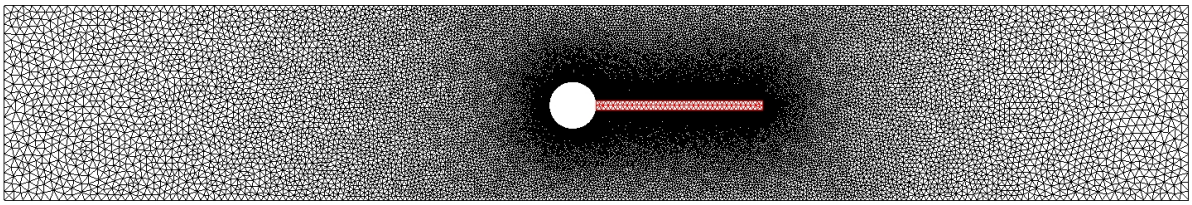
geometrical parameters		value [m]
reduced channel length	l	1.5
additional channel offset	l_{off}	1
channel height	h	0.41
radius of cylinder	r	0.05
center of cylinder	C	(0.2, 0.2)
length of cantilever	l_C	0.35
width of cantilever	w	0.02
watchpoint (at t = 0)	A	(0.6, 0.2)

Table 6.8: Geometrical parameters of the benchmark scenario shown in Figure 6.10.

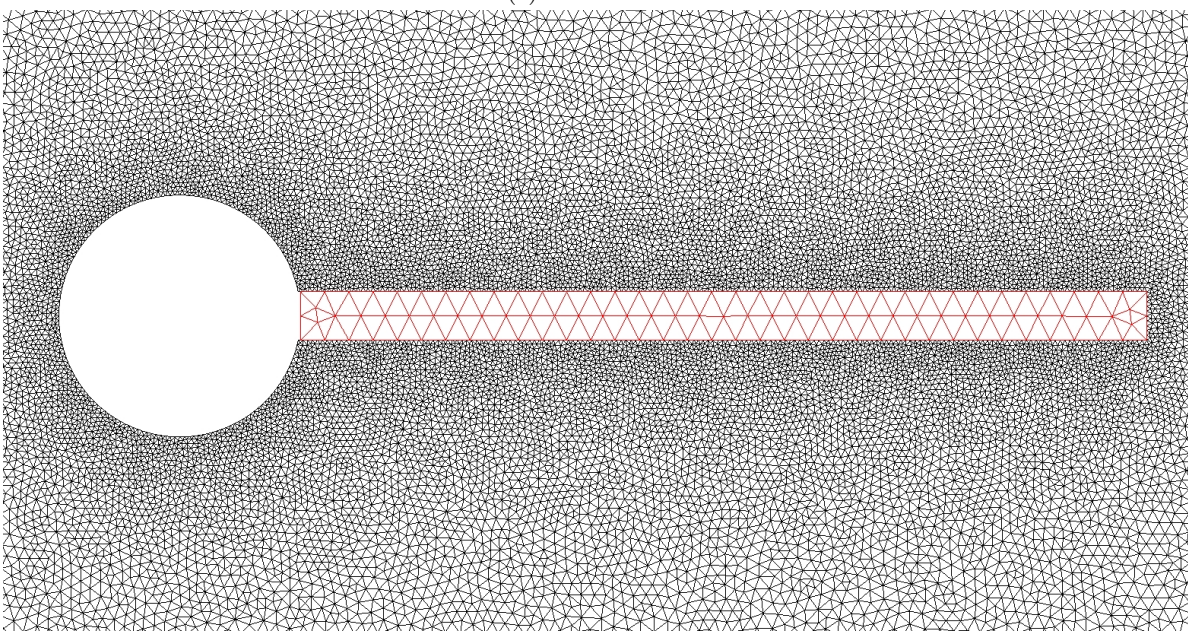
The necessity of the additional channel offset as described above becomes apparent by considering the single-physics CFD test *CFD1*, which is also proposed in [39]. For this test the cantilever is considered as rigid obstacle just as the cylinder. The test is run for the original and the channel geometry with additional offset under same solver conditions. Also, adaptive refinement at the cylinder and the cantilever is the same for both used meshes, preserving the comparability of the results. Lift and drag forces acting on cylinder and cantilever serve as benchmark quantities. Table 6.10 compares the obtained force values for both geometries with the reference results of CFD1. The values obtained by using the additional offset are significantly closer to the reference results.
Moreover, experimenting with different offset lengths for the FSI3 test highlights an optimum of 1 m as depicted by the velocity profiles in Figure 6.12. The profile corresponding to the original geometry, i.e. without offset length, is almost perfectly constant. The only disturbances occur at the no-slip walls. An offset of 0.5 m results in a more developed profile. However, this is not sufficient to obtain a parabolic shape as is the case for an offset of 1 m. Consequently, the latter length is chosen for the FSI simulation.

discretization parameters		value
number of degrees of freedom, fluid mesh	$n_{\text{dof,F}}$	26347
number of degrees of freedom, solid mesh	$n_{\text{dof,S}}$	111
number of elements, fluid mesh	$n_{\text{elem,F}}$	51682
number of elements, solid mesh	$n_{\text{elem,S}}$	146
number of elements at wet surface, fluid mesh	$n_{\text{elem},\Gamma_{\text{F}}}$	362
number of elements at wet surface, solid mesh	$n_{\text{elem},\Gamma_{\text{S}}}$	72

Table 6.9: Discretization parameters of the FSI3 testcase meshes shown in Figure 6.11.



(a) Whole mesh.



(b) Close up of mesh at cylinder and cantilever.

Figure 6.11: Fluid and solid meshes of the FSI3 benchmark testcase. The total mesh is shown in Figure 6.11a while Figure 6.11b focuses on the zone nearby the cylinder and the attached cantilever. Further descriptions of the meshes are given in Table 6.9.

	lift force [N]	drag force [N]
no offset	1.005	13.39
offset	1.076	14.36
reference	1.119	14.29

Table 6.10: Comparison of geometry without and with 1 m channel offset for the test CFD1 ([39]).

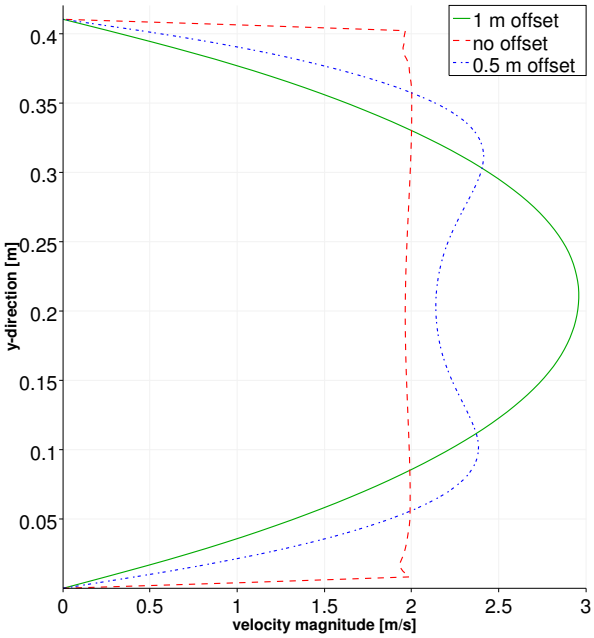


Figure 6.12: Velocity profiles at $x = 0$ for different offset lengths l_{off} (compare Figure 6.10). A parabolic shape is achieved for an additional length of 1 m.

Simulation Settings

As explained before, the testcase is usually simulated with incompressible flows. Nevertheless, due to the already mentioned limitations of SU^2, the flow is described by the compressible, laminar NSE in this case. Choosing a very low Mach number is supposed to minimize compressible effects. It is defined in Table 6.1 in line with all other physical parameters of the simulation. The Reynolds number of this testcase is determined using the cylinder diameter (see Table 6.8) as Reynolds length.

The flow is initialized from a fluid-only start solution. The cantilever is modeled as rigid obstacle during this period and does not move. See Table 6.2 for all other solver settings. The start solution serves also as simple scale-up test to determine the optimum number of processes to parallelize SU^2 with. Table 6.11 summarizes the corresponding findings. Based on these results, the coupled FSI simulation is run with 120 processes on the fluid domain, while five processes handle the structural computation.

procs	nodes after decomposition	ghost nodes	nodes per proc	computation time [s]
60	30532	4185	508.87	109.2
100	32000	5653	320	73.3
110	32371	6024	294.28	72.3
120	32720	6373	272.67	69
140	33184	6837	237.03	69.5
≥ 160	-	-	-	-

Table 6.11: Scale-up test for parallelization of the FSI3 benchmark in SU^2. 2 s of single-physics solution are calculated. The best result is obtained with 120 processes. For 160 processes or more the domain decomposition procedure in SU^2 does not finish successfully.

The computation time for 6 s of the coupled FSI simulation is about 12.5 hours. Per time step 2.29 iterations are needed for convergence of the coupled fixed-point system with a single FSI iteration having a run time of 3.29 s. The latter two values are averaged.

Results

It takes about 3 s until the cantilever movement converges to a periodic oscillation as expected. Figure 6.13 presents the velocity field in vicinity of the test object at time instance $t = 6$ s. The periodic

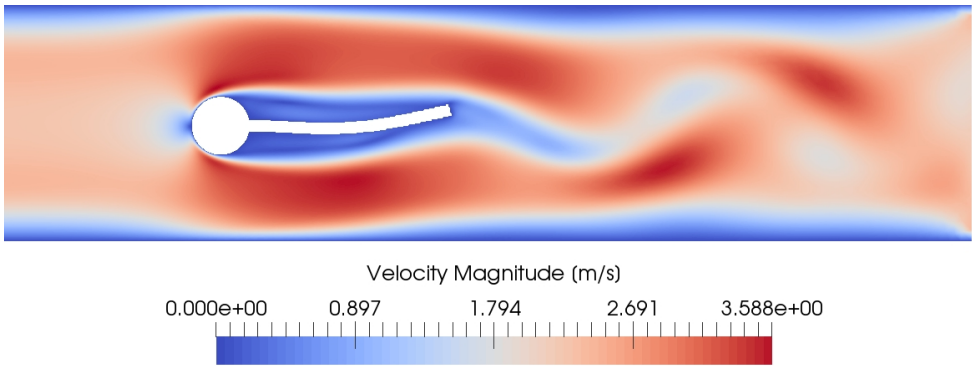
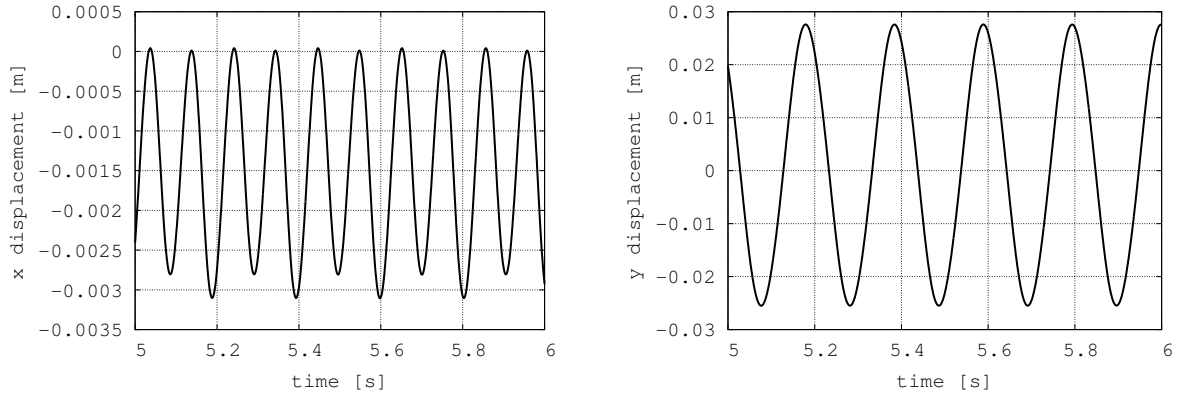


Figure 6.13: Velocity magnitude at $t = 6$ s. Note the periodic, alternating low and high velocity zones in the wake of the cantilever.

oscillation can be observed in the wake of the cantilever, where high and low velocity regions alternate. Note that the low velocity region is dragged behind the cantilever tip. The oscillation of the watchpoint is described in Figure 6.14, where the displacements in x- and y-direction are plotted for the last second of the simulation. Table 6.12 captures the oscillations with respect to mean and amplitude values, as well as the frequency. The obtained results are compared to the FSI3 reference given in [39]. Both displacements in x- and y-direction agree quite well with the reference values. This validates the implementation of the coupling adapter quantitatively.
The differences to the reference results have several reasons. First of all, even though the Mach number is chosen very small, slight compressible effects may still be present such that this approach cannot be expected to yield the exactly same values as obtained in an incompressible simulation. Next, refining both meshes should produce more accurate results[2]. Especially, the solid mesh is very coarse. This yields less degrees of freedom of the cantilever and consequently restricts its movement, which explains why the absolute values (both mean and amplitude) of the displacements are too small compared to the reference. Moreover, the FSI3 reference results were calculated using a smaller time step size of $\Delta t = 0.0005$ s with a monolithic solver. Also, configuring SU^2 and preCICE more strictly in terms of the convergence criteria should improve the simulation outcome. Yet, the probably most severe source of errors is the mapping procedure. Since the meshes are far from matching in a node-to-node manner, conservatively mapping forces by the NN method causes oscillations in the forces, which are received at the solid interface. This is due to the fact that the number of fluid nodes, which are assigned to a single solid node, is not necessarily constant over the structural FSI interface. E.g. one solid node may receive forces from five fluid nodes, while a neighboring structural node gathers force values from six fluid nodes. For a high number of assigned fluid nodes, difference of one or two fluid nodes is not significant (relative influence is small). However, if only a few fluid nodes are assigned to the solid nodes, even a difference of one node can have a large impact on the force value, as the relative influence of a single fluid node is then considerably higher. Several simulation attempts with finer structural meshes failed due to these oscillations. The choice of the relatively coarse structure mesh used for the FSI3 benchmark is, therefore, not arbitrary. It smoothes out the influence of single fluid nodes on the overall forces received at structural nodes. Yet, this incorporates the drawback of less degrees of freedom for the cantilever movement and introduces spatial oscillations in the fluid mesh with large wavelength. NN mapping causes steps in the fluid mesh as depicted in Figure 6.15. The displacements of single solid nodes are transferred consistently (recall Section 4.1.3). Consequently, all fluid nodes associated with the same solid node are in line. Overall, spatial oscillations result in the fluid mesh, which reduce computational accuracy. The wavelength of these oscillations is larger, the greater the difference in refinement of both meshes is at the wet surface. The computational problems arising from mapping forces conservatively are known ([7]). They can be reduced by using meshes, which match to a greater extent and by applying more elaborate mapping methods, such as RBF-based interpolation. Especially, the latter improvement can easily be realized, as preCICE offers a wide variety of sophisticated RBF mapping methods (recall Section 4.1.3). However, at the time of obtaining these simulation results, those methods have not yet been implemented with support for parallel simulations in preCICE.

[2]The fluid mesh used for obtaining the reference results contains about 10 times more degrees of freedom.



(a) Displacement in x-direction.

(b) Displacement in y-direction.

Figure 6.14: Displacements of the watchpoint with respect to time in x- and y-direction. The oscillations are further characterized in Table 6.12.

		mean ± amplitude [frequency]
achieved	u_x	$(-1.46 \pm 1.48) \cdot 10^{-3}$ m [9.77 Hz]
	u_y	$(1.05 \pm 26.53) \cdot 10^{-3}$ m [4.88 Hz]
reference	u_x	$(-2.69 \pm 2.53) \cdot 10^{-3}$ m [10.9 Hz]
	u_y	$(1.48 \pm 34.38) \cdot 10^{-3}$ m [5.3 Hz]

Table 6.12: Further description of the oscillations (shown in Figure 6.14) of the watchpoint A in terms of mean and amplitude value, as well as frequency of the displacements u in x- and y-direction. The quantities are obtained by averaging over the time period $[5\text{ s}, 6\text{ s}]$.

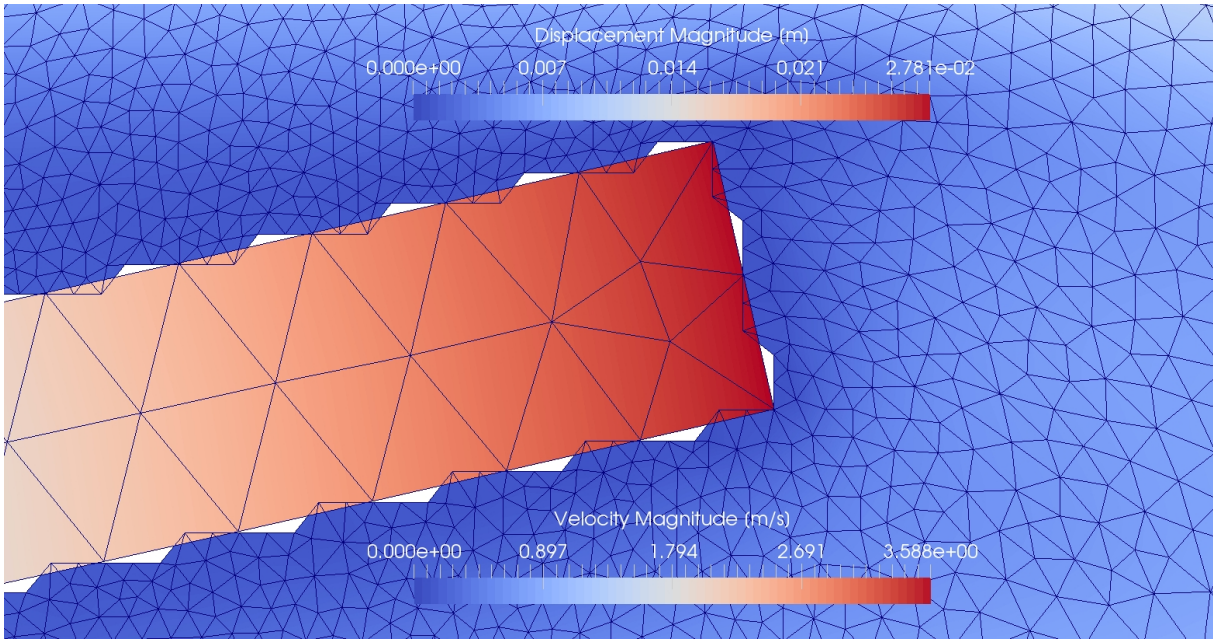


Figure 6.15: Oscillations in the fluid mesh at the end of the cantilever at $t = 6$ s due to the simple NN mapping and the large differences in element sizes of the two meshes.

6.4 Practical Application: Slender Cylinder

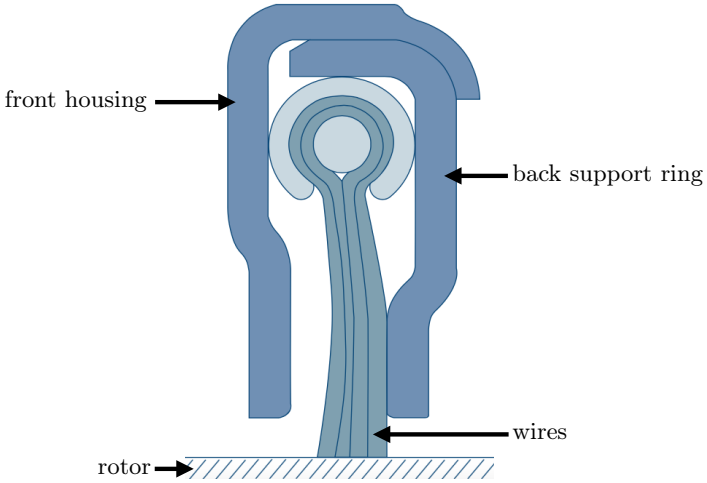
Case Definition

The chair of Flight Propulsion at Technical University of Munich and MTU Aero Engines work on improving brush seals. These sealing elements are used in a wide variety of applications throughout the discipline of mechanical engineering, including flight propulsion systems, gas turbines or steam turbines. Efficient sealing is needed between static (stator) and dynamic elements (rotor) of a machine, separating two zones of different operational conditions. Brush seals allow for large relative velocities between stator and rotor (up to 500 m/s) and entail significantly lower leakage than alternative, conventional technologies such as labyrinth seals. It is possible to separate gases from gases but also gases from liquids with brush seals ([9]).

The name of these seals originates from their characteristic structure. See Figure 6.16 for an example of a seal brush and its composition. Many deformable, slender wires are clamped together to form a brush, which is spatially quite loose in its initial configuration. The wires are in contact with the rotor and allow for both axial and radial relative motion between stator and rotor because of their elastic properties. Due to the movement of the rotor, fluid flows against the brush wires. As a consequence, they are pushed against the back support ring. The wires align with each other and the brush becomes more and more compact, yielding a very leak-tight barrier for fluids. Of course, this behavior is also favored by friction arising from the contact of the wires with the rotor. However, the fluid flow significantly influences the deformation of the wires, which then impact the flow field in reverse, as the tight sealing obstacle develops. Thus, brush seals are clearly governed by the mutual influence of fluid and solid, allowing to further investigate the physics of these seals with FSI simulations in order to finally, develop improved, more sophisticated brush seals.

In collaboration with the chair of Flight Propulsion, we plan to stepwise approximate the real conditions in brush seals, beginning with the reduced-model research on single wires. A corresponding simulation is executed with SU^2 and preCICE using the newly developed coupling adapter. This demonstrates its applicability for real-world, three-dimensional simulations of larger scale.

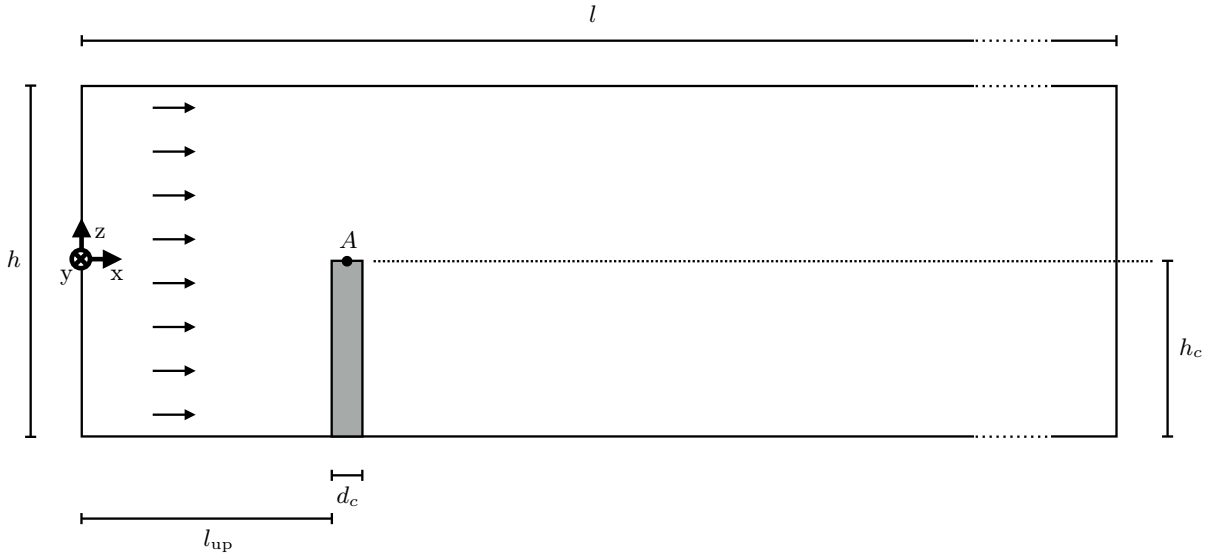
(a) Sealing element of a brush seal. Picture taken from [9].



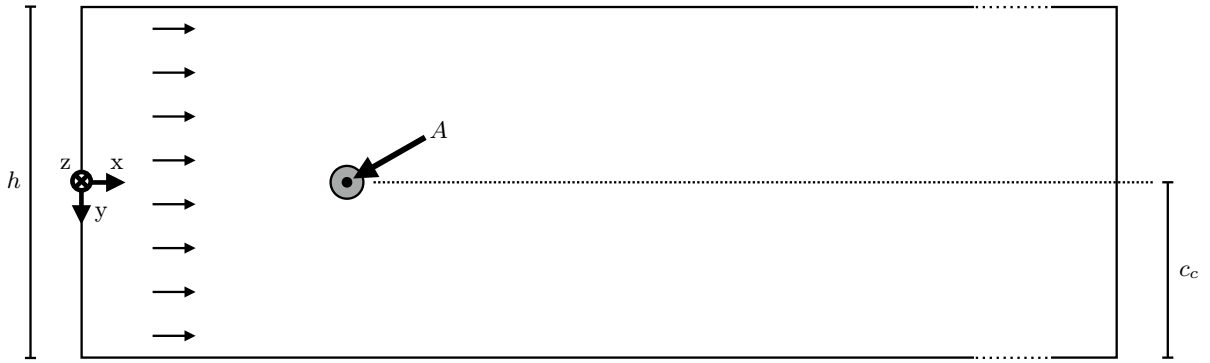
(b) Schematic cross section of the sealing element with its housing. Figure adapted from [9].

Figure 6.16: A typical sealing element of a brush seal is shown in Figure 6.16a. The composition of it is depicted in Figure 6.16b. Note that the housing and the rotor surface are shown in addition.

The geometrical and physical parameters of the simulation are taken from [9]. A deformable, slender cylinder, which is clamped to the channel bottom, represents a single wire of the brush seal. In [9] the cylinder is simulated with a reduced three-dimensional model, using a symmetry boundary condition at the upper (free) end of the cylinder. Thus, the cylinder top and its influence on the behavior of the overall simulation is not included. In contrast, in the scope of this work a fully three-dimensional FSI simulation is run, including resolving the cylinder top. This introduces another degree of freedom for the movement of the structure, making the problem more complex. Also, the simulation is computationally more costly due to resolution of the cylinder top. The geometrical setup of the scenario is shown in Figure 6.17,



(a) Geometry in the xz-plane.



(b) Geometry in the xy-plane.

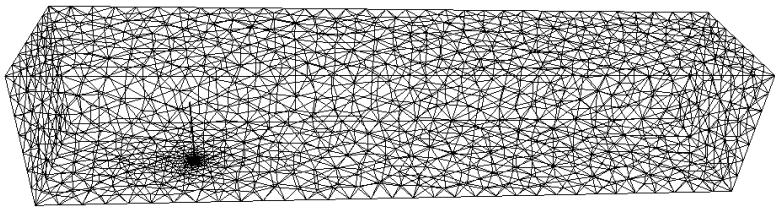
Figure 6.17: Geometry of the three-dimensional, slender cylinder scenario depicted from two different perspectives. The geometrical parameters are defined in Table 6.13.

geometrical parameters		value [m]
channel length	l	0.5
channel height/depth	h	0.1
channel length upstream	l_{up}	0.09975
cylinder height	h_c	0.05
cylinder diameter	d_c	0.005
cylinder center offset	c_c	0.05
watchpoint (at t = 0)	A	(0.1, 0, 0)

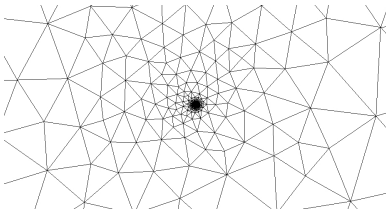
Table 6.13: Geometrical parameters of the three-dimensional showcase as defined in Figure 6.17.

discretization parameters		value
number of degrees of freedom, fluid mesh	$n_{\mathrm{dof,F}}$	38171
number of degrees of freedom, solid mesh	$n_{\mathrm{dof,S}}$	646
number of elements, fluid mesh	$n_{\mathrm{elem,F}}$	146396
number of elements, solid mesh	$n_{\mathrm{elem,S}}$	2504
number of elements at wet surface, fluid mesh	$n_{\mathrm{elem},\Gamma_{\mathrm{F}}}$	52218
number of elements at wet surface, solid mesh	$n_{\mathrm{elem},\Gamma_{\mathrm{S}}}$	1088

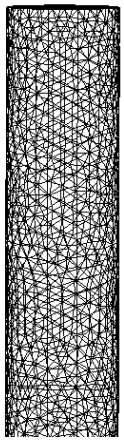
Table 6.14: Discretization parameters of the showcase meshes displayed in Figure 6.18.



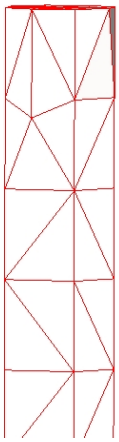
(a) Whole channel surface mesh.



(b) Refinement at the cylinder.



(c) Cylinder surface fluid mesh.



(d) Cylinder surface solid mesh.

Figure 6.18: Surface meshes of the slender cylinder problem. The total fluid mesh is shown in Figure 6.18a while Figure 6.18b focuses on the zone nearby the cylinder. The wet surface meshes (at cylinder top) are depicted in Figures 6.18c and 6.18d for the fluid and solid domain, respectively. Further descriptions of the meshes are given in Table 6.14.

corresponding parameters are defined in Table 6.13. A constant inlet velocity profile is chosen (depicted by the set of arrows in Figure 6.17). The opposite side of the channel is the outlet and prescribes no conditions on the flow. Furthermore, the lower wall in Figure 6.17a enforces the no-slip condition, as does the cylinder surface (wet surface). The upper wall (in Figure 6.17a) allows the fluid to slip freely. Both boundaries with normal vectors pointing in y-direction, i.e. upper and lower wall of Figure 6.17b, are chosen as symmetry conditions.
The problem is spatially discretized by unstructured, tetrahedral meshes. They are shown in Figure 6.18. Note that just the surface meshes are shown, as it is difficult to display the volumetric grids in a reasonable way. Additionally, a numerical description of the grids is given in Table 6.14. The fluid grid is adaptively refined in vicinity of the cylinder, while the solid elements are mostly of constant size. The fluid grid is much finer than the structural mesh at the FSI interface. This is depicted in Figures 6.18c and 6.18d at the top of the cylinder. In order to track the deformation of the cylinder, a watchpoint is set to the middle of its top surface in preCICE.

According to the results in [9], the cylinder is expected to start oscillating in all three-directions. While movement in z-direction is solely due to the geometrical constraints, two independent oscillations should be observed for x- and y-displacements of the cylinder.

Simulation Settings

The flow is characterized as compressible, viscous and turbulent, thus it is governed by the RANS equations. The turbulence closure problem is solved by using the *Menter Shear Stress Transport* (SST) model ([28]) with a turbulence intensity of 5%. All other material and physical parameters are defined in Table 6.1. The cylinder diameter (see Table 6.13) serves as Reynolds length.

The FSI simulation is initialized from a fluid-only start solution. During this period, the cylinder represents a rigid, non-moving obstacle in the channel. As in the previous benchmark scenario, the start solution is used for determining the optimal number of processes to parallelize SU^2 with, in order to reduce the overall computation time. The results are stated in Table 6.15. Thus, SU^2 is parallelized using 230 processes. Five processes work on the structure domain. No interfield parallelism is used for

this simulation as calculating a single time step with the fluid solver exceeds the runtime of the solid solver by far. The latter is fast enough such that no significant difference in the overall runtime of the simulation is expected.

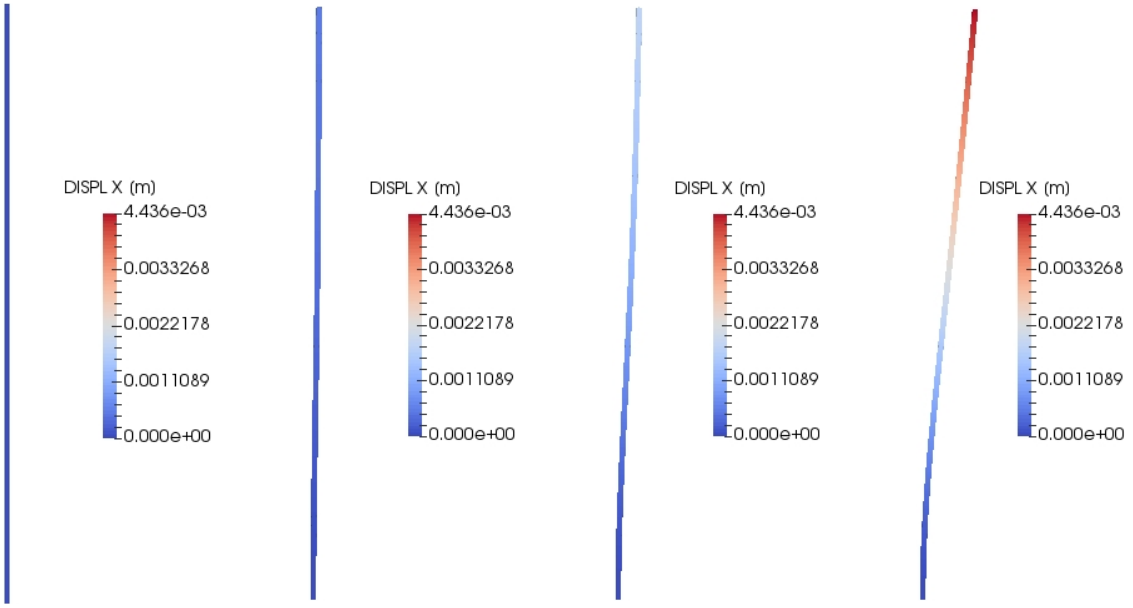
procs	nodes after decomposition	ghost nodes	nodes per proc	computation time [s]
210	65142	26971	310.2	2392
220	65712	27541	298.69	2246
230	66414	28243	288.76	2171
245	67242	29071	274.46	2290
300	70418	32247	234.73	2387

Table 6.15: Scale-up test for parallelization of SU^2 for the slender cylinder scenario. 10^{-2} s of single-physics solution are calculated. The best result is obtained with 230 processes.

The turbulence model is solved with an upwind spatial discretization scheme of first order, while the implicit Euler method deals with time discretization.
An implicit, serial coupling algorithm with acceleration via dynamic Aitken relaxation is chosen (for a reminder, see Section 4.1.1) in preCICE. The relaxation value at the beginning of a time step is set to 0.1. See Table 6.2 for the rest of the solver configurations.
The runtime for $2.3 \cdot 10^{-3}$ s of the FSI simulation is roughly 19.7 hours. An average of 3.98 iterations per time step is needed, thus, a single FSI subiteration lasts 77.47 s.

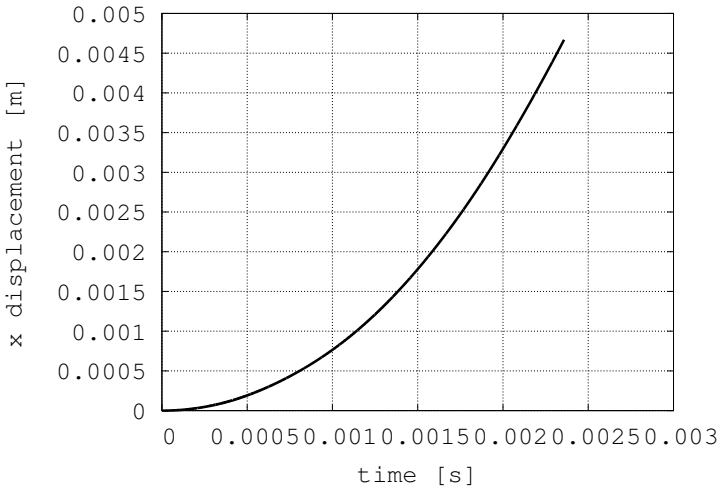
Results

Due to crashes of the simulation, no oscillation of the cylinder can be observed in the obtained results. However, the crashes do not relate to errors in the developed adapter, but arise from physical restrictions on the simulation and its initialization. Large deformations occur at the beginning of the simulation yielding highly distorted solid mesh elements, such that the structural solver crashes within the first 250 time steps (time step size as stated above). Figure 6.19 depicts the displacement of the cylinder in x-direction for different time instances at the beginning of the simulation. Compared to reference values given in [9], the displacements of the cylinder are too large. They are plotted in Figure 6.20 with respect to the watchpoint. The initialization of the FSI simulation is the most critical aspect of this scenario. Starting from a fluid-only solution with the original free-stream velocity is not sufficient to obtain meaningful simulation results, since the forces initially exerted on the cylinder are too large. Therefore, an initialization procedure is necessary, which reduces these forces during startup of the simulation. A possible solution might be to start the simulation with a low free-stream velocity and continuously increase it to the original value during the first few time steps. This, however, is not easily possible as SU^2 does currently not support time-dependent inflow profiles. Alternatively, the forces can be decreased artificially by a "load ramping" procedure. Therefore, the force vector, which is transferred to preCICE, can be simply multiplied by a scalar factor (< 1). This factor is then continuously increased within the first time steps (up to 1) in order to recover the physically correct force vector. Such a method is already implemented in preCICE, yet it requires to build preCICE with Python support. However, it might also be useful to directly integrate this feature in the developed coupling adapter, because implementation is rather easy as the force vector can be accessed and manipulated within the adapter before transferring it to preCICE. A corresponding option could be added to the SU^2 configuration file by analogy with the approach of Section 5.1.

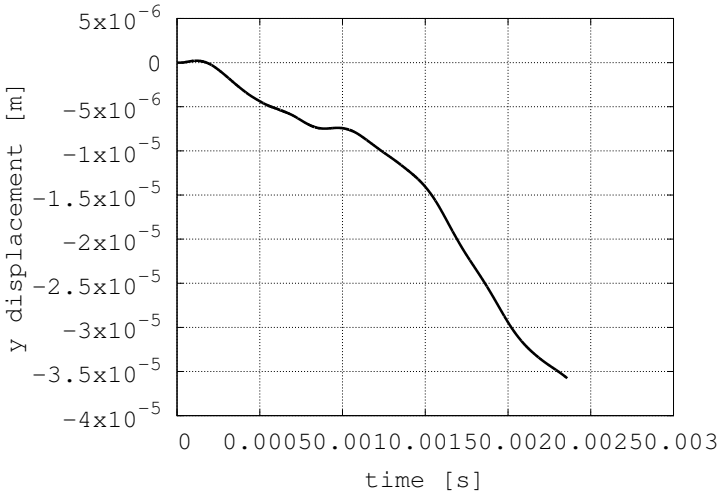


(a) $t = 10 \cdot 10^{-5}$ s. (b) $t = 80 \cdot 10^{-5}$ s. (c) $t = 150 \cdot 10^{-5}$ s. (d) $t = 230 \cdot 10^{-5}$ s.

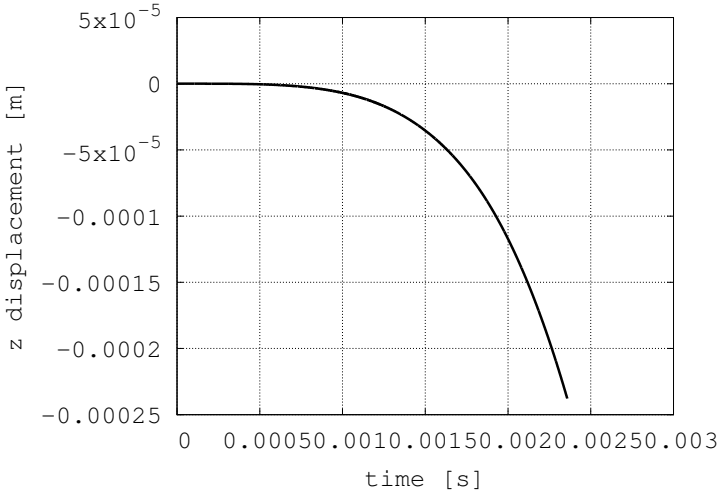
Figure 6.19: Deformation of the cylinder in xz-plane at different time instances. The orientation conforms with Figure 6.17a.



(a) Displacement in x-direction.



(b) Displacement in y-direction.



(c) Displacement in z-direction.

Figure 6.20: Displacements of the watchpoint with respect to time for all three spatial directions. The simulation ends untimely due to a crash of the solid solver.

7. Conclusion and Outlook

An adapter for linking the CFD solver SU^2 with the multiphysics coupling tool preCICE is developed in this work and validated both qualitatively and quantitatively with several testcases.

This work gives insights into the basics of FSI simulations and allows to classify the implemented partitioned coupling among the wide variety of solver strategies. The integration of the adapter into the code structure of SU^2 is explained, as well as its implementation itself. The coupling approach is validated via generic two- and three-dimensional testcases, and the well-known numerical benchmark scenario FSI3 ([39]). The adapter is capable of handling three-dimensional, real-world examples. This is exemplified by simulating a long, slender cylinder under turbulent flow conditions. The practical background of this simulation is research on brush seals for e.g. gas turbines. However, time did not suffice to obtain meaningful simulation results in the scope of this thesis. Thus, work on this topic will be continued. Once feasible single-cylinder simulation results are achieved, scenarios involving multiple cylinders in a channel should be simulated in order to include possible mutual interactions of these structures. Thereby, the real conditions in brush seals can be approximated more precisely. In its current version, the adapter is limited to a single wet surface in the SU^2 mesh file. While it is technically possible to define the surface of multiple cylinders as a single wet surface, it is more accurate to extend the adapter such that an arbitrary number of separate FSI interfaces can be defined. Moreover, it might be useful to integrate a load ramping functionality in the adapter, such that at the beginning of an FSI simulation the forces exerted on the structure are reduced by a factor, which is consequently increased to recover the real force values after a certain number of time steps. This might help to stabilize simulations as the initially occurring, large displacements can be reduced by this method. Recently, several RBF mapping methods have been included in preCICE, which can be used for parallel simulations. Applying these elaborate interpolation procedures instead of NN mapping allows to use finer solid meshes without encountering oscillatory forces at the wet surface, such that simulation results are expected to significantly improve (compared to simulations done with NN mapping). Furthermore, the spatial oscillations at the FSI interface are expected to vanish. Consequently, further testing with the FSI3 benchmark scenario will be done in order to verify this.

The adapter is implemented such that it perfectly integrates into SU^2, allowing to reuse characteristic functionalities like turbulence modeling, convergence acceleration via dual time stepping and multi-grid functionalities. Moreover, usage of preCICE for coupled FSI simulations can be configured via the native SU^2 configuration file. A single executable of SU^2 can be used for different FSI scenarios and even fluid-only computations without the need for recompilation. The adapter allows for inter- and intrafield parallel simulations. While no ALE implementation is currently available for the incompressible solver in SU^2, the adapter is already prepared for this feature, possibly extending the FSI capabilities to the incompressible regime in the future.

Although intrinsic FSI functionalities were recently added to SU^2, the coupling with preCICE offers more flexibility, as even commercial partner solvers can be chosen for multiphysics simulations. Moreover, the coupling algorithms and data mapping methods of preCICE are more elaborate. The possibility of running fully-parallel simulations outperforms the intrinsic FSI capabilities with regard to HPC.

Another practically very useful outcome of this work is the description of the procedure for integrating the adapter into SU^2 and building it with preCICE. Thus, users can easily extend SU^2 to FSI via preCICE guided by this thesis, which addresses both SU^2 and preCICE communities. The adapter will soon be included in the preCICE repository.

Appendices

A. Details on Integrating the Coupling Adapter into SU^2

Changes Concerning SU^2 Configuration

The necessary code changes in SU^2 for using the four new configuration options described in Section 5.1 need to be applied to the class *CConfig*. However, the class is defined by a header (*config_structure.hpp*), an inline (*config_structure.inl*) and a source file (*config_structure.cpp*). All of them have to be adapted slightly. In *config_structure.hpp* the member variables of *CConfig* are declared. In order to store the values of the above mentioned configuration options, first of all, four new private variables need to be introduced. Moreover, public getter functions are needed to access these variables. The necessary changes are shown in the following:

```
class CConfig {
private:
  ...
  bool precice_usage;
  bool precice_verbosityLevel_high;
  string preciceConfigFileName;
  string preciceWetSurfaceMarkerName;
  ...
public:
  ...
  bool GetpreCICE_Usage(void);
  bool GetpreCICE_VerbosityLevel_High(void);
  string GetpreCICE_ConfigFileName(void);
  string GetpreCICE_WetSurfaceMarkerName(void);
  ...
};
```

Listing A.1: Code changes in *config_structure.hpp*.

The four simple getter functions declared above are implemented in the file *config_structure.inl*:

```
...
inline bool CConfig::GetpreCICE_Usage(void){return precice_usage;}

inline bool CConfig::GetpreCICE_VerbosityLevel_High(void){return
precice_verbosityLevel_high;}

inline string CConfig::GetpreCICE_ConfigFileName(void){return
preciceConfigFileName;}

inline string CConfig::GetpreCICE_WetSurfaceMarkerName(void){return
preciceWetSurfaceMarkerName;}
...
```

Listing A.2: Code changes in *config_structure.inl*.

Eventually, the configuration file interpreter needs to be informed about the new options. In *config_structure.cpp* an assignment for each of the configuration file options to the member variables of *CConfig* is defined as well as the default values. This must be done in *CConfig::SetConfig_Options* for the four new options as follows:

```
...
void CConfig::SetConfig_Options(...) {
  ...
  addBoolOption("PRECICE_USAGE", precice_usage, false);

  addBoolOption("PRECICE_VERBOSITYLEVEL_HIGH",
  precice_verbosityLevel_high, false);

  addStringOption("PRECICE_CONFIG_FILENAME", preciceConfigFileName,
  string("precice.xml"));

  addStringOption("PRECICE_WETSURFACE_MARKER_NAME",
  preciceWetSurfaceMarkerName, string("wetSurface"));
  ...
}
```

Listing A.3: Code changes in *config_structure.cpp*.

The new mesh movement option PRECICE_MOVEMENT in SU^2 as introduced in Section 5.1 needs to be added to the configuration procedure. In contrast to the other options stated above, no changes to the *CConfig* class are needed since the option name GRID_MOVEMENT_KIND already exists. However, the new value PRECICE_MOVEMENT must be introduced. Therefore, an adaption in *option_structure.hpp* is necessary. Whenever multiple values[1] for an option are possible in the SU^2 configuration file, internally each value is assigned with an identifying number. After parsing the configuration file, such option values are mapped to their respective identifiers. These are further used in the source code of SU^2. The *option_structure.hpp* file is included in *config_structure.hpp*. This way, *CConfig* can use such value mappings, although they are not directly defined in its own source code. The following changes simply extend the mapping of already existing values for GRID_MOVEMENT_KIND by another entry:

```
...
enum ENUM_GRIDMOVEMENT {
NO_MOVEMENT = 0,
...
RIGID_MOTION = 2,
...
ROTATING_FRAME = 8,
...
PRECICE_MOVEMENT = 13
};

static const map<string, ENUM_GRIDMOVEMENT> GridMovement_Map =
CCreateMap<string, ENUM_GRIDMOVEMENT>
("NONE", NO_MOVEMENT)
...
("RIGID_MOTION", RIGID_MOTION)
...
("ROTATING_FRAME", ROTATING_FRAME)
...
("PRECICE_MOVEMENT", PRECICE_MOVEMENT);
```

Listing A.4: Code changes in *option_structure.hpp*. Only the entries corresponding to PRECICE_MOVEMENT need to be added. All other options are shown for illustrative reasons. 13 is assigned to the new option value because of consecutive numbering. The class *CCreateMap* defines how the mapping is done technically. It is not described here, as this detail is not necessary.

[1] Note that "value" can correspond to strings such as PRECICE_MOVEMENT, see Equation 4.10.

The new mesh movement sequence as mentioned in Section 5.1 is defined as follows: A switch-case-statement in *iteration_structure.cpp* determines, which movement procedure to use based on the configuration information. The list of possible cases is now extended by PRECICE_MOVEMENT as follows:

```
void SetGrid_Movement(CGeometry **geometry_container, ...,
CVolumetricMovement *grid_movement, CConfig *config_container, ...,
unsigned long ExtIter) {
  ...
  unsigned short Kind_Grid_Movement =
  config_container->GetKind_GridMovement(...);
  ...
  switch (Kind_Grid_Movement) {
  ...
  case ROTATING_FRAME:
    ...
  case RIGID_MOTION:
    ...
  case PRECICE_MOVEMENT:
    grid_movement->SetVolume_Deformation(geometry_container[MESH_0],
    config_container, true);
    geometry_container[MESH_0]->SetGridVelocity(config_container,
    ExtIter);
    grid_movement->UpdateMultiGrid(geometry_container, config_container);
    break;

  case NO_MOVEMENT:
    ...
}
```

Listing A.5: Code changes in *iteration_structure.cpp*. Only the entries corresponding to PRECICE_MOVEMENT need to be added. All other options are shown for illustrative reasons.

Adaption of the Main Solver Routine of SU^2

First of all, the header file of the adapter, *precice.hpp*, needs to be included so that the adapter can be instantiated within the code of the solver:

```
#include "../include/precice.hpp"
```

Listing A.6: Including the header file of the adapter, *precice.hpp* in *SU2_CFD.cpp*.

For positioning newly added lines of code in *SU2_CFD.cpp* I refer to Algorithm 2. The initialization of the boolean flag, which determines whether preCICE should be used and the conditional startup of the coupling via the adapter is achieved by the following lines, which have to be inserted right before the main solver while-loop of SU^2 (i.e. between lines 8 and 9 of Algorithm 2):

```
bool precice_usage = config_container[ZONE_0]->GetpreCICE_Usage();
Precice *precice;
double *max_precice_dt, *dt;
if (precice_usage) {
  precice = new Precice(rank, size, geometry_container,
  solver_container, config_container, grid_movement);
  dt = new double(config_container[ZONE_0]->GetDelta_UnstTimeND());
  precice->configure(config_container[ZONE_0]->GetpreCICE_ConfigFileName());
  max_precice_dt = new double(precice->initialize());
}
```

Listing A.7: Insertion of the declarations and memory allocation of coupling-related variables, as well as startup of the coupling procedure.

The flag *precice_usage* is set via the newly added getter function *GetpreCICE_Usage()* from the *config_container*, which stores all configuration information. Upon initialization of the adapter object *precice*, MPI rank and size as well as the containers holding geometry, solver, configuration and grid movement information are passed to it. For configuration of preCICE, name and location of its configuration file need to be forwarded to the adapter. This is done by calling the new getter function *GetpreCICE_ConfigFileName()*. The variable *dt* refers to the current physical time step size in SU^2 and is needed for timing issues, as is *max_precice_dt*.

Next, the condition of the main solver while-loop (starting at line 9 of Algorithm 2) must be modified such that preCICE is able to shut down SU^2 if a simulation should be ended. This is the case if *isCouplingOngoing()* evaluates to false:

```
while ((ExtIter < config_container[ZONE_0]->GetnExtIter() &&
precice_usage && precice->isCouplingOngoing()) || (ExtIter <
config_container[ZONE_0]->GetnExtIter() && !precice_usage)) {
  ...
}
```

Listing A.8: Modification of main solver while-loop condition, which allows to shut down SU^2 via preCICE.

SU^2 cannot differentiate, whether a solver iteration is a new time step or a coupling subiteration. Therefore, preCICE is solely in charge of this decision and signalizes it via a flag. The following lines must be added right after the beginning of the while-loop body (between lines 9 and 10, Algorithm 2):

```
if(precice_usage && precice->isActionRequired(precice->getCowic())){
  precice->saveOldState(&StopCalc, dt);
}
```

Listing A.9: Saving the current solver state for checkpointing of implicit solver strategies in preCICE.

preCICE signals the necessity of saving the solver state via the *isActionRequired()* function. Its argument specifies that the required action relates to writing an iteration checkpoint (*getCowic()*).

The following lines need to be inserted right after the code of Listing A.9 in order to allow preCICE to enforce time steps[2] smaller than the current value in SU^2:

```
if(precice_usage){
  dt = min(max_precice_dt, dt);
  config_container[ZONE_0]->SetDelta_UnstTimeND(*dt);
}
```

Listing A.10: Determining the minimal time step size for the next solver run.

preCICE is triggered, once a new fluid solution is computed:

```
if(precice_usage){
  *max_precice_dt = precice->advance(*dt);
}
```

Listing A.11: Advancing preCICE after a solver run of SU^2.

This code excerpt must be added between lines 14 and 15 of Algorithm 2, i.e. after convergence in SU^2 is checked, but before output files are written. *advance()* uses the current solver time step size as input and returns the new maximum limit for the next time instance. This is determined by comparing time step sizes of the coupled solvers up to the next instance when coupling data needs to be exchanged (compare Figure 4.2).

If a subiteration has not converged in preCICE, SU^2 is reset to the last iteration checkpoint. Thus, the following code is the counterpart of the *saveOldState()* function of Listing A.9 and must be inserted right after the *advance()* step:

[2] *dt* refers to the black and red arrows of Figure 4.2, while the green arrow corresponds to *max_precice_dt*.

```
if(precice_usage && precice->isActionRequired(precice->getCoric())){
  ExtIter--;
  precice->reloadOldState(&StopCalc, dt);
}
else if (solutionNeedsToBeOutput){
  writeOutputFiles();
}
```

Listing A.12: Reloading the old SU^2 solver state if another subiteration is necessary. Note that *solutionNeedsToBeOutput* and *writeOutputFiles()* are simplified pseudo-code expressions.

By analogy with *saveOldState()*, *reloadOldState()* is initiated by preCICE as it signalizes if an action is required. In contrast, this time the task does not relate to writing but rather reading an iteration checkpoint (*getCoric()*).

The following code is executed at the end of a coupled simulation and, thus, must be added between lines 22 and 23 of Algorithm 2:

```
if(precice_usage){
  precice->finalize();
  delete [] precice;
}
```

Listing A.13: Shutting down the coupling between SU^2 and preCICE and deallocating memory.

Algorithm 4 sums up the SU^2 solver run extended by the coupling-related, minimally invasive code changes to SU2_CFD in pseudo-code, which are necessary to integrate the coupling adapter *Precice* into SU^2.

Algorithm 4 Conceptual SU^2 solver run extended for coupling with preCICE in pseudo-code. The shown code is reduced to main functionalities.

```
1: stopCalculation = false;
2: externalIteration = 0;
3: parseConfigurationFileAndStoreRespectiveInformation();
4: geometricalPreprocessing();
5: driverPreprocessing();
6: if dynamicMeshSimulation then
7:    setDynamicMeshStructure();
8: end if
9: preciceUsage = getPreciceUsage();
10: if preciceUsage then
11:    initializeAdapter();
12:    configurepreCICE();
13:    initializepreCICE();
14:    initializeTimeSteppingVariables();
15: end if
16: while ((externalIteration < maxNumberOfExternalIteration) AND preciceUsage AND CouplingIsOngoing) OR ((externalIteration < maxNumberOfExternalIteration) AND NOT preciceUsage) do
17:    if preciceUsage AND iterationCheckpointNeedsToBeWritten then
18:       saveOldState();
19:    end if
20:    if preciceUsage then
21:       determineAndSetNewTimeStepSize();
22:    end if
23:    driverRun();
24:    updateConvergenceHistory();
25:    if convergence then
26:       stopCalculation = true;
27:    end if
28:    if preciceUsage then
29:       advancepreCICE();
30:    end if
31:    if preciceUsage AND iterationCheckpointNeedsToBeRead then
32:       externalIteration−−;
33:       reloadOldState();
34:    else if solutionNeedsToBeOutput then
35:       writeOutputFiles();
36:    end if
37:    if stopCalculation then
38:       break;
39:    end if
40:    externalIteration++;
41: end while
42: if preciceUsage then
43:    finalizepreCICE();
44:    deallocateMemoryForAdapter();
45: end if
46: return exitSuccess;
```

B. SU^2 Installation Description

Building without preCICE

It is assumed that the reader of this description has some basic experience in working with git. If not, I recommend an introduction as e.g. can be found here:

`https://www.atlassian.com/pt/git/tutorial/git-basics`

The open-source Computational Fluid Dynamics (CFD) Code Stanford University Unstructured (SU^2) can be accessed on Github:

`https://github.com/su2code`

The code can either be directly downloaded from there or cloned via:

```
$ git clone https://github.com/su2code/SU2.git
```

The installation procedure uses the very common *autoconf*, *automake* tools. After downloading and extracting or cloning the code, navigate to the directory, which contains the SU^2 source code:

```
$ cd /path/to/SU2SourceCode
```

In this manual, I assume the code is to be built with support for the parallel features of SU^2. Therefore, the user needs to have a version of Message Passing Interface (MPI) installed, e.g. OpenMPI, MPICH or Intel MPI.

To find out where the compiler wrappers MPICC(C) and MPICXX(C++) are located, simply run:

```
$ which mpicc
$ which mpicxx
```

Also by

```
$ mpicc --version
$ mpicxx --version
```

the compilers specified in the wrappers can be checked. To start the configuration of SU^2 run:

```
$ ./configure --prefix=/installation/location/for/SU2/
--enable-mpi --with-cc=/location/of/mpicc
--with-cxx=/location/of/mpicxx
```

With prefix, the installation location for SU^2 can be specified. If none is defined, SU^2 is installed in

```
/usr/local/bin
```

by default. The second argument enables MPI communication (needed for parallel support) and the latter two arguments specify which MPI wrapper compilers should be used (in case you want to use a different wrapper than the standard one, just specify it here with its full path). The configuration now starts and the system is checked for all necessary resources and requirements. If you encounter any errors here, just have a look at the console output, which probably indicates the problem. It is possible that your version of autoconf/automake is not the same as the one of the developers, who generated the makefiles. In such a case it is necessary to run

```
$ autoreconf
```

once, so that the makefiles are adapted to your version of the tools. If this is not sufficient, it may be necessary to adjust some environment variables in order for SU^2 to find the necessary resources on your system. Therefore, you may want to have a look at the files *README.md* and *INSTALL*, which are located in the current directory. Just run

```
$ vi README.md
```

or

```
$ vi INSTALL
```

After the configuration process has finished successfully, the console output reads some commands, which need to be added to your *.bashrc* file. These define some necessary environment variables (SU2_RUN, SU2_HOME) and expand existing ones (PATH, PYTHONPATH). The lines might look like this:

```
export SU2_RUN="/home/yourUserName/bin"
export SU2_HOME="/home/yourUserName/SU2_source"
export PATH=$PATH:$SU2_RUN
export PYTHONPATH=$PYTHONPATH:$SU2_RUN
```

When opening a new shell, those commands set the path for the executeables and the source code of SU^2. Moreover, the path for some python scripts is defined, which are e.g. used for parallel computations in SU^2. In a user-friendly fashion the shown command lines are already adapted for your system so that you only need to copy-paste them. The *.basrc* is located in your home directory and can be edited via

```
$ vi ~/.bashrc
```

The next step is to run the make command:

```
$ make -j 8
```

followed by

```
$ make install
```

To save some time, the build process can be accelerated using the -j flag with a number of processes, as SU^2 can be built in parallel. In this example, the build process would run on 8 processes. However, it is reasonable to only use a number corresponding to the amount of physical cores available, since the build process almost fully uses the computational resources of each core and thus, multithreading may even slow down the process.
SU2 can also support the mesh and output format CFD General Notation System (CGNS - `http://cgns.github.io`). This is not standard though and therefore, not included in this short manual. For further information I refer to the official wiki of SU^2 on Github:

`https://github.com/su2code/SU2/wiki`

Building with preCICE

In this part, I describe how to build SU^2 linked with preCICE. Therefore, the adapter files *precice.hpp* and *precice.cpp* developed in this thesis and explained in Chapter 5 are included in the SU^2 source code. It is assumed that preCICE has been installed successfully beforehand. Concerning installation instructions for preCICE, I refer to:

`https://github.com/precice/precice/wiki/Building`

It is assumed that code changes as stated in Appendix A have already been applied to SU^2.
First of all, copy the adapter files to the following locations:

```
/path/to/SU2SourceCode/SU2_CFD/include/precice.hpp
```

and

```
/path/to/SU2SourceCode/SU2_CFD/src/precice.cpp
```

Now, navigate to the source code of the SU2_CFD module. It is located here:

```
/path/to/SU2SourceCode/SU2_CFD/src/SU2_CFD.cpp
```

The header file of the adapter, (*precice.hpp*), needs to be included in order for the coupling to be established. Therefore, simply add the following line to the inclusions already given in *SU2_CFD.cpp*:

```
#include "../include/precice.hpp"
```

Now we can start linking SU^2 and preCICE via the static library *libprecice.a*, which is generated upon successfully building preCICE. To do so, some environment variables have to be adapted so that SU^2 knows what library to look after and in which directory:

```
$ export LDFLAGS=-L/path/to/preCICESourceCode/build/last
$ export LIBS=-lprecice
```

Here it is assumed that the last successful build of preCICE is used for linking. Of course, this can be customized in order to link specific builds.
Before using autoconf and automake, the makefiles of SU^2 have to be edited such that the adapter files (*precice.hpp* and *precice.cpp*) are compiled with SU^2. Navigate to the following directory:

```
$ cd /path/to/SU2SourceCode/SU2_CFD/obj/
```

Now edit the file *Makefile.am*, in which all the files to be compiled with SU2_CFD are given.

```
$ vi Makefile.am
```

Search for the following line in that file:

```
su2_cfd_sources = \
```

If you copied *precice.hpp* and *precice.cpp* to the suggested locations, add the following lines below that expression:

```
../include/precice.hpp \
../src/precice.cpp \
```

If you copied them somewhere else, you need to adapt these paths. Make sure to use spacings and backslashes analogously to the other lines there. Now the changes in *Makefile.am* have to be transferred to *Makefile.in*, which finally leads to a new *Makefile* in the configuration process. This *Makefile* is then used to build SU^2 coupled with preCICE, again using the make command. Navigate back to the main directory of SU^2 by simply typing:

```
cd ../../
```

Now run:

```
$ automake
```

to update *Makefile.in*. If this fails, it might be necessary to run

```
$ autoreconf
```

as described in the first part of this manual.
Once more, follow the configuration procedure:

```
$ ./configure --prefix=/installation/location/for/SU2/
--enable-mpi --with-cc=/location/of/mpicc
--with-cxx=/location/of/mpicxx
```

Depending on whether you changed the position/names of your SU^2 source code or not you may or may not have to add the command lines adapting the SU^2 environment variables to your *.bashrc*, which are given as console output at the end of a successful configuration process. Analogous to the build without preCICE, now run make (with optional parallelization):

```
$ make -j 8
```

and

```
$ make install
```

The new executeable of SU2_CFD, now coupled with preCICE, can be found in

```
/path/to/SU2SourceCode/SU2_CFD/bin
```

if the installation finished successfully.

Bibliography

[1] Armin Beckert and Holger Wendland. Multivariate interpolation for fluid-structure-interaction problems using radial basis functions. *Aerospace Science and Technology*, 5(2):125–134, 2001.

[2] Tomáš Bodnár, Giovanni P Galdi, and Šárka Nečasová. *Fluid-Structure Interaction and Biomedical Applications*. Springer, 2014.

[3] Leonid M Brekhovskikh and Valery Goncharov. *Mechanics of continua and wave dynamics*, volume 1. Springer Science & Business Media, 2012.

[4] Hans-Joachim Bungartz, Florian Lindner, Bernhard Gatzhammer, Miriam Mehl, Klaudius Scheufele, Alexander Shukaev, and Benjamin Uekermann. preCICE – A Fully Parallel Library for Multi-Physics Surface Coupling. *Computers & Fluids*, 2016.

[5] Paola Causin, Jean-Frédéric Gerbeau, and Fabio Nobile. Added-mass effect in the design of partitioned algorithms for fluid–structure problems. *Computer methods in applied mechanics and engineering*, 194(42):4506–4527, 2005.

[6] Aukje de Boer, Alexander H van Zuijlen, and Hester Bijl. Review of coupling methods for non-matching meshes. *Computer methods in applied mechanics and engineering*, 196(8):1515–1525, 2007.

[7] Aukje de Boer, Alexander H van Zuijlen, and Hester Bijl. Comparison of conservative and consistent approaches for the coupling of non-matching meshes. *Computer Methods in Applied Mechanics and Engineering*, 197(49):4284–4297, 2008.

[8] Joris Degroote, Klaus-Jürgen Bathe, and Jan Vierendeels. Performance of a new partitioned procedure versus a monolithic procedure in fluid–structure interaction. *Computers & Structures*, 87(11):793–801, 2009.

[9] Yannick Diekmann. Möglichkeiten der Kopplung von Fluid- und Struktursimulationen. Semester thesis, Technische Universität München, 2015.

[10] Jean Donea, Antonio Huerta, Jean-Philippe Ponthot, and Antonio Rodriguez-Ferran. Arbitrary Lagrangian-Eulerian Methods. In Erwin Stein, René De Borst, and Thomas JR Hughes, editors, *Encyclopedia of Computational Mechanics*, volume 1: Fundamentals, chapter 14. Wiley & Sons, 2004.

[11] Thomas D Economon, Francisco Palacios, Juan J Alonso, Gaurav Bansal, Dheevatsa Mudigere, Anand Deshpande, Alexander Heinecke, and Mikhail Smelyanskiy. Towards High-Performance Optimizations of the Unstructured Open-Source SU2 Suite. *AIAA Paper*, 1949, 2015.

[12] Charbel Farhat and Michael Lesoinne. Two efficient staggered algorithms for the serial and parallel solution of three-dimensional nonlinear transient aeroelastic problems. *Computer methods in applied mechanics and engineering*, 182(3):499–515, 2000.

[13] Charbel Farhat, Kristoffer G van der Zee, and Philippe Geuzaine. Provably second-order time-accurate loosely-coupled solution algorithms for transient nonlinear computational aeroelasticity. *Computer methods in applied mechanics and engineering*, 195(17):1973–2001, 2006.

[14] Joel H Ferziger and Milovan Peric. *Numerische Strömungsmechanik*. Springer-Verlag, 2008.

[15] Christiane Förster, Wolfgang A Wall, and Ekkehard Ramm. The artificial added mass effect in sequential staggered fluid-structure interaction algorithms. In *ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics, Egmond aan Zee, The Netherlands, September 5-8, 2006*. Delft University of Technology; European Community on Computational Methods in Applied Sciences (ECCOMAS), 2006.

[16] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. University of Tennessee, 3.1 edition, 2015.

[17] Giovanni P Galdi. *An introduction to the mathematical theory of the Navier-Stokes equations: Steady-state problems*. Springer Science & Business Media, 2011.

[18] Bernhard Gatzhammer. *Efficient and flexible partitioned simulation of fluid-structure interactions*. PhD thesis, Technische Universität München, 2015.

[19] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[20] Gene Hou, Jin Wang, and Anita Layton. Numerical methods for fluid-structure interaction—a review. *Commun. Comput. Phys*, 12(2):337–377, 2012.

[21] Bruce M Irons and Robert C Tuck. A version of the Aitken accelerator for computer iteration. *International Journal for Numerical Methods in Engineering*, 1(3):275–277, 1969.

[22] George Karypis. METIS and ParMETIS. In *Encyclopedia of Parallel Computing*, pages 1117–1124. Springer, 2011.

[23] George Karypis and Vipin Kumar. *MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System*. University of Minnesota, 4.0 edition, 2009.

[24] George S Kell. Effects of isotopic composition, temperature, pressure, and dissolved gases on the density of liquid water. *Journal of Physical and Chemical Reference Data*, 6(4):1109–1131, 1977.

[25] Ulrich Küttler and Wolfgang A Wall. Fixed-point fluid–structure interaction solvers with dynamic relaxation. *Computational Mechanics*, 43(1):61–72, 2008.

[26] W Michael Lai, David H Rubin, and Erhard Krempl. *Introduction to Continuum Mechanics*. Butterworth-Heinemann, 2009.

[27] Miriam Mehl, Benjamin Uekermann, Hester Bijl, David Blom, Bernhard Gatzhammer, and Alexander van Zuijlen. Parallel coupling numerics for partitioned fluid–structure interaction simulations. *Computers & Mathematics with Applications*, 71(4):869–891, 2016.

[28] Florian R Menter. Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA journal*, 32(8):1598–1605, 1994.

[29] US National Institute of Standards and Technology. The NIST Reference on Constants, Units, and Uncertainty - Avogadro constant. http://physics.nist.gov/cgi-bin/cuu/Value?na, 2016. Online, accessed 15.03.2016.

[30] Francisco Palacios, Michael R Colonno, Aniket C Aranake, Alejandro Campos, Sean R Copeland, Thomas D Economon, Amrita K Lonkar, Trent W Lukaczyk, Thomas WR Taylor, and Juan J Alonso. Stanford University Unstructured (SU2): An open-source integrated computational environment for multi-physics simulation and design. *AIAA Paper*, 287:2013, 2013.

[31] Charles S Peskin. The immersed boundary method. *Acta numerica*, 11:479–517, 2002.

[32] Ekkehard Ramm and Wolfgang A Wall. Fluid-structure interaction based upon a stabilized (ALE) finite element method. In *4th World Congress on Computational Mechanics: New Trends and Applications, CIMNE, Barcelona*, pages 1–20, 1998.

[33] Thomas CS Rendall and Christian B Allen. Unified fluid–structure interpolation and mesh motion using radial basis functions. *International Journal for Numerical Methods in Engineering*, 74(10):1519–1559, 2008.

[34] Philip L Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of computational physics*, 43(2):357–372, 1981.

[35] Youcef Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing*, 14(2):461–469, 1993.

[36] Ruben Sanchez, Rafael Palacios, Thomas D Economon, Heather L Kline, Juan J Alonso, and Francisco Palacios. Towards a Fluid-Structure Interaction solver for problems with large deformations within the open-source SU2 suite. In *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2016.

[37] Alexander Shukaev. A Fully Parallel Process-to-Process Intercommunication Technique for preCICE. Master's thesis, Technische Universität München, June 2015.

[38] Galina Sieber. *Numerical simulation of fluid-structure interaction using loose coupling methods*. PhD thesis, Technische Universität Darmstadt, 2002.

[39] Stefan Turek and Jaroslav Hron. Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow. In Hans-Joachim Bungartz and Michael Schäfer, editors, *Fluid-Structure Interaction: Modelling, Simulation, Optimisation*, volume 53 of *Lecture Notes in Computational Science and Engineering*, pages 371–385. Springer Science & Business Media, 2006.

[40] Benjamin Uekermann, Hans-Joachim Bungartz, Bernhard Gatzhammer, and Miriam Mehl. A parallel, black-box coupling algorithm for fluid-structure interaction. In *Proceedings of 5th International Conference on Computational Methods for Coupled Problems in Science and Engineering*, pages 1–12, 2013.

[41] Benjamin Uekermann, Juan Carlos Cajas, Bernhard Gatzhammer, Guillaume Houzeaux, Miriam Mehl, and Mariano Vázquez. Towards partitioned fluid-structure interaction on massively parallel systems. *Proceedings of WCCM XI/ECCM V/ECFD VI, Barcelona*, 2014.

[42] E Harald van Brummelen. Added mass effects of compressible and incompressible flows in fluid-structure interaction. *Journal of Applied mechanics*, 76(2):021206, 2009.

[43] Raoul van Loon, Patrick D Anderson, Frans N van de Vosse, and Spencer J Sherwin. Comparison of various fluid–structure interaction methods for deformable bodies. *Computers & structures*, 85(11):833–843, 2007.

[44] Mariano Vázquez, Guillaume Houzeaux, Seid Koric, Antoni Artigues, Jazmin Aguado-Sierra, Ruth Arís, Daniel Mira, Hadrien Calmet, Fernando Cucchietti, Herbert Owen, et al. Alya: Multiphysics engineering simulation toward exascale. *Journal of Computational Science*, 2016.

[45] Peter Wriggers. *Nonlinear finite element methods*. Springer Science & Business Media, 2008.