Technische Universität München
Uppsala Universität

Structural multi-model coupling with CalculiX and preCICE

by

Alexandre Trujillo Boqué

Master Thesis for the completion of the
degree of MSc in Computational Science

carried out in
Chair of Scientific Computing
Faculty of Informatics
Technische Universität München

for presentation in
Department of Information Technology
Faculty of Science and Technology
Uppsala Universität

Summer 2018

Technische Universität München
Uppsala Universität

Abstract

Efficient multi-physics simulations are vital to study interacting systems in science and engineering. In the field of structural mechanics, multi-physics simulations target interactions between structures with different characteristics. Here, we take a partitioned approach to the simulation of structure-structure interaction with the FEM program CalculiX and the coupling library preCICE. We aim to speed up the simulation of nonlinear phenomena via the coupling of linear and nonlinear simulations, restricting nonlinear treatment only to where required. After validation of the preCICE adapter for CalculiX with a linear test case, we study the convergence of implicit coupling and the effect of quasi-Newton methods. The latter prove successful in the acceleration of the coupling. Afterwards, we study the performance in a nonlinear case. Nevertheless, nonlinear effects are not strong enough to bring in a performance improvement; thus, we point to subsequent experimentation.

Contents

Chapter 1

Introduction

Complex phenomena often involve interactions between multiple systems, of which each has its own physical characteristics. Individual systems can be simulated with single-physics solvers, i.e. programs that simulate one physical system with a unique set of governing equations. Some examples of interaction between systems are fluid-structure interaction (FSI) in blood vessels, or between the air inside a jet engine and the structures that make up the engine itself. Systems with these kind of interactions cannot be simulated by a single-physics solver. Therefore, it is necessary to produce multi-physics software that is capable of simulating simultaneously several of the subdomains of the full model.

The exponential growth of computational power during the last decades, specially regarding the advances in massive parallelization, has made multi-physics simulations possible for large problem setups in science and engineering, with enough precision to provide realistic and meaningful results.

We can distinguish two trends in the development of multi-physics software: monolithic approaches try to incorporate the full multi-physics simulation into a single solver program, and partitioned approaches implement a coupling between single-physics solvers. In this project, we follow a partitioned approach, which allows us to reuse existing software. Development tasks are accelerated when software is reused, since it eliminates the necessity of producing specific monolithic solvers for each type of interaction problem.

Let us consider the multi-physics concept of modelling interactions between subdomains, but restricted to working with the same governing equations in all of them. In this thesis, we study the interaction between subdomains of a structural system, thus the governing equations belong to the field of structural mechanics. Structure-structure interaction (SSI) is the name for the phenomenon we aim to model. Our hypothesis is that this kind

of interaction modelling offers an opportunity to improve performance, if the numerical analysis is adapted to precision requirements and properties of each subdomain.

We use the term multi-model for a multi-physics setup where the subdomains have the same governing equations, considering that we derive a model that is numerically different for each subdomain. Specifically, in our setup we aim to couple linear with nonlinear structure models.

In this project, we implement a framework for partitioned structure-structure interaction modelling with the finite element program CalculiX [1] as structural mechanics solver, and the coupling library preCICE [2, 3]. To this end, a non-intrusive, black-box philosophy is employed, meaning that CalculiX is adapted[4] with minimal changes to the code and all the coupling operations are performed externally by preCICE, with the software setup shown in Figure 1.1.

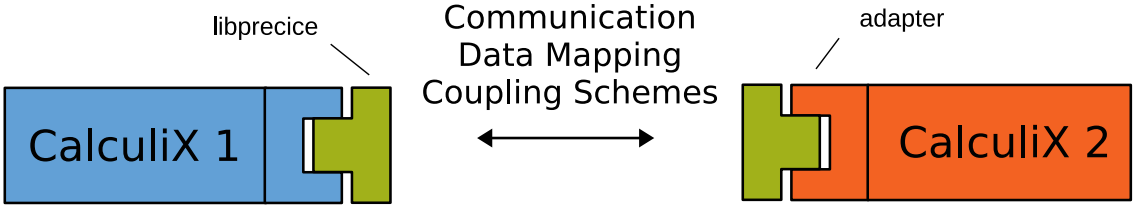

FIGURE 1.1: Software architecture of a two-solver preCICE setup. Adapted CalculiX solvers use the preCICE library for coupling tasks without depending on a central instance.

This report is structured as follows: in the remaining of this chapter, we detail the motivation of the thesis and provide the background with a literature review. Next, in Chapter 2, we introduce the concepts and formulation of the theoretical framework. After that, in Chapter 3, we take a look at the existing software that plays a role in this project: the coupling library preCICE and the finite element program CalculiX. In Chapter 4, we describe the implementation of the preCICE adapter for CalculiX and how SSI functionality has been incorporated into it. Following that, in Chapter 5, we present numerical studies and give an interpretation of the results obtained. Finally, we discuss the conclusions of the project in Chapter 6.

1.1 Motivation

A partitioned structure-structure interaction model implies a decomposition of the spatial domain with integration steps performed independently in each subdomain. Our strategy towards a more efficient simulation is based on this decomposition.

We aim to execute multiple instances of single-physics structural solvers in parallel, restricting nonlinear treatment to subdomains known to present nonlinear behaviour. Examples of such subdomains are impact or crack regions. Each instance of the solver is referred to as a participant of the simulation. Participants should only use the amount of computational resources needed to achieve the desired precision in the subdomain they have been assigned to. For this purpose, in this project, we restrict to switching between linear and nonlinear treatment of the problem by the single-physics solver.

Thus, a local numerical adaptation of the model to the requirements of the system is expected to optimize resources and improve performance, but only as long as the overhead from coupling the solvers does not impede it.

Besides model customization, we should also be able to achieve speedup by means of scaling the parallelization of the setup. This is almost a requirement, as commonplace problem sizes get bigger. Currently, CalculiX only supports shared-memory parallelization, a situation that limits the number of threads we can run the simulation on; with a partitioned approach to modelling we know that we can get past this limitation.

A further reason to use a partitioned coupling scheme with a black-box philosophy is that it contributes to a fast-paced development; since we make use of already existing single-physics solvers, which have been tested in many applications and constitute the standard of academia and industry. This methodology does not require the additional development of stand-alone specific tools for each kind of interaction problem.

Finally, despite being outside the scope of this project, it is worthwhile to mention that SSI can also be applied to models of independent structures whose interactions need to be modelled, a common situation in engineering. Since we implement and test a general structure-structure coupling tool, it can also be applied to this type of SSI problems, not only to individual structures that are internally partitioned into subdomains.

1.2 Background

Research efforts in multi-physics have been growing in the last decades as many issues come along with the scaling of hardware architecture [5]. The multi-physics community has faced the challenge of designing algorithms that fulfill convergence and stability requirements introduced by coupling schemes, without hampering performance. Many researchers have focused on the numerical methods inherent in multi-physics simulations, while others have studied their software architecture aspects.

The diversity of applications and strategies possible in multi-physics makes the amount of existing software solutions impossible to review here. For an exhaustive survey of multi-physics frameworks developed until 2015, see [6].

Instead, we focus the literature review on structure-structure interaction. We distinguish monolithic and partitioned solution strategies. It is convenient to review the research in partitioned approaches for FSI, as many of the concepts developed therein have been incorporated into partitioned approaches for SSI.

The monolithic approach to the coupled structural mechanics problem is usually addressed with the application of dual Schur decomposition in Newmark based integrators [7], an algebraic perspective where the equations of all subdomains are integrated in composite matrix systems, by means of connectivity matrices and Lagrange multipliers. Explicit and implicit integrators can be coupled this way, imposing continuity of velocities between subdomains, at the interface or coupling surface, as in [8–10]. Coupling schemes have incorporated multi-time stepping, multi-scale, and other techniques to adapt the discretization to local properties of subdomains. Geometric and time step incompatibilities are handled with strategies such as the Mortar method or the Arlequin framework [11–14]. Another method used for domain decomposition in structural mechanics is the finite element tearing and interconnection (FETI) [15]. Model order reduction methods [16] reduce the amount of degrees of freedom in the system, by defining a superelement in the mesh, where inner nodes are removed and only boundary nodes remain. The reduced model is solved separately and incorporated as part of the original one, in what can be seen as a coupling between both models. Monolithic methods have been shown to converge in presence of strong nonlinear behaviour, in all or part of the structure [17].

Partitioned coupling [18] in an FSI simulation has been studied extensively, whereas for SSI it has not been exhaustively explored. A partitioned method for FSI is implemented in [19]. The use of implicit coupling, with a fixed point iteration, as the coupling method has been introduced along with imposition of boundary conditions. To this end, partitioned Newton methods have been used to evaluate the exact Jacobian of the coupling surface [20], but this is an expensive method. Besides, it is not suitable for a black-box situation, where we do not know the discretization of the coupling surface performed by the solver. Therefore, quasi-Newton schemes are often necessary to approximate the Jacobian [21–23]; these methods accelerate the fixed-point iterations in the implicit coupling procedure. Further advancements have extended quasi-Newton schemes pursuing stabilization and convergence acceleration, by using information of previous iterations [24–26]. Since quasi-Newton schemes are relevant for our project, we return to them in Section 2.3.2. Multigrid techniques [27] and reduced order models [28] have also been incorporated into the partitioned approach. Partitioned techniques, like monolithic ones, often involve

a dual Schur decomposition framework, but with a non-intrusive methodology [22, 29], that leads to a less explicit formulation. As a side note, FSI simulations with partitioned coupling present a source of stability and convergence problems, due to time splitting, known as the added-mass effect [30, 31]. To the best of our knowledge, this issue has not been observed nor described for SSI simulations.

Chapter 2

Theory

In the previous chapter, we have introduced the concept of multi-physics, and more specifically, structure-structure interaction. We have given an overview of recent developments that have contributed to the feasibility of large-scale SSI simulations.

In this chapter, we present the mathematical framework of SSI. We start with a simple computational method for the simulation of mechanics in solid structures, applying a finite element analysis. The formulation of the numerical method is then extended via domain decomposition to an analysis of a union of subdomains. This is done within monolithic and partitioned frames. The latter is the approach we have implemented and tested in this project.

2.1 Basic Notions of Computational Structural Mechanics

In the following lines we present the equations of a computational structural mechanics problem without damping[8, 10, 32], derived from a finite element method (FEM). Our model defines and computes dynamic variables in a mesh. As can be seen in Figure 2.1, a mesh is determined by the position of nodes and the configuration of elements, following the FEM formulation.

Let us consider an elastic, deformable solid that occupies a domain Ω. Newton's second law of motion states that the acceleration of each particle in the solid depends on its mass and the forces that act upon it. This law can be expressed for all particles as a system of differential equations. Since these particles belong to a solid structure, there are strong internal forces acting between them. Additionally, there might be external forces coming from mechanical loads or an external force field.

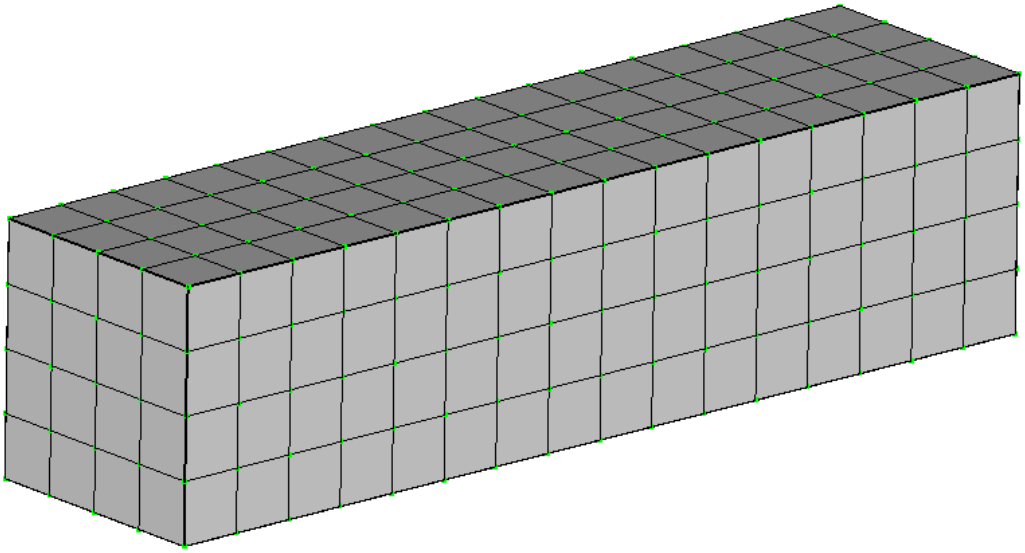

FIGURE 2.1: Example of a typical 3D FEM model of a solid. Green dots are the nodes of the mesh, and each cube is an element. Element type here is 8-node brick element.

As a first step of the FEM, we write the differential equations of motion in weak form. Afterwards, we discretize them in the spatial domain, expressing the continuous physical functions as a linear combination of test functions. This formulation defines a mass matrix for the structure, that depends on the mass and connection geometry of its particles.

Thus, the FEM leads us to the spatially discretized equations of motion for the solid, in matrix form,

$$\mathbf{M}\ddot{\mathbf{U}}(t) + \mathbf{F}_{int}(\mathbf{U}(t)) = \mathbf{F}_{ext}(t) \tag{2.1}$$

for times $t \in [0, T]$, where $\mathbf{U}(t)$ is the vector of nodal displacements for all nodes of a mesh defined in Ω, at time t; $\mathbf{F}_{int}$ and $\mathbf{F}_{ext}$ are the vectors of internal and external forces acting on the structure, respectively; and $\mathbf{M}$ is the mass matrix. Assuming linear elasticity, we have

$$\mathbf{F}_{int}(\mathbf{U}(t)) = \mathbf{K}\mathbf{U}(t), \tag{2.2}$$

with a stiffness matrix $\mathbf{K}$, containing the elastic properties of the structure.

For time discretization of (2.1), we use the Newmark [7] family of numeric time schemes[1]. Firstly, we divide the time domain in N time steps, with $\Delta t = T/N$. We use the conventional notation $\mathbf{U}(t_n) \equiv \mathbf{U}_n$ to indicate values at time step n.

Displacements and velocities $\dot{\mathbf{U}}(t) = \frac{d\mathbf{U}(t)}{dt}$ are computed for each time step with

[1] CalculiX does not run with the original Newmark time integrator, but the principles of discretization in the method implemented there are the same as here.

$$\mathbf{U}_{n+1} = \mathbf{U}_n + \Delta t \dot{\mathbf{U}}_n + \Delta t^2 (\frac{1}{2} - \beta) \ddot{\mathbf{U}}_n + \Delta t^2 \beta \ddot{\mathbf{U}}_{n+1} \tag{2.3}$$

$$\dot{\mathbf{U}}_{n+1} = \dot{\mathbf{U}}_n + \Delta t (1 - \gamma) \ddot{\mathbf{U}}_n + \Delta t \gamma \ddot{\mathbf{U}}_{n+1} \tag{2.4}$$

where parameters $\beta \in [0, \frac{1}{2}]$ and $\gamma \in [0, 1]$ characterize the numeric method.

CalculiX uses a revision [33] of the HHT α-method [34] for direct integration dynamic analyses [35]. The latter, in turn, builds on the Newmark method, incorporating an additional α parameter to the motion equations, in order to introduce numerical dissipation for an improved stability:

$$\mathbf{M}\ddot{\mathbf{U}}_{n+1} + (1 + \alpha)\mathbf{K}\mathbf{U}_{n+1} - \alpha \mathbf{K}\mathbf{U}_n = \mathbf{F}_{ext,\ n+1}, \quad n = 0, 1 \ldots N - 1. \tag{2.5}$$

When $\alpha = 0$ we recover a classic Newmark method.

It is also important that we mention the type of element used to define the mesh for the finite element method in our test cases. In the first one (see Section 5.1.1), we use volume elements, specifically twenty-node brick elements with reduced integration. In the second one (see Section 5.1.2), we use three-node shell elements. These elements have eight and three integration points, respectively, where displacements and forces are computed. In the case of twenty-node elements, values at integration points must be extrapolated to the nodes. Location of nodes and integration points in the elements is shown in Figure 2.2.

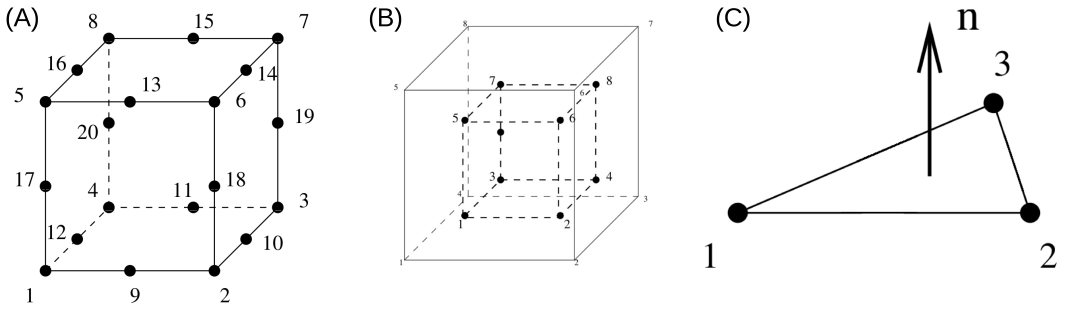

FIGURE 2.2: Location of: (A) nodes in a twenty-node brick element. (B) integration points in a twenty-node brick element with reduced integration. (C) nodes and integration points in a three-node shell element. Images taken from the CalculiX Manual [35].

To prevent confusion in the following chapters, please note that the terms implicit and explicit, when used with respect to the contents of this section, refer to time integration. An explicit method computes values of the next time step using information of the current one. A priori, this requires relatively small time step lengths for stability. Contrarily, an implicit method computes values of the next step as a function of variables both of the current time step and the next one. This usually requires a fixed-point iteration or more complex procedures, but implicit methods are more stable and can thus be implemented

with larger time step lengths. In the description of our implementation, however, we use the terms explicit and implicit mainly to refer to a classification of partitioned coupling schemes, as explained in Section 2.3.

2.2 Domain Decomposition within the Monolithic Frame

We presented the time integrator of a Newmark method for structural mechanics. Now, we define a partition of the full domain Ω into subdomains Ω_k, connected by coupling surfaces[2] $\Gamma_{kl} \equiv \Omega_k \cap \Omega_l$, as in Figure 2.3. Note that all pairs of adjacent subdomains overlap at the respective interface; therefore, as long as the meshes match, nodes in that interface are inside both subdomains. From now on, we indicate data structures corresponding to each spatial subdomain by using the subindices k.

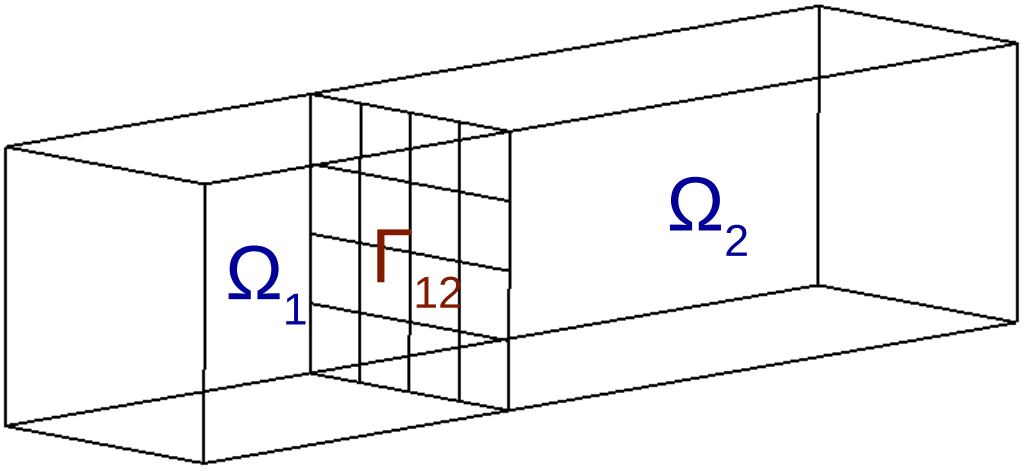

FIGURE 2.3: Partition of a 3D domain Ω into subdomains Ω_1 and Ω_2, by means of the definition of a surface as the interface Γ_{12}. In the case we have matching meshes, nodes at the interface are shared between subdomains.

This spatial decomposition implies a partition of the Newmark time integrator, often referred to as time splitting. A partition of the global system of motion equations, by means of a dual Schur decomposition, results in an additional term, the linking forces, and a continuity condition for one of the kinematic quantities of the system ($\mathbf{W}$),

$$\begin{aligned} \mathbf{M}\ddot{\mathbf{U}}^k(t) + \mathbf{F}^k_{int}(\mathbf{U}(t)) &= \mathbf{F}^k_{ext}(t) + \mathbf{F}^k_{link} \\ \sum_k \mathbf{C}^k \mathbf{W}^k &= 0 \end{aligned} \tag{2.6}$$

[2]The terms interface or patch are also used to refer to coupling surfaces.

The structure of connectivity matrices $\mathbf{C}^k$ depends on the geometry of the model and its partition into subdomains. Linking forces verify the equilibrium through Lagrange multipliers $\mathbf{\Lambda}$,

$$\mathbf{F}^k_{link} = \mathbf{C}^k \mathbf{\Lambda}, \tag{2.7}$$

then displacements and velocities can be integrated as in the previous section, for all subdomains. For further details, a partition of the time integrator of the HHT α-method is studied in [36].

In this section we have presented the formulation of domain decomposition within a monolithic frame. On that account, let us define clearly what we mean by a monolithic approach, as opposed to partitioned. We use the term monolithic when the governing equations (2.6) include the linking forces $\mathbf{F}^k_{link}$, and they are solved by a program working with a mesh that represents the entire domain. Alternatively, the approach is partitioned when solver programs consider the equations without linking forces terms. Then, we incorporate extra algorithmic steps, in which variables are exchanged between solvers in order to impose boundary conditions at the coupling surface. Note that within a monolithic frame, we still perform an algebraic partition, or splitting, of data structures and time integrators.

Many monolithic approaches implement subcycling along with splitting of the time integrator. Subcycling, also known as multi-time stepping, means that, for each subdomain, time discretization is performed with different step lengths. Additional considerations are necessary in that case. However, we do not cover this case here.

2.3 Domain Decomposition within the Partitioned Frame

The partitioned approach also involves a dual Schur decomposition, but it is performed following the idea of non-intrusive coupling [29].

Starting from the monolithic decomposition we have just seen, a first step towards a partitioned frame is to uncouple the system of equations [10]. We can separate, in (2.6), those terms corresponding to each subdomain (free problem), and the linking terms, in such a way that the governing motion equations take the form:

$$\begin{aligned} \mathbf{M}^k \ddot{\mathbf{U}}^k_{free} + \mathbf{F}^k_{int}(\mathbf{U}^k) &= \mathbf{F}^k_{ext} \\ \mathbf{M}^k \ddot{\mathbf{U}}^k_{link} &= \mathbf{F}^k_{link}. \end{aligned} \tag{2.8}$$

Once in this form, a parallel computing strategy seems mostly convenient. Variables corresponding to the free problem of each subdomain, and the linking terms, are computed in separate threads every time step.

However, this is still a monolithic approach, according to the definitions given in the previous section. We can go one step further, and avoid the calculation of the linking terms altogether. Instead, we calculate only the free problem of each subdomain, and then we force the synchronization of dynamic quantities at the coupling surface. To this end, we exchange certain variables at the interface after every time step.

In a differential formulation, continuity of all the dynamic quantities has to be guaranteed at the interface. However, for numerical reasons, in a discrete formulation, only one dynamic quantity can be prescribed at the boundary of a subdomain. Thus, we can couple the simulations of each pair of adjacent subdomains with the prescription of two different dynamic quantities at their interface. For instance, we can exchange displacements and forces, with each solver sending one and receiving the other.

Formally, the linking term of (2.8) is substituted in the partitioned approach for an imposition of Dirichlet and Neumann boundary conditions in every interface, which brings the dynamic equilibrium. This is explained in detail in the following section.

From an implementation point of view, vectors of displacements and forces must be accessible by an external coupling agent during the simulation, so that they can be exchanged. In this project, the coupling library preCICE takes care of these tasks.

Depending on how variables are exchanged, we consider two types of coupling schemes (see flow diagrams in Figure 2.4):

- Explicit/weak coupling: every time step, interface values are exchanged and the simulation is advanced without convergence check. Therefore, solvers run just once per time step, but neither stability nor accuracy are guaranteed.
- Implicit/strong coupling: every time step, after the exchange of values, solvers are executed repeatedly in a fixed-point iteration until convergence is reached, according to a measure based on displacements and/or forces at the interface, e.g. a residual. Provided that simulations are stable in each subdomain, when the coupling is implicit the global simulation is unconditionally stable, since we recover the monolithic solution every time step. The fixed-point iteration add an extra layer of complexity to the computation, therefore, it is convenient to use convergence acceleration techniques such as Aitken underrelaxation or quasi-Newton schemes, to reduce execution time. Quasi-Newton schemes are introduced in Section 2.3.2.

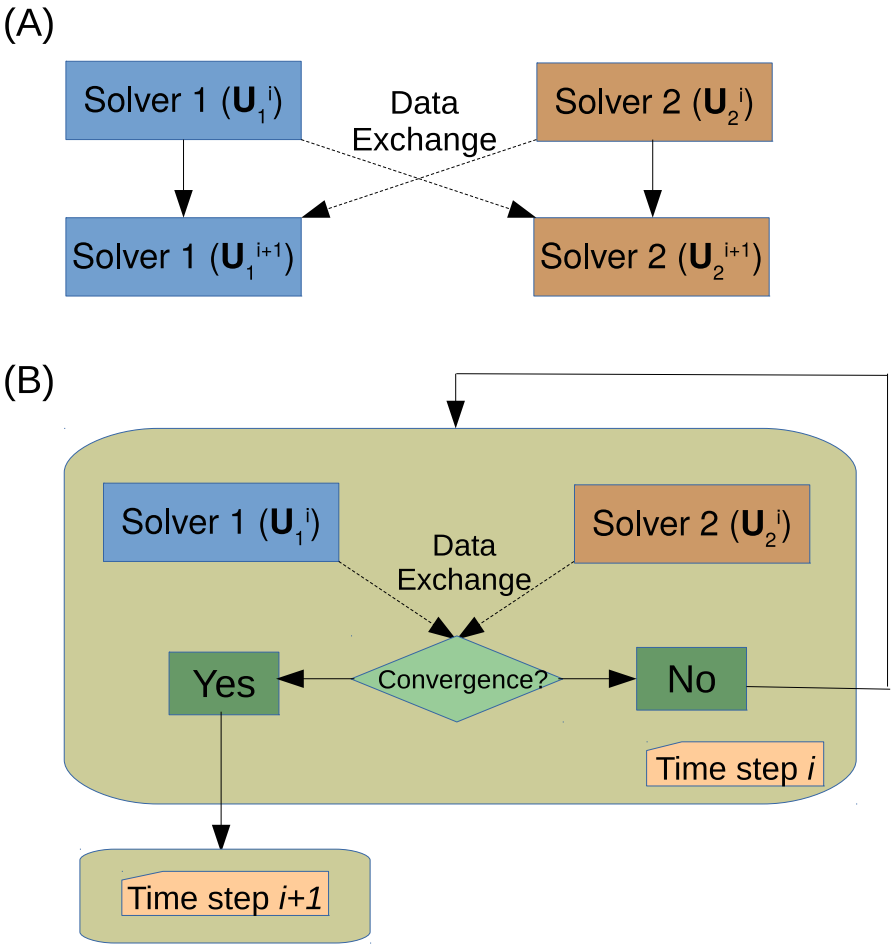

FIGURE 2.4: Implicit/explicit coupling schemes for a bipartite domain. Continuous line arrows indicate execution flow. Superindices indicate time discretization, subindices indicate subdomain. (A) Explicit coupling scheme. Data is directly exchanged and the simulation is advanced to the next time step. (B) Implicit coupling scheme. After data exchange, the simulation is advanced only if convergence is reached, otherwise the solvers must iterate again for the same time step.

Aside from being explicit or implicit, coupling schemes are classified as (see Figure 2.5):

- Serial: if the participants run alternately in one processor.
- Parallel: if the execution is carried out simultaneously in separate threads, and synchronized when variables are exchanged.

Regarding multi-time stepping, in this explanation, for simplicity, we restrict to the case of equal and constant time step for all subdomains. We do the same in our implementation, as can be seen in Chapter 4. Later on, once the method is validated, subcycling can be incorporated.

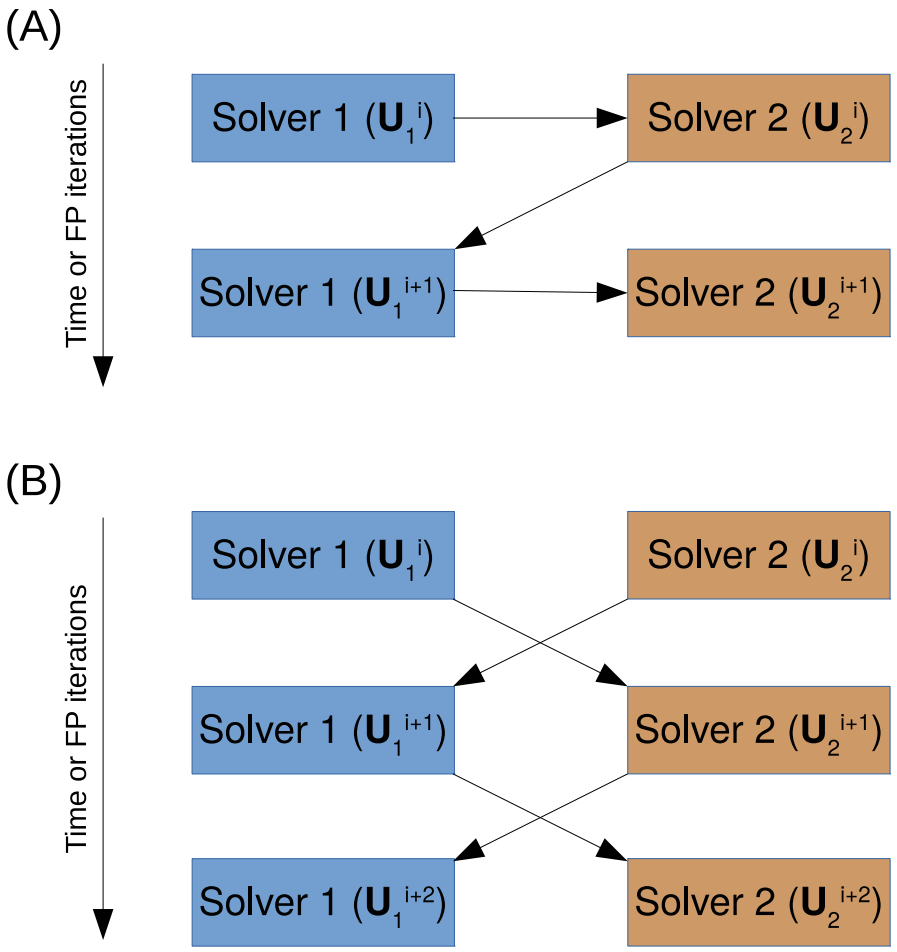

FIGURE 2.5: Serial/parallel coupling schemes for a bipartite domain. Execution flow and data exchange indicated by the arrows. Superindices indicate generic iterations (time step or fixed-point iterations), subindices indicate subdomain. (A) Serial execution. Only one solver is executed at the same time, using interface data from the other one, which means that for the second solver this data is from the same iteration. (B) Parallel execution. Both solvers are executed in parallel, using interface data from the previous iteration.

2.3.1 Dirichlet-Neumann Coupling

To implement partitioned coupling, we prescribe boundary conditions (BCs) at the interfaces Γ_{kl} of all pairs of adjacent subdomains Ω_k and Ω_l.

In models of computational mechanics, continuity of kinematic quantities and momentum conservation through the interface are guaranteed via prescription of Dirichlet and Neumann BCs. Before jumping into the explicit formulation of the BCs, we must remember the expression of the conservation of momentum in a solid structure with no external forces,

$$\rho\ddot{\mathbf{U}}(t) - \bigtriangledown \cdot \sigma = 0, \tag{2.9}$$

where ρ is the density of the solid; σ is the Cauchy stress tensor, and can be understood as an expression of internal forces acting between subdomains k and l in the structure

in relation to the deformation caused. Subsequently, it is related to the change of the areas of their mutual interfaces Γ_{kl} [24]. This conservation law is expressed in terms of an equilibrium between the second derivative of displacements, acceleration, and the divergence of the stress tensor. The divergence here can be interpreted as the local outward flux of the stress field, or the gradient of pressure. For each bipartite system, considered independently, the consequence is that two conditions are needed at the interface to impose a mechanical equilibrium: a kinematic condition and a dynamic one, which are the Dirichlet and Neumann BCs, respectively.

A Dirichlet BC prescribes values for a function on the boundary. This can be formulated for the displacements field as

$$\mathbf{U}(t)\big|_{\Gamma} = f(\mathbf{r}, t), \quad \mathbf{r} \in \Gamma, \tag{2.10}$$

which is the kinematic equilibrium condition.

A Neumann BC prescribes values for the derivative of a function on the boundary. In the case of structural mechanics, the Neumann BC is defined with the normal derivative. The normal derivative gives the rate of variation of a function in the direction normal to the boundary. Conventionally, the normal vector is taken pointing outwards from the domain where the BC is prescribed. In fluid-structure interaction, the conservation law (2.9) is formulated in the continuum, which leads to the dynamic condition

$$\hat{n}_f \cdot \sigma_f = -\hat{n}_s \cdot \sigma_s \tag{2.11}$$

where $\hat{n}$ are unit vectors normal to Γ, and subscripts f and s indicate the fluid and structure subdomains, respectively. However, in the case of structure-structure interaction, we can integrate the stress tensor and obtain the forces at the nodes instead of differential stresses. Thus we rewrite the dynamic condition, and Neumann BC for partitioned SSI, as

$$\mathbf{F}(t)\big|_{\Gamma} = g(\mathbf{r}, t), \quad \mathbf{r} \in \Gamma, \tag{2.12}$$

where $\mathbf{F}(t)$ is the vector of nodal forces, containing internal forces as well as external loads. In Chapter 4 we see how boundary conditions are applied as force loads.

In practice, we prescribe Dirichlet and Neumann BCs in a conjugate manner, to ensure equilibrium. The method consists in the appropriate definition of functions $f(\mathbf{r}, t)$ and $g(\mathbf{r}, t)$, in such a way that values of displacements and forces at Γ are exchanged between subdomains every time step.

To this end, for an interface Γ_{kl}, we identify displacements $\mathbf{U}_l(t)$ as $f(\mathbf{r}, t)$ in (2.10), and forces $\mathbf{F}_k(t)$ as $g(\mathbf{r}, t)$ in (2.12). Thus, we write the prescription of Dirichlet and Neumann

BCs, with superscripts indicating time discretization,

$$\begin{aligned} \mathbf{U}_k^{i+1}\big|_{\Gamma_{kl}} &= \mathbf{U}_l^i\big|_{\Gamma_{kl}} \\ \mathbf{F}_k^{i+1}\big|_{\Gamma_{kl}} &= \mathbf{F}_l^i\big|_{\Gamma_{kl}}. \end{aligned} \tag{2.13}$$

We explain the implementation of this scheme in Section 4.1.

2.3.2 Quasi-Newton Schemes

Fixed-point iterations for implicit partitioned coupling have a high computational cost if they are implemented without an acceleration technique.

A possibility is to use quasi-Newton (QN) schemes, based on multisecant methods (see, for instance, [23]). The residual system $\mathbf{R}$ at the coupling interface, i.e. the difference between the solution of the two last iterations in the subset of nodes located at the interface, is used to perform Newton-like update steps. An exact method requires the inverse Jacobian of the residual, $\mathbf{J}_R^{-1}$, which is expensive to compute. QN schemes re-use vectors of previous iterations, in order to find an approximation of the inverse Jacobian of $\mathbf{R}$ after every iteration, $\widetilde{\mathbf{J}}_R^{-1}$.

Once a specific residual $\mathbf{R}$ is defined for displacements or forces, a QN method starts building two matrices. Columns of $\mathbf{V}$ are the displacements or forces computed in previous iterations, and columns of $\mathbf{W}$ are the residuals, or a related quantity, of previous iterations as well.

Finding an approximation to the inverse interface Jacobian corresponds to solving the overdetermined system

$$(\widetilde{\mathbf{J}}_R^i)^{-1}\mathbf{V}^i = \mathbf{W}^i, \tag{2.14}$$

with subindices i used to indicate the time step. In our implementation, the condition to find $(\widetilde{\mathbf{J}}_R^i)^{-1}$ is to minimize its norm in a least squares problem. This type of QN scheme is referred to as interface quasi-Newton – inverse least-squares (IQN-ILS).

The performance of quasi-Newton schemes can be increased incorporating information of previous time steps. Matrices $\mathbf{V}$ and $\mathbf{W}$ are extended accordingly.

Additionally, by filtering out columns with values below a threshold to prevent linear dependence between these columns, a significant performance improvement can be achieved [37]. We study the effects of filtering in the numerical studies of Section 5.3.

Chapter 3

Software in the Project

We have presented the theoretical framework of structure-structure coupling, following both the monolithic and partitioned approach. Specially, we have focused on the concepts of partitioned coupling and the necessary techniques for validity, efficiency and stabilization of the corresponding algorithms.

In this chapter, we give an overview of the software that plays a role in our implementation of partitioned structure-structure coupling. On the one hand, we have the finite element program CalculiX. It plays the role of the single-physics solver in our framework, providing a solution for the structural mechanics problem in every subdomain. On the other hand, we have the coupling library preCICE, that provides the functionality to couple the single-physics simulations. Every instance of CalculiX that we execute is seen by preCICE as a participant of the simulation.

3.1 preCICE - A Coupling Library for Partitioned Multi-Physics Simulations

The coupling library preCICE (Precise Code Interaction Coupling Environment) was developed as an external tool to be called by the participants of the simulation, i.e. the single-physics solvers, while they run in parallel, in such a way that turns a set of independent simulations into a multi-physics one, via surface coupling. It is written in C++ and has a high-level API for integration with the solvers.

The basic idea of preCICE is to bring flexibility to implementations of partitioned coupling schemes with the possibility of using existing state-of-the-art solvers, both opensource and commercial, and not necessarily written in C++; instead of having to develop new software solutions adapted to each kind of interaction problem. Following this idea, integration

of preCICE with these solvers is minimally invasive, it requires but a few changes to the main routines of the solvers. This is done with an **adapter**.

A preCICE adapter consists of code that connects the preCICE library with a solver. Part of this code is in independent files, while the rest is written in the form of modifications introduced to the solver routines.

Due to the black-box philosophy of preCICE, as few assumptions as possible must be made regarding the solvers. Considering this, a variety of functions are required to guarantee that convergence and stability can be transferred from the subdomains to the full simulation.

We proceed to describe this functionality and the configuration process. A thorough description is out of scope here, for exhaustive practical details please refer to the wiki on GitHub [38].

3.1.1 Overview of the Functionality of preCICE

Let us briefly explain the functionality of preCICE:

- Explicit/implicit and serial/parallel coupling: all four combinations of coupling schemes, described in Section 2.3, are available in preCICE.
- preCICE introduces acceleration techniques in fixed-point iterations, to improve stability and to reduce the number of iterations needed for convergence. The application of these techniques is referred to as post-processing. Numerical methods available in preCICE for post-processing are constant and Aitken underrelaxation, as well as two quasi-Newton methods.
- Technical realizations for communication can be done by means of MPI ports, TCP/IP sockets or communication via files. Sockets are the recommended option for an optimal balance between robustness and performance.
- In the general case, we assume non-matching meshes of the models at the interface. To handle this, preCICE incorporates a set of functions for data mapping between the meshes used by the solvers. There are two alternatives to define a constraint for these functions, consistent and conservative, indicating whether constant values must be exactly mapped or integral values preserved, respectively. Mapping methods can be projection-based or use radial basis functions to interpolate the values. Two projection-based methods are available. Nearest-neighbour mapping connects each node in the target mesh with the closest node from the source mesh.

Nearest-projection mapping firstly orthogonally projects nodes of the target mesh onto elements of the source mesh, and then performs linear interpolation using nodal values from the element of the source mesh. When meshes are known to be matching at the interface, mapping is not numerically necessary; still, in practice, a nearest-neighbour criterion is used for a coincident one-to-one mapping. This is the case in all the setups of this project.

- Non-matching time step sizes are supported, currently with a constant extrapolation in time. Further methods for handling non-matching temporal discretizations are currently under development [39].

3.1.2 Configuration of the Coupling

To configure any preCICE coupled simulation, it is necessary to prepare an XML file that is read at runtime, where we indicate parameters for several areas:

- Coupling data: indicate which variables are exchanged in the simulation.
- Coupling meshes: define the meshes and associated variables.
- Coupling participants: establish the relations between participants of the simulation, meshes and the coupling data previously defined. Indicate the direction of data exchange (read-write) for each participant, and the type of interpolation for data mapping between meshes.
- Communication: specify the channel of communication between processors.
- Coupling scheme: indicate whether the coupling is serial or parallel, and explicit or implicit. Also give numerical parameters for the simulation such as time step length or maximum time steps allowed. Depending on the choice of scheme, further details must be given to configure the convergence criteria for the fixed-point iteration; and which post-processing technique, if any, will be used, as well as parameters for it.

A sample XML configuration file is presented in Annex A.1.

Extra YAML configuration files are required by the preCICE adapter for CalculiX, one per participant. In these files, we point to the corresponding XML file, and configure the adapter. A sample YAML configuration file is presented in Annex A.2.

3.2 CalculiX - A Three-Dimensional Structural Finite Element Program

CalculiX [1] is a software package that solves several types of field problems with the finite element method [32]. Analysis types available in CalculiX belong mostly, but not exclusively, to structural mechanics and thermodynamics. The coupling of transient structural mechanics simulations requires a direct integration dynamic analysis. In such an analysis, global equations of motion are integrated in time, which can be done with an implicit or explicit method. For our setup, we have opted for an implicit dynamic analysis, which is specially convenient for nonlinear systems.

For the integration of CalculiX with preCICE, we need an adapted version of the main routines of the former, which make up part of the preCICE adapter for CalculiX (see Chapter 4).

The main routine of CalculiX, and a small part of the subroutines directly called by it, are written in C; while the rest of the subroutines are in FORTRAN.

3.2.1 Overview of the Functionality of CalculiX

Dynamic analyses with CalculiX calculate the response of a structure to dynamic loading. CalculiX uses an improved version of the Hilber-Hughes-Taylor method [33] to integrate the equations of motion. The resulting system of equations is solved with a direct solver, that can be chosen among several alternatives.

The analysis can take into account geometrically nonlinear effects, which is necessary for large displacements. In that case, a nonlinear strain tensor is used, and every time step a Newton iteration is performed.

Boundary conditions are defined for arbitrary node sets, and stored in internal variables of the Calculix routines. During the simulation, BCs are prescribed every time step to the indicated nodes. This procedure is suitable for the exchange of boundary values between participants, by manipulating the internal variables of CalculiX.

Time step length in CalculiX can be adaptive or fixed. For a coupling scheme without subcycling, such as the one in our implementation, a fixed time step length is adequate.

3.2.2 Configuration of the Simulation

A CalculiX job is configured with an input file where the user specifies required and optional parameters, indicated in the input file by predefined keywords. The amount of options is far too large for an exhaustive review, therefore we describe only those relevant for our project. For a detailed explanation, please refer to the CalculiX documentation [35].

Here we provide an explanation of keywords:

- INCLUDE: this keyword is used to import mesh files, which contain nodal coordinates as well as element type and nodes-element correspondence. For coupling purposes, additional files are imported to specify the subset of nodes located at the coupling surface. Furthermore, if external forces act on the structure, another file contains the subset where external boundary conditions (not related to coupling operations) need to be prescribed.
- BOUNDARY: here we prescribe Dirichlet BCs, giving values for the displacements in a set of nodes previously defined. Dirichlet BCs in CalculiX are also called single point constraints (SPC), since they are prescribed independently to single node.
- CLOAD: here we prescribe Neumann BCs, giving values for the nodal forces in a set of nodes previously defined. Discrete Neumann BCs in CalculiX are also referred to as concentrated loads, since they are applied independently to each node. To be able to apply a concentrated load to a node, it can't be already fixed by a SPC. A time-dependent amplitude can be defined to prescribe variable loads.
- MATERIAL: here we give values for the physical parameters of the structure. For a linear elastic model of a solid, CalculiX requires a mass density ρ, a Young modulus E, and a Poisson ratio ν.
- DYNAMIC: this keyword is used to indicate that we run a dynamic analysis; and to specify various parameters including but not restricted to:
  - Time step length Δt.
  - Whether Δt is adaptive or not.
  - Total duration of the simulation.
  - Use of explicit or implicit integration.
- STEP: in CalculiX, we can configure a job so that it consists of more than one consecutive simulation. Each simulation is a step, not to be confused with the time step Δt, which in the CalculiX environment is referred to as time increment.

Each step will generally have different parameters. The STEP keyword is used to separate parts of a configuration file belonging to different simulation steps. It can also indicate additional options, for instance whether the simulation is taking into account geometric nonlinearities.

For a sample CalculiX input file, see Annex B.

3.2.3 Mesh

A mesh is a discretized geometric description of the structure, which is needed by a finite element solver to run an analysis. Formally, a mesh consists of:

- A list of nodes, each defined as a point in a 3D space with a unique identifier.
- A list of elements, indicating their type. Each element is defined as the set of nodes that lie in the boundary of the element, with a unique identifier.

The CalculiX package includes a graphic tool, CalculiX GraphiX, which can be used to define a geometry for a finite element model. In this way, a mesh representing the geometry can be obtained in the native format of CalculiX. Then, the mesh is prepared to be read by the solver program.

Chapter 4

Implementation

In the previous chapter, we have presented the software tools that constitute the basis upon which we have developed an extension to deal with structure-structure interaction problems. In this chapter, we describe the preCICE adapter for CalculiX [4], which was firstly implemented for conjugate heat transfer, later on for dynamic fluid-structure interaction [40], and has now been extended to handle SSI cases.

Firstly, we must understand the way we augment FSI functionality. Simulations of FSI with the preCICE adapter for CalculiX rely also on another solver with its respective preCICE adapter, to simulate the fluid side of the physical system. Just as in the SSI case, the simulation is coupled by means of the prescription of Dirichlet and Neumann boundary conditions on the interface (see Section 2.3.1). To this end, the adapters obtain and map values of displacements and forces at the interface from the solvers, and keep them in a specific data structure. Then, they send these values to the other solver. Specifically, the following exchange takes place:

- The preCICE adapter for CalculiX **writes** displacements from the internal data structures of CalculiX into the preCICE interface, and **reads** forces from the preCICE interface into the internal variables of CalculiX.
- The preCICE adapter for the fluid solver **writes** forces from the internal variables of the fluid solver into the preCICE interface, and **reads** displacements from the preCICE interface into the internal variables of the fluid solver.

Accordingly, in the FSI implementation, two of the exchange operations were implemented in the adapter for CalculiX, and the other two in the adapter for the fluid solver. Thus, in order to incorporate SSI, it is necessary to extend the adapter for CalculiX with the implementation of two new operations:

- **Read** displacements from the preCICE interface into the internal variables of CalculiX.
- **Write** forces from the internal variables of CalculiX into the preCICE interface.

Once these operations are implemented, we can run an SSI simulation with two CalculiX participants. The implementation of all read-write operations in the adapter for CalculiX is described in Section 4.1. Note that on the one hand, one of the two participants exchanges values in the same direction as the structural participant in an FSI simulation. On the other hand, the other participant sends and receives variables in the same direction as the fluid participant. Thus, the functions that need to be implemented in the adapter are needed for the latter, as can be seen in Figure 4.1.

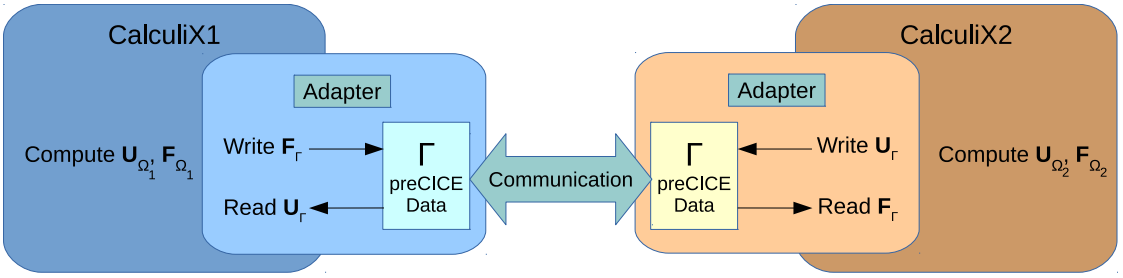

FIGURE 4.1: Simulation setup and exchange of variables with two CalculiX participants, coupled via calls to preCICE functions from the adapted main routines of CalculiX (the adapter). Data structures are created by each adapter, to prepare the interface data (Γ) to be exchanged when the simulation is advanced, following a decentralized peer-to-peer approach. CalculiX1 needs the new read and write operations, while CalculiX2 makes use of the functions already implemented for FSI.

The basic idea of the adapter is to introduce commands to the main solver loop in CalculiX, in order to:

- create a data structure, **SimulationData**, to handle access to internal CalculiX variables and store coupling data to control the simulation, such as the time step.
- create another data structure, **PreciceInterface**, a member of **SimulationData**, to keep the configuration of the coupling interface. For instance, node identifiers and data mapping vectors.
- take control of the time step in CalculiX, and set up implicit coupling iterations with checkpointing.
- steer the simulation and execute the exchange of variables, in accordance with the configuration and preparation of data.

The adapter code is completely open-source and accessible in [4]. For a close examination of the relevant sections of code, indications are given in Section 4.3.

4.1 Inter-Field Prescription of Boundary Conditions

Let us describe in detail now the procedure of exchange of values at the interface, that implements the prescription of boundary conditions needed for a partitioned coupling approach. The definition and indexing of all internal CalculiX variables mentioned here can be found in the CalculiX manual [35].

Every time step, each participant of the simulation executes internal solving subroutines that read the current vector of nodal displacements (state of the simulation) and the boundary conditions, and then compute nodal forces and displacements for the next time step. These results are kept in the internal variables **vold** (displacements) and **fn** (forces) of CalculiX.

Boundary conditions, which are read and applied by the solver subroutines, are kept in the variables **xboun** (displacements, single point constraints) and **xforc** (forces, concentrated loads). Values stored in these variables, before being modified by the adapter, are those defined in the CalculiX input file described in Section 3.2.2.

The adapter stores pointers to all these vectors, and calls the high-level preCICE functions *Precice_ReadCouplingData* and *Precice_WriteCouplingData*. The former is called every time the adapter needs to read the results from the current iteration from **vold** and **fn**; the latter, when the adapter needs to modify the values of **xforc** and **xbound** so that in the current iteration, values received from the other solver are prescribed as BCs. How CalculiX internally incorporates the prescribed BCs into the differential equations is beyond scope here, the reader is referred to the CalculiX manual for insight.

We describe now the high-level read and write functions:

- *Precice_ReadCouplingData*: the function takes a **SimulationData** instance as argument and checks from the corresponding **PreciceInterface** which type of data has to be read, displacements or forces. As we mentioned earlier, the adapter was initially implemented for conjugate heat transfer, and therefore all functions support working with temperature, heat flux, and convection. Displacement increments are also supported, and needed for FSI. A switch statement handles each of these cases.

  Knowing the data type, the function calls a lower-level function to configure the reading, this function is a generic one from the preCICE core and not part of the adapter, *precicec_readBlockVectorData*. It takes as arguments the data type ID, the number and ID of nodes of the coupling surface and a pointer to a buffer, **nodeVectorData**, to hold the values.

  Next, the buffer is passed to a helper function. Values are mapped from **PreciceInterface** to **SimulationData**; this requires an index mapping function.

Let us now take a look at an excerpt of the function code. In the case of reading forces, *Precice_ReadCouplingData* executes the following commands:

```
case FORCES:
    precicec_readBlockVectorData( interfaces[i]->forcesDataID,
    interfaces[i]->numNodes, interfaces[i]->preciceNodeIDs,
    interfaces[i]->nodeVectorData );
    setNodeForces( interfaces[i]->nodeVectorData, interfaces[i]->numNodes,
    interfaces[i]->xforcIndices, sim->xforc);
    break;
```

and for reading displacements:

```
case DISPLACEMENTS:
    precicec_readBlockVectorData( interfaces[i]->displacementsDataID,
    interfaces[i]->numNodes, interfaces[i]->preciceNodeIDs,
    interfaces[i]->nodeVectorData );
    setNodeDisplacements( interfaces[i]->nodeVectorData,
    interfaces[i]->numNodes, interfaces[i]->xbounIndices,
    sim->xboun );
    break;
```

where **sim** is an instance of **SimulationData** and **interfaces[i]** is an instance of **PreciceInterface** in which data of one particular coupling surface is stored; for our case, a bipartite simulation, there is only one interface, but the adapter is prepared to handle coupling with additional subdomains.

After seeing the high-level functions, we present the helper ones:

```
void setNodeForces( double * forces, ITG numNodes, int * xforcIndices,
double * xforc )
{
    ITG i;
    for ( i = 0 ; i < numNodes ; i++ )
    {
        // x-component
        xforc[xforcIndices[3 * i]] = forces[3 * i];
        // y-component
        xforc[xforcIndices[3 * i + 1]] = forces[3 * i + 1];
        // z-component
        xforc[xforcIndices[3 * i + 2]] = forces[3 * i + 2];
    }
}

void setNodeDisplacements( double * displacements, ITG numNodes,
int * xbounIndices, double * xboun )
{
    ITG i;
    for( i = 0 ; i < numNodes ; i++ )
    {
        // x-component
        xboun[xbounIndices[3 * i]] = displacements[3 * i];
        // y-component
        xboun[xbounIndices[3 * i + 1]] = displacements[3 * i + 1];
        // z-component
        xboun[xbounIndices[3 * i + 2]] = displacements[3 * i + 2];
    }
}
```

where we can see how the three components of the vector values from **PreciceInterface** are copied into the BC vectors for each node of the interface, according to two index mappings, **xforcIndices** and **xbounIndices**.

For further details on the low-level functions of the preCICE API, see for example [37].

- *Precice_WriteCouplingData*: the writing function takes a **SimulationData** instance as argument, and firstly checks from **PreciceInterface** which type of data has to be written.

  Then, depending on the data ID, helper functions store in the preCICE buffer a copy of the force vector **fn**, or the displacement vector **vold**, from **SimulationData**. The ID of nodes and an auxiliary variable **mt** are needed for indexing, but unlike the case of reading, no additional mapping function is used.

  Afterwards, the function calls a lower-level preCICE function to configure the writing of the buffer, *precicec_writeBlockVectorData*, which needs to be passed the same arguments as *precicec_readBlockVectorData*.

We show here an excerpt of the code of Precice_WriteCouplingData for writing forces:

```
case FORCES:
    getNodeForces( interfaces[i]->nodeIDs, interfaces[i]->numNodes,
    sim->fncopy, sim->mt, interfaces[i]->nodeVectorData );
    precicec_writeBlockVectorData( interfaces[i]->forcesDataID,
    interfaces[i]->numNodes, interfaces[i]->preciceNodeIDs,
    interfaces[i]->nodeVectorData );
    break;
```

and for writing displacements:

```
case DISPLACEMENTS:
    getNodeDisplacements( interfaces[i]->nodeIDs, interfaces[i]->numNodes,
    sim->vold, sim->mt, interfaces[i]->nodeVectorData );
    precicec_writeBlockVectorData( interfaces[i]->displacementsDataID,
    interfaces[i]->numNodes, interfaces[i]->preciceNodeIDs,
    interfaces[i]->nodeVectorData );
    break;
```

Then, the respective helper functions:

```
void getNodeForces( ITG * nodes, ITG numNodes, double * fn, ITG mt,
double * forces )
{
    ITG i;
    for ( i = 0 ; i < numNodes ; i++ )
    {
        int nodeIdx = nodes[i] - 1;
        //x-component
        forces[3 * i] = fn[nodeIdx * mt + 1];
        //y-component
        forces[3 * i + 1] = fn[nodeIdx * mt + 2];
        //z-component
        forces[3 * i + 2] = fn[nodeIdx * mt + 3];
    }
}

void getNodeDisplacements( ITG * nodes, ITG numNodes, double * v, int mt,
double * displacements )
{
    ITG i;
    for( i = 0 ; i < numNodes ; i++ )
    {
        int nodeIdx = nodes[i] - 1;
        //x-component
        displacements[3 * i] = v[nodeIdx * mt + 1];
        //y-component
        displacements[3 * i + 1] = v[nodeIdx * mt + 2];
        //z-component
        displacements[3 * i + 2] = v[nodeIdx * mt + 3];
    }
}
```

where we can see how the three components of the CalculiX vectors are copied into the interface vectors, for each node, according to the node IDs from **PreciceInterface**.

We have seen that, in order to map node indices from the BC vectors of CalculiX to the coupling interface vectors of preCICE, we need specific mapping functions. The adapter already supported this functionality for forces in **xforc**, in this project it has been extended to work for displacements in **xboun**. We proceed to show the code for this task:

```
void getXforcIndices( ITG * nodes, ITG numNodes, int nforc, int * ikforc,
int * ilforc, int * xforcIndices )
{
    ITG i;
    for( i = 0 ; i < numNodes ; i++ )
    {
        //x-component
        int idof = 8 * ( nodes[i] - 1 ) + 1;
        int k;
        FORTRAN( nident, ( ikforc, &idof, &nforc, &k ) );
        k -= 1;
        int m = ilforc[k] - 1;
        xforcIndices[3 * i] = m;
        //y-component
        idof = 8 * ( nodes[i] - 1 ) + 2;
        FORTRAN( nident, ( ikforc, &idof, &nforc, &k ) );
        k -= 1;
        m = ilforc[k] - 1;
        xforcIndices[3 * i + 1] = m;
        //z-component
        idof = 8 * ( nodes[i] - 1 ) + 3;
        FORTRAN( nident, ( ikforc, &idof, &nforc, &k ) );
        k -= 1;
        m = ilforc[k] - 1;
        xforcIndices[3 * i + 2] = m;
    }
    //...
}

void getXbounIndices( ITG * nodes, ITG numNodes, int nboun, int * ikboun,
int * ilboun, int * xbounIndices, enum CouplingDataType couplDataType )
{
    //...
    case DISPLACEMENTS:
        for( i = 0 ; i < numNodes ; i++ )
        {
            // x-component
            int idof_x = 8 * ( nodes[i] - 1 ) + 1;
            // y-component
            int idof_y = 8 * ( nodes[i] - 1 ) + 2;
            // z-component
            int idof_z = 8 * ( nodes[i] - 1 ) + 3;
            int kx, ky, kz;
            FORTRAN( nident, ( ikboun, &idof_x, &nboun, &kx ) );
            FORTRAN( nident, ( ikboun, &idof_y, &nboun, &ky ) );
            FORTRAN( nident, ( ikboun, &idof_z, &nboun, &kz ) );
            xbounIndices[3 * i] = ilboun[kx - 1] - 1;
            xbounIndices[3 * i + 1] = ilboun[ky - 1] - 1;
            xbounIndices[3 * i + 2] = ilboun[kz - 1] - 1;
        }
    //...
}
```

Note that FORTRAN subroutines must be called to obtain the right indices, but no changes in these routines are necessary.

Following from what has been already mentioned in the introduction of this chapter and in Figure 4.1, the functions that have been implemented as extension of the adapter for SSI are *setNodeDisplacements* and *getNodeForces*, i.e. read $\mathbf{U}_\Gamma$ and write $\mathbf{F}_\Gamma$, respectively.

4.2 Implementation of Implicit Coupling

The operations above can be part of an explicit or implicit coupling scheme. Accordingly, the adapter must be able to handle both cases.

An implicit scheme requires, apart from the time stepping loop, an additional checkpointing strategy to incorporate coupling iterations. The adapter introduces checkpoints that store the state of the simulation (values of displacements) at the beginning of the simulation loop, and reload them at the end, as long as the fixed-point iteration has not converged yet. Once it does converge, a new checkpoint is written with the updated displacements.

On the contrary, following an explicit scheme, the simulation is advanced to the next time step every time the solvers are executed, thus checkpointing is not necessary. Therefore, the code for the explicit case is essentially included in the code for the implicit case. For that reason, we explain directly the implicit case, which is also the one that offers more improvement and is thus studied in Chapter 5.

The main routine of CalculiX originally includes a simulation loop over the time steps. The adapter adds the possibility of reloading the previous time step with checkpointing, creating a coupling loop over the iterations of an implicit coupling scheme. This loop runs until the convergence criteria set in the preCICE configuration file are met.

Checkpointing is implemented mainly with two functions:

- *Precice_WriteIterationCheckpoint*: called at the beginning of the time step iteration when the time step is advanced, stores the values of **vold** in the auxiliary vector **coupling_init_v**, as well as the current time step and time step length. Note that latter is not strictly necessary, in absence of subcycling.

The code of the function reads:

```
void Precice_WriteIterationCheckpoint( SimulationData * sim, double * v )
{
    // Save time step
    sim->coupling_init_theta = *( sim->theta );
    // Save time step length
    sim->coupling_init_dtheta = *( sim->dtheta );
    // Save solution vector v
    memcpy( sim->coupling_init_v, v, sizeof( double ) * sim->mt * sim->nk );
}
```

where theta is an internal CalculiX variable for the time step.

- *Precice_ReadIterationCheckpoint*: called at the end of the time step, after the internal solver iterations, if the coupling has not converged. It copies the displacement values, written in the last checkpoint to **coupling _init_v**, into the solution vector **vold**. It also loads the old time step and time step size stored in the checkpoint.

  The code of the function reads:

```
void Precice_ReadIterationCheckpoint( SimulationData * sim, double * v )
{
    // Reload time step
    *( sim->theta ) = sim->coupling_init_theta;
    // Reload time step size
    *( sim->dtheta ) = sim->coupling_init_dtheta;
    // Reload solution vector v
    memcpy( v, sim->coupling_init_v, sizeof( double ) * sim->mt * sim->nk );
}
```

4.3 Structure of the Adapter Code

In order to integrate CalculiX with preCICE, we need to introduce slight modifications into the main routine *ccx_<version>.c*[1] of CalculiX, so that it calls the adapted solver routine *nonlingeo_precice.c*. The latter handles both geometrically linear and nonlinear cases, despite its name. Furthermore, the adapter also consists of *PreciceInterface.c*, *CCXHelpers.c*, *ConfigReader.cpp*, and the respective header files. We present a diagram of the module hierarchy of the adapter in Figure 4.2.

[1] In this thesis, we employed ccx_2.13.c.

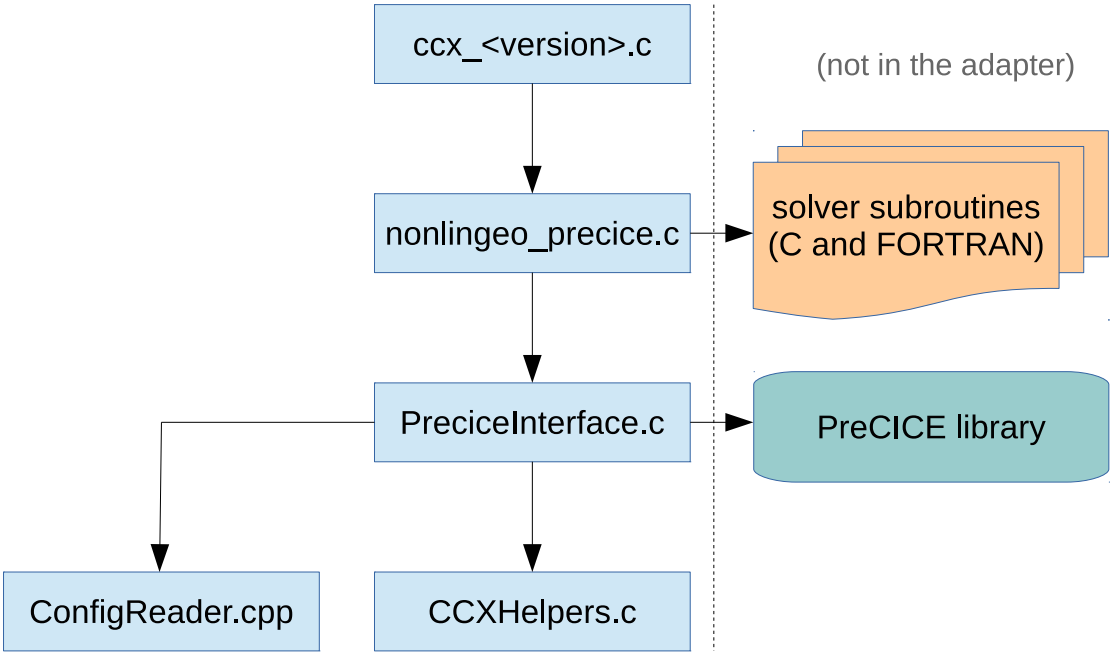

FIGURE 4.2: Module hierarchy of the preCICE adapter for CalculiX. Arrows in the diagram mean that a routine calls another one or one of its functions. The main program, *ccx_<version>*, calls *nonlingeo_precice*, a modified CalculiX routine. The latter contains the main simulation loop, and makes calls to the unmodified CalculiX solver subroutines, as well as to *PreciceInterface*, which is not modified but original code. *PreciceInterface* contains all the high-level functions for coupling, including the creation of specific data structures as explained in Section 4.1, and calls *ConfigReader* to read the YAML configuration file. *PreciceInterface* calls low-level preCICE functions and auxiliary functions from *CCXHelpers* to manipulate CalculiX data. *ConfigReader* and *CCXHelpers* are also original code of the adapter.

Chapter 5

Numerical Studies

We have presented the preCICE adapter for CalculiX in the previous chapter, explaining the implementation of structure-structure interaction functionality.

In this chapter, we begin with a description of two test cases we have set up for experimentation. The first one is a simple model of a beam with linear elastic behaviour, whereas the second one is more complex: a model of a reinforced pipe with nonlinear behaviour. In both cases, the system is bipartite; therefore the simulation is run on two instances of CalculiX, **CalculiX1** and **CalculiX2**.

Afterwards, we present the research questions, along with the tests carried out in order to answer them, and a discussion of the results. The first research objective is to validate the coupling, i.e. to confirm the equality of the solutions obtained with a partitioned approach and a monolithic one. Secondly, we study the dependence of the numerical behaviour of the model on several parameters. Then, we compare the error caused by the splitting of the model with the accuracy loss that may arise from the lack of a nonlinear analysis. Finally, we analyze and compare the performance of monolithic and partitioned simulations.

5.1 Test Cases

To begin with, we describe the test case geometries, specified as a mesh for the finite element method. In each case, a **full mesh** is generated for the complete spatial domain, and afterwards, it is split in **mesh1** and **mesh2** via the definition of a splitting plane, which is the coupling surface. Thus, we guarantee matching meshes on the interface. Additionally, different mesh sizes are used for the beam test case. We refer to the different mesh sizes as mesh size A, B and C, starting by the coarsest one. Where we do not

indicate it explicitly, we refer to the coarsest one, mesh size A, since it is used in most experiments. Furthermore, we explain all the options and values set in the configuration files of CalculiX and preCICE.

5.1.1 Beam Fixed at Both Ends and Loaded by Point Forces

The first test case is adapted from one of the CalculiX example problems, described in the manual [35] under the name *Cantilever beam*. The bending of a simple beam only requires an inexpensive linear analysis, therefore we employ this example in the validation of our implementation, in Section 5.2; in parameter dependence studies, in Section 5.3; and to analyze the influence of convergence measures on the error, in section 5.4.

Description of the Geometry

Consider a rectangular beam of dimensions $1 \times 1 \times 8\,\text{mm}$. A splitting plane is situated at $z = 6\,\text{mm}$, dividing the beam in two subdomains, **beam1** and **beam2**. The domain is discretized in a **full mesh** containing 261 nodes and 32 elements, in mesh size A. Since we have 3 spatial degrees of freedom (dof) for each node, this means a total of 783 dof. The coupling surface has 21 nodes, therefore it has 63 dof. Elements, as mentioned in Section 2.1, are chosen to be twenty-node brick elements with reduced integration. This is not changed from the original example problem, as they show stable behaviour. Finer meshes are created with 2945 and 19809 nodes respectively. The distribution of nodes and elements among meshes can be seen in Table 5.1.

	No. of nodes				No. of elements			
	Beam A	Beam B	Beam C	Pipe	Beam A	Beam B	Beam C	Pipe
Full mesh	261	2945	19809	5736	32	512	4096	11264
Mesh 1	201	2225	14913	4320	24	384	3072	8448
Mesh 2	81	785	5121	1488	8	128	1024	2816
Coupling surface	21	65	225	72				

TABLE 5.1: Number of nodes and elements for each of the meshes utilized, the partitions and the coupling surface.

Boundary and Initial Conditions

Boundary conditions are applied to node sets as shown in Figure 5.1.

- Dirichlet BCs: the beam is fixed at both of its $1 \times 1\,\text{mm}$ faces.
- Neumann BCs: a mechanical load $F = 0.1\,\text{N}$ is applied to five nodes situated in **beam2**, in one of the $1 \times 8\,\text{mm}$ faces. The force is normal to the surface, with the

vector pointing inwards. The loading is applied as a step load, i.e. only during a reduced number of time steps; specifically, after $t = 0.01\,\mathrm{s}$, F is zero.

- Initial conditions: U_0 and $\dot{U}_0$ are both zero in the entire domain.

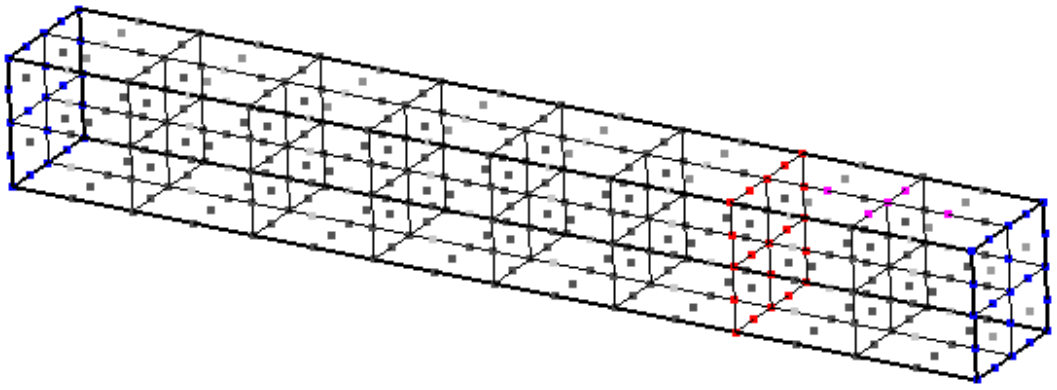

FIGURE 5.1: Mesh A of the beam test case. In blue: fixed nodes. In magenta: loaded nodes. In red: nodes of coupling surface. Element edges are shown as well, as black lines.

Parameters and Configuration

The following configuration is common for all tests with the beam:

- The material is a solid with linear elastic behaviour. The following parameter ranges are chosen to explore the numerical robustness of the method:
  - For density: $7.8 \times 10^{-14}\,\mathrm{kg\,m^{-3}} < \rho < 7.8 \times 10^{9}\,\mathrm{kg\,m^{-3}}$
  - For Young modulus: $210\,\mathrm{N\,mm^{-2}} < E < 2.1 \times 10^{8}\,\mathrm{N\,mm^{-2}}$
  - For Poisson ratio: $0.1 < \nu < 0.5$
- The type of analysis is dynamic, without taking into account geometric nonlinearities.
- Single point constraints and concentrated loads are indicated, according to the boundary conditions presented above.
- Time step length, Δt, is made constant throughout the simulation. Besides, there is no subcycling, thus Δt is the same for both participants, **CalculiX1** and **CalculiX2**. We keep Δt in the range $[2.5 - 50] \times 10^{-3}$s.
- Simulation time is fixed at 0.5 s, and the number of time steps is adjusted accordingly, depending on Δt.

- The coupling scheme is parallel and implicit, meaning that both participants are executed simultaneously. The stopping criterion for coupling iterations is a relative convergence measure, in the range $[10^{-8}, 0.1]$.
- A quasi-Newton inverse least-squares method is used as post-processing technique, with further parameters.
  - Results and residuals from previous iterations of the implicit coupling loop are reused and incorporated as columns of the matrices from (2.14), used to estimate the Jacobian of the coupling surface. However, we must indicate a maximum of reused iterations, to avoid having a similar number of rows and columns as we incorporate new columns into the matrices, considering that the number of rows is equal to the number of dof in the coupling surface. For instance, in mesh A there are $21 \times 3 = 63$ dof at the coupling surface, therefore a maximum of 60 reused iterations avoids the problem.
  - To avoid linear dependency between the columns of the interface system, we use filters that remove columns with entries below a threshold during QR decomposition. Two filter types are tested: QR1 and QR2 (for details, see [26] and [37]). Setting a filter requires that we specify a threshold, which we keep in the range $[10^{-14}, 0.1]$. Additionally, we perform tests without filtering.

5.1.2 Reinforced Pipe Fixed at Both Ends and Loaded by Point Forces

The second test case is replicated from one of the examples in [8] using the CalculiX graphic tool cgx. In the reference, an elastic-plastic nonlinear constitutive law with strain hardening is considered for a reinforced pipe structure.

Therefore, we assume that, in order to obtain accurate values for the displacements with this test case, its geometric nonlinearities must be taken into account. For this reason, we use this test case, as explained in Section 5.4, to study the loss of accuracy in the solution that occurs when these nonlinearities are ignored, i.e. if only a linear analysis is performed. The difference between results of a linear and nonlinear analysis is referred to as the error due to nonlinear effects. Then, we compare the error due to nonlinear effects and the error introduced by another source, the splitting of the model. Later on, we test the performance of our implementation with this test case, as presented in Section 5.5.

Description of the Geometry

Consider a cylindrical pipe composed of shell elements, of radius 1 m, 8 m long and with 5 mm thickness. The structure has longitudinal and circular reinforcements, 0.1 m high

and 5 mm thick. Circular reinforcements are situated 1 m from the end of the pipe and 2 m from each other. Longitudinal ones are placed 90° from one another, as seen in Figure 5.2.

The domain is partitioned into two meshes in the same way as the beam; a splitting plane is situated at $z = 6\,\text{m}$ and divides the pipe in **pipe1** and **pipe2**.

The mesh has 5736 nodes and 11264 elements, meaning a total of 17208 dof. The coupling surface has 72 nodes with 216 dof. In the reference article, the mesh has 8664 dof. We maintain the choice of element type from the article, three-node shell elements. Furthermore, in the reference, elements are distributed irregularly over the surface, in an unstructured mesh. In our reproduction, we do not use this type of mesh, even though CalculiX and preCICE can handle it. The reason is that we need to set a splitting plane that defines two subdomain meshes and a coupling surface, which is easier with a structured mesh. The distribution of nodes and elements among meshes can be seen in Table 5.1.

Boundary and Initial Conditions

Boundary conditions are applied to node sets as shown in Figure 5.2.

- Dirichlet BCs: the pipe is fixed at both ends.
- Neumann BCs: a mechanical load is applied onto a small set of nodes close to one end of the pipe, and directed towards the inside. The total force applied is $F = 2 \times 10^5\,\text{N}$, distributed among the nodes. The loaded region is located in **pipe2**. In this case, the load is constant throughout the simulation, as in the reference example.
- Initial conditions: U_0 and $\dot{U}_0$ are both zero in the entire domain.

Parameters and Configuration

With respect to the beam test case, we make the following changes and additions to the configuration:

- Shell elements require that we specify the thickness of the material. Nodal thickness is set to 5 mm.
- In [8], the critical time step for the nonlinear case is $\Delta t = 2.84 \times 10^{-7}\,\text{s}$. Defined by the Courant condition, this value depends on element size. However, we take it as a reference and perform tests with $\Delta t \in [10^{-8}, 10^{-6}]\text{s}$.

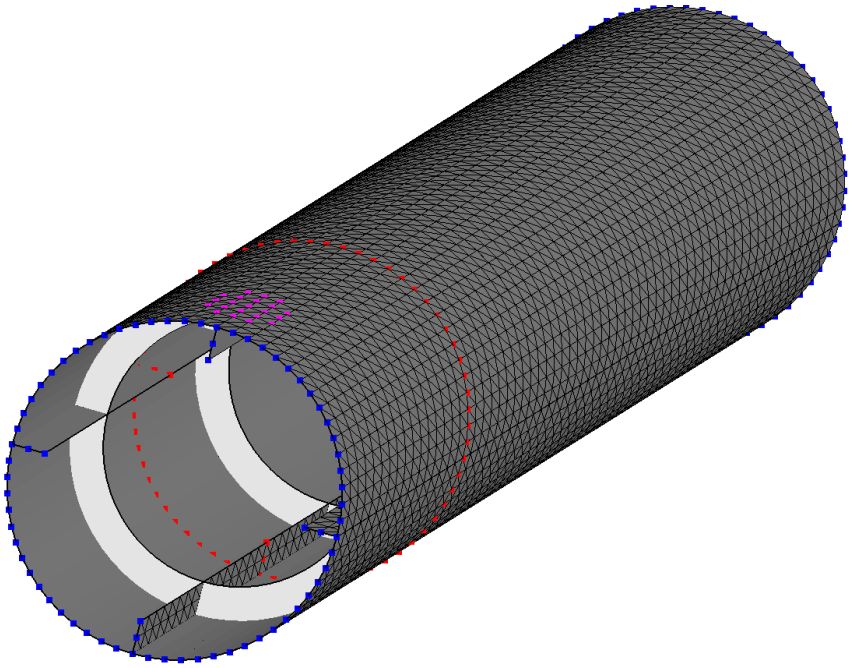

FIGURE 5.2: Mesh of the reinforced pipe test case. In blue: fixed nodes. In magenta: loaded nodes. In red: nodes of coupling surface at $z = 6\,\text{m}$. Elements are shown as well, with edges in black. Note the structured mesh and the reinforcements on the inside; circular reinforcements are displayed in a lighter colour.

- The number of time steps is 10000.
- We incorporate, as a variable, considering geometric nonlinearities or not.
- The relative convergence measure limit is set to 2×10^{-14}. The choice of such a strict convergence measure is due to the fine time scale, which allows us to reduce the error by setting a smaller convergence measure than in the beam test case.

5.2 Validation of the Coupling

We begin this section with the definition of the error measures used in the numerical studies, and then give the results of initial validation experiments. We have compared the displacement fields obtained with our implementation against those obtained with a monolithic simulation. Both simulations are configured with equal parameters. Naturally, the partitioned one needs additional coupling configuration.

To quantify the error between two displacement fields, we compute the magnitude of the difference at each node with an ℓ^2-norm, then we compute another ℓ^2-norm over all nodes,

this is:

$$\begin{aligned} \mathbf{d}_i &= \mathbf{u}_{part,i} - \mathbf{u}_{mono,i} \\ e &= \sqrt{\sum_i (d_{i,x}^2 + d_{i,y}^2 + d_{i,z}^2)} \end{aligned} \tag{5.1}$$

where $u_{part,i}$ and $u_{mono,i}$ are the displacement vectors at node i in the partitioned and monolithic solution, respectively; d_i are local differences, and e is the absolute error measure. This measure is used throughout the experiments in the project.

We perform validation tests with the beam test case and the following parameters:

- Material properties corresponding to steel:
  - Density $\rho = 7800\,\mathrm{kg\,m^{-3}}$.
  - Young modulus $E = 2.1 \times 10^5\,\mathrm{N\,mm^{-2}}$.
  - Poisson ratio $\nu = 0.3$.
- Time step length $\Delta t = 0.01\,\mathrm{s}$.
- Relative convergence measure for implicit coupling sub-iterations: 10^{-4} both for displacements and forces.

Firstly, we confirm the continuity of displacements at the coupling surface, i.e that values at the interface nodes in both **beam1** and **beam2** are equal. In Figure 5.3 we show a comparison between the maximum error magnitude and the smallest magnitude of the displacements, at every time step, at the coupling surface. We ascertain that the error is negligible, and consequently, continuity is confirmed.

After the comparison of values at the coupling surface, we compare displacements in the entire domain at $t = 0.5\,\mathrm{s}$. Observations from Figure 5.4 show that the implementation of structure-structure coupling is validated, however, the difference we observe in Figure 5.5 points towards an error introduced by the lack of numerical precision in the splitting, to which we refer in the following sections. Nevertheless, we confirm that the perturbation from the loading in **beam2** is transmitted to **beam1**, and the resulting displacement field has a similar profile to that of a monolithic simulation.

5.3 Parameter Dependence Studies

Baseline parameters for parameter dependence studies with the beam test case are:

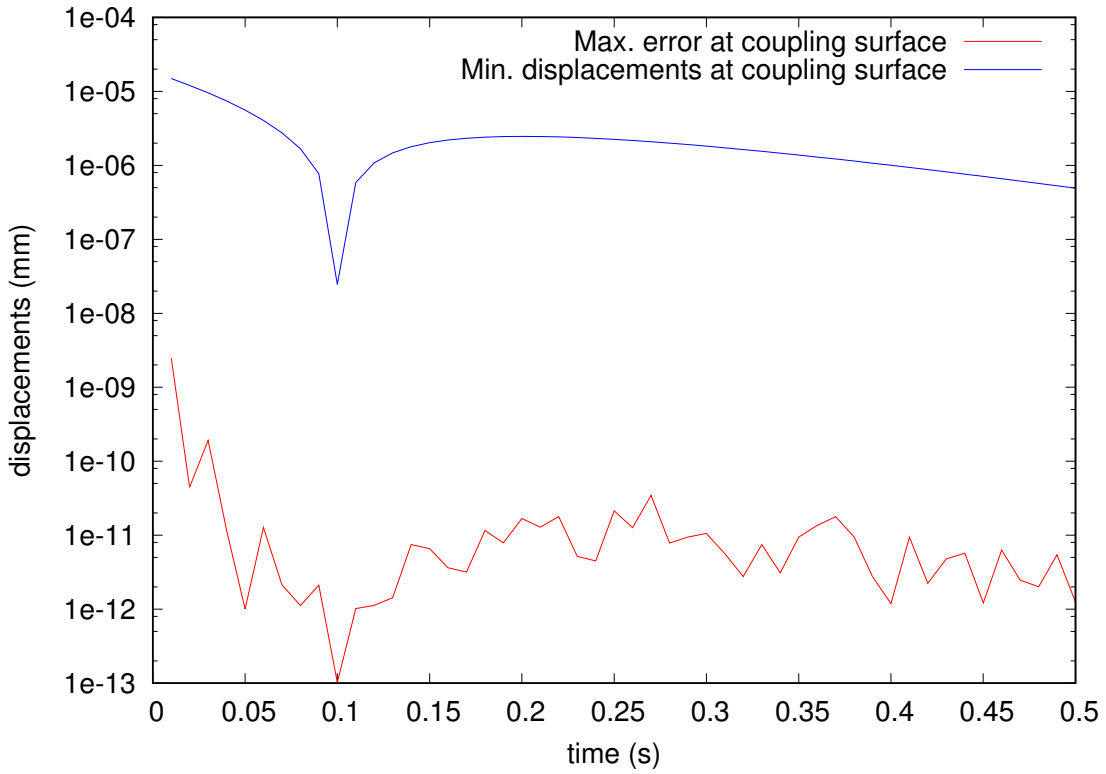

FIGURE 5.3: In red, we show an upper bound for local errors at the interface in the partitioned simulation, $\max_i(||u_{i,beam1} - u_{i,beam2}||_{l^2})$, where i refers to the nodes in the coupling surface, and u is the displacement in one node. We compare it with a lower bound of the displacements at the interface in a reference monolithic simulation, $min_i(||u_{i,mono}||_{l^2})$, shown in blue. The error is negligible with respect to the values of displacements, thus, we confirm the continuity of displacements through the interface.

- Material parameters for steel, as in Section 5.2.
- $\Delta t = 5 \times 10^{-3}$ s and $N_{steps} = 100$.
- Relative convergence measure for implicit coupling: 10^{-4} for both displacements and forces.
- Quasi-Newton inverse least-squares post-processing, with filter type QR1 and filter limit set to 10^{-4}.

We study the variation in the number of iterations required for the implicit coupling scheme to converge, and we find the range of physical parameters where the simulation converges. Besides, we identify optimal parameters, where the simulation converges in a minimum number of iterations. Results are presented in Table 5.2. We see that ranges of values that correspond to real-world solid materials are found inside the regions of the parameter space that lead to convergence in the simulations. Values corresponding to steel show an optimal performance.

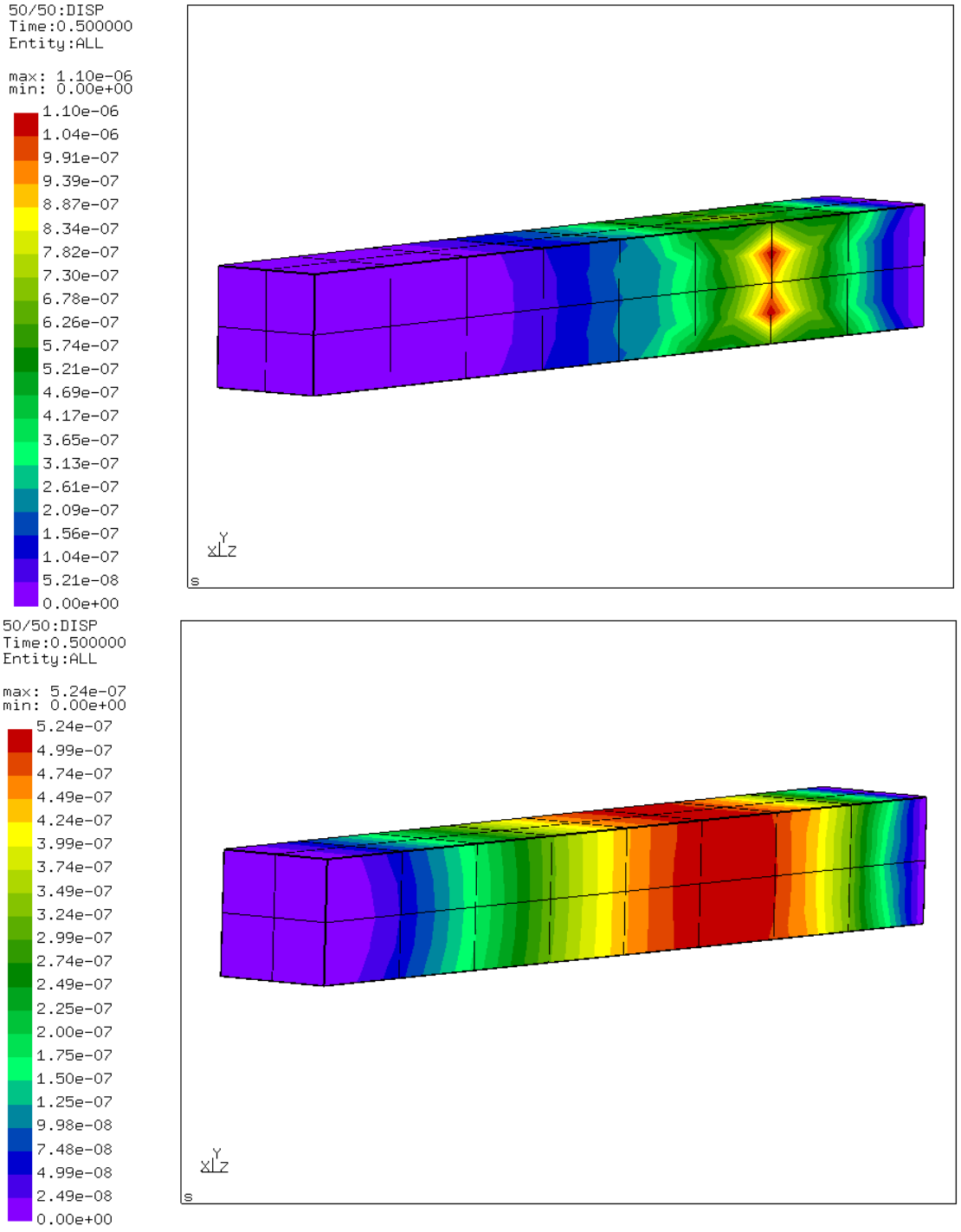

FIGURE 5.4: Magnitude of nodal displacements (in mm) at $t = 0.5\,\mathrm{s}$ for the beam test case. Below, results running one CalculiX instance on the entire domain, i.e. a monolithic simulation, and above, results of a bipartite simulation with two CalculiX instances, coupled using the preCICE-CalculiX adapter. We observe a factor of 2 between the maximum magnitudes of both beams, with the gradient in the partitioned simulation being more abrupt.

Next, we study the effects of filtering in the performance of quasi-Newton schemes, in terms of the mean coupling iterations needed to converge in each time step, for several time step lengths. For this, we vary filter type and limit.

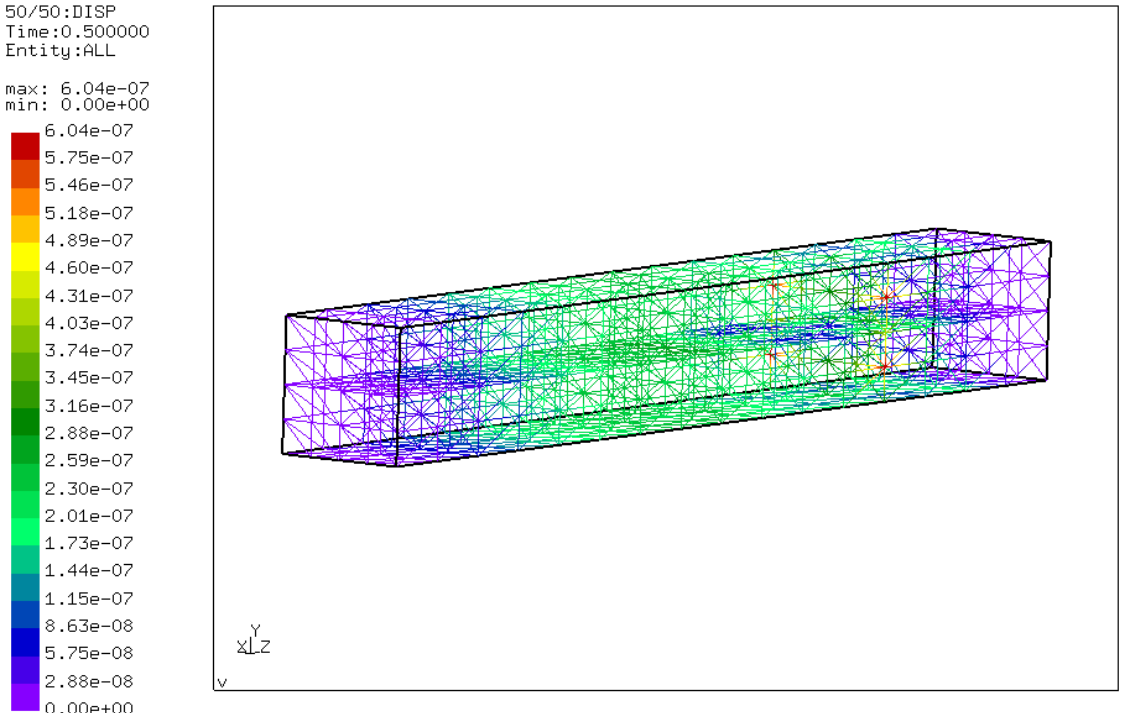

FIGURE 5.5: Visualization of the difference (in mm) between displacement fields obtained with a partitioned and a monolithic simulation. We show the inside of the model, where the error peaks at four regions near the coupling surface.

ρ ($\times$ 7800 kg m^{-3})	10^6	10^5	10^4	10^2	10	1	10^{-1}	10^{-3}	10^{-5}
Mean iter.	-	33.12	10.58	7.87	7.69	6.82	7.67	7.28	7.3

ρ ($\times$ 7800 kg m^{-3})	10^{-7}	10^{-9}	10^{-10}	10^{-11}	10^{-13}	10^{-15}	10^{-16}	10^{-17}
Mean iter.	8.18	11.05	10.08	21.38	18.96	24.55	76.83	-

E ($\times$ 210 000 N mm^{-2})	0.001	0.01	0.1	1	10	100	1000
Mean iter.	-	7.94	7.37	6.82	7.68	7.29	-

ν	0.1	0.2	0.3	0.4	0.5
Mean iter.	7.5	7.77	6.82	7.25	-

TABLE 5.2: Dependence of mean coupling iterations per time step, with respect to density ρ, Young modulus E and Poisson ratio ν. Parameter ranges corresponding to real-world solid materials are near $\rho = 7800\,\mathrm{kg\,m^{-3}}$, $E = 210\,000\,\mathrm{N\,mm^{-2}}$, and $\nu = 0.3$; optimal convergence is achieved for these parameters. No result is shown for those parameter values that do not converge.

We observe a larger amount of iterations in the initial 2-3 time steps, which translates into an increment of the mean iterations. This occurs also if, instead of applying the load only in $t \in [0, 0.01]$ s, it is applied constantly throughout the simulation. Since we fix the simulation time and let the number of steps vary, this increment in mean iterations

depends on time step length. In order to avoid this perturbation, we only consider the last half of the simulation for the calculation of mean coupling iterations. A comparison between filter types can be seen in Table 5.3, and results for several time step lengths in Figure 5.6. Both variants, QR1 and QR2, improve the performance of the quasi-Newton method, reducing the iterations needed for convergence; QR2 is the more robust one. We observe that the influence of filtering is highly dependent on time step length: it is stronger for coarser time steps, and weaker for the finer ones.

Filter limit		10^{-12}	10^{-11}	10^{-10}	10^{-9}	10^{-8}	10^{-7}
Filter type	QR1	7.48	6.52	6.78	6.40	6.72	7.34
	QR2	7.48	7.48	6.52	6.54	6.40	6.72

Filter limit		10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}
Filter type	QR1	6.50	44.00	415.36	809.50	-	-
	QR2	6.40	7.08	6.70	6.78	6.90	8.18
No filter		7.48					

TABLE 5.3: Mean coupling iterations per time step for the last half of a 0.5 s simulation with $\Delta t = 5 \times 10^{-3}$ s. We compare two variants of filtering techniques for quasi-Newton methods, QR1 and QR2, using a range of filter limits. When the limit is set to 10^{-12} or lower for the QR1 variant (10^{-11} for QR2), no columns are filtered; therefore, iteration numbers are the same.

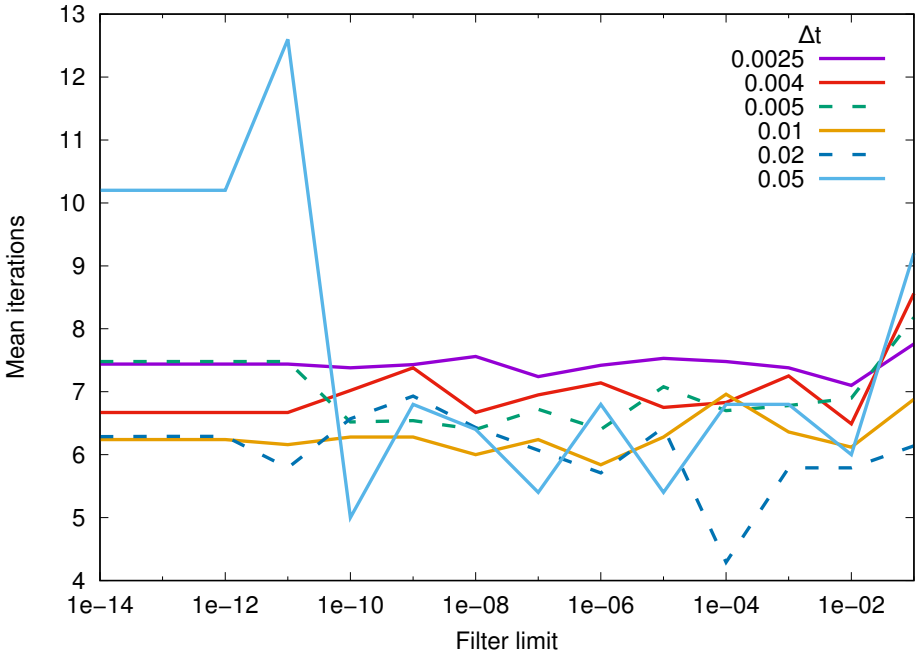

FIGURE 5.6: Mean coupling iterations per time step for the last half of a 0.5 s simulation with $\Delta t \in \{2.5,\ 4,\ 5,\ 10,\ 20,\ 50\} \times 10^{-3}$s. Filtering technique QR2 is used for its stability. As a reference, for filter limits below 10^{-12}, we see the performance of a QN method without filtering.

To finish the parameter dependence studies, we perform tests with meshes of different size. We aim to see the effects of a finer discretization on e and quasi-Newton convergence. In

finer meshes, we increase the maximum reused iterations, since the coupling surface has more dof, and more columns from previous iterations can be kept without it leading to an over-determined system. In Table 5.4, we show an error measure relative to the number of nodes in each mesh of the beam test case, as well as the mean coupling iterations and execution time. The simulation does not converge with the same parameters in the three meshes. The finest mesh requires a different configuration to reach convergence. No overall accuracy improvement is observed for a finer mesh, whereas the coupling is significantly slower.

	e/N_{nodes}(mm)	Mean iterations	Exec. time (s)
Mesh A	4.23×10^{-4}	8.52	210
Mesh B	6.01×10^{-4}	49.66	5414
Mesh C	-	-	-

TABLE 5.4: Absolute ℓ^2-norm of the error (e), measured against the monolithic solution and normalized to the number of nodes, shown for each mesh size of the beam test case. Mean coupling iterations per time step and execution time are shown as well. The simulation does not converge with mesh C if parameters are not changed.

5.4 Study of Error Sources: Splitting Error and Nonlinear Effects

One strategy for performance improvement in simulations of nonlinear phenomena is to perform a linear analysis, ignoring nonlinearities. Another one is the partitioned approach we develop in this thesis: a linear analysis in one subdomain, and a nonlinear analysis in the other one. Both strategies introduce an error: the former lacks nonlinear corrections to the solution, while the latter incorporates a splitting error into the numerical integration method, as we have observed in the previous section.

To see the origin of the splitting error, we study the dependence of the global error on the relative convergence measure limit, on the beam test case. Two error measures are computed for partitioned simulations: with respect to a monolithic reference (e from (5.1)), and with respect to a partitioned simulation with the minimum relative convergence measure limit of the range tested, that is, with respect to the most accurate partitioned simulation. Results are shown in Figure 5.7, for $\Delta t \in \{2.5,\ 4,\ 5,\ 10,\ 20,\ 50\} \times 10^{-3}$s. Both error measures are reduced with the convergence measure in the range $\Delta t \in [0.004, 0.01]$ s. An error offset puts a limit to this reduction, but only for the error measure with respect to the monolithic reference simulation. Thus, we conclude that this limit to the accuracy of our method is introduced by the splitting of the model. We observe that the splitting error decreases with time step length, in the range $\Delta t \in [0.0025, 0.02]$ s.

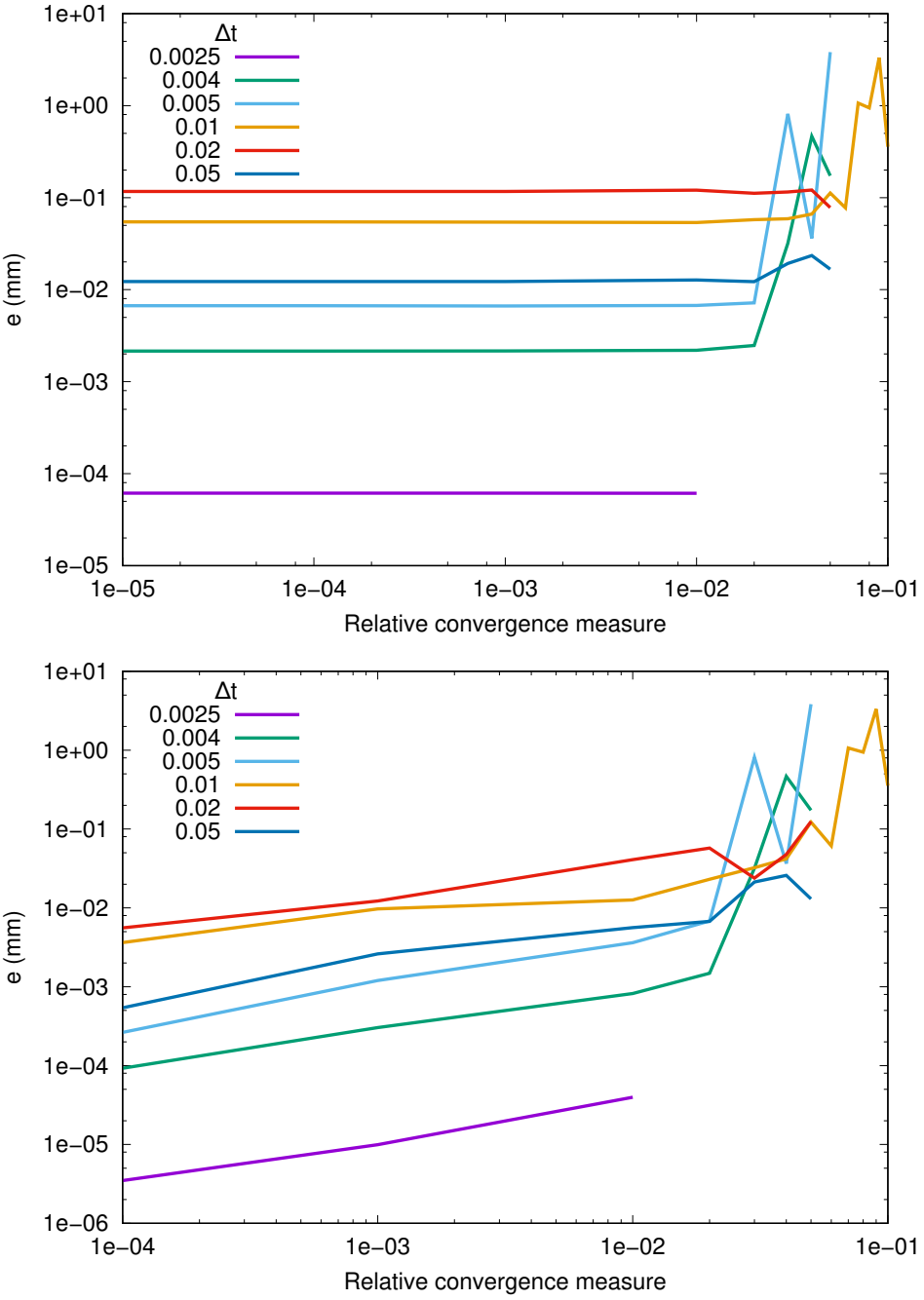

FIGURE 5.7: Dependence of the error, in the partitioned simulation, on the relative convergence measure for implicit coupling, for several Δt. Above, the error is measured against the monolithic reference, below, against the partitioned simulation with the strictest convergence criterion. For $\Delta t \in \{0.004, 0.005, 0.01\}$ we observe a region where the error is reduced proportionally to the relative convergence measure in both plots. Then, an error offset puts a limit to this reduction, but only in the upper plot.

Comparing errors from both splitting and nonlinear effects, we can evaluate the efficiency of our partitioned approach, and whether it is beneficial for accuracy. A partitioned linear-nonlinear approach aims to avoid the accuracy loss of a monolithic linear analysis, but the splitting error can hamper the strategy if it exceeds the error of the linear analysis. In Figure 5.8 we show the temporal evolution of both errors in a run of the nonlinear pipe test case, with $\Delta t = 10^{-7}$s. For the partitioned simulation, we set QR2 filtering with limit 10^{-3}. We observe a dominance of the splitting error, even though nonlinear effects grow with time. Increasing the number of time steps in the simulation does not make the error due to nonlinear effects surpass the splitting error, as the latter escalates faster. For $\Delta t = 10^{-8}$s, results do not change.

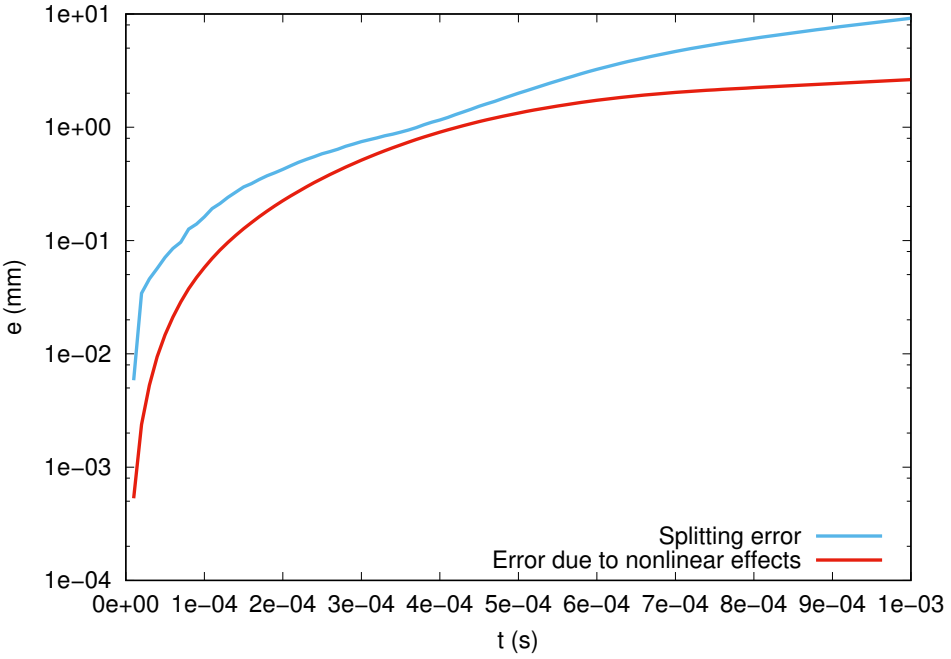

FIGURE 5.8: Error measure e, plotted against time and computed using a monolithic nonlinear simulation as a reference. It is shown for a partitioned simulation with nonlinear analyses running on both solvers (blue) and a monolithic linear simulation without nonlinear corrections (red). For the monolithic linear simulations, e is identified as nonlinear corrections. For the partitioned nonlinear simulation, as the splitting error. We see how the latter is larger than nonlinear corrections during all the simulation, and the difference between the two quantities increases over time.

Considering that the source of the splitting error lies in the low order of the approximation we make in the exchange of variables, we attempt to reduce it via a more sophisticated interpolation of displacements between time steps. Concretely, we take $\frac{1}{2}(\mathbf{U}_{i-1} + \mathbf{U}_i)$, instead of $\mathbf{U}_i$ where the new values for one solver in time step i come from the other solver. Error measures show that the error is reduced a 0.0002%, therefore we deem this modification ineffective.

From the experiments in this section, we conclude that the current scenario is only weakly nonlinear. However, we expect a different situation for another test case.

5.5 Performance Study

We study the performance of partitioned and monolithic setups of the pipe test case. All simulations are configured to run with $\Delta t = 10^{-7}$s. As in the previous section, for the partitioned setup we use the QR2 filtering with limit 10^{-3}. Monolithic and partitioned simulations run on one and two ranks, respectively; with one dedicated rank for each participant in the partitioned case. We run all tests on a laptop with an 4-core processor at 2.3 GHz and 16 GB of RAM; running Ubuntu Linux 16.04.

We show a comparison of execution times for all setups in Table 5.5. The partitioned setup needs an average of 3.71 coupling iterations per time step to converge; together with a smaller contribution from communication tasks, this produces an overhead, which makes the partitioned setup 2.5 times slower than any of the monolithic ones. Consider that the partitioned setup follows a parallel implicit scheme, meaning that both participants run simultaneously. Timing measures in the partitioned simulation show that the nonlinear

Setup		Exec. time (min)
Monolithic	Linear	349
	Nonlinear	351
Partitioned Linear-nonlinear		879

TABLE 5.5: Execution times for several simulation setups, with $\Delta t = 10^{-7}$s and 10000 time steps.

participant, **CalculiX2**, spends 652 min in *advance*; while the linear participant, **CalculiX1**, only spends 12 s in it. The time spent by a solver in this function is an indicator of the time it spends waiting for the other one. For **CalculiX2**, this represents 74% of its total runtime with its assigned processor in an idle state, while it waits for **CalculiX1** to obtain the solution for the current time step. This result does not meet our expectation of a nonlinear simulation that is more expensive than a linear one, and leads to poor load balancing.

Therefore, in absence of stronger nonlinear effects, to achieve a better load balancing and be able to benefit from a partitioned approach, we need to assign more processors to the linear participant. Additionally, we need a linear domain that is much larger than the nonlinear one. In that way, the difference in execution time between linear and nonlinear simulations has more weight in the execution time of the partitioned simulation. Consequently, the overhead from coupling operations is relatively lower, which makes the

partitioned setup more efficient. However, the necessary ratio of domain sizes makes this solution impractical for the current setting.

Thus, we conclude that a partitioned linear-nonlinear setup of the reinforced pipe test case does not improve the overall efficiency of the simulation.

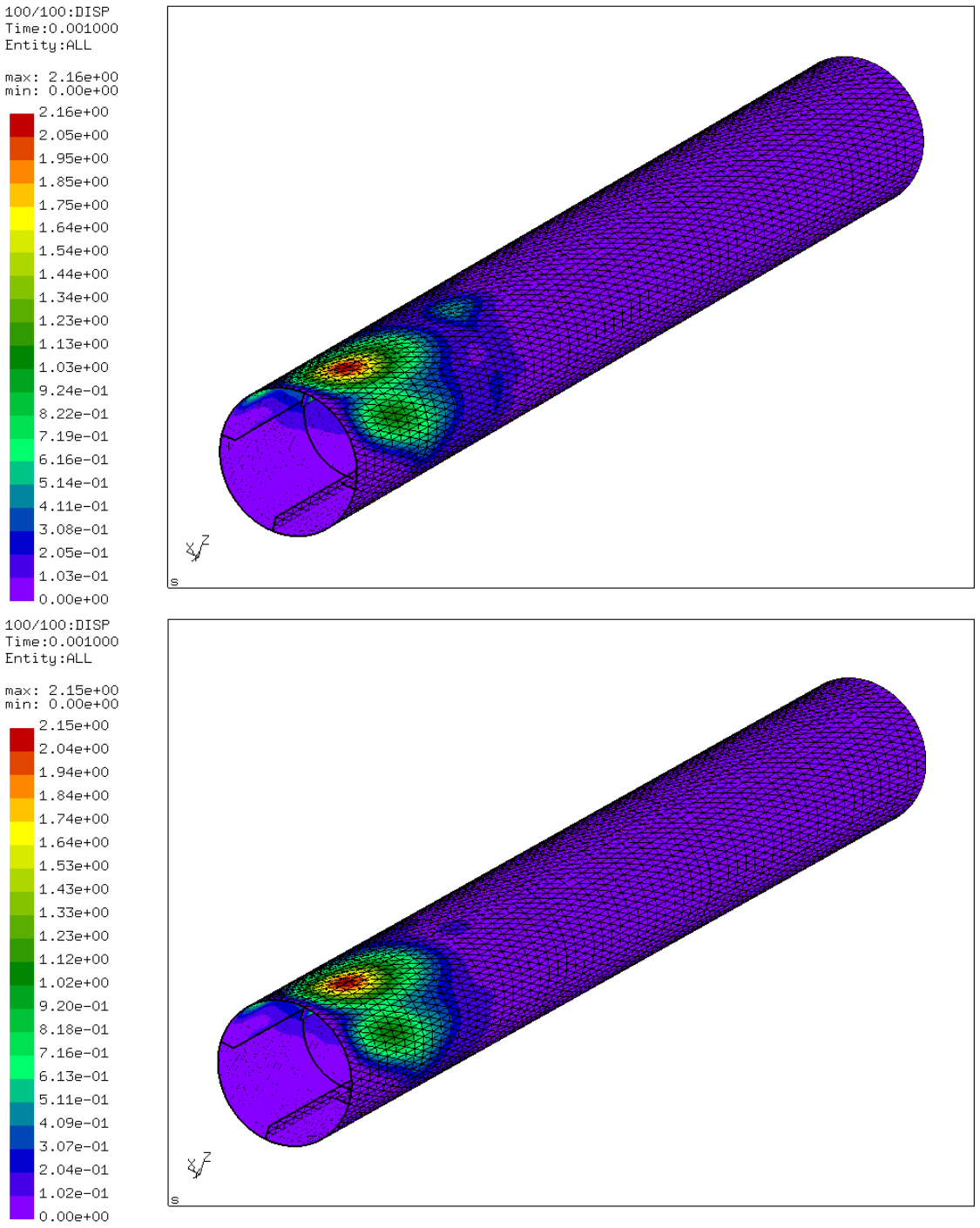

FIGURE 5.9: Magnitude of nodal displacements (in mm) at $t = 10^{-3}$s for a reinforced pipe in a partitioned linear-nonlinear (above) and a monolithic nonlinear setup (below).

For illustrative purposes, we plot displacement fields at $t = 10^{-3}$s in Figure 5.9, for the monolithic nonlinear and partitioned linear-nonlinear setup.

This section ends the description of the experimentation for the validation of the implementation, and the studies of parameter dependence, error sources and performance. Based on the results obtained, in the next chapter we draw the conclusions of our project.

Chapter 6

Conclusions

We have extended the range of physical interactions that can be handled by the coupling library preCICE, with the addition of structure-structure coupling functionality into the preCICE adapter for the structural FEM solver CalculiX. In order to test this functionality, we have set up partitioned simulations in which two instances of CalculiX are coupled. For each instance, it is possible to choose whether geometric nonlinearities are taken into account or not. To guarantee stability of the partitioned approach, we sub-iterate between both structure solvers in every time step, which we refer to as implicit coupling.

As a starting point, we have validated our implementation with two test cases: a beam and a reinforced pipe fixed at both ends and loaded by point forces. In order to do this, we have defined and measured a global error in the displacement field. This error quantifies the difference between the solution of monolithic and partitioned simulations. All combinations of analysis types (linear-linear, linear-nonlinear, and nonlinear-nonlinear) are found to be stable and converge to valid solutions. Next, we have identified a lower bound for the global error. This limit can be seen as we reduce the convergence criteria of the implicit coupling, and depends on the time step length. It has been recognized as a numerical error caused by the time splitting; thus, we refer to it as splitting error. Better time splitting methods are expected to reduce this error. Currently, research is being carried out to incorporate them into preCICE [39].

Furthermore, we have studied numerical behaviour over a range of physical parameters; as well as the effect of tuning several numerical parameters to improve performance. Simulations converge for a range of physical parameters comprising those that correspond to realistic solid materials. Quasi-Newton methods with filtering techniques have proved useful to improve the convergence of implicit coupling schemes, provided that their parameters are correctly tuned.

Additionally, we have considered the scaling of the partitioned approach, constructing a large parallel setup with each participant of the simulation running on one rank, with the definition of additional coupling surfaces for adjacent subdomains. The reason is to overcome the limitation of shared-memory parallelization in CalculiX. Nevertheless, due to lack of time, we leave this step for future efforts.

Our main motivation for the implementation of structure-structure coupling is the potential performance benefit yielded by a partitioned simulation. Nonlinear treatment can be restricted to a small subdomain that presents nonlinearities, keeping the rest of the simulation linear, and therefore, computationally cheaper. However, nonlinear effects observed in the pipe test case are not strong enough to cause a significant difference between linear and nonlinear simulations, neither in the values of the solution nor in performance.

In particular, the weak nonlinear effects pose two issues for our test case:

- Firstly, the splitting error dominates the global error as it surpasses the error due to nonlinear effects. Therefore, a partitioned linear-nonlinear simulation does not produce a more accurate solution than a monolithic linear one.
- Secondly, a linear analysis does not result in a significantly faster simulation, since the nonlinear implicit integrator of CalculiX converges almost as fast. Thus, considering the overhead due to implicit coupling iterations, the difference in execution time between linear and nonlinear solver is not enough to make the partitioned approach faster than a monolithic nonlinear simulation.

To sum up, for a partitioned approach to give benefits, it must be applied to a strongly nonlinear case. In such a case, the increase in execution time as a result of taking into account nonlinear effects is presumably larger than the coupling overhead, and the error due to nonlinear effects is presumably larger than the splitting error. Our pipe test case is a reproduction from a model in [8]. In that work, it is considered with a nonlinear constitutive law, thus presenting a strong nonlinearity that justifies a partitioned analysis. However, in the time frame of our project, we have been unable to reproduce entirely the nonlinear behaviour of the test case with CalculiX.

In future work, we consider the addition of the option *PLASTIC in the CalculiX input file, to work with an elastic plastic model instead of simply a linear elastic one, as we have done in this project. The structural simulation with this parameter might exhibit the aforementioned nonlinear effects with enough intensity.

Acknowledgements

First of all, I would like to show my most sincere gratitude towards my thesis advisor Benjamin Uekermann, for his guidance and careful explanations, and for giving me the opportunity of being part of the development of preCICE.

Secondly, I would also like to thank the people who collaborate in making such a great open-source tool: for their brilliant work, and for the comments and ideas provided.

For the help in getting started with the adapter code, I also want to thank Alexander Rusch.

Besides, a special mention to my family, for their unconditional and constant support throughout these months.

And last but not least, my gratitude also goes to those friends who always asked me about the thesis work and, through this, helped me organize my ideas.

Annex A.1: Sample preCICE Configuration xml File

```
<?xml version="1.0" encoding='UTF-8'?>

<precice-configuration xmlns:data="data" xmlns:coupling-scheme="coupling-scheme" xmlns:mapping="mappi

   <solver-interface dimensions="3">
      <data:vector name="Forces0"/>
      <data:vector name="Displacements0"/>

      <mesh name="Calculix_Mesh1">
         <use-data name="Forces0"/>
         <use-data name="Displacements0"/>
      </mesh>

      <mesh name="Calculix_Mesh2">
         <use-data name="Displacements0"/>
         <use-data name="Forces0"/>
      </mesh>

      <participant name="Calculix1">
         <use-mesh name="Calculix_Mesh1" provide="yes"/>
                 <use-mesh name="Calculix_Mesh2" from="Calculix2"/>
         <write-data name="Forces0" mesh="Calculix_Mesh1"/>
         <read-data  name="Displacements0"      mesh="Calculix_Mesh1"/>
         <mapping:nearest-neighbor
            direction="write" from="Calculix_Mesh1" to="Calculix_Mesh2"
            constraint="conservative" timing="initial"/>
         <mapping:nearest-neighbor
                direction="read" from="Calculix_Mesh2" to="Calculix_Mesh1"
            constraint="consistent" timing="initial"/>
      </participant>

      <participant name="Calculix2">
         <use-mesh name="Calculix_Mesh2" provide="yes"/>
         <write-data name="Displacements0" mesh="Calculix_Mesh2"/>
         <read-data  name="Forces0"      mesh="Calculix_Mesh2"/>
      </participant>

      <m2n:sockets from="Calculix1" to="Calculix2"  exchange-directory="../" distribution-type="gathe

      <coupling-scheme:parallel-implicit>
           <participants first="Calculix1" second="Calculix2"/>
           <max-timesteps value="100"/>
           <timestep-length value="5e-3"/>
                   <exchange data="Displacements0" mesh="Calculix_Mesh2" from="Calculix2" to="Calculi
           <exchange data="Forces0" mesh="Calculix_Mesh2" from="Calculix1" to="Calculix2"/>

           <relative-convergence-measure limit="1e-4" data="Displacements0" mesh="Calculix_Mesh2"/>
           <relative-convergence-measure limit="1e-4" data="Forces0" mesh="Calculix_Mesh2"/>

           <post-processing:IQN-ILS>
            <data name="Displacements0" mesh="Calculix_Mesh2"/>
            <data name="Forces0" mesh="Calculix_Mesh2"/>
            <preconditioner type="residual-sum"/>
```

```
            <filter type="QR2" limit="1e-4"/>
            <initial-relaxation value="0.1"/>
            <max-used-iterations value="60"/>
            <timesteps-reused value="10"/>
          </post-processing:IQN-ILS>

      </coupling-scheme:parallel-implicit>

   </solver-interface>

</precice-configuration>
```

Annex A.2: Sample Adapter Configuration YAML File

```
participants:

    Calculix1:
        interfaces:
        - nodes-mesh: Calculix_Mesh1
          patch: surface
          read-data: [Displacements0]
          write-data: [Forces0]

precice-config-file: ../precice-config.xml
```

Annex B: Sample CalculiX Input File

```
**
**   Structure: cantilever beam.
**
**
*INCLUDE, INPUT=CalculixMesh/all.msh
*INCLUDE, INPUT=CalculixMesh/fix1_beam.nam
*BOUNDARY
FIX, 1
*BOUNDARY
FIX, 2
*BOUNDARY
FIX, 3
*NSET,NSET=Nload
185, 186, 187, 188, 189
*MATERIAL,NAME=EL
*ELASTIC
 210000.0,         .3
*DENSITY
7.8E-9
*SOLID SECTION,ELSET=Eall,MATERIAL=EL
*AMPLITUDE, NAME=A1
0., 1.,
.01, 0.,
5., 0.
*STEP, INC=10000
*DYNAMIC
1E-2, 0.5, 1E-2, 1E-2
*CLOAD, AMPLITUDE=A1
Nload, 2, -.1
*NODE FILE
U
*NODE PRINT,NSET=Nall
U,RF
*END STEP
```

Bibliography

[1] Guido Dhondt and Klaus Wittig. CALCULIX – A Free Software Three-Dimensional Structural Finite Element Program. http://www.calculix.de/.

[2] preCICE – A coupling library for partitioned multi-physics simulation. http://www.precice.org/, .

[3] 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 and Fluids*, 141:250–258, 2016. ISSN 0045-7930. doi: https://doi.org/10.1016/j.compfluid.2016.04.003. URL http://www.sciencedirect.com/science/article/pii/S0045793016300974. Advances in Fluid-Structure Interaction.

[4] preCICE-adapter for the CSM code CalculiX. https://github.com/precice/calculix-adapter.

[5] David E Keyes, Lois C McInnes, Carol Woodward, William Gropp, Eric Myra, Michael Pernice, John Bell, Jed Brown, Alain Clo, Jeffrey Connors, et al. Multiphysics simulations: Challenges and opportunities. *The International Journal of High Performance Computing Applications*, 27(1):4–83, 2013.

[6] Önder Babur, Tom Verhoeff, and Mark van den Brand. Multiphysics and multiscale software frameworks: an annotated bibliography. *Computer Science Reports*, 1501, 2015. ISSN 0926-4515.

[7] Nathan M Newmark. A method of computation for structural dynamics. *Journal of the engineering mechanics division*, 85(3):67–94, 1959.

[8] Anthony Gravouil and Alain Combescure. Multi-time-step explicit–implicit method for non-linear structural dynamics. *International Journal for Numerical Methods in Engineering*, 50(1):199–225, 2001.

[9] Anthony Gravouil and Alain Combescure. Multi-time-step and two-scale domain decomposition method for non-linear structural dynamics. *International Journal for Numerical Methods in Engineering*, 58(10):1545–1569, 2003.

[10] Alain Combescure and Anthony Gravouil. A numerical scheme to couple subdomains with different time-steps for predominantly linear transient analysis. *Computer methods in applied mechanics and engineering*, 191(11-12):1129–1157, 2002.

[11] Vincent Faucher and Alain Combescure. A time and space mortar method for coupling linear modal subdomains and non-linear subdomains in explicit structural dynamics. *Computer methods in applied mechanics and engineering*, 192(5-6):509–533, 2003.

[12] B Herry, L Di Valentin, and Alain Combescure. An approach to the connection between subdomains with non-matching meshes for transient mechanical analysis. *International Journal for Numerical Methods in Engineering*, 55(8):973–1003, 2002.

[13] P.J. Blanco, R.A. Feijóo, and S.A. Urquiza. A variational approach for coupling kinematically incompatible structural models. *Computer Methods in Applied Mechanics and Engineering*, 197:1577–1602, 2007. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2007.12.001.

[14] Assaf Ghanem, Mohamed Torkhani, Najib Mahjoubi, TN Baranger, and Alain Combescure. Arlequin framework for multi-model, multi-time scale and heterogeneous time integrators for structural transient dynamics. *Computer methods in applied mechanics and engineering*, 254:292–308, 2013.

[15] A Prakash and KD Hjelmstad. A feti-based multi-time-step coupling method for newmark schemes in structural dynamics. *International Journal for Numerical Methods in Engineering*, 61(13):2183–2204, 2004.

[16] Ali Cagatay Cobanoglu, Simon Möß ner, Majid Hojjat, and Fabian Duddeck. Model order reduction methods for explicit fem. *Science in the Age of Experience*, 2016.

[17] Arun Prakash, Ertugrul Taciroglu, and Keith D Hjelmstad. Computationally efficient multi-time-step method for partitioned time integration of highly nonlinear structural dynamics. *Computers & Structures*, 133:51–63, 2014.

[18] Carlos A Felippa, KC Park, and Charbel Farhat. Partitioned analysis of coupled mechanical systems. *Computer methods in applied mechanics and engineering*, 190 (24-25):3247–3270, 2001.

[19] Serge Piperno and Charbel Farhat. Partitioned procedures for the transient solution of coupled aeroelastic problems–part ii: energy transfer analysis and three-dimensional applications. *Computer methods in applied mechanics and engineering*, 190(24-25):3147–3170, 2001.

[20] Miguel Ángel Fernández and Marwan Moubachir. A newton method using exact jacobians for solving fluid–structure coupling. *Computers & Structures*, 83(2-3):127–142, 2005.

[21] Hermann G Matthies and Jan Steindorf. Partitioned but strongly coupled iteration schemes for nonlinear fluid–structure interaction. *Computers & structures*, 80(27-30): 1991–1999, 2002.

[22] Lionel Gendre, Olivier Allix, Pierre Gosselet, and François Comte. Non-intrusive and exact global/local techniques for structural problems with local plasticity. *Computational Mechanics*, 44(2):233–245, 2009.

[23] 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(11-12):793–801, 2009.

[24] Joris Degroote and Jan Vierendeels. Multi-level quasi-newton coupling algorithms for the partitioned simulation of fluid–structure interaction. *Computer Methods in Applied Mechanics and Engineering*, 225:14–27, 2012.

[25] 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.

[26] R Haelterman, Alfred EJ Bogaers, K Scheufele, B Uekermann, and M Mehl. Improving the performance of the partitioned qn-ils procedure for fluid–structure interaction problems: Filtering. *Computers & Structures*, 171:9–17, 2016.

[27] AH van Zuijlen, S Bosscher, and H Bijl. Two level algorithms for partitioned fluid–structure interaction computations. *Computer Methods in Applied Mechanics and Engineering*, 196(8):1458–1470, 2007.

[28] Jan Vierendeels, Lieve Lanoye, Joris Degroote, and Pascal Verdonck. Implicit coupling of partitioned fluid–structure interaction problems with reduced order models. *Computers & structures*, 85(11-14):970–976, 2007.

[29] Lionel Gendre, Olivier Allix, and Pierre Gosselet. A two-scale approximation of the schur complement and its use for non-intrusive coupling. *International Journal for Numerical Methods in Engineering*, 87(9):889–905, 2011.

[30] 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(42-44):4506–4527, 2005.

[31] Sergio R Idelsohn, Facundo Del Pin, Riccardo Rossi, and Eugenio Oñate. Fluid–structure interaction problems with strong added-mass effect. *International journal for numerical methods in engineering*, 80(10):1261–1294, 2009.

[32] Guido Dhondt. *The finite element method for three-dimensional thermomechanical applications*. John Wiley & Sons, 2004.

[33] Isidoro Miranda, Robert M. Ferencz, and Thomas J. R. Hughes. An improved implicit-explicit time integration method for structural dynamics. *Earthquake Engineering & Structural Dynamics*, 18(5):643–653, 1989. doi: 10.1002/eqe.4290180505. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/eqe.4290180505.

[34] Hans M Hilber, Thomas JR Hughes, and Robert L Taylor. Improved numerical dissipation for time integration algorithms in structural dynamics. *Earthquake Engineering & Structural Dynamics*, 5(3):283–292, 1977.

[35] Guido Dhondt. *CalculiX CrunchiX USERS MANUAL version 2.13*, 2017.

[36] T. Belytschko and Y.Y. Lu. Stability analysis of elemental explicit-implicit partitions by fourier methods. *Computer Methods in Applied Mechanics and Engineering*, 95: 87–96, 1992. ISSN 0045-7825.

[37] Benjamin Walter Uekermann. *Partitioned fluid-structure interaction on massively parallel systems*. PhD thesis, Technische Universität München, 2016.

[38] The preCICE Wiki. https://github.com/precice/precice/wiki, .

[39] Benjamin Rüth, Benjamin Uekermann, Miriam Mehl, and Hans-Joachim Bungartz. Time stepping algorithms for partitioned multi-scale multi-physics in preCICE. In *6th European Conference on Computational Mechanics, 7th European Conference on Computational Fluid Dynamics*, 2018.

[40] Benjamin Uekermann, Hans-Joachim Bungartz, Lucia Cheung Yau, Gerasimos Chourdakis, and Alexander Rusch. Official precice adapters for standard open-source solvers. In *Proceedings of the 7th GACM Colloquium on Computational Mechanics for Young Scientists from Academia*, 2017.